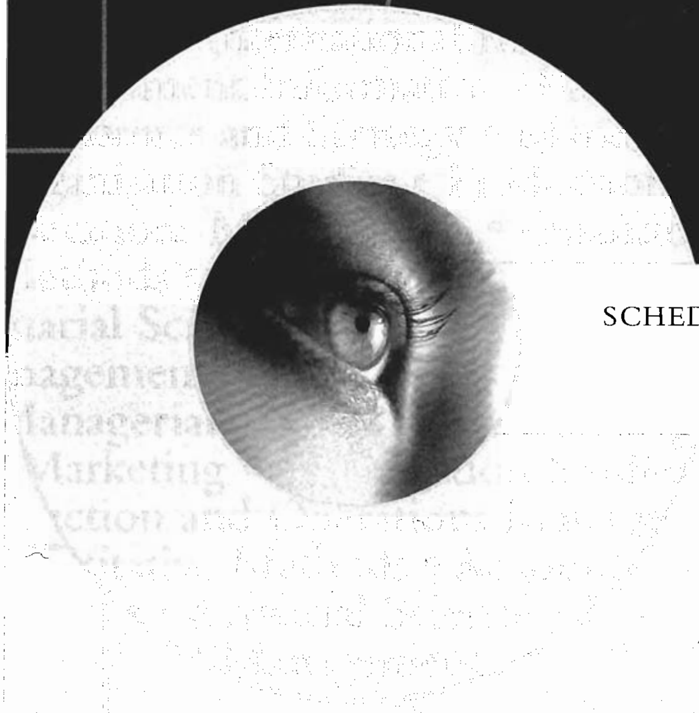
DEPARTMENT OF APPLIED ECONOMICS



RESEARCH REPORT

SCHEDULING TRAINEES AT A HOSPITAL DEPARTMENT USING A BRANCH-AND-PRICE APPROACH

JEROEN BELIEN & ERIK DEMEULEMEESTER

OR 0403

Scheduling trainees at a hospital department using a branch-and-price approach

Jeroen Belien[1], **Erik Demeulemeester**[1]

[1]Department of Applied Economics, K.U.Leuven, Belgium
e-mail: <first name>.<family name>@econ.kuleuven.ac.be

Abstract

Scheduling trainees (graduate students) is a complicated problem that has to be solved frequently in many hospital departments. We will describe a trainee scheduling problem encountered in practice (at the ophthalmology department of the university hospital Gasthuisberg, Leuven). In this problem a department has a certain number of trainees at its disposal, which assist specialists in their activities (surgery, consultation, etc.). For each trainee one has to schedule the activities in which (s)he will assist during a certain time horizon, usually one year. Typically, these kind of scheduling problems are characterized by both hard and soft constraints. The hard constraints consist of both work covering constraints and formation requirements, whereas the soft constraints include trainees' preferences and setup restrictions. In this paper we will describe an exact branch-and-price method to solve the problem to optimality.

Key words: Staff scheduling, branch-and-price, health care

Introduction

The problem of scheduling medical trainees to perform a number of activities over a given time horizon is addressed. Recently, a very good bibliographic survey on medical staff rostering problems has appeared (Cheang et al (2002)). Several studies in the literature have utilized mathematical programming techniques to assist in finding efficient staff schedules (see e.g. Warner (1976) or Beaumont (1997)). The main problem of these integer programs lies in the large computation times needed for many practical instances, even to obtain just a feasible solution. To overcome this problem both heuristic approaches and exact approaches that exploit specific features of the problem structure have been developed and described for plenty of staff scheduling applications. The most important heuristic approaches include simulated annealing (e.g. Brusco and Jacobs (1993)), tabu search (e.g. Burke et al. (1998)) and genetic algorithms (e.g. Aickelin and Dowsland (2000)). Regarding exact approaches we can distinguish between constraint programming, branch-and-bound (see e.g. Bosi and Milano (2001)) and branch-and-price approaches. Branch-and-price, a technique in which bounds are calculated by solving the LP relaxation of the problem using a column generation scheme, has gained considerable attention during the last decade.

Most of the encountered scheduling problems studied in the literature are short-term shift scheduling problems involving some kind of set covering or set partitioning formulation (e.g. Mason and Smith (1998)). Alternatively, 0-1 multi-commodity flow formulations are proposed (e.g. Cappanera and Gallo (2001)). To the best of our knowledge all branch-and-price approaches for staff scheduling problems decompose on staff members, i.e. generate columns per staff member (see e.g. Mehrotra et al (2000)). In contrast, we study a long-term scheduling problem for which we propose a decomposition scheme on the tasks, further referred to as activities, instead of decomposing on the employees. The main advantage of this decomposition strategy is that it results in smaller network problems while pricing out new columns. This approach enables us to find optimal solutions for real-life data sets. The problem will be written as a 0-1 multi-commodity flow problem with side-constraints, where each activity corresponds to a commodity.

The paper is organized as follows. Firstly, the problem will be stated and written as an integer program. Secondly, to solve the problem to optimality a branch-and-bound algorithm will be elaborated. Next, starting from the drawbacks of the branch-and-bound scheme, a branch-and-price algorithm will be introduced. Thirdly, several features of the branch-and-price scheme are discussed followed by an extensive overview of computational results. Finally, the paper ends with a conclusion and some ideas for future research.

1. Problem definition

As the problem originates from a practical context, the exact statement of the problem is not straightforward. The main reason is the presence of both hard and soft constraints. Hard constraints are constraints that cannot be violated, whereas soft constraints are constraints that must be met 'as much as possible'. Firstly, the problem will be described in an informal way by means of an example. Secondly, an integer linear programming (ILP) formulation of the problem will be given.

Consider a hospital department in which trainees have to perform a number of activities over a certain time horizon. Since the trainees have different experience levels, they can be divided in a number of experience groups. Most of the activities include assisting a specialist in a very specific field of the health care. Firstly, for each activity it is known how many trainees of each experience group are required in each period. Secondly, for each trainee it is known which activities (s)he has to perform in order to meet formation objectives. Thirdly, for each trainee it is known for each time period whether or not (s)he is available to be scheduled. Finally, in order to maximize both the efficiency and the quality of the service provided, it is not allowed that activities are split up per trainee. The efficiency decreases with each new activity start of a trainee, because it takes (again) some time to master the skills required for the activity. Moreover, patients have a smaller chance to be treated by one and the same trainee, resulting in less efficient care. In the ideal case each trainee starts each activity only once and performs it for a minimum number of consecutive periods. The last two constraints are soft constraints meaning that they can be violated at a certain 'cost'. Since a split-up in activities is considered to be worse than the violation of a non-availability constraint, we will concentrate on the problem solution in which we only relax the non-availability constraints. Therefore, the trainees have to quantify their preferences for having weeks-off. This happens by dividing a number of points per trainee over the scheduling horizon. The higher the number of points a certain period receives, the stronger the trainee feels about not being scheduled during that period.

Let us illustrate this problem with a simple example (see Figure 1). Suppose we have a problem with three activities, four trainees and ten periods. Furthermore, assume that all assistants have the same level of experience. Finally, suppose that each assistant has to perform each activity and this between a minimum of two and maximum of three consecutive periods. This example is graphically represented in Figure 1. The columns represent the trainees and the rows represent the periods. The periods of non-availability for each trainee are marked in grey and the respective 'costs' are indicated. Note that each trainee has divided in total five points over the ten periods. It is realistic that these points are concentrated in a small number of periods.

tr 1	tr 2	tr 3	tr 4
4			
		4	
	1		
			2
1			
	4		
		1	
			3

min. nr. consecutive periods:

	tr 1	tr 2	tr 3	tr 4
act 1	2	2	2	2
act 2	2	2	2	2
act 3	2	2	2	2

max. nr. consecutive periods:

	tr 1	tr 2	tr 3	tr 4
act 1	3	3	3	3
act 2	3	3	3	3
act 3	3	3	3	3

Figure 1: Example 1

A possible solution for this problem is represented in figure 2. In this solution, trainees 1 and 3 are both scheduled during a period in which they actually prefer not to be scheduled, respectively period 7 and period 9, making up for a total cost of 1+1 = 2. In practice, this means that either the trainee has to give up his/her preference for having a period off or the trainee has to be replaced by someone else in this period, resulting in a decrease of the quality of care. As a final remark, observe that in this solution during four time periods no activity is scheduled for particular trainees. During these periods, the trainees will perform activities for which no specific skills are required and for which consequently both experience level and minimal formation requirements are less important. An example of such an activity is consultation. A two-phase approach is thus being used to solve this problem. In the first phase the *difficult* (hard constrained) activities are scheduled. In the second phase the partial schedule is completed with the *easy* activities. The scheduling of the easy activities is straightforward and can easily be done manually. Therefore, we will only concentrate on the scheduling of the difficult activities in this paper.

tr 1	tr 2	tr 3	tr 4
	act 1	act 2	act 3
4	act 1	act 2	act 3
act 3	act 1	4	act 2
act 3	1	act 1	act 2
	act 3	act 1	act 2
act 2	act 3	act 1	2
act 2	act 3		act 1
act 2	4	act 3	act 1
act 1	act 2	act 3	
act 1	act 2	act 3	3

min. nr. consecutive periods:

	tr 1	tr 2	tr 3	tr 4
act 1	2	2	2	2
act 2	2	2	2	2
act 3	2	2	2	2

max. nr. consecutive periods:

	tr 1	tr 2	tr 3	tr 4
act 1	3	3	3	3
act 2	3	3	3	3
act 3	3	3	3	3

Figure 2: A solution for example 1

The problem can be mathematically stated as follows. Consider the following binary decision variables:

x_{ijk} = 1 if during period i trainee j is scheduled to perform activity k;
= 0 otherwise.
y_{ijk} = 1 if trainee j starts activity k during period i;
= 0 otherwise.

Let p_{ij} be the penalty cost charged for assigning trainee j to period i. It must be clear that p_{ij} equals 0 if trainee j is available during period i. Let l_{jk} and u_{jk} be the respective minimum and maximum number of periods assistant j has to perform activity k. Finally, Let S_k represent the set of trainees that will perform activity k in the given time horizon (i.e. all trainees j for which $u_{jk} > 0$). The integer linear programming model (ILP) is given below.

$$\text{MIN} \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ijk} \qquad [1.1]$$

S.T.

1. A trainee can perform no more than one activity per period:

$$\sum_{k=1}^{p} x_{ijk} \leq 1 \qquad \forall\, i = 1,..,n \text{ and } \forall j = 1,..,m \qquad [1.2]$$

2. At each period every activity has to be performed by exactly one trainee:

$$\sum_{j \in S_k} x_{ijk} = 1 \qquad \forall\, i = 1,..,n \text{ and } \forall k = 1,..,p \qquad [1.3]$$

3. Each trainee has to perform each activity between a minimum and maximum number of periods:

$$\sum_{i=1}^{n} x_{ijk} \geq l_{jk} \qquad \forall j = 1,..,m \text{ and } \forall k = 1,..,p \qquad [1.4]$$

$$\sum_{i=1}^{n} x_{ijk} \geq u_{jk} \qquad \forall j = 1,..,m \text{ and } \forall k = 1,..,p \qquad [1.5]$$

4. Each trainee starts each activity only once:

$$y_{1jk} = x_{1jk} \qquad \forall j = 1,..,m \text{ and } \forall k = 1,..,p \qquad [1.6]$$

$$y_{ijk} \geq x_{ijk} - x_{(i-1)jk} \qquad \forall\ i = 2,.., n,\ \forall j = 1,..,m \text{ and } \forall k = 1,..,p \qquad [1.7]$$

$$\sum_{i=1}^{n} y_{ijk} \leq 1 \qquad \forall j = 1,..,m \text{ and } \forall k = 1,..,p \qquad [1.8]$$

5. Integrality constraints:

$$x_{ijk}, y_{ijk} \in \{0,1\} \qquad \forall\ i = 1,..,n,\ \forall j = 1,..,m \text{ and } \forall k = 1,..,p \qquad [1.9]$$

For a problem with n time periods, m trainees and p activities, this notation requires $2nmp$ binary decision variables, a number that is growing rapidly. Even very simple problems require long solution times using this formulation and a commercial ILP solver. Obviously, better formulations and thus smaller solution times could be found by introducing other (integer) decision variables. It seems, however, that a specifically developed solution procedure like branch-and-bound is more suitable to solve this problem.
A last remark concerns activities that require more than one trainee during each period. We replace each of these activities by two or more artificial activities dividing the trainees in the original set S_k over these new activities. Consider for instance an activity for which each time period two trainees are required and twelve trainees have to perform the activity. Then, this activity will be replaced by two new activities each of which has to be performed by six trainees. This assumption facilitates the construction of an enumeration scheme at the cost of possibly excluding an optimal solution, since we only consider one particular division of trainees over the newly introduced activities.
In the remainder of this paper an effort will be made to solve the problem to optimality. To prove optimality all possible solutions have to be enumerated (either explicitly or implicitly). Firstly, a simple branch-and-bound scheme will be elaborated. The difficulties encountered in this approach will serve as a link towards a branch-and-price approach.

2. A branch-and-bound approach

The most intuitive way of enumerating all solutions is to run through a triple loop. In the first loop all activities are enumerated. The second loop runs through all trainees that have to perform the activity under consideration. The third loop walks from the minimum number of consecutive periods till the maximum number of consecutive periods for the activity and trainee under consideration. Each time a non-available period is assigned, its cost is added to the total cost of the schedule. Before starting the enumeration the activities k and the trainees within each activity ($j \in S_k$) are sorted. Tests indicated that the best results are obtained when the activities are sorted in descending order of (average) difference between maximum and minimum number of consecutive periods of the trainees performing the activity. The trainees are sorted within each activity in descending order of occupation. By occupation we refer to the number of activities a trainee has to perform. In the pseudo-code below cost(a,j,t_1,t_2) represents the cost of scheduling trainee j to perform activity a from period t_1 until period t_2 (periods t_1 and t_2 are included) and tc is the total cost of a (partial) schedule.

```
SORT();

DO RECURSION(1,1);

RECURSION(a,t)
{
    calculate_lower_bound;
    IF (lower_bound ≥ best_solution_found) RETURN;
    ELSE IF ($a = p+1$)
    {
        register schedule;
        best_solution_found = $tc$;
        $a = a-1$;
    }
    ELSE IF ($a \leq p$)
    {
        IF ($t = n$)
        {
            DO RECURSION(a+1,1);
        }
        ELSE IF ($t < n$)
        {
            FOR EACH trainee $j \in S_a$
            {
                FOR ($d = l_{ja}$; $d \leq u_{ja}$; $d = d+1$)
                {
                    IF (trainee $j$ is not yet scheduled between periods $t$ and $t+d$) AND
                    (SUM($l_{j'a} | j' \in S_a \setminus j$) $\leq n-t-d$) AND (SUM($u_{j'a} | j' \in S_a \setminus j$) $\geq n-t-d$)
                    {
                        Schedule trainee $j$ to perform activity $a$ from period $t$ until period
                        $t+d$;
                        $tc = tc + \text{cost}(a,j,t,t+d)$;
                        $S_a = S_a \setminus \{j\}$;
                        DO RECURSION(a, $t+d$);
                        Undo scheduling of activity $a$ from period $t$ until period $t+d$;
                        $S_a = S_a \cup \{j\}$;
                        $tc = tc - \text{cost}(a,j,t,t+d)$;
                    }
                }
            }
        }
    }
}
```

This algorithm explicitly enumerates all possible schedules. It should be clear that this number grows exponentially with the number of activities, the number of trainees per activity and the difference between the minimum and maximum number of consecutive periods required per activity per trainee. Obviously, the construction of many schedules could be interrupted at an early stage if one could prove that completing the current (partial) schedule never leads to a better schedule than the best schedule already found. To that purpose one should be able to calculate a strong (high) lower bound for the total cost of each (still to complete) partial schedule. The construction of partial schedules could also be terminated because of symmetry with an earlier generated schedule.

This branch-and-bound enumerating scheme has two disadvantages. The main problem is the calculation of strong lower bounds. Our lower bound calculation works as follows. For each period a minimum cost assignment problem is generated, in which the remaining activities are assigned to the available trainees. Each activity-trainee assignment cost equals the minimum cost to schedule the activity such that it 'covers' the time period for the considered trainee. A time period is covered by an activity for a certain trainee if the trainee is scheduled to perform the activity in the given time period. To find the minimum cost for each activity-trainee assignment, only the activity schedule at the minimum number of consecutive periods has to be considered, since longer activity schedules can only increase this cost. The construction of such a minimum cost assignment problem for period 1 is represented in figure 3.

tr 1	tr 2	tr 3	tr 4
	act 1		
4	act 1		
	act 1	4	
	1	act 1	
		act 1	
		act 1	2
1			act 1
	4		act 1
act 1		1	
act 1			3

min. nr. consecutive periods:

	tr 1	tr 2	tr 3	tr 4
act 1	2	2	2	2
act 2	2	2	2	2
act 3	2	2	2	2

max. nr. consecutive periods:

	tr 1	tr 2	tr 3	tr 4
act 1	3	3	3	3
act 2	3	3	3	3
act 3	3	3	3	3

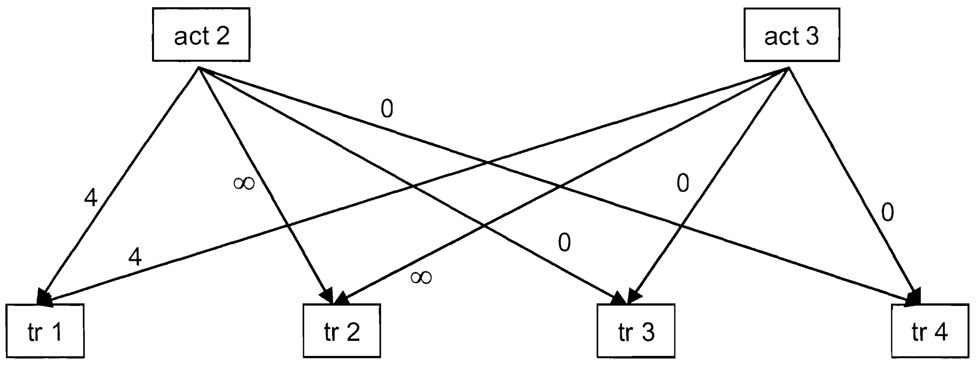


Figure 3: Assignment problem for period 1 in the partial schedule for example 1 represented above

In this figure the numbers next to the arcs represent the assignment costs. For instance, to cover period 1 for trainee 1 with activity 2, a cost of 4 has to be incurred. All the trainee 2 assignment costs are ∞, since it is impossible to schedule an activity to cover period 1 for trainee 2. Such an assignment problem can be solved very efficiently with a branch-and-bound algorithm. For each activity, the available trainees are sorted in increasing order of assignment costs. Then, the activities are subsequently considered and the first still available trainee is assigned. After each trainee assignment, the trainee has to be excluded from being assigned to other activities. A lower bound for the assignment problem can be calculated by summing up all costs of the first still available trainee for each activity. The optimal solution value for the assignment problem in figure 3 is of course 0. This is also the case for all assignment problems until period 7. The optimal costs for the assignment problems in period 8, 9 and 10 are equal to 1. If, however, for a certain time period an increase in the lower bound is obtained, then the algorithm cannot simply continue with solving the assignment problem of the next period, because the scheduling of an activity to cover the current period could also cover some later periods. Since a lower bound is searched, a best-case scenario has to be assumed. This means that the highest possible number of next periods have to be omitted. This is the maximum number of consecutive periods over all still to schedule activity-trainee combinations. In the example above, the lower bound can thus only be increased with 1, obtained from solving the assignment problem in period 8. Summarizing, given a partial schedule, a lower bound is found by solving at most *n* assignment problems. The bound relaxation lies in two facts. Firstly, it is assumed that each activity can be scheduled for each trainee during its minimum number of consecutive periods, which obviously cannot be the case for all trainees in a feasible solution. Secondly, if during a certain time period a cost is inevitable, then it is assumed that the cost is incurred by scheduling an activity, such that a maximum number of next periods is also covered by this activity. The second relaxation could be tightened by doing the lower bound calculation a second time, but now starting at the end of the scheduling horizon and working backwards. A second way to tighten the second relaxation is to assume that each trainee has to perform each activity for only one period. In this way, assignment problems for next periods will never be omitted. This happens at the expense of a decrease in the assignment costs and thus an increase of the gap from the first relaxation. Obviously, the highest of the three bounds provides the best lower bound. This lower bound calculation is fast, but does not provide good lower bounds. Moreover, the quality of the bounds is highly dependent on the difference between the minimum and maximum number of consecutive periods (first relaxation) and the relative magnitude of the maximum number of consecutive periods (second relaxation).
A second problem with the branch-and-bound scheme is the way of traversing the search tree. The tree is searched in a depth-first way, which entails that, if accidentally a bad decision is made near the root of the tree, detection of the optimum could happen at a very late stadium in the search. A best-first search would solve this matter. However, since the number of active branches can be very large, this would happen at the cost of an important increase in memory usage.
Since both problems originate from the fact that optimality has to be proven, a heuristic search procedure seems to be the only alternative. A branch-and-price approach, however, provides an answer to both difficulties, while still maintaining the ability to prove the optimality of solutions.

3. A branch-and-price approach

3.1 An alternative formulation

The integer program of [1.1]-[1.9] could be formulated in a totally different way. Observe that the problem can be seen as the scheduling of p activity patterns. An activity pattern includes the scheduling of all trainees having to perform the activity. In figure 3 such a pattern is represented for activity 1. For reasons that will become clear in a moment, an activity pattern will be called a *column* in the rest of this paper. Now, we can introduce new binary decision variables that explicitly incorporate these columns. Let binary decision variable z_{kt} be defined as follows:

$$z_{kt} \quad = 1, \text{ if column } t \text{ was chosen for activity } k;$$
$$= 0, \text{ otherwise.}$$

Let c_{kt} be the total cost of column t for activity k and NC_k the total number of different columns for activity k. Let a_{ijkt} equal 1 if trainee j is scheduled during period i in column t for activity k. The model can then be formulated as follows:

$$\text{MIN} \sum_{k=1}^{p} \sum_{t=1}^{NC_k} c_{kt} z_{kt} \qquad [1.10]$$

S.T.

$$\sum_{k=1}^{p} \sum_{t=1}^{NC_k} a_{ijkt} z_{kt} \leq 1 \qquad \forall\, i = 1,..,n \text{ and } \forall j = 1,..,m \qquad \lambda_{ij} \qquad [1.11]$$

$$\sum_{t=1}^{NC_k} z_{kt} = 1 \qquad \forall k = 1,..,p \qquad \gamma_k \qquad [1.12]$$

$$z_{kt} \in \{0,1\} \qquad \forall k = 1,..,p \text{ and } \forall t = 1,.., NC_k \qquad [1.13]$$

The objective function is again the minimization of costs, but now expressed in terms of the new z_{kt} variables. Constraint [1.11] states that each trainee can perform no more than one activity at the same time. Constraint [1.12] implies that exactly one column has to be selected for each activity. Remark that a_{ijkt} is merely a coefficient in the constraint matrix of the new model, whereas x_{ijk} was a decision variable in the old model. Tests revealed that the LP relaxation of this formulation provides a much stronger lower bound than that from the original formulation of [1.1]-[1.9]. The main drawback, however, is that this new model can have far more variables than can be reasonably attacked directly. Indeed, the number of columns increases dramatically with growing problem dimensions. It is however not necessary to enumerate all possible columns to solve the LP to optimality. The LP can be solved by using only a subset of the columns and can generate more columns as needed. This way of LP optimizing is called *column generation*. We iteratively add new columns and solve the restricted model until no more columns *price out*, i.e. no more columns with negative reduced cost can be found. Let λ_{ij} represent the dual prices of restrictions [1.11] and let γ_k represent the dual prices of restrictions [1.12]. The reduced cost of a new column *t* for activity *k* is given by:

$$\gamma_k + \sum_{i=1}^{n}\sum_{j=1}^{m}\left(p_{ij} + \lambda_{ij}\right)a_{ijkt}$$

$$= \gamma_k + c_{kt} + \sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}a_{ijkt} \qquad [1.14]$$

In this expression γ_k is non-positive and can be seen as the 'discount' for introducing a new column for activity *k*. This reward has to outperform the 'price' of the new column which is given by the remaining part in [1.14]. This price consists of two non-negative parts: the real price c_{kt} and the price charged for 'consuming' timetable cells (*i,j*) expressed by the dual prices λ_{ij}. Consequently, the LP is solved to optimality when no more columns can be found with negative reduced cost. The search is started by solving the *master problem*, stated in [1.10]-[1.12][1] for a limited number of columns. These initial columns are originated from a heuristic solution procedure. The master returns an objective value (which is an upper bound for the LP solution) and dual prices λ_{ij} and γ_k. λ_{ij} serves as a direct input for the objective function of the *pricing problem* (stated below), whereas γ_k is needed to check the negativity of the reduced cost of a newly found column for activity *k*. Then, to check the optimality of the LP solution, a subproblem, called the pricing problem, is solved for each activity *k*. Let b_{ijkt} equal 1 if in column t activity *k* starts during period *i* for trainee *j*. The *t*th pricing problem for activity *k* is given by:

$$\text{MIN}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(p_{ij} + \lambda_{ij}\right)a_{ijkt} \qquad [1.15]$$

$$\sum_{i=1}^{n} a_{ijkt} \geq l_{jk} \qquad \forall j = 1,..,m \qquad [1.16]$$

$$\sum_{i=1}^{n} a_{ijkt} \leq u_{jk} \qquad \forall j = 1,..,m \qquad [1.17]$$

$$b_{1jkt} = a_{1jkt} \qquad \forall j = 1,..,m \qquad [1.18]$$

$$b_{ijkt} \geq a_{ijkt} - a_{(i-1)jkt} \qquad \forall\ i = 2,..,n \text{ and } \forall j = 1,..,m \qquad [1.19]$$

$$\sum_{i=1}^{n} b_{ijkt} \leq 1 \qquad \forall j = 1,..,m \qquad [1.20]$$

$$a_{ijkt}, b_{ijkt} \in \{0,1\} \qquad \forall i = 1,..,m \text{ and } \forall j = 1,..,m \qquad [1.21]$$

In this formulation the objective function [1.15] minimizes the reduced cost of the new column. Constraints [1.16] to [1.21] imply the restrictions each column has to satisfy.

[1] Remark that the integrality constraints are omitted

Observe that in the pricing problem a_{ijkt} is a decision variable instead of a coefficient. Columns with a negative reduced cost are added to the master problem and the master is again optimized. This process continues until no columns price out any more. As will be shown in a moment, the pricing problem can be solved very efficiently by means of a dynamic programming formulation.

When an integer problem is solved by an enumeration scheme in which the lower bound is calculated by a LP relaxation and the LP is solved through column generation, one speaks of a branch-and-price approach. One could ask why we didn't apply the original formulation [1.1]-[1.8] as a lower bound calculation in the enumeration scheme described above. This would avoid the need of column generation, but entails two important drawbacks. The first problem is that the number of branches (recursive calls) would still be large and thus the lower bound calculation, which is still computationally intensive, has to be done many times. The second and most important problem is the weakness of the lower bounds. This observation is completely in line with Barnhart et al. (1998) when they state that the LP relaxation frequently can be tightened by a reformulation that involves a huge number of variables.

In the next sections the branch-and-price algorithm will be elaborated. Firstly, an overview of the algorithm is given. Secondly, the pricing problem is discussed. Thirdly, four possible branching strategies are elaborated. Finally, some improvements to speed up the computation time will be studied and their importance will be discussed on the basis of test results.

3.2 Branch-and-price algorithm overview

In Figure 4 an overview of the branch-and-price algorithm is given. The algorithm starts with a heuristic search for an initial solution. This heuristic successively generates activity patterns (columns) without violating the no-overlap constraint. If the algorithm succeeds in finding a feasible solution, this solution is saved and an initial upper bound is registered. The number of trials is adjustable. The master is initialized with both the p columns making up the best found solution and p supercolumns (one per activity). Supercolumns have a very high objective coefficient c_{tk} and all a_{ijkt} equal to 0. Consequently, supercolumns are not likely to enter the basis but are needed to ensure feasibility of the master at each stage of the branching scheme. Next, the algorithm enters the LP optimization loop. Every iteration of this loop consists of solving a number of pricing problems and adding columns to the master LP. When no more columns price out, the LP is optimized and the solution value is registered as the lower bound. If the solution is non-fractional, the solution is feasible and thus optimal. In the other case, branching has to be applied in order to find an integer solution. The algorithm again ends up in the LP optimization loop to determine a lower bound. Whenever this lower bound exceeds an already found upper bound, the loop is terminated and the search tree is backtracked. This process continues until an integer solution is found. The solution is saved and the solution value is registered as the new upper bound. The algorithm ends if the upper bound equals the lower bound or when backtracking leads back to the root node.

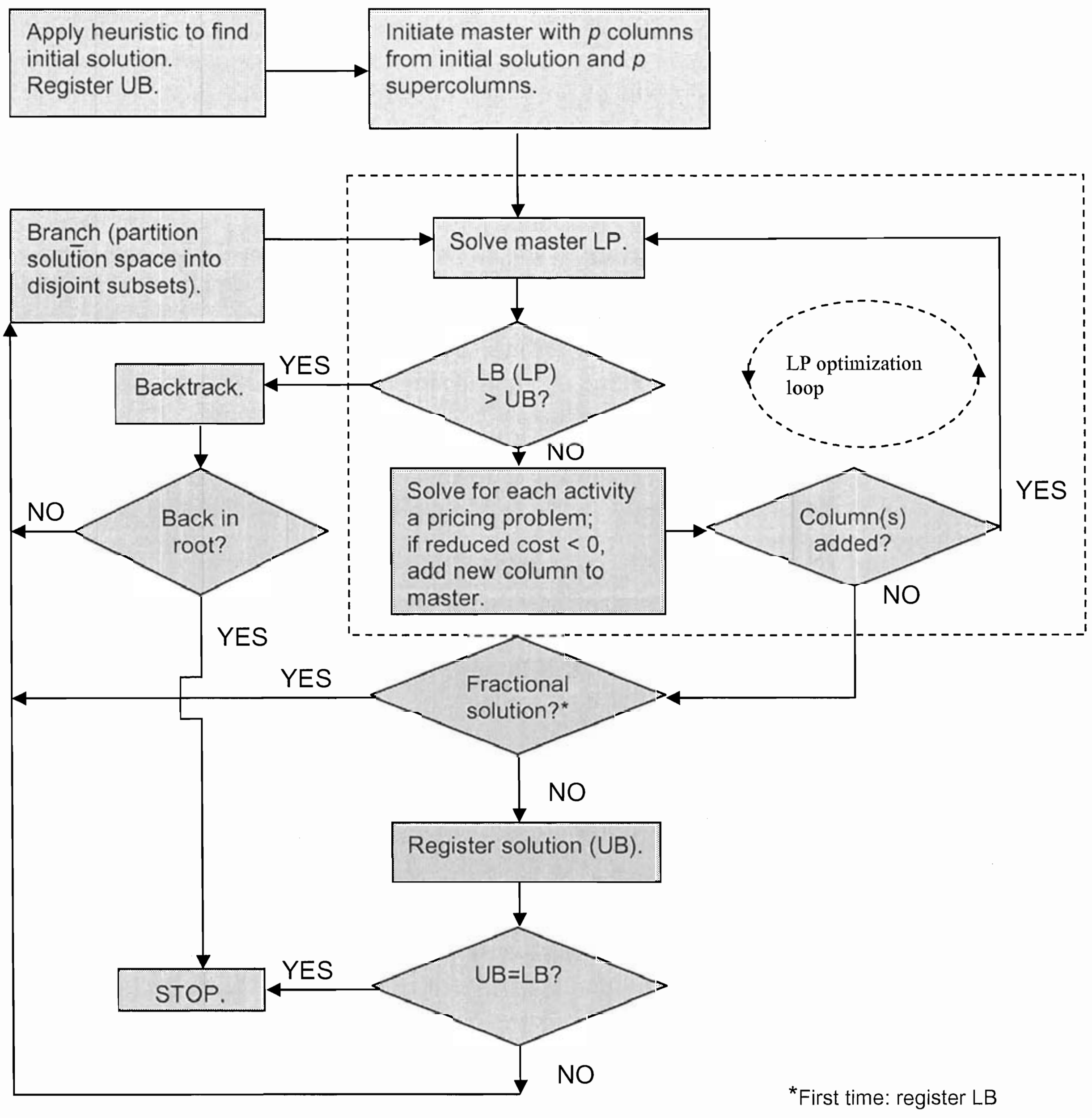
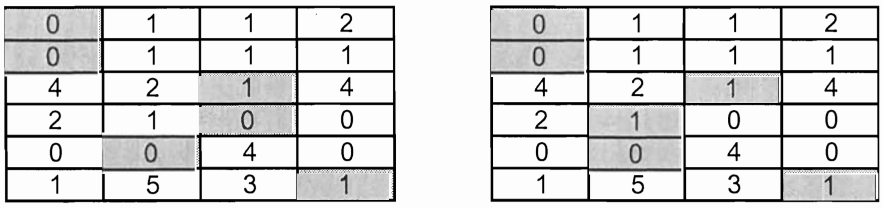

Apply heuristic to find initial solution. Register UB.
Initiate master with *p* columns from initial solution and *p* supercolumns.
Branch (partition solution space into disjoint subsets).
Solve master LP.
YES
Backtrack.
LB (LP) > UB?
LP optimization loop
NO
Solve for each activity a pricing problem; if reduced cost < 0, add new column to master.
Column(s) added?
YES
NO
Back in root?
NO
YES
YES
Fractional solution?*
NO
Register solution (UB).
STOP.
YES
UB=LB?
NO
*First time: register LB

Figure 4: Branch-and-price algorithm overview



0	1	1	2
0	1	1	1
4	2	**1**	4
2	1	**0**	0
0	**0**	4	0
1	5	3	**1**

0	1	1	2
0	1	1	1
4	2	**1**	4
2	**1**	0	0
0	**0**	4	0
1	5	3	**1**

Figure 5: Pricing problem (left: optimal solution; right: 2nd best solution)

3.3 The pricing problem

The pricing problem can be described as a restricted shortest path problem. Given is a matrix of costs. This matrix has to be traversed from top to bottom in the cheapest possible way, while visiting each column exactly once between a minimum and maximum number of rows. Figure 5 represents the best and one of the several alternative second best solutions for an instance of a pricing problem for an activity of four trainees that all have to be scheduled between one and two periods in a time horizon of six periods. The pricing problem is solved with a forward *dynamic programming* approach. Dynamic programming (Bellman (1957); Dreyfus and Law (1977)) is a decomposition technique that first decomposes the problem into a nested family of subproblems. A possible way to divide the pricing problem in a set of nested subproblems is as follows. Let T denote a set of trainees and $t_l \in T$ a trainee that is scheduled as the last one. The subproblem can be described as finding the cheapest way to reach a period $i \leq n$ with all trainees in T scheduled and trainee t_l scheduled as the last one. Let cost($i|T|t_l$) represent the cost of the solution to this problem. If $c_{i,j}$ is the cost to assign period i to trainee j and we search a column for activity k, then cost($i|T|t_l$) can be formulated recursively as follows:

$$\text{cost}(\,i|T|t_l) = \underset{l_{t_l k} \leq d \leq u_{t_l k}}{\text{MIN}} \left\{ \underset{t_j \in T \setminus \{t_l\}}{\text{MIN}} \left\{ \text{cost}(\,i\text{-}d|T \setminus \{t_l\}|t_j) + \sum_{b=1}^{d} c_{(i\text{-}d+b),t_l} \right\} \right\} \qquad [1.22]$$

The different values for cost($i|T|t_l$) can be calculated working forward from the beginning until period n. All values are initialized with $+\infty$ except for each trainee $t_j \in S_k$ and for each $i = l_{t_j k}..u_{t_j k}$:

$$\text{cost}(i|\{t_j\}|t_j) = \sum_{b=1}^{i} c_{b,t_j} \qquad [1.23]$$

Once all calculations are done, the cheapest way to reach period n can be found easily. Firstly, one searches the trainee that is to be scheduled last by solving the following expression:

$$\underset{t_l \in S_k}{\text{MIN}} \{\text{cost}(\,n|S_k|t_l)\} \qquad [1.24]$$

Assume trainee t_m is found with a total column cost of $c^* = \text{cost}(n|S_k|t_m)$. Then, the schedule is constructed backward step-by-step by searching for which trainee assignment the following expression holds:

$$c^* = \text{cost}(\,n - d|S_k \setminus \{t_m\}|t_j) + \sum_{\text{b}=1}^{\text{d}} c_{(n\text{-}b),t_m} \qquad [1.25]$$

This expression is checked for each trainee $t_j \in S_k \setminus \{t_m\}$ and for each $d = l_{t_j k}..u_{t_j k}$. If a match is encountered, the trainee is scheduled and both the current period and the set of already scheduled trainees are updated. This process continues until the last trainee is scheduled at the beginning of the scheduling horizon. Note that the major part of the required computation time goes to the calculations of all cost values. Once the cost values obtained, the schedule can be constructed in a relatively limited number of steps. The number of cost calculations obviously grows exponentially with the number of trainees that have to be scheduled. For realistic data (number of trainees) this number is not too large which results in acceptable solution times (<1s). We tried to extend the dynamic program with a lower bound calculation giving rise to an A* algorithm. Preliminary results, however, indicated that the time spent in the lower bound calculation exceeds the time gained from node pruning. If larger networks (i.e. activities with more than ten trainees) are considered, a lower bound calculation is however indispensable.

An important advantage of the pure dynamic programming approach is that it can be easily extended to find the b^{th} best solution instead of only the optimal solution. This property is very useful in a branch-and-price environment with branching on the column variables z_{kt} (see *3.5.1*). Our algorithm to find the b^{th} best column reflects the same idea as the algorithm proposed by Jiménez and Marzal (1999) for computing the K shortest paths in a network. To find the second best solution one first searches for the optimal solution as described above. Then, starting at the beginning of the time horizon all cost values being part of the found column are adapted to represent the second cheapest cost values. During this cost recalculation phase $|S_k|$ cost recalculations are made. For the example of figure 5 the recalculations are as follows. cost(2|{1}|1) with initial value zero is now updated to ∞, since there is no other possibility to reach period 2 with trainee 1 scheduled and trainee 1 scheduled as the last one. cost(4|{1,3}|3) with initial value one is now updated to ∞, since the only way to reach period 4 with trainees 1 and 3 scheduled and trainee 3 scheduled as the last one is to schedule trainee 1 during periods 1 and 2 and trainee 3 during periods 3 and 4. The value of cost(5|{1,2,3}|2) changes from one to two, since the cheapest way to reach period 5 with trainees 1, 2 and 3 scheduled and 2 as the last one is now by scheduling trainee 1 during periods 1 and 2, trainee 3 during period 3 and trainee 2 during periods 4 and 5. Finally, cost(6|{1,2,3,4}|4) is updated from two to three as the cheapest way to reach period 6 with all trainees scheduled and trainee 4 scheduled as the last one is now represented at the right of figure 5. Note that the same cost can also be obtained when scheduling trainee 2 only during period 4 and trainee 4 during periods 5 and 6. When the $|S_k|$ cost values are updated, the backward construction algorithm described above will now generate the second best column.

The old cost values (and the partial paths represented by these values), however, have to be saved in a list, because they could be part of the second best column and thus could be necessary during backward construction. This will be the case if the second best column has the same 'head' as the optimal column but a different 'tail', which is the case for the example in Figure 5. When during backward construction no trainee assignment can be found that matches [1.25], the list has to be scanned. In this case the list always contains a matching cost value. Note that once a cost value is retrieved in the list, the remaining trainee assignments can also be found in the list; they occupy the immediately preceding positions. Also during the cost recalculation phase, one should consider the list values as possible starting points when searching for the second best cost value to reach a certain node in the network. It should be clear that the extra computation time for finding the b^{th} best solution only requires $2^*b^*\sum_{j\in S_k}(u_{jk}-l_{jk})$ computation steps and $b^*|S_k|$ write instructions, which is negligible compared to the number of computation steps needed to calculate all initial cost values.

3.4 Column addition

An important characteristic of the branch-and-price algorithm is the number of columns that are added per master LP optimization. Three strategies can be distinguished. Firstly, we could add the most negative reduced cost column for each activity. Secondly, we could add only one column for all activities, i.e. the column with the overall most negative reduced cost. Finally, we could add the most negative reduced cost column for activity k, re-optimize the master and search for a new column for activity k+1. Remark that in this last strategy it is no longer possible to prune nodes based on Lagrange relaxation (see section *3.6.2*), since the reduced costs of all activities, needed to calculate the lower bound, are no longer available. Obviously, for all three strategies, columns with non-negative reduced cost are never added.

3.5 Branching

The LP relaxation of the master problem may not have an integral optimal solution. Branching refers to the process of partitioning the solution space to eliminate the current fractional solution. After branching it may be the case that there exists a column that would price out favorably, but is not present in the column pool. Applying standard branch-and-bound procedures to the master problem over the existing columns is unlikely to find an optimal, or good, or even feasible solution. To illustrate this point, a branch-and-bound algorithm was written to find the best possible integral solution given the column pool after LP optimization. When the algorithm was run on the problem set, it never succeeded in finding a feasible solution, because the columns could not be combined into an integer solution.
Four binary branching schemes were implemented and extensively tested: branching on the column variables, branching on timetable cells, branching on immediate precedence relations and branching on normal precedence relations.

3.5.1 Branching on column variables

In this branching scheme branching happens by fixing the largest fractional variable z_{kt} either to one (left branch) or to zero (right branch). It is however well known that 'direct' partitioning of the solution space, i.e. by fixing (or bounding) individual column variables, is not appropriate because of two reasons. Firstly, it could require significant alterations to the pricing problem and secondly, it yields an unbalanced branch-and-bound tree (Vanderbeck (2000)). The first problem is encountered along a branch, where a variable has been set to zero. Recall that z_{kt} represents a particular schedule for activity k. Thus z_{kt} = 0 means that this schedule is excluded. However, it is possible (and quite likely) that the next time the pricing problem is solved for the kth activity the optimal solution is precisely the one represented by z_{kt}. In that case it would be necessary to find the second best solution. At depth l in the branch-and-bound tree we may need to find the lth best solution. We already showed that the dynamic programming approach for the pricing problem (see section *3.3*) can be easily extended to handle this need and this at a negligible computational effort. The unbalanced branch-and-bound tree remains a problem, but also involves an advantage as faster detection of (sub)optimal integral solutions may be expected.

3.5.2 Branching on timetable cells

Since timetable cells are represented by the original variables, this branching scheme will also be referred to as branching on the original variables. When columns can be associated with paths in a network, a possible branching scheme consists of fixing single components of the arc incidence vector (Vanderbeck (2000)). If this branching principle is applied to our problem, it results in branching on the original x_{ijk} variables. The next x_{ijk} to branch on is found by selecting the largest fractional column k' and searching for this column the first timetable cell (i,j) for which there exists another fractional column that includes the same time table cell (i,j). $x_{ijk'}$ is set to one in the left branch and to zero in the right branch. Alternatively, this branching rule can be seen as Ryan-Foster branching. Ryan-Foster branching, proposed by Ryan and Foster (1981) for set partitioning problems, identifies two *rows*, say r and s, covered by different fractional columns. In the left branch rows r and s have to be covered by the same column and in the right branch by different columns. Observe that in our application row r corresponds to one of the capacity constraints [1.11] and s corresponds to one of the convexity constraints [1.12]. The main advantage of this branching scheme is that it does not destroy the structure of the pricing problem, because the resulting modifications simply entail amending the cost of the corresponding arc in the underlying network. If x_{ijk} is set to zero, $c_{i,j}$ is set to $+\infty$ in the pricing problem of activity k. Else if x_{ijk} is set to one, $c_{i,j}$ is set to $+\infty$ for all trainees $j \in S_k$ with $j \neq j'$. A second advantage is the fact that this branching scheme yields a balanced branch-and-bound tree. The main drawbacks of this branching scheme are the large number of arcs (x_{ijk}'s) to choose from and the fact that a branching constraint that involves a single arc might not be very restrictive.

3.5.3 Branching on immediate precedence relations

In this branching scheme branching happens on immediate precedence relations within the trainees performing an activity. An immediate precedence relation for an activity k between two trainees, say j and j', implies that trainee j has to perform activity k either immediately before trainee j'. In the twin node trainee j cannot be scheduled immediately before trainee j'. Upon detection of a fractional solution, the algorithm searches for two fractional columns for the same activity with different orderings in trainee assignments over the scheduling horizon. Then, an immediate precedence relation, which is satisfied by only one of both fractional columns, is implied. The main drawback of this branching scheme is that it is not guaranteed that it drives the solution completely to integrality. In other words, this branching scheme is not complete. Theoretically, it is possible that an optimal fractional solution is found in which all fractional columns for each activity have the same trainee ordering. If this would be the case, the algorithm rounds all fractional values to 1 and verifies whether or not the resulting solution contains an overlap. In the case of an overlap, only a lower bound instead of a feasible solution could be provided.

Preliminary tests, however, indicated that this scenario occurs rarely. Similar to the case in which branching occurs on the timetable cells, application of this branching scheme preserves the structure of the pricing problem. An immediate precedence relation can be implied easily by simply amending the costs of certain arcs in the dynamic programming network. It is not immediately clear whether or not this branching scheme leads to more balanced branch-and-bound trees than the trees yielded when branching on the timetable cells. However, this branching scheme certainly leads to more balanced branch-and-bound trees than in the case of branching on the column variables.

3.5.4 Branching on normal precedence relations

This branching scheme is equal to the previous branching scheme, except that branching now occurs on normal precedence relations instead of immediate precedence relations. A normal precedence relation for an activity k between two trainees, say j and j', simply states that trainee j has to perform activity k either before or after trainee j'. Obviously, this branching scheme also preserves the pricing problem structure and is also incomplete. Compared to the other three branching schemes, this branching scheme clearly yields the most balanced branch-and-bound trees. The restrictions implied in the left branches are similar to the ones implied in the right branches, i.e. if trainee j cannot perform activity k before trainee j', it means that trainee j' has to perform activity k before trainee j.[2]

3.6 Speed-up techniques

Once the column generator and branching scheme(s) are developed, we obtain a working branch-and-price algorithm. The performance of the algorithm is, however, strongly dependent on a number of speed-up techniques. The most important amongst these are discussed below.

3.6.1 Initial heuristic

The column pool is initialized with the columns making up an initial solution. The heuristic works as follows. Firstly, the activities k are sorted such that the most constrained activities are considered first. Then, the first activity is scheduled optimally using the dynamic program of the pricing algorithm. All occupied timetable cells receive large costs (+1000) in order to exclude overlaps when scheduling the next activities. In addition, the timetable cell costs just before and after the already scheduled activity are slightly decreased (-0.05), making them more attractive for scheduling the following activities. This prevents as much as possible the appearance of “holes”, i.e. blocks of timetable cells that are too small to fit an activity in. Then, the next activity is considered taking into account the new timetable costs. This process continues until either all activities are scheduled or an activity cannot be scheduled any more due to an overlap. If all activities are scheduled, the total cost is compared with an earlier found solution and if lower, the upper bound is decreased. The changes to the timetable cell costs are made undone and the process restarts with the first activity. In order to avoid generation of the same columns as

[2] Observe that this is not the case when branching occurs on timetable cells or on immediate precedence relations.

much as possible, the costs of the occupied timetable cells in the previous iteration are increased slightly (+0.01) before starting a new iteration. The heuristic ends if it fails to improve the current best solution in a predetermined number of iterations.

3.6.2 Lower bound calculation

Our column generation scheme exhibits the tailing-off effect, i.e. requiring a large number of iterations to prove LP optimality. Instead of solving the linear program to optimality, i.e. generating columns as long as profitable columns exist, we could end the column generation phase based on bound comparisons. It is well known that Lagrangian relaxation can complement column generation in that it can be used in every iteration of the column generation scheme to compute a lower bound to the original problem with little additional computational effort (see e.g. Van den Akker and Hoogeveen (2000); Vanderbeck and Wolsey (1996)). If this lower bound exceeds an already found upper bound, the column generation phase can end without any risk of missing the optimum. Using the information from solving the reduced master and the information provided by solving a pricing problem for each activity *k*, it can be shown (see e.g. Hans, 2001) that a lower bound is given by:

$$\delta + \sum_{k=1}^{p} RC_k \theta_k, \tag{1.26}$$

where δ is the objective value of the reduced master, RC_k is the reduced cost of a newly found column for activity k and θ_k is a binary variable equal to 1 when RC_k is non-negative and set to zero, otherwise. This lower bound is referred to as the Lagrangian lower bound, since it can be shown that it equals the bound obtained by Lagrange relaxation. In addition with an upper bound it can also be used to fix variables. When the reduced cost of a variable z_{kt} is larger than *UB-LB*, we know from linear programming theory that $z_{kt} = 0$ in any solution with a value less than *UB*. Hence, that variable can be fixed in the current node and in all nodes below that node. Analogously, when the reduced cost is smaller than *LB-UB* then $z_{kt} = 1$ in any solution with a value less than *UB*.

3.6.3 Initial network restriction

Recall that, to price out a new column, a shortest path network problem is solved by applying a forward dynamic program approach. For problems in which these networks are very large, the pricing problems are the bottleneck of the algorithm. We distinguish two ways of decreasing the required solution times of the pricing problems. Firstly, one could initially restrict these networks. Concretely, arcs with positive non-availability costs are excluded during the early phase of each LP optimization loop. When no more columns can be found with negative reduced cost, these arcs are restored. The benefits are two-fold. Firstly, the required time of the pricing algorithm dramatically decreases during the early phase of column generation. Secondly, from the start on, the algorithm is forced to price out qualitatively good columns.

3.6.4 Dynamic programming with upper bound pruning

A second way of reducing the computational effort to price out a new column was already mentioned in section *3.3* and involves an extension of the forward dynamic recursion with bound pruning. Whenever the forward dynamic program reaches a leaf node in the network, the cost value of the path can be compared with the best found solution and, if lower, registered as the new upper bound. This enables us to stop the construction of partial paths as soon as the associated cost value exceeds the upper bound. Remark that the cost value of a partial network cannot be decreased when extending the path, since all non-availability costs and dual prices of the non-overlap constraints are non-negative. As also mentioned in section *3.3*, the partial cost value could be increased with a lower bound calculation for scheduling the remaining trainees. Preliminary tests, however, showed that this didn't turn out to be beneficial. Finally, remark that bound pruning is not possible if it could be required to find the 2nd, 3rd, …, *k*th best path, since these could be pruned. Hence, when branching takes place on the column variables, a pure dynamic program is applied to price out new columns.

3.6.5 Master LP optimization

An important computational issue relates to the optimization of the master linear program. When new columns are added and the master is re-optimized, the (dual) simplex algorithm could be started either from an empty base or from the optimal base of the previous iteration. Tests revealed that the LP is optimized fastest when started from an advanced base.

3.6.6 Cost varying horizon

To limit the solution space as much as possible, we implemented the idea of a cost varying horizon. This idea is equivalent with a time varying horizon in exact algorithms for the *Resource Constrained Project Scheduling Problem* (Demeulemeester and Herroelen, 1992).
When implementing a cost varying horizon, one could distinguish between a maximum and minimum bounding search strategy. Both strategies are different with respect to the value of the upper bound. In minimum bounding search the upper bound reflects the best found solution. When it is important to prove the optimality of a solution, a maximum bounding approach can be more effective than a minimum one. In maximum bounding search the upper bound is set to the first integer equal to or higher than the LP lower bound. If the algorithm succeeds in finding a solution with a total cost equal to this upper bound, we found an optimal solution. Otherwise, both the upper and lower bound are increased with one, the column pool is re-initiated with the columns making up the LP optimum and the algorithm tries to find a solution equal to this new upper bound. This approach corresponds to best-first search in branch-and-bound, as the first solution obtained is also the optimal solution. Tests indicated that maximum bounding search slightly decreases computation times at the expense of not providing (sub)optimal solutions during search.

3.6.7 Column elimination

The idea of column elimination is inherent in all branching schemes except for the column-based branching scheme. To fully exploit the column-based branching strategy, the branching scheme was extended in that it also inherits the idea of column elimination.
The solution time of the master LP grows strongly with the number of columns in the master, even when their associate column variables z_{kt} cannot be positive in a feasible solution. After branching, an important number of already generated columns could be excluded from the master. If a particular column, say $z_{k't'}$, is set to one, all other columns $z_{k't}$ with $t \neq t'$ are excluded implicitly because of the convexity constraint [1.12] in the master. To speed up the computation time of the master, these columns can be excluded explicitly from the master (by eliminating them). Similarly, all columns having an overlap with column $z_{k't'}$ can be excluded as well, due to the non-overlap constraints [1.11]. Observe that eliminated columns have to be saved, since they have to be reentered upon backtracking. Obviously, if column $z_{k't'}$ is set to zero, no columns but $z_{k't'}$ can be left out.
Column elimination is inherent when branching occurs on the original variables. Consider the situation in which $x_{i'j'k'}$ is set to one. All columns $z_{k't}$ not including timetable cell (i',j') (i.e. having $a_{i'j'k't} = 0$) will be left out. Similarly, all columns z_{kt} with $k \neq k'$ including timetable cell (i',j') (i.e. having $a_{i'j'kt} = 1$) will be removed as well. If $x_{i'j'k'}$ is set to zero, the reverse applies. Column elimination is also inherent in the immediate and normal precedence relation branching schemes. Columns that do not satisfy the introduced precedence relations will be eliminated explicitly out of the master.
The same reasoning leads to the artificial adaptation of dual prices when branching occurs on the column variables. During preliminary tests of the algorithm, columns were generated that share timetable cells with already branched-to-one columns. Obviously, these columns can never enter the basis. The algorithm was adapted in that the dual prices of all timetable cells making up branched-to-one columns are increased with an artificially high value. Observe again that these artificial cost adaptations are inherent in the case branching is done on timetable cells and on immediate or normal precedence relations.

3.6.8 Subgradient optimization

Instead of solving the master problem with a standard simplex algorithm, subgradient optimization applied on the Lagrangian relaxation of the master could be used to find the dual prices. Excellent expositions on how to exploit Lagrangian relaxation and subgradient optimization techniques in combination with column generation can be found in Peeters (2002) and Jans (2002). If we relax the capacity constraints [1.11], dualizing them into the objective function [1.10], we obtain the Lagrange problem of [1.10]-[1.13]:

$$\text{MIN} \sum_{k=1}^{p} \sum_{t=1}^{NC_k} \left(c_{kt} + \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ijkt} \lambda_{ij} \right) z_{kt} - \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_{ij} \qquad [1.27]$$

$$\sum_{t=1}^{NC_k} z_{kt} = 1 \qquad \forall k = 1,..,p \qquad \gamma_k \qquad [1.28]$$

$$z_{kt} \in \{0,1\} \qquad \forall k = 1,..,p \text{ and } \forall t = 1,.., NC_k \qquad [1.29]$$

We have implemented a standard subgradient optimization scheme for setting the dual prices λ_{ij}. At step r+1, the dual prices are updated as follows:

$$\lambda_{ij}^{r+1} = \max\left\{0, \lambda_{ij}^{r} - \sigma\left(1 - \sum_{k=1}^{p}\sum_{t=1}^{NC_k} a_{ijkt} z_{kt}^{r}\right)\right\} \qquad \forall\ i = 1,..,n \text{ and } \forall j = 1,..,m \qquad [1.30]$$

$$\text{with } \sigma = \frac{\omega\left(UB - Z_{LAG}(\lambda_{ij}^{r})\right)}{\sum_{i=1}^{n}\sum_{j=1}^{m}\left(1 - \sum_{k=1}^{p}\sum_{t=1}^{NC_k} a_{ijkt} z_{kt}^{r}\right)} \qquad [1.31]$$

In this expression σ can be seen as a stepsize. $Z_{LAG}(\lambda_{ij}^{r})$ is the optimal objective value [1.27] of the Lagrange dual problem for a given set of dual prices λ_{ij}^{r} at step r and z_{kt}^{r} indicates the corresponding optimal value of column t for activity k. The optimal solution to the Lagrange dual problem [1.27]-[1.29] is always integral and easily found by setting for each activity the z_{kt} with the lowest cost equal to 1 and all other z_{kt}'s equal to 0. UB is the best known integer solution. ω is initially set to 1.5 and is halved each time the lower bound has failed to increase for a fixed number of iterations. After a predetermined number of iterations we stop the updating and we obtain the approximate dual prices λ_{ij}. In order to determine whether or not a new column prices out and thus should be added to the master, one has to calculate the reduced cost of the new column. The reduced cost could be calculated using the approximating dual prices given in [1.28] or the optimal dual prices obtained from the last solved master. If the approximating dual prices are used, one still needs to calculate the dual price γ_k of the convexity constraint [1.12]. It can be shown that the dual price of the convexity constraint for activity k is approximated by the cost of the optimal column, as defined in the first term of [1.27].

4 Computational results

4.1 Test set

In order to study the computational performance of the algorithm, a test set was generated. Firstly, the six factors that have an influence on the complexity of the problem were identified. These are the number of periods, the number of trainees, the number of activities, for each activity the number of trainees performing the activity, the difference between the maximum and minimum number of consecutive weeks further referred to as the range and finally the magnitude of the costs. Consider the following settings for these six factors:

Table I: Design of experiment

Factor setting	Nr. periods	Nr. trainees	Nr. activities (% of nr. trainees)	Nr. trainees per activity (% of nr. trainees)	Range	Magnitude of costs
1	18	6	60	60	1	1
2	35	8	75	75	2	U(1,5)
3	52	10	90	90	3	
4		12		random(75)	4	
5					random(2)	
6					random(3)	

Observe that the number of activities and the number of trainees having to perform an activity is expressed as a percentage of the number of trainees. For instance, a test problem with 10 trainees and 90% activities includes 9 activities. Note also that the number of activities cannot exceed the number of trainees, because otherwise not all activities can be performed. The ratio nr. activities over nr. trainees represents the total schedule occupation percentage. Recall that the remaining part of the schedule has to be filled up with activities for which the consecutiveness is not important. random(x) indicates that the factor setting is uniformly distributed with an average of x. For instance, the range setting random(2) means that the ranges are generated randomly in such a way that the average amounts to 2. If the magnitude of the costs is 1, it means that all non-available time periods, which are generated randomly for each trainee, have a cost of 1. Alternatively, these cost values are drawn from a uniform distribution between 1 and 5.
According to these factor settings, problem instances were generated with randomness on both the activity-trainee assignments and the non-available periods. In order to exclude non-feasible and trivial problems as much as possible, the trainee occupations were kept more or less at the same level. Without loss of generality all non-availability costs are assumed to be integral. The total number of periods containing positive costs equals 3, 4 or 5 for problems with respectively 18, 35 and 52 periods. If we generate three problem instances per factor setting, we obtain 3*(3*4*3*4*6*2) = 5184 problem instances. In order to decrease this number, we decided to subsequently fix the first three factors and the next two factors (the fourth and fifth factor) at an intermediate level, making us end up with 3*(4*6*2)+ 3*(3*4*3*2) = 360 problem instances.

4.2 Proven optimal solutions

In order to find (proven) optimal solutions, all above discussed speed-up techniques turned out to be useful to reduce computation times, except, somewhat surprisingly, for subgradient optimization (see section *3.6.7*). In this part we present optimal solutions and computation times of all problems instances. All our experiments were performed on a 2.4 GHz Pentium 4 PC with the Windows XP operating system. The algorithm was written in MS Visual C++.NET and linked with the CPLEX 8.1 optimization library. All speed-up techniques described above were incorporated in the algorithm, except for subgradient optimization (see Section 3.6.7), since it could not improve solution times. We apply maximum bounding search on all 360 problem instances and distinguish between the four branching strategies. All computation times are given in seconds. The results are represented in Appendix 1. The first column contains the name of the problem instance. All problems are named "DOEabcdef_g", which can be interpreted as follows:

- a is the factor setting for the number of periods,
- b is the factor setting for the number of trainees,
- c is the factor setting for the number of activities,
- d is the factor setting for the number of trainees for each activity,
- e is the factor setting for the range,
- f is the factor setting for the magnitude of costs,
- g is a replication number.

The second column contains the optimal LP objective value at the root node (before branching), the heuristic solution, and the optimal integer solution value (after branching). The next columns contain respectively the required CPU time (in seconds), the number of generated columns and the number of nodes in the branch-and-bound tree for each branching scheme. The time limit for each problem was 600 seconds. If the optimum was not found (or proven) within this time limit, this is indicated with a ">600". Recall that, since a maximum bounding search is applied, no solution is found when the algorithm fails to find the optimum. If the precedence-related branching schemes could not branch until a non-fractional solution, this is indicated with "Fract" equal to 1.

4.3 Discussion of results

In this section, we summarize the most important findings from our computational experiments.
In table II the number of problems that could be solved to optimality within 10 minutes is given for each branching scheme together with the average computation times. The second row (*) contains the average times for only those problems for which all four branching schemes succeeded in finding (and proving) the optimal solution within 600 seconds. In the third row (**) average times are calculated based on all problems. For these calculations, 600 seconds were assigned to those problems for which no optimal solution was found within 600 seconds.

Table II: Comparison of branching schemes

Branch on:	column variables	timetable cells	immediate prec. relations	normal prec. relations
Nr. solved to optimality	311	319	281	281
Average comp. time* (s)	14.5	19.0	22.9	42.1
Average comp. time** (s)	112.0	114.1	161.1	171.5

A first important observation is that the two precedence-related branching schemes are clearly outperformed by the first two branching schemes. A second observation is that, although branching on timetable cells yields more problems solved to optimality, the required computation times are generally higher than those for the column-based branching scheme. This is a clear indication of the appearance of unbalanced branch-and-bound trees when branching occurs on the column variables.

In the following graphs the results are visualized per factor combination. The first four graphs (Figure 6) summarize the results as a function of varying number of trainees for each activity and range. The number of periods, trainees and activities are fixed at an intermediate level and the non-availability costs are always 1. The columns represent average computation times over three test instances per factor setting for each branching scheme. The first column indicates branching on the column variables, the second branching on the timetable cells and the third and the fourth branching on immediate and normal precedence relations.

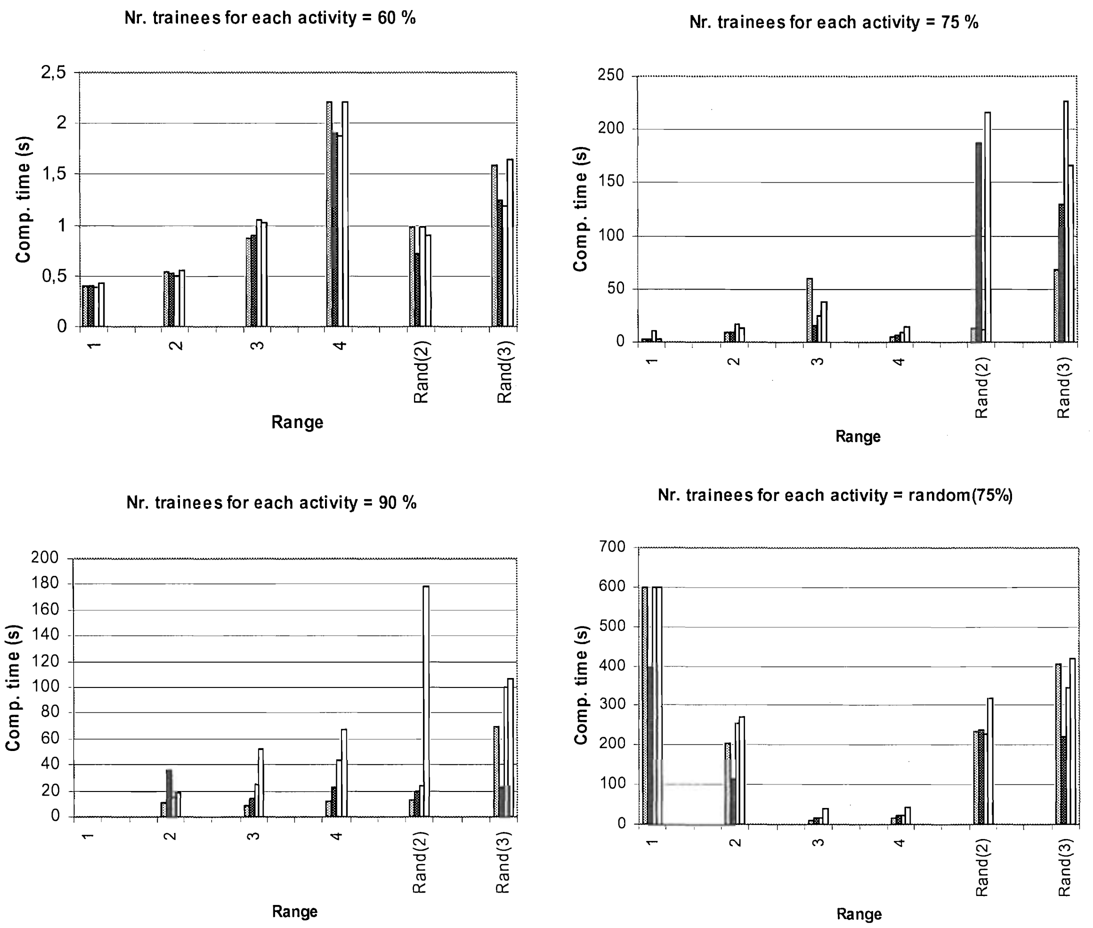


Figure 6: Comp. times for nr. periods = 35, nr. trainees = 8, nr. activities = 6 and magnitude of costs = 1

The conclusions with respect to these graphs are three-fold. Firstly, the computation times grow exponentially with the number of trainees per activity. The results in graph 4 suggest that randomness on this factor complicates the problem only if the range is small. Secondly, the influence of the range on the complexity of the problem is far less important. The reason is that the number of possible columns for each activity grows exponentially with the number of trainees (all possible permutations) whereas this number grows only linearly with the range. All four graphs indicate that randomness on the range tends to complicate the problem. Thirdly, branching on the columns and timetable cells seem to outperform the two precedence-related branching schemes.

The next set of four graphs (Figure 7) show the same information with the only difference that the magnitude of the costs are now randomly distributed between 1 and 5.

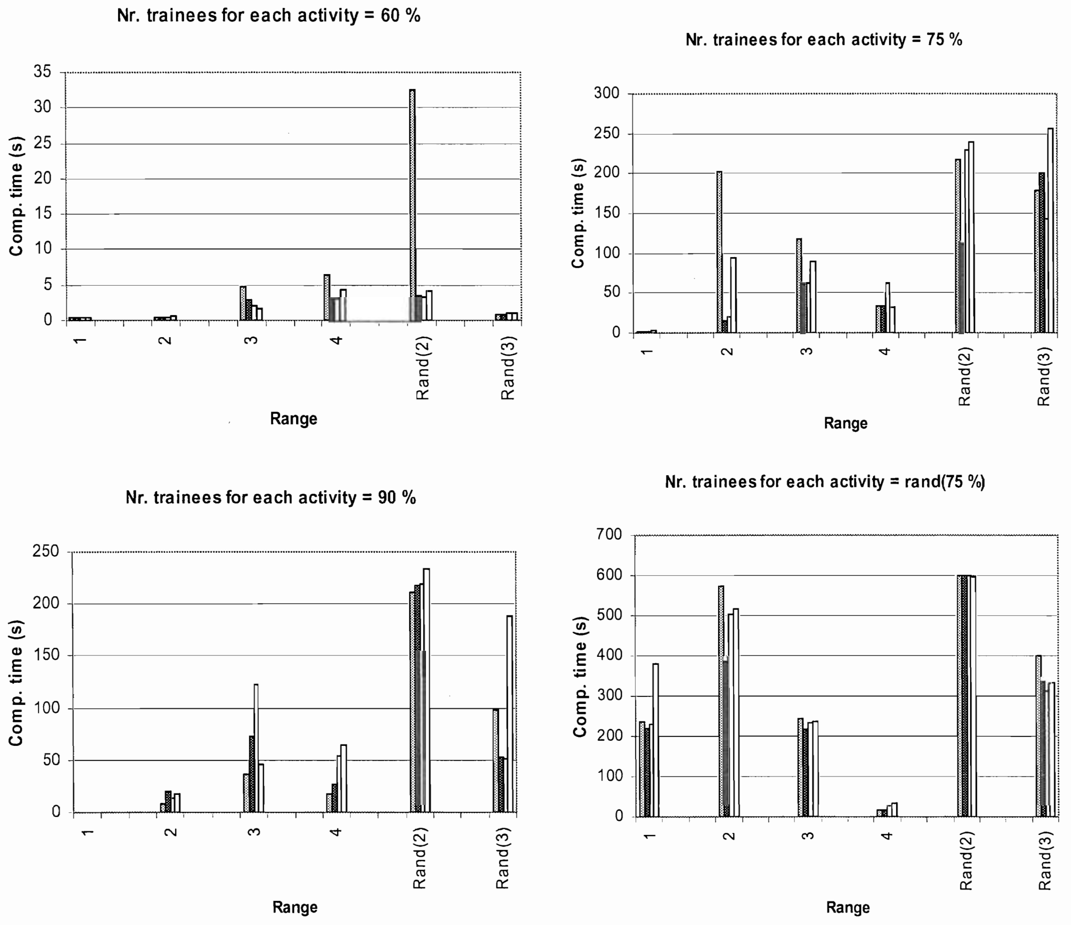
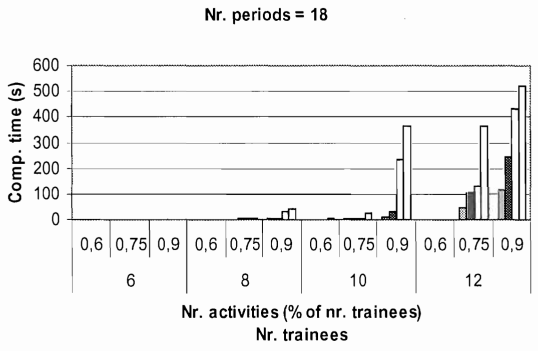
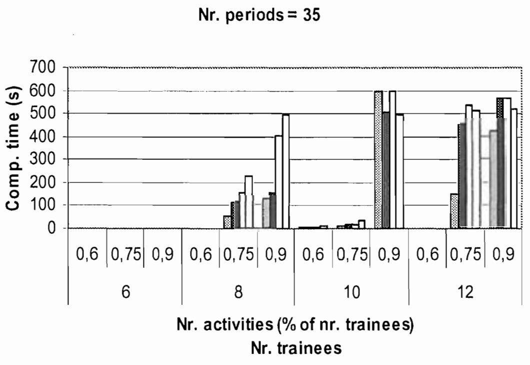


Figure 7: Results for nr. periods = 35, nr. trainees = 8, nr. activities = 6 and magnitude of costs = U(1,5)

These graphs confirm all previous conclusions. With respect to the comparison of the first two branching schemes, one can observe that branching on the columns now performs significantly worse. This can be explained by the fact that the randomness on the magnitude of the costs makes the quality of the LP lower bound decrease, which of course brings more harm to the unbalanced column branching than to the balanced timetable cell branching.

The next three graphs show the results for varying number of periods, trainees and activities while the fourth and the fifth factor are fixed at an intermediate level. We distinguish again between the case in which the non-availability costs are always 1 and the case in which these costs are uniformly distributed between 1 and 5.





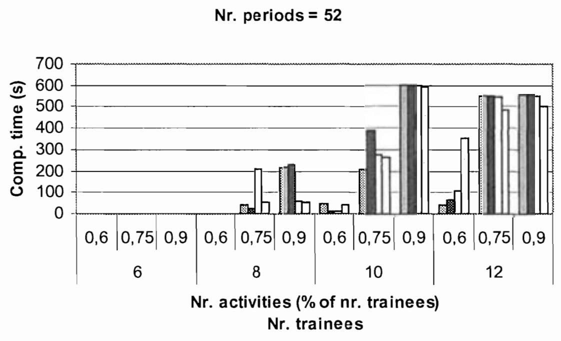
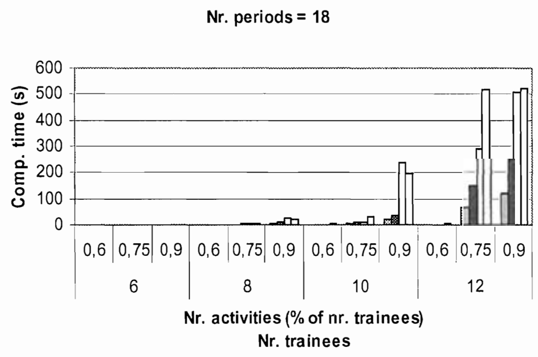
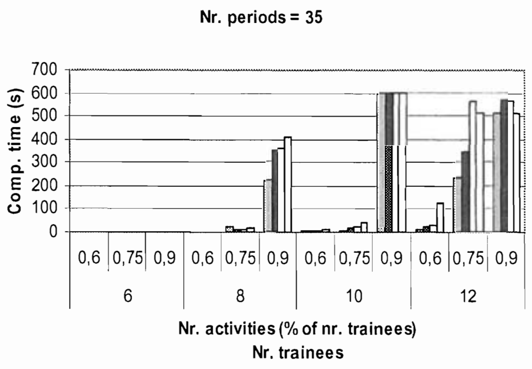


Figure 8: Results for nr. trainees for each activity = 75%, range = 3 and magnitude of costs = 1





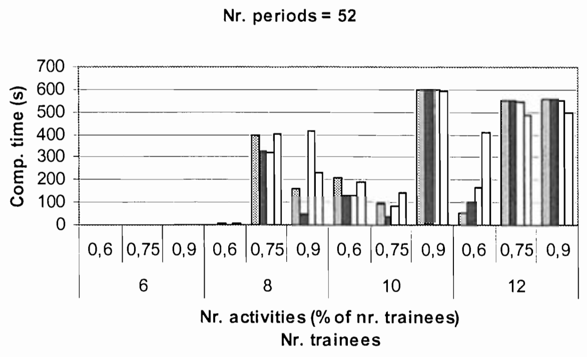


Figure 9: Results for nr. trainees for each activity = 75%, range = 3 and magnitude of costs = U(1,5)

From these graphs one may conclude that both the number of activities and the number of trainees have a significant impact on the complexity of the problem. With a few exceptions the algorithm was not able to solve problems with 12 trainees and 11 activities to optimality within the time limit of 600 seconds. Furthermore, the results indicate that the impact of the number of periods on the complexity of the problem is rather small. Similar to the previous results, the precedence-related branching schemes are outperformed by the first two branching schemes. Branching on the columns seems to perform best when the magnitude of the costs is always 1, whereas when these cost values are uniformly distributed between 1 and 5, branching on the timetable cells seems to be the most robust way of branching.

4 Contributions of speed-up techniques

In order to gain some insights into the contributions of the different speed-up techniques, an experiment was performed including all 307 problems for which an optimal solution was found within 600 seconds for both the first and the second branching scheme. Besides all eight speed-up techniques (see section *3.6*) the influence of the two alternative ways of column addition (see section *3.4*) was investigated. The results are presented in Table 3 and visualized in Figure 10. The first row of Table 3 contains the average computation times for the basic algorithm, i.e. the branch-and-price algorithm including all speed-up techniques except for subgradient optimization (see section *4.2*). Rows 2 to 9 contain the average computation times when the respective speed-up technique was omitted (or included in the case of subgradient optimization). Note that the effects are not cumulative, i.e. the algorithm always included all but one speed-up technique. Row 10 gives the computation times when only one column, i.e. the overall best (most negative reduced column), was added after each master optimization. Finally, row 11 contains the computation times when the master was re-optimized after the generation of only one column, instead of one column for each activity. Recall that the dynamic program could not be extended with upper bound pruning in the first branching scheme, since the 2nd, 3rd, …, kth best column may be required. Recall also that column elimination is inherent in the second branching scheme.

Conclusions with respect to this experiment are twofold. Firstly, the way of column addition plays a major role in fast convergence of column generation. Adding only one (optimal) column per master optimization seems to be outperformed by adding *k* (suboptimal) columns per master optimization. The main reason for the large difference between (10) and (11) is probably the impossibility in (11) to prune nodes based on Lagrange relaxation.

Secondly, the results clearly indicate the positive impact of all speed-up techniques except for subgradient optimization. The initial heuristic and initial network restrictions turned out to have the smallest impact on the computation times.

Table III: Contributions of speed-up techniques

		Average computation time (s)	
		Branching on columns	Branching on timetable cells
Basic algorithm	(1)	36.00	36.17
Without initial heuristic (section *3.6.1*)	(2)	37.94	41.60
Without initial network restriction (section *3.6.3*)	(3)	36.14	42.60
DP without upper bound pruning (section *3.6.4*)	(4)	-	49.51
Without Lagrange dual pruning (section *3.6.2*)	(5)	42.57	49.96
Minimum bounding search strategy (section *3.6.6*)	(6)	56.66	60.97
Solving master LP starting from empty basis (section *3.6.5*)	(7)	55.92	67.80
With subgradient optimization[3] (section *3.6.8*)	(8)	65.88	81.44
Without column elimination first branching strategy (section *3.6.7*)	(9)	93.46	-
Search *k* columns, add 1 column per master optim. (section *3.4*)	(10)	72.33	76.03
Search 1 column, add 1 column per master optim. (section *3.4*)	(11)	141.09	148.31

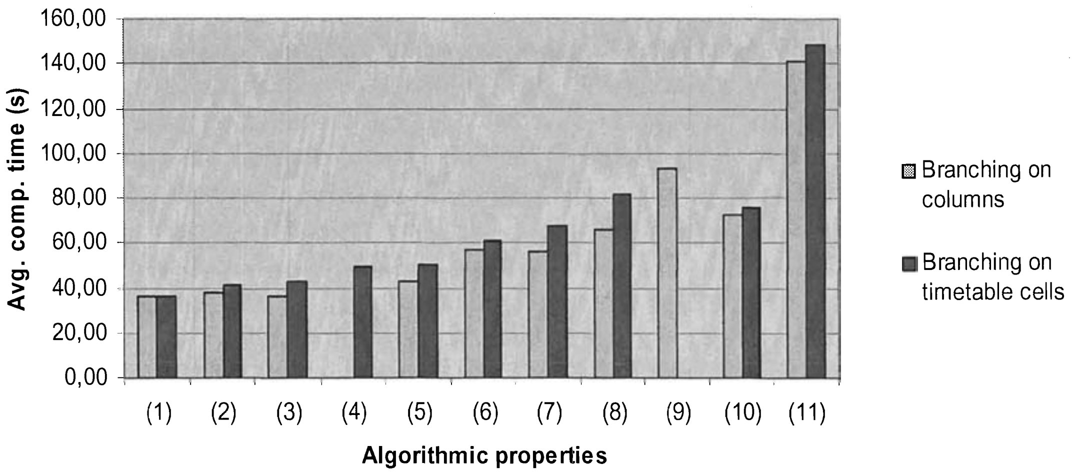


Figure 10: Contributions of algorithmic improvements

[3] Subgradient optimization settings include:
- 10 subgradient optimization iterations per simplex iteration;
- 50 iterations of dual price updating within each subgradient optimization iteration;
- new columns priced out based on the optimal dual prices.

5 Conclusions and future research

In this paper the problem of scheduling trainees at a hospital department was addressed. In the first part, the problem was stated and formulated as an integer program. Next, a branch-and-bound algorithm was proposed to solve the problem to optimality. The drawbacks of this approach served as a link towards a branch-and-price approach. Therefore, the problem was reformulated as a zero-one multi-commodity flow problem with side-constraints in which we decomposed on the activities. In the next sections the different parts of the branch-and-price algorithm were discussed extensively. The pricing problem could be formulated as a constrained shortest path problem and can be solved efficiently using a forward dynamic programming approach. A very important feature of this dynamic program is the ability to find also the 2^{nd}, 3^{rd} or k^{th} shortest path at a very low computational extra cost. This property enabled us to develop a branching scheme based on the column variables. Alternatively, a branching scheme based on timetable cells and two precedence relations based branching schemes were elaborated. Finally, several speed-up techniques were discussed. In the next part, extensive computational results were presented. An experiment was set up in which the influence of six factors on the complexity of the problem was investigated and the four branching schemes were compared. Concerning theoretical issues, there are two main conclusions. The first one is that the branch-and-price algorithm clearly outperforms the branch-and-bound algorithm. The second is that, within the branch-and-price algorithm, branching on the timetable cells turned out to provide the best results in the most consistent way. Compared to branching on the column variables, this branching scheme is more robust when the magnitude of the non-availability costs contains variability. The branching schemes based on precedence relations converge more slowly to an optimal solution and moreover are not guaranteed to branch until a completely integer solution has been found.

Concerning practical issues, the application makes it possible to find better solutions in less time compared to previous ways of scheduling. To illustrate this, earlier schedules were built for 18 periods. These 18 periods represented 52 weeks (16 3-week periods and 2 2-week periods). If a trainee was not available during a certain week, the full period was made unavailable (for scheduling the difficult activities). The developed application is able to deal with scheduling problems for 52 periods. Also, the formerly seniority based division of weeks-off can now be replaced by an approach that takes as much as possible all preferences of all trainees into account. Of course, senior trainees may still be given more priority by assigning them a larger total amount of non-availability costs.

Despite all the improvements, the borders of optimality searching within reasonable time were reached when considering problems starting from twelve trainees and ten activities. Obviously, the exact branch-and-price algorithm can also provide heuristic solutions by allowing a gap between the lower and upper bound. It would, however, be interesting to develop robust heuristic solution procedures that provide good solutions in small computation times. Another interesting research direction would be to adapt the existing branch-and-price algorithm to handle setup costs explicitly. Setup costs would occur each time an assistant (re)starts a certain activity. In this way, one could search for the optimal tradeoff between assigning preferred weeks-off and splitting up activities within trainees.

6 Acknowledgements

We acknowledge the support given to this project by the *Fonds voor Wetenschappelijk Onderzoek* (*FWO*) - *Vlaanderen*, Belgium under contract number G.0463.04. We are grateful to Prof. Dr. W. Sermeus and Kris Vanhaecht of the *Centrum voor Ziekenhuis- en Verplegingswetenschap* (*CZV*) for drawing our attention to challenging scheduling problems in health care management and to Jenny Cristael, (*Oogziekenhuis UZ Leuven*, Belgium) for providing case study data.

7 References

Aickelin, U. and Dowsland, KA., *Exploiting problem structure in a genetic algorithms approach to a nurse rostering problem*, Journal of Scheduling, Vol. 31, pp. 139-153, 2000.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Martin, W.P., Savelsbergh, W.P. and Vance P.H., *Branch-and-price: column generation for solving huge integer programs*, Operations Research, Vol. 46(3), pp. 316-329, 1998.

Beaumont, N., *Scheduling staff using mixed integer programming*, European Journal of Operational Research, Vol. 98, pp. 473-484, 1997.

Bellman, R., *Dynamic Programming*, Princeton University Press, 1957.

Brusco, MJ and Jacobs, LW, *A simulated annealing approach to the cyclic staff-scheduling problem*, Naval Research Logistics, Vol. 40, pp. 69-84, 1993.

Bosi, F. and Milano, M., *Enhancing constraint logic programming branch and bound techniques for scheduling problems*, Software Practice & Experience, Vol. 31, John Wiley & Sons, 2001.

Burke, E.K., De Causmaecker, P. and Vanden Berghe, G., *A hybrid tabu search algorithm for the nurse rostering problem*, in *Selected Papers from the 2nd Asia Pacific Conference on Simulated Evolution and Learning*, Springer Verlag, Vol. 1585 of LNAI, pp. 187-194, 1998.

Cappanera, P. and Gallo, G., *A multi-commodity flow approach to the crew rostering problem*, Technical Report, TR-01-08, Dip. di Informatica, Univ. di Pisa, 2001.

Cheang B., Li H., Lim A., Rodrigues B., *Nurse rostering problems – A bibliographic survey*, European Journal of Operations Research, Vol. 151, pp. 447-460, 2003.

Demeulemeester, E. and Herroelen, W.S., *A branch-and-bound procedure for the multiple resource-constrained project scheduling problem*, Management Science, Vol. 38(12), pp. 1803-1818, 1992.

Dreyfus, S.E. and Law, A.M., *The art and theory of dynamic programming*, Academic Press, Inc. Ltd., 1977.

Hans, E.W., *Resource loading by branch-and-price techniques*, Twente University Press, Enschede, The Netherlands, pp. 38-42, 2001.

Jans, R., *Capacitated lot sizing problems: New applications, formulations and algorithms*, Doctoraal Proefschrift, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002.

Jiménez, V.M. and Marzal, A., *Computing the K shortest paths: A new algorithm and an experimental comparison*, in *Lecture Notes in Computer Science Series*, J.S. Vitter and C.D. Zaroliagis (eds.), Spinger-Verlag, Vol. 1668, pp. 15-29, 1999.

Mason, A.J. and Smith, M.C., *A nested column generator for solving rostering problems with integer programming*, in *International Conference on Optimisation: Techniques and Applications*, L. Caccetta; K. L. Teo; P. F. Siew; Y. H. Leung; L. S. Jennings, and V. Rehbock (eds.), Curtin University of Technology, Perth, Australia, pp. 827-834, 1998.

Mehrotra, A., Murphy, K.E. and Trick, M.A., *Optimal shift scheduling: A branch-and-price Approach,* Naval Research Logistics, Vol. 47, pp. 185-200, 2000.

Peeters, M., *One dimensional cutting and packing: New problems and algorithms*, Doctoraal Proefschrift, Faculteit Economische en Toegepaste Economische Wetenschappen, 2002.

Ryan, D.M. and Foster, B.A., *An integer programming approach to scheduling*, Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, A. Weren (ed.), North-Holland, Amsterdam, pp. 269-280, 1981.

Van den Akker, M., Hoogeveen H. and Van de velde, S.L., *Combining column generation and lagrangian relaxation to solve a single-machine common due date problem*, INFORMS Journal on Computing, Vol. 14(1), pp. 37-51, 2002.

Vanderbeck, F., *On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm*, Operations Research, Vol. 48(1), pp. 111-128, 2000.

Vanderbeck, F. and Wolsey, L.A., *An exact algorithm for IP column generation*, Operations Research Letters, Vol. 19, pp. 151-159, 1996.

Warner, M.D., *Scheduling nursing personnel according to nurses preference: A mathematical programming approach*, Operations Research, Vol. 24(5), pp. 842-856, 1976.

Appendix 1: Computational results

				Branching on column variables			Branching on timetable cells			Branching on immediate precedence relations				Branching on normal precedence relations			
Problem	Root LP	Heur	Opt	Time	Cols	Nodes	Time	Cols	Nodes	Time	Cols	Nodes	Fract	Time	Cols	Nodes	Fract
DOE222111_1	13	14	13	0.22	69	1	0.2	63	0	0.2	63	0	0	0.2	63	0	0
DOE222111_2	16	20	16	0.36	95	3	0.28	80	2	0.28	83	2	0	0.27	79	1	0
DOE222111_3	16	17	16	0.2	60	1	0.25	69	4	0.25	65	5	0	0.31	78	8	0
DOE222112_1	36	41	36	0.2	60	0	0.25	73	2	0.38	104	3	0	0.38	107	3	0
DOE222112_2	50	53	50	0.22	69	0	0.28	84	2	0.28	89	2	0	0.44	116	5	0
DOE222112_3	45	45	45	0.13	43	0	0.11	39	0	0.09	39	0	0	0.11	39	0	0
DOE222121_1	11	12	11	0.25	75	0	0.28	80	1	0.31	90	1	0	0.36	91	1	0
DOE222121_2	15	19	15	0.44	121	2	0.28	83	1	0.28	81	2	0	0.3	83	2	0
DOE222121_3	16	17	16	0.36	103	3	0.41	106	4	0.31	88	2	0	0.53	130	8	0
DOE222122_1	42	44	42	0.42	110	4	0.38	103	3	0.41	108	4	1	0.36	96	3	0
DOE222122_2	33.5	43	34	0.34	94	1	0.33	94	1	0.31	90	1	0	0.31	90	1	0
DOE222122_3	31	-	31	0.33	94	0	0.33	91	0	0.31	91	0	0	0.33	91	0	0
DOE222131_1	10	14	10	0.63	154	3	0.64	167	2	0.58	143	2	0	0.5	133	1	0
DOE222131_2	13.5	16	14	0.34	95	0	0.55	138	4	0.67	163	6	0	0.55	136	4	0
DOE222131_3	10.66	16	12	2.91	480	28	1.64	334	12	1.45	295	9	0	1.91	359	17	0
DOE222132_1	35.75	45	37	3.17	404	60	0.98	216	9	1.38	278	15	0	0.72	156	8	1
DOE222132_2	33.5	50	35	1.03	218	7	1.01	223	6	1.05	224	8	0	1.06	224	8	0
DOE222132_3	31.5	-	33	1.88	340	18	1.3	312	8	1.89	413	9	0	1.98	360	16	1
DOE222141_1	9.56	13	10	0.83	197	2	0.95	214	7	0.98	221	5	0	1.16	242	8	0
DOE222141_2	12.3	-	13	0.72	161	5	1.05	222	12	0.97	215	7	0	1.23	256	9	0
DOE222141_3	13.33	-	14	0.42	112	1	0.69	156	6	0.66	155	5	0	0.59	133	5	0
DOE222142_1	28.83	45	31	2.81	433	25	2.25	419	16	2.22	421	16	0	4.48	712	38	0
DOE222142_2	30.8	40	32	2.17	380	19	2.13	401	16	2.14	388	16	0	2.23	397	15	0
DOE222142_3	22.1	25	24	8.47	705	120	2.38	425	20	2.77	488	21	0	4.03	652	29	0
DOE222151_1	14	16	14	0.38	100	2	0.77	183	5	0.75	182	5	0	0.59	138	7	0
DOE222151_2	13.38	16	14	0.55	134	2	0.56	143	3	0.73	168	5	0	0.73	159	6	0
DOE222151_3	13.33	16	14	0.41	105	2	0.47	114	6	0.42	109	4	0	0.56	128	9	0
DOE222152_1	43	54	43	0.25	74	1	0.45	116	2	0.42	113	1	0	0.47	119	1	0
DOE222152_2	42	49	43	3	469	41	1.58	312	15	1.33	263	11	1	1.5	288	13	0
DOE222152_3	37	44	37	0.34	95	1	0.33	94	0	0.33	93	0	0	0.33	93	0	0
DOE222161_1	12	14	12	0.86	198	3	0.8	189	4	0.63	150	2	0	0.66	147	4	0
DOE222161_2	10.63	13	11	0.88	192	4	0.72	166	5	0.97	211	5	0	1.44	282	10	0
DOE222161_3	12.5	14	13	0.78	176	7	0.45	119	3	0.69	172	4	0	0.45	113	3	0
DOE222162_1	32	47	32	0.56	146	0	0.77	191	2	0.83	201	2	0	0.8	194	1	0
DOE222162_2	37	45	37	0.39	106	0	0.44	120	0	0.44	120	0	0	0.44	120	0	0
DOE222162_3	33.58	43	35	1.64	349	8	1.64	341	7	1.75	368	7	0	2.36	454	10	0
DOE222211_1	13	14	13	1.03	227	4	1.53	325	14	1.31	267	10	0	2.05	336	23	0
DOE222211_2	14	14	14	0.69	153	6	0.31	90	0	0.3	84	0	0	0.3	84	0	0
DOE222211_3	13	16	13	1.34	281	7	1.36	286	7	1.59	317	7	0	1.84	319	17	0
DOE222212_1	33	41	33	0.88	210	3	1.42	299	12	1.38	286	11	0	1.8	342	16	0
DOE222212_2	31	36	31	0.34	90	0	1.3	273	12	1.36	295	10	0	1.73	297	24	0
DOE222212_3	26	32	26	1.06	246	5	1.63	352	12	1.53	309	10	0	1.88	327	19	0
DOE222221_1	6	9	6	3.39	507	3	7.61	1025	10	5.75	718	9	0	15.56	1271	16	0
DOE222221_2	9	12	9	2.11	315	2	3.41	526	10	5.69	765	10	0	5.88	637	12	0
DOE222221_3	8	15	8	2.59	385	3	3.39	507	7	4.25	603	7	0	3.91	422	11	0
DOE222222_1	26	35	26	4.38	656	5	4.69	691	4	4.02	562	6	0	5.98	634	14	0
DOE222222_2	31.57	42	33	157.94	3844	290	61.45	2984	141	23.72	1732	39	0	74.88	2831	147	0
DOE222222_3	18	27	18	7.75	1005	11	3.61	534	3	3.66	522	2	0	5.19	576	5	0
DOE222231_1	6	12	6	3.48	432	1	6.99	737	6	19.88	1417	18	0	20.34	1211	18	0
DOE222231_2	6	10	7	123.25	3854	130	45.22	2791	45	42.8	2496	38	0	61.94	2585	67	1
DOE222231_3	6	9	6	3.2	336	0	4.06	502	1	5.7	614	2	0	9.72	745	5	0
DOE222232_1	17	27	18	361.89	8807	354	119.61	4431	111	66	2947	63	0	110.95	3738	89	0
DOE222232_2	15.1	21	16	9.59	1062	7	78.73	3639	100	14.13	1172	15	0	16.53	1072	21	0
DOE222232_3	15	25	15	5.88	711	5	6.86	790	11	26.14	1840	22	0	35.47	1819	31	0
DOE222241_1	7	9	7	5.84	581	20	6.22	733	12	7.13	755	16	0	7.27	474	20	0
DOE222241_2	8	9	8	4.13	376	3	6.56	652	16	8.31	671	18	0	12.28	697	32	0
DOE222241_3	5	9	5	15.3	1285	19	7.06	679	7	8.64	666	7	0	25.97	1224	18	0
DOE222242_1	18	25	18	57.14	3258	46	11.02	1001	7	89.09	3126	61	0	67.38	2306	29	0
DOE222242_2	8	24	8	8	717	5	8.42	970	7	13.39	1078	11	0	16.06	887	9	0
DOE222242_3	11	34	11	10.52	751	2	8.89	789	2	19.06	1255	6	0	34.13	1553	11	0
DOE222251_1	8.1	12	9	2.86	432	4	2.55	390	4	5.34	671	8	0	11.2	1035	28	0
DOE222251_2	9	13	10	9.88	973	15	37.73	2540	69	273.53	4984	252	0	>600	7134	506	0
DOE222251_3	6.51	13	7	9.08	1063	50	29.54	2015	50	12.99	1193	16	0	25.47	1515	30	0
DOE222252_1	10.81	27	13	>600	13252	1163	453.45	10680	684	374.58	8504	465	0	561.53	10154	592	0
DOE222252_2	22.58	38	24	63.9	3410	146	9.57	955	14	20.52	1634	27	0	39.15	2275	58	0
DOE222252_3	19.49	29	22	512.87	9910	589	174.91	7359	211	257.19	5919	225	0	171.37	4916	192	0
DOE222261_1	6	11	6	18.61	1635	27	19.77	1728	20	12.24	1007	10	0	11.47	731	16	0
DOE222261_2	7	10	8	121.47	3804	127	36.12	2485	44	57.07	2936	59	0	35.17	1759	42	0
DOE222261_3	4.01	10	5	35.36	2033	108	30.58	2425	44	7.91	719	10	0	10.52	693	23	0
DOE222262_1	14.14	38	15	9.73	860	7	10.97	997	12	19.95	1349	21	0	15.11	1006	15	0
DOE222262_2	20.29	27	21	14.25	1254	15	25.14	2123	29	40.59	2184	32	0	133.01	3504	74	0
DOE222262_3	13.98	28	16	344.36	7687	1198	113.02	5798	243	184.55	5694	274	0	193.96	6041	312	0
DOE222311_1	15	15	15	0.28	70	0	0.33	87	0	0.28	77	0	0	0.31	77	0	0
DOE222311_2	15	15	15	0.42	95	0	0.34	89	0	0.34	89	0	0	0.36	89	0	0
DOE222311_3	12	13	12	0.41	96	0	1.19	238	15	1.28	252	14	0	1.31	193	17	0
DOE222312_1	37	37	37	0.55	125	0	0.36	89	0	0.39	100	0	0	0.44	100	0	0
DOE222312_2	34	39	34	0.39	90	0	1.42	300	15	1.2	216	14	0	2.23	327	32	0
DOE222312_3	40	40	40	0.52	117	0	0.37	96	0	0.39	100	0	0	0.42	100	0	0

				Branching on column variables			Branching on timetable cells			Branching on immediate precedence relations				Branching on normal precedence relations			
Problem	Root LP	Heur	Opt	Time	Cols	Nodes	Time	Cols	Nodes	Time	Cols	Nodes	Fract	Time	Cols	Nodes	Fract
DOE222321_1	6	9	6	6.78	561	2	15.27	1412	15	12.15	910	18	0	21.52	1076	40	0
DOE222321_2	5	11	5	7.25	642	2	18.13	1380	11	9.11	628	7	0	122.44	3554	62	0
DOE222321_3	8	11	8	5.69	527	4	15.33	1429	18	10.83	880	15	1	19.6	1068	28	1
DOE222322_1	14	24	14	24.9	1555	7	18.8	1409	7	41.35	1886	17	0	69.25	2114	16	0
DOE222322_2	18	28	18	10.14	820	6	13.85	1167	13	12.97	839	7	0	24.9	948	13	0
DOE222322_3	10	26	10	12.77	1283	16	6.6	805	6	6.72	755	11	0	27.45	1664	28	0
DOE222331_1	5	8	5	8.49	501	2	14.74	1182	18	15.2	705	10	0	52.66	1432	29	0
DOE222331_2	5	9	5	13.85	916	5	13.55	1251	16	20.78	1236	18	0	27.13	827	27	0
DOE222331_3	6	9	6	10.02	664	5	16.02	1112	15	29.52	1324	17	0	140.12	3406	58	0
DOE222332_1	17	26	17	14.2	831	6	10.78	969	9	13.13	999	18	0	31.7	1134	30	0
DOE222332_2	12	24	12	14	772	5	12.5	902	8	18.78	1011	10	0	42.2	1425	19	0
DOE222332_3	8.6	20	9	29.31	1586	20	90.05	3384	58	29.16	1532	15	0	58.11	1757	24	0
DOE222341_1	4	6	4	12.44	494	2	25.91	1159	19	32.47	1052	16	0	67.01	1109	27	0
DOE222341_2	5	8	5	18.72	777	4	18.81	1281	17	32.16	1404	25	0	104.15	1716	46	0
DOE222341_3	2	6	2	13.56	567	2	14.45	891	10	27.3	966	9	0	43.86	925	19	0
DOE222342_1	13	19	13	24.11	1015	14	24.59	1443	18	22.13	851	11	0	50.77	929	28	0
DOE222342_2	4	11	4	40.08	1455	27	29.33	1467	16	34.59	1226	16	0	77.47	1383	30	0
DOE222342_3	4	17	4	20.31	827	4	19.11	1075	12	32.99	1178	14	0	49.38	1077	28	0
DOE222351_1	5	8	5	11.72	677	3	18.81	1442	16	9.48	772	12	0	13.91	786	18	0
DOE222351_2	9.37	12	10	20.89	1424	34	15.05	1378	24	20.44	1351	14	0	378.05	6198	164	0
DOE222351_3	5.69	10	6	165.3	4671	255	18.7	1236	15	46.28	1819	32	0	225.8	3486	95	0
DOE222352_1	10.83	19	11	14.22	1144	9	15.89	1257	11	71.78	2520	41	0	150.34	3472	63	0
DOE222352_2	18.29	27	19	>600	9249	623	247.14	5900	99	256.78	5050	104	0	>600	6964	229	0
DOE222352_3	18	24	18	9.84	703	9	8.16	646	7	36.75	1341	18	0	160.02	2938	51	0
DOE222361_1	7	10	7	14.08	649	4	16.64	933	12	24.82	919	11	0	55.21	1077	30	0
DOE222361_2	5	7	5	23.05	1056	13	14.8	810	7	>600	7174	102	0	259.96	3616	59	0
DOE222361_3	5	9	5	65.8	2606	38	>600	9508	173	393.65	7522	132	0	298.31	4419	69	0
DOE222362_1	8	19	8	34.73	1681	23	38.97	2354	29	47.78	1855	17	0	39.08	1012	16	0
DOE222362_2	15	26	15	83.53	3091	20	14.13	1046	8	61.13	2432	18	0	97.81	2619	22	0
DOE222362_3	6	15	7	234.16	6940	273	>600	10322	226	>600	7921	155	0	>600	7615	140	0
DOE222411_1	12.25	-	15	>600	11797	3031	129.77	5981	515	>600	7214	1159	0	>600	7502	1346	0
DOE222411_2	10.78	-	-	>600	12467	1856	>600	13338	2064	>600	7322	1200	0	>600	7991	1534	0
DOE222411_3	10.64	-	-	>600	10479	1409	>600	12713	1941	>600	9326	1304	0	>600	8044	1383	0
DOE222412_1	26.43	-	29	53.36	2955	328	24.56	2436	94	19.05	1997	59	0	37.77	2641	110	0
DOE222412_2	24.5	-	28	11.86	1447	70	10.02	1266	65	26.86	2226	189	1	83.03	3911	561	0
DOE222412_3	32.25	-	-	>600	12525	3968	>600	17638	3646	>600	12019	2860	0	>600	13329	2583	0
DOE222421_1	6.57	12	7	9.97	1216	27	21.72	1928	33	53.23	2848	46	0	7.27	649	12	0
DOE222421_2	7.41	13	8	4.5	460	3	6.69	687	13	8.45	745	10	0	13.23	842	18	0
DOE222421_3	8	10	8	13.59	1119	37	5.89	622	8	6.14	502	4	0	12.84	626	10	0
DOE222422_1	13	29	-	>600	12668	864	>600	13691	452	>600	11584	357	0	>600	11748	385	0
DOE222422_2	17.6	29	20	394.74	8729	425	75.22	4610	66	154.2	5828	90	0	364.47	7436	182	0
DOE222422_3	12.29	25	13	8.31	928	10	14.09	1336	11	8.86	809	5	0	12.28	755	10	0
DOE222431_1	5	7	6	25.14	1352	17	15.44	1241	14	41.98	1892	29	1	31.03	1162	27	1
DOE222431_2	4	10	4	4.7	405	1	9.42	780	5	7	554	2	0	9.09	530	4	0
DOE222431_3	4.89	9	6	19.42	1098	14	14	960	14	22.44	1184	15	0	53.45	1787	37	0
DOE222432_1	9.7	19	12	64.34	2693	90	77.61	4047	76	46.56	2610	36	0	71.06	2707	54	0
DOE222432_2	20.49	33	21	7.44	564	3	8.16	607	6	9.33	606	5	0	18.19	745	8	0
DOE222432_3	10	23	10	8.86	528	1	5.89	553	2	6.78	576	2	0	7.92	498	1	0
DOE222441_1	5.04	8	6	29.3	1489	20	27.94	1835	36	41.39	1824	17	0	54.14	1550	17	0
DOE222441_2	2	5	2	14.69	746	4	11.06	654	3	18.14	815	6	0	19.89	592	3	0
DOE222441_3	5	7	5	22.3	1105	13	15.2	1120	15	12.19	775	11	0	22.81	975	21	0
DOE222442_1	3	12	3	28.92	1077	7	16.11	925	3	39.25	1640	12	0	72.5	1731	12	0
DOE222442_2	9	16	9	14.83	721	2	35.8	1743	10	45.14	1564	6	0	48.42	1321	16	0
DOE222442_3	12	21	13	69.5	2487	80	26.83	1754	12	26.88	1498	13	0	115.6	3603	47	0
DOE222451_1	6.78	12	8	23.44	1822	42	268.03	6138	287	8.63	800	10	1	223.12	5367	261	0
DOE222451_2	7.98	12	10	>600	5747	447	340.81	9565	374	601	11059	580	0	>600	11060	530	0
DOE222451_3	9.56	-	11	>600	4990	556	99.52	4264	167	220.3	5614	289	0	424.22	7684	516	0
DOE222452_1	14.56	35	19	254.66	8220	481	140	7243	140	211.41	8606	181	0	242.19	8003	182	0
DOE222452_2	14.83	33	16	15.27	1079	9	12.88	1073	12	14.59	1089	8	1	25.8	1310	18	0
DOE222452_3	22	36	25	344.08	10188	727	245.36	9327	258	>600	13211	396	0	>600	13195	486	0
DOE222461_1	5.19	9	7	565.67	7296	586	152.07	4598	111	230.35	5295	109	0	>600	6326	236	0
DOE222461_2	7	10	7	7.94	425	4	105	3253	55	31.23	934	6	0	65.44	1204	9	0
DOE222461_3	4	7	4	38.48	2042	23	18.27	1206	6	9.97	643	4	0	17.45	699	6	0
DOE222462_1	10	18	10	4.22	413	2	3.41	412	3	7.06	680	7	0	10.2	724	12	0
DOE222462_2	14.65	37	18	>600	11515	564	327	10757	250	515.46	12880	340	0	>600	10379	308	0
DOE222462_3	11.56	25	13	86.82	3838	91	86.05	3867	47	79.45	2910	38	0	152.89	3269	74	0
DOE111231_1	1	1	1	0.03	7	0	0.06	14	0	0.06	14	0	0	0.06	14	0	0
DOE111231_2	6	6	6	0.03	6	0	0.02	6	0	0.01	6	0	0	0.02	6	0	0
DOE111231_3	0	0	0	0	3	0	0	3	0	0	3	0	0	0	3	0	0
DOE111232_1	1	3	1	0.09	18	0	0.11	20	0	0.11	20	0	0	0.11	20	0	0
DOE111232_2	1	2	2	0.36	48	5	0.3	38	3	0.23	29	2	0	0.22	29	2	0
DOE111232_3	8	8	8	0.05	10	0	0.03	8	0	0.03	8	0	0	0.03	8	0	0
DOE112231_1	3	4	3	0.14	34	0	0.22	47	2	0.25	55	2	0	0.25	58	2	0
DOE112231_2	3	7	3	0.34	89	0	0.28	77	0	0.28	77	0	0	0.28	77	0	0
DOE112231_3	4	5	4	0.17	50	0	0.47	120	4	0.36	88	4	0	0.45	116	4	0
DOE112232_1	9.25	18	10	0.38	94	1	0.48	111	3	0.63	136	5	0	0.44	102	2	0
DOE112232_2	13	16	13	0.09	27	0	0.3	70	3	0.47	96	4	0	0.41	92	4	0
DOE112232_3	11	17	11	0.25	68	0	0.23	64	0	0.34	88	1	0	0.33	90	1	0
DOE113231_1	8	-	8	0.45	127	1	0.83	215	3	0.69	187	3	0	1.59	330	13	0
DOE113231_2	7	9	7	0.72	179	3	0.73	192	4	1.05	250	6	0	0.88	222	6	0
DOE113231_3	8	9	8	0.58	165	1	0.55	156	1	0.91	230	5	0	0.95	228	7	0

				Branching on column variables			Branching on timetable cells			Branching on immediate precedence relations				Branching on normal precedence relations			
Problem	Root LP	Heur	Opt	Time	Cols	Nodes	Time	Cols	Nodes	Time	Cols	Nodes	Fract	Time	Cols	Nodes	Fract
DOE113232_1	16	22	16	0.58	165	0	0.92	243	2	0.73	196	1	0	0.73	196	1	0
DOE113232_2	21.67	-	22	1.19	298	3	1.06	289	3	0.92	253	3	0	1.44	339	6	0
DOE113232_3	15	-	15	0.53	155	0	0.58	157	1	0.58	156	2	0	0.63	156	3	0
DOE121231_1	0	0	0	0	4	0	0	4	0	0	4	0	0	0	4	0	0
DOE121231_2	0	0	0	0	4	0	0	4	0	0	4	0	0	0	4	0	0
DOE121231_3	0	0	0	0	4	0	0	4	0	0	4	0	0	0	4	0	0
DOE121232_1	0	0	0	0	4	0	0	4	0	0	4	0	0	0	4	0	0
DOE121232_2	5	5	5	0.11	20	0	0.13	27	0	0.11	26	0	0	0.13	26	0	0
DOE121232_3	0	0	0	0	4	0	0	4	0	0	4	0	0	0	4	0	0
DOE122231_1	3	4	3	2.81	430	5	3.88	614	13	4.06	538	12	0	6.24	616	23	0
DOE122231_2	2	5	2	2.61	430	3	3.31	548	10	3.89	609	10	0	12.45	1248	26	0
DOE122231_3	1	3	1	1.38	240	2	2.64	497	6	3.7	525	12	0	15.11	1468	46	0
DOE122232_1	9	17	9	3.11	475	4	3.27	550	11	3.67	603	13	0	6.44	633	21	0
DOE122232_2	8	15	8	1.72	261	3	1.88	322	7	3.91	566	12	0	5.63	617	22	0
DOE122232_3	7	9	7	4.03	547	6	4.77	671	15	5.06	585	10	0	8.89	732	21	0
DOE123231_1	11	-	11	4.94	639	9	9.27	1067	20	9.2	899	20	0	24.72	1584	39	0
DOE123231_2	11	15	11	4.61	619	4	9.38	1277	20	23.48	1852	38	0	18.63	1246	43	0
DOE123231_3	9	14	9	4.63	674	4	10.23	1332	21	10.95	1088	20	0	19.17	1253	41	0
DOE123232_1	30	39	30	5.75	769	3	10.77	1277	23	9.23	995	14	0	15.25	1038	35	0
DOE123232_2	28	-	28	7.45	922	6	12.08	1422	19	10.83	1070	16	0	29.75	1822	40	0
DOE123232_3	26	38	26	9.08	1180	7	15.66	2052	23	15.66	1440	17	0	19.77	1405	30	0
DOE131231_1	0	0	0	1.61	145	3	0	6	0	0	6	0	0	0	6	0	0
DOE131231_2	1	1	1	0.48	50	0	0.34	66	0	0.34	65	0	0	0.42	65	0	0
DOE131231_3	0	0	0	0	6	0	0	6	0	0	6	0	0	0	6	0	0
DOE131232_1	2	2	2	0.88	89	0	0.31	76	0	0.28	72	0	0	0.3	72	0	0
DOE131232_2	0	0	0	0	6	0	0	6	0	0	6	0	0	0	6	0	0
DOE131232_3	3	4	3	2.19	161	3	4.05	313	14	4.41	331	20	0	10.17	300	35	0
DOE132231_1	3	4	3	0.5	66	0	5.36	618	30	7.01	658	29	0	11.14	722	70	0
DOE132231_2	1	2	1	4.72	366	5	8.94	685	27	12.28	835	29	0	26.66	1018	49	1
DOE132231_3	0	2	0	3.05	284	5	4.58	637	25	8.66	845	20	0	8.97	728	41	0
DOE132232_1	5	8	5	4.22	330	5	4.45	420	26	8.92	663	24	0	18.89	685	57	0
DOE132232_2	0	3	0	0.01	7	0	4.01	539	35	5.89	719	18	0	8.08	672	41	0
DOE132232_3	5	6	5	5.48	456	2	8.94	740	26	8.86	866	31	0	13.66	689	50	0
DOE133231_1	18	-	18	16.17	1136	7	42.33	2959	50	221.22	5228	152	1	91.09	2509	109	1
DOE133231_2	16	-	16	17.2	1124	7	38.7	2508	47	247.74	4938	172	1	>600	7251	236	0
DOE133231_3	14	-	14	20.2	1413	7	50.58	3558	51	223.36	5524	112	0	>600	8227	210	0
DOE133232_1	26	-	26	26.56	1591	6	64.53	3980	41	128.35	3810	42	0	>600	8558	171	0
DOE133232_2	38	-	38	24.72	1636	8	61.63	3892	47	195.52	5014	58	0	103.22	2228	70	0
DOE133232_3	44	-	44	29.59	1985	11	69.7	3861	47	71.5	2996	56	0	262.07	4916	158	1
DOE141231_1	0	0	0	0	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE141231_2	0	0	0	0	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE141231_3	0	0	0	0	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE141232_1	0	0	0	0	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE141232_2	0	0	0	0	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE141232_3	1	1	1	4.77	95	0	0.84	77	0	0.88	77	0	0	1.69	77	0	0
DOE142231_1	2	2	2	65.42	659	6	3.64	310	0	3.08	271	0	0	5.59	271	0	0
DOE142231_2	5	5	5	24.95	305	0	2.73	235	0	3.05	244	0	0	5.81	244	0	0
DOE142231_3	2	4	2	68.41	740	8	54.78	1242	69	107.72	1870	61	0	497.53	2392	148	0
DOE142232_1	12	14	12	77.33	787	9	109.82	1401	63	175.47	1828	60	0	563.6	1285	103	0
DOE142232_2	2	4	2	57.65	639	7	94.09	1086	65	169.45	2264	70	1	>600	1035	93	0
DOE142232_3	11	13	11	88.53	978	6	65.02	1257	59	113.76	1985	64	1	>600	1480	149	0
DOE143231_1	6	10	6	218.26	2500	10	349.81	5015	74	>600	6550	124	0	>600	1477	108	0
DOE143231_2	8	9	8	140.03	1486	7	166.36	2363	84	>600	5406	104	0	>600	871	61	0
DOE143231_3	12	13	12	50.73	578	0	274.35	3151	81	245.24	1982	80	1	>600	844	78	0
DOE143232_1	21	29	21	154.42	1411	7	349.31	4078	92	>600	6103	159	0	>600	912	58	0
DOE143232_2	16	26	16	150.01	1397	10	275.31	2341	81	>600	4601	73	0	>600	772	55	0
DOE143232_3	23	27	23	146.28	1271	7	315.14	4105	68	>600	4064	73	0	>600	858	56	0
DOE211231_1	5	5	5	0.05	8	0	0.06	10	0	0.05	10	0	0	0.05	10	0	0
DOE211231_2	5.5	6	6	0.06	12	0	0.03	7	0	0.03	7	0	0	0.05	7	0	0
DOE211231_3	4	4	4	0.06	10	0	0.06	11	0	0.06	11	0	0	0.06	11	0	0
DOE211232_1	8	8	8	0.03	6	0	0.03	6	0	0.03	6	0	0	0.03	6	0	0
DOE211232_2	5	12	5	0.17	31	0	0.16	33	0	0.17	33	0	0	0.17	33	0	0
DOE211232_3	11	21	11	0.11	20	0	0.11	22	0	0.11	22	0	0	0.11	22	0	0
DOE212231_1	5	13	5	0.25	50	1	0.33	70	2	0.33	69	3	0	0.38	80	3	0
DOE212231_2	7	8	8	0.48	94	3	0.61	124	4	0.61	120	4	0	0.64	122	5	0
DOE212231_3	8	12	8	0.38	81	1	0.33	78	2	0.34	74	3	0	0.36	79	3	0
DOE212232_1	24	29	25	0.89	160	8	0.75	152	5	0.69	142	2	0	0.95	184	6	1
DOE212232_2	20.5	24	21	0.28	58	2	0.25	58	1	0.28	64	2	0	0.3	63	3	0
DOE212232_3	24.33	36	25	0.34	77	0	0.45	90	4	0.52	105	4	1	0.48	89	5	0
DOE213231_1	13	14	13	0.7	165	0	0.94	208	1	0.94	208	1	0	0.95	205	1	0
DOE213231_2	13	15	13	0.67	154	1	0.8	172	4	0.81	172	5	0	0.97	196	6	0
DOE213231_3	13	15	13	1.06	237	1	0.83	186	0	1.03	232	1	0	1	223	1	0
DOE213232_1	41	44	41	0.63	143	1	1.24	248	6	0.75	160	2	0	1.08	213	9	0
DOE213232_2	39	44	39	0.5	111	2	0.75	156	3	0.88	183	5	0	1	196	7	0
DOE213232_3	45	47	45	1.03	211	4	0.52	121	2	0.81	171	4	0	0.66	145	3	0
DOE221231_1	1	1	1	0.16	28	0	0.17	33	0	0.16	32	0	0	0.17	32	0	0
DOE221231_2	2.17	3	3	0.75	86	0	0.66	99	0	0.47	75	0	0	0.53	75	0	0
DOE221231_3	0	1	0	0.38	50	1	0.3	48	1	0.56	82	3	0	0.67	90	4	0
DOE221232_1	0	0	0	0.41	50	1	0	4	0	0	4	0	0	0	4	0	0
DOE221232_2	1	1	1	0.17	24	0	0.2	40	0	0.22	38	0	0	0.23	38	0	0
DOE221232_3	0	0	0	0	4	0	0.02	4	0	0	4	0	0	0	4	0	0

				Branching on column variables			Branching on timetable cells			Branching on immediate precedence relations				Branching on normal precedence relations			
Problem	Root LP	Heur	Opt	Time	Cols	Nodes	Time	Cols	Nodes	Time	Cols	Nodes	Fract	Time	Cols	Nodes	Fract
DOE222231_1	6	11	6	17.31	1456	18	12.19	1089	10	22.03	1326	16	0	27.47	1445	23	0
DOE222231_2	5.82	12	7	147.08	3888	200	69.8	3325	86	76.03	3676	82	0	121.39	3829	155	0
DOE222231_3	6.42	10	7	36.25	2252	58	8.25	840	11	133.17	3774	86	0	449.3	6863	233	0
DOE222232_1	12.39	22	13	57.16	2824	74	25.98	1742	21	18.64	1197	15	0	71.73	2569	49	0
DOE222232_2	17	32	17	6.31	614	6	14.45	1273	13	7.51	672	3	0	16.42	1082	13	0
DOE222232_3	16	34	16	12.06	1007	7	7.22	700	4	5.19	495	2	0	8.55	642	9	0
DOE223231_1	13.09	-	14	41.61	3230	28	>600	9976	191	71.52	3154	25	0	>600	8596	156	0
DOE223231_2	13	-	-	>600	12231	286	>600	11827	239	>600	10656	148	0	>600	9782	148	0
DOE223231_3	14.17	21	15	24.56	1826	14	435.92	8091	119	37.47	1868	14	0	41.08	1735	28	0
DOE223232_1	37	-	37	12.95	1070	6	32.45	2006	15	38.94	1792	14	0	51.66	1980	22	0
DOE223232_2	39.14	-	-	>600	8328	189	>600	9718	141	>600	8950	126	0	>600	8583	135	0
DOE223232_3	29	77	-	>600	13427	234	>600	12784	171	>600	10899	82	0	>600	11191	87	0
DOE231231_1	0	0	0	1.77	141	3	0	6	0	0	6	0	0	0	6	0	0
DOE231231_2	3	3	3	1.2	119	0	0.64	95	0	0.53	83	0	0	0.59	83	0	0
DOE231231_3	1	2	1	3.88	250	3	3.06	251	10	5.19	415	15	0	5.25	227	18	0
DOE231232_1	3	5	3	3.28	198	4	4.42	319	10	3.64	247	11	0	6.23	245	23	0
DOE231232_2	0	1	0	2.34	172	6	2.23	265	15	3.34	364	15	0	4.39	400	20	0
DOE231232_3	0	4	0	2.03	172	3	3.05	362	12	3.34	362	13	0	4.08	350	15	0
DOE232231_1	3	6	3	11.95	699	9	9.36	793	17	43.83	2123	35	0	27.78	983	23	0
DOE232231_2	1	5	1	15.69	904	4	58.59	2606	31	15.5	809	9	0	33.14	1075	12	0
DOE232231_3	6	10	6	7.95	470	4	18.34	1034	17	23.28	1068	18	0	27.44	792	35	0
DOE232232_1	5	12	5	19.5	1177	7	32.11	1865	25	31.36	1612	30	0	11.61	613	17	0
DOE232232_2	11	18	11	10.2	491	4	11.73	803	8	14.27	832	15	0	35.02	1125	28	0
DOE232232_3	2	9	2	11.31	681	5	6.84	576	13	16.34	925	13	0	21.58	834	25	0
DOE233231_1	18	-	-	>600	11742	136	>600	9396	66	>600	5875	29	0	>600	5329	48	0
DOE233231_2	19	-	-	>600	12674	199	>600	8739	36	>600	5818	25	0	>600	6123	42	0
DOE233231_3	20	-	-	>600	15023	171	602.1	8948	48	>600	5440	18	0	494.19	5056	45	0
DOE233232_1	41	-	-	>600	13107	99	601.2	7883	20	>600	6007	11	0	>600	4680	22	0
DOE233232_2	52	-	52	81.61	2950	10	242.16	4867	22	>600	5923	27	0	>600	5334	46	0
DOE233232_3	56	-	-	>600	14705	216	>600	9205	88	>600	6871	30	0	>600	6237	60	0
DOE241231_1	0	0	0	0.02	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE241231_2	0	0	0	0	7	0	0	7	0	0.02	7	0	0	0	7	0	0
DOE241231_3	0	0	0	0.02	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE241232_1	0	0	0	0	7	0	0	7	0	0.01	7	0	0	0	7	0	0
DOE241232_2	0	0	0	21.86	248	5	0	7	0	0	7	0	0	0	7	0	0
DOE241232_3	0	0	0	0.01	7	0	0	7	0	0	7	0	0	0.02	7	0	0
DOE242231_1	4	9	4	244.51	2376	14	>600	4877	75	432.5	3005	41	0	>600	1096	70	0
DOE242231_2	0	4	0	81.16	1089	9	226.99	4527	86	>600	5294	44	0	>600	3755	95	0
DOE242231_3	2	6	-	>600	4575	341	>600	7025	72	>600	3711	39	0	>600	1143	66	0
DOE242232_1	2	15	2	159.63	1694	27	>600	6241	118	583	4695	40	0	>600	2222	95	0
DOE242232_2	3	20	3	173.55	2135	11	>600	8812	102	183.64	2694	32	0	>600	4003	82	0
DOE242232_3	1	15	1	288.18	2795	11	362.04	4606	45	>600	3867	37	0	>600	1471	61	0
DOE243231_1	9	19	-	>600	5310	33	>600	4308	34	>600	2276	27	0	>600	973	21	0
DOE243231_2	8	-	-	>600	3986	6	>600	3506	20	>600	2011	21	0	>600	913	16	0
DOE243231_3	6	-	-	>600	4178	4	>600	4652	25	>600	2529	28	0	>600	1215	26	0
DOE243232_1	19	-	-	>600	3551	8	>600	3450	10	>600	2115	15	0	>600	1134	11	0
DOE243232_2	20	43	-	>600	3445	4	>600	4772	24	>600	2129	19	0	>600	1142	15	0
DOE243232_3	30	68	30	479.74	3195	8	>600	3197	13	>600	1992	18	0	>600	1042	13	0
DOE311231_1	9	9	9	0.06	9	0	0.11	17	0	0.13	17	0	0	0.13	17	0	0
DOE311231_2	6	6	6	0.06	10	0	0.05	9	0	0.05	9	0	0	0.06	9	0	0
DOE311231_3	4	4	4	0.13	13	0	0.09	12	0	0.08	12	0	0	0.09	12	0	0
DOE311232_1	23	23	23	0.06	8	0	0.08	11	0	0.06	11	0	0	0.06	11	0	0
DOE311232_2	12	12	12	0.09	14	0	0.05	7	0	0.05	7	0	0	0.03	7	0	0
DOE311232_3	14	14	14	0.08	13	0	0.08	14	0	0.09	14	0	0	0.08	14	0	0
DOE312231_1	11	12	11	0.08	14	0	0.38	50	3	0.25	35	1	0	0.25	35	1	0
DOE312231_2	13	17	13	0.5	76	1	0.39	63	0	0.38	63	0	0	0.39	63	0	0
DOE312231_3	11	12	11	0.14	24	0	0.27	43	1	0.3	42	1	0	0.28	42	1	0
DOE312232_1	28	33	28	0.33	47	1	0.39	58	1	0.33	50	1	0	0.34	53	1	0
DOE312232_2	30	32	30	0.48	71	3	0.33	51	0	0.33	51	0	0	0.33	51	0	0
DOE312232_3	33	33	33	0.14	23	0	0.09	19	0	0.11	19	0	0	0.09	19	0	0
DOE313231_1	18.36	-	19	0.75	133	0	1.36	214	7	1.44	226	5	0	1.72	255	11	0
DOE313231_2	19.5	20	20	0.53	93	0	0.44	81	0	0.44	81	0	0	0.45	81	0	0
DOE313231_3	18	-	18	1.13	175	4	0.64	107	1	0.63	104	1	0	0.63	104	1	0
DOE313232_1	53	-	53	0.88	142	2	0.84	131	3	1.44	219	5	0	1.09	169	6	0
DOE313232_2	51	55	51	1.11	175	2	1.28	207	2	1.56	234	4	0	1.86	272	8	0
DOE313232_3	55	-	55	0.92	157	1	0.89	155	0	0.89	155	0	0	0.91	155	0	0
DOE321231_1	2	3	2	1.31	95	1	1.2	127	1	0.69	74	1	0	0.73	74	1	0
DOE321231_2	1	2	1	0.83	73	1	2.94	231	12	4.99	374	18	0	5.75	410	18	0
DOE321231_3	1	1	1	0.72	55	0	0.2	28	0	0.2	28	0	0	0.22	28	0	0
DOE321232_1	1.21	4	2	1.42	105	0	1.83	179	8	1.78	172	5	0	1.28	116	5	0
DOE321232_2	4.33	7	5	1.13	78	1	1.13	106	1	2.02	168	5	0	3.09	222	11	0
DOE321232_3	10.29	13	11	3.81	231	7	2.48	220	9	1.58	150	5	0	4.2	307	12	0
DOE322231_1	8.15	15	9	29.24	1609	21	9.63	650	4	31.39	1574	16	0	90.04	3065	62	0
DOE322231_2	14	18	14	5.49	353	0	4.56	338	0	4.66	341	0	0	5.63	341	0	0
DOE322231_3	9.34	16	10	26.05	1351	67	9.94	664	11	11.47	718	10	0	12.42	660	17	0
DOE322232_1	23	51	23	6.25	407	1	6.8	502	4	6.41	470	5	0	5.95	380	5	0
DOE322232_2	20	44	-	>600	13527	488	>600	11901	366	302.46	6115	102	0	>600	9358	253	0
DOE322232_3	22.8	40	25	515.91	9512	341	345.18	7984	224	255.94	5362	180	0	326.94	6457	181	0
DOE323231_1	22	-	22	18.75	1003	5	44.33	2118	15	51.3	1868	13	0	83.17	2142	24	0
DOE323231_2	24	-	24	15.98	923	6	23.62	1379	24	31.75	1247	15	0	47.72	1651	30	0
DOE323231_3	23	30	23	267.89	7368	108	158.56	4388	41	177.7	3876	19	0	331.77	5281	38	0

				Branching on column variables			Branching on timetable cells			Branching on immediate precedence relations				Branching on normal precedence relations			
Problem	Root LP	Heur	Opt	Time	Cols	Nodes	Time	Cols	Nodes	Time	Cols	Nodes	Fract	Time	Cols	Nodes	Fract
DOE323232_1	61	-	61	14.02	793	5	30.83	1514	13	56.94	2098	15	0	50.97	1874	28	0
DOE323232_2	63	-	63	15.11	871	4	47.91	2039	14	66.44	2051	11	0	76.61	1966	18	0
DOE323232_3	53	-	53	58.02	2877	24	61.31	2447	13	77.78	2073	8	0	>600	6628	53	0
DOE331231_1	4	7	4	7.27	272	4	11.17	491	11	8.31	346	8	0	15.41	467	22	0
DOE331231_2	0	3	0	9.48	470	4	8.83	543	3	14.84	524	2	0	28.36	717	4	0
DOE331231_3	4	7	4	50.55	1237	34	14.27	681	11	13.3	600	10	0	9.45	389	7	0
DOE331232_1	4	8	4	171.14	2505	191	23.63	1184	18	7.42	402	7	0	21.27	919	13	0
DOE331232_2	1	9	1	10.56	500	2	8.64	588	7	7.09	471	6	0	10.64	560	10	0
DOE331232_3	6	15	7	>600	6563	422	84.44	2626	58	383.95	6809	227	0	>600	8228	348	0
DOE332231_1	9	14	9	172.48	3395	66	345.11	5089	78	597.3	6078	88	0	>600	5541	88	0
DOE332231_2	9	12	9	54.42	1453	8	33.42	1139	14	35.39	826	8	0	60.31	998	12	0
DOE332231_3	5	10	5	>600	6504	377	358.31	6260	154	143.86	2958	36	0	153.4	2906	53	0
DOE332232_1	20	35	20	251.77	3725	135	77.66	2055	15	58.08	1228	11	0	50.18	969	17	0
DOE332232_2	21	37	21	36.42	1136	7	40.67	1524	13	350.62	4744	46	0	63.46	1410	22	0
DOE332232_3	9	24	9	>600	6706	659	27.89	1175	8	>600	6146	107	0	>600	5770	114	0
DOE333231_1	27	-	-	598.6	9247	27	>600	7268	30	>600	4769	16	0	>600	4443	27	0
DOE333231_2	28	-	28	440.78	8727	106	>600	8012	31	>600	4275	15	0	>600	4404	29	0
DOE333231_3	27	-	-	>600	8807	86	>600	6648	23	>600	5259	26	0	>600	3775	46	0
DOE333232_1	68	-	-	>600	6247	4	>600	4423	5	>600	4102	4	0	>600	4003	4	0
DOE333232_2	75	-	-	>600	9474	6	>600	5398	5	>600	4724	4	0	>600	4358	4	0
DOE333232_3	68.08	-	-	>600	7545	8	>600	5056	6	>600	5125	7	0	>600	4649	7	0
DOE341231_1	1	2	1	46.86	489	6	68.54	820	34	105.35	973	26	0	347.27	1428	64	0
DOE341231_2	2	2	2	12.36	132	0	3.72	186	0	4.61	166	0	0	6.75	166	0	0
DOE341231_3	1	1	1	9.14	130	0	2.59	151	0	2.27	148	0	0	2.67	148	0	0
DOE341232_1	0	3	0	0	7	0	14.71	631	35	28.78	1014	25	0	72.65	1477	67	0
DOE341232_2	0	2	0	33.93	330	7	46.26	808	38	73.93	1105	23	0	129.48	932	65	0
DOE341232_3	0	0	0	0	7	0	0	7	0	0	7	0	0	0	7	0	0
DOE342231_1	4	10	-	>600	4225	25	>600	3914	40	>600	1919	15	0	>600	1211	40	0
DOE342231_2	4	15	-	>600	3401	10	>600	3153	17	>600	1815	15	0	>600	1060	13	0
DOE342231_3	7	14	-	>600	3740	55	>600	3334	20	>600	1969	26	0	>600	989	27	0
DOE342232_1	4	38	-	>600	3013	6	>600	3397	10	>600	1973	9	0	>600	1191	12	0
DOE342232_2	7	44	7	431.45	2403	5	>600	4587	25	>600	2608	18	0	>600	1406	32	0
DOE342232_3	6	10	-	>600	3895	28	>600	3527	32	>600	2050	21	0	>600	984	33	0
DOE343231_1	13.78	30	-	>600	1510	0	>600	1805	0	>600	1695	0	0	>600	1105	0	0
DOE343231_2	11	34	-	>600	2235	1	>600	2581	5	>600	1666	5	0	>600	1145	1	0
DOE343231_3	13.17	-	-	>600	1460	0	>600	1590	0	>600	1546	0	0	>600	956	0	0
DOE343232_1	26.45	-	-	>600	1360	0	>600	1710	0	>600	1687	0	0	>600	1107	0	0
DOE343232_2	26.37	-	-	>600	1320	0	>600	1607	0	>600	1584	0	0	>600	1024	0	0
DOE343232_3	31.62	97	-	>600	1410	0	>600	2334	4	>600	1928	2	0	>600	1318	0	0