Open access • Journal Article • DOI:10.1007/S10479-007-0220-2

On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem — Source link

Jeroen Belien, Erik Demeulemeester

Institutions: Katholieke Universiteit Leuven

Published on: 07 Jul 2007 - Annals of Operations Research (Springer US)

Topics: Column generation, Nurse scheduling problem, Job shop scheduling, Dynamic priority scheduling and Fair-share scheduling

Ann Oper Res (2007) 155: 143–166
DOI 10.1007/s10479-007-0220-2

On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem

Jeroen Beliën · Erik Demeulemeester

Published online: 7 July 2007

Abstract In this paper a comparison is made between two decomposition techniques to solve a staff scheduling problem with column generation. In the first approach, decomposition takes place on the staff members, whereas in the second approach decomposition takes place on the activities that have to be performed by the staff members. The resulting master LP is respectively a set partitioning problem and a capacitated multi-commodity flow problem. Both approaches have been implemented in a branch-and-price algorithm. We show a trade-off between modeling power and computation times of both techniques.

Keywords Decomposition · Staff scheduling · Set partitioning · Multi-commodity flow · Branch-and-price

When dealing with staff scheduling problems a common approach is to formulate the problem as an integer linear program (ILP) and solve it with a general-purpose ILP solver (see, e.g., Billionnet 1999). It is however well known that computation times can be dramatically reduced if the problem is decomposed. Decomposition of linear programs, known as Dantzig-Wolfe decomposition, was originally introduced by Dantzig and Wolfe (1960). Decomposition involves the division of the problem into several subproblems that are smaller than the original problem, that contain none (or less) so-called 'complicating constraints' and hence can be solved more efficiently and independently of each other. The question is how to decompose a particular problem in order to efficiently solve the problem. To the best of our knowledge, all decomposition approaches for staff scheduling problems decompose on staff members, i.e., generate columns per staff member (see, e.g., Mehrotra et al. 2000;

J. Beliën (✉) · E. Demeulemeester
Faculty of Economics and Applied Economics, Department DSIM: Decision Sciences & Information Management, Research Center for Operations Management, Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium
e-mail: jeroen.belien@econ.kuleuven.be

E. Demeulemeester
e-mail: erik.demeulemeester@econ.kuleuven.be

Mason and Smith 1998; Jaumard et al. 1998). For certain problems, however, a decomposition scheme on the tasks (further referred to as activities) instead of decomposing on the employees, could also be applied.

The objective of this paper is to compare an activity-based and a staff-based decomposition approach for a particular staff scheduling problem encountered in many hospitals. The problem involves scheduling trainees (graduated students) to perform a number of activities over a time horizon. This problem distinguishes from the classic nurse scheduling problem (NSP) in the fact that the NSP usually deals with detailed shift scheduling, e.g., determining the exact hours nurses have to work during the next month, whereas trainees are scheduled over a much longer time horizon (usually one year). Moreover, in contrast to nurses, trainees still have to complete an education. This education requires that they have to perform a number of widely divergent activities and the capacity of the trainee posts is often limited. Consequently, the set covering constraints in NSP are replaced by set partitioning constraints. A second important difference is the undesirability of a situation where a trainee alternates too much between the activities. Each activity (re)start represents a discontinuity in his/her education and involves a considerable mastering time for the trainee. Hence, when a trainee has a week-off, (s)he cannot simply be replaced by another trainee, due to the difference in qualification and due to the mastering time. Nurses, however, can usually exchange weeks-off rather easily by mutual agreement.

The paper is organized as follows. In Sect. 1 we define the problem and state it mathematically. Section 2 proposes the two different decomposition techniques and discusses their properties. Section 3 compares the resulting pricing problems. Section 4 proposes a branching scheme that could be applied in both approaches. Section 5 provides computational results, while Sect. 6 discusses some modeling issues. Finally, Sect. 7 draws conclusions and lists some topics for further research.

1 Problem statement

The problem addressed in this paper involves the construction of trainee schedules at a hospital department. Trainees are graduate medical science students who wish to specialize further in a specific field of health care. Trainees are a unique type of human resources, for they have an important contribution to the provision of services but at the same time are still in education. This makes the scheduling of the trainees a challenging practice. On the one hand the hospital service level has to be assured, on the other hand the education progress of the trainees has to be monitored. The considered problem deals with the following constraints. First of all, there are a number of coverage constraints that ensure that each activity has to be performed at each period by exactly one trainee out of a given set. These trainee sets are basically composed by trainees having the same experience level. Second, in order to meet formation objectives each trainee has to perform each activity between a minimum and maximum number of periods. Third, trainees are not always available to be scheduled; it is known for each time period which trainees are available to be scheduled and which are not. On beforehand, the trainees have to quantify their preferences for having weeks-off. To this aim, they divide a number of points 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. Finally, there is some setup cost when a trainee starts a new activity. Indeed, each new activity start of a trainee takes some time to master the skills required for the activity. Therefore, in order to maximize the efficiency of the service provided, we cannot spit activities up per trainee. This restriction also improves the quality of service, as

the patients have a larger chance to be treated by one and the same trainee. Hence, 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.

Due to all these constraints, the development of trainee schedules is an extremely complicated task. In order to provide more insight into the problem, we will shortly describe how this task was carried out up till now. In a first step, the responsible scheduler collects the required data. Coverage constraints and formation requirements are provided by the head of the department. Non-availability constraints are collected in a hierarchical way. A list circulates in which the trainees successively indicate during which weeks they will be absent and during which weeks they would like to take vacation. To ensure that vacation periods are sufficiently spread, the number of trainees having vacation at the same time is limited. Next, using pencil and paper, the scheduler tries to find a schedule that satisfies as many constraints as possible. She mainly concentrates on the satisfaction of the coverage constraints. At certain moment, typically when 75% of the schedule is completed, she fails to satisfy the next coverage constraints without violating one or more of the formation, non-availability or setup constraints. At that moment, she tries to solve the schedule conflict by making a number of assignments undone or performing a number of switches. If she fails to solve the conflict in a limited number of tries, she accepts the violation of one (or more) constraints and continues construction. Upon completion, the schedule is communicated to all the people involved (trainees and surgeons). Since this task essentially involves the solution of a complex combinatorial puzzle and PC's are typically more suitable to solve such problems than humans, we believe that a well-thought-out algorithm could save construction time as well as generate qualitatively better schedules. The nature of the problem makes it suitable to be solved by a decomposition approach that decomposes on the trainees as well as an approach that decomposes on the activities. But first, we state the problem as an integer linear program. Consider the following binary decision variables:

$$x_{ijk} = \begin{cases} 1, & \text{if during period } i \text{ trainee } j \text{ is scheduled to perform activity } k; \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ijk} = \begin{cases} 1, & \text{if trainee } j \text{ starts activity } k \text{ during period } i; \\ 0, & \text{otherwise.} \end{cases}$$

Let p_{ij} denote the penalty cost charged for assigning trainee j to period i. Let l_{jk} and u_{jk} be the respective minimum and maximum number of periods assistant j has to perform activity k. It is assumed that $u_{jk} > 0$ implies $l_{jk} > 0$ although this assumption is not necessary for the algorithms developed hereafter. 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 $l_{jk} > 0$) and A^j represent the set of all activities that have to be performed by trainee j, i.e., all activities k for which $l_{jk} > 0$). The integer linear programming model (ILP) is as follows:

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{k=1}^{p} \sum_{j \in S^k} p_{ij} x_{ijk} \tag{1}$$

subject to:

$$\sum_{k \in A^j} x_{ijk} \leq 1 \quad \forall i = 1, \ldots, n \text{ and } \forall j = 1, \ldots, m, \tag{2}$$

$$\sum_{j \in S^k} x_{ijk} = 1 \quad \forall i = 1, \ldots, n \text{ and } \forall k = 1, \ldots, p, \tag{3}$$

$$\sum_{i=1}^{n} x_{ijk} \geq l_{jk} \quad \forall k = 1, \ldots, p \text{ and } \forall j \in S^k, \tag{4}$$

$$\sum_{i=1}^{n} x_{ijk} \leq u_{jk} \quad \forall k = 1, \ldots, p \text{ and } \forall j \in S^k, \tag{5}$$

$$y_{1jk} = x_{1jk} \quad \forall k = 1, \ldots, p \text{ and } \forall j \in S^k, \tag{6}$$

$$y_{ijk} \geq x_{ijk} - x_{(i-1)jk} \quad \forall i = 2, \ldots, n \text{ and } \forall k = 1, \ldots, p \text{ and } \forall j \in S^k, \tag{7}$$

$$\sum_{i=1}^{n} y_{ijk} \leq 1 \quad \forall k = 1, \ldots, p \text{ and } \forall j \in S^k, \tag{8}$$

$$y_{ijk}, x_{ijk} \in \{0, 1\} \quad \forall i = 1, \ldots, n \text{ and } \forall k = 1, \ldots, p \text{ and } \forall j \in S^k. \tag{9}$$

The objective function (1) minimizes the total schedule cost. Constraint set (2) ensures that each trainee can perform at most one activity at each period. Constraint set (3) guarantees that every activity is performed by exactly one trainee during each time period. The fact that each trainee performs each activity between a minimum and a maximum number of periods is reflected in constraint sets (4) and (5). Constraint sets (6), (7) and (8) ensure that each trainee starts each activity only once. Finally, (9) defines x and y as binary variables.

Table 1 displays a simple example of this problem. This example involves four trainees, three activities and ten periods. All trainees are assumed to have the same level of experience and each trainee has to perform each activity between a minimum of two and a maximum of three consecutive periods. In Table 1, the columns correspond to the trainees and the rows represent the periods. The numbers indicate the non-availability costs for each trainee.

Table 1 A problem instance

Period	Non-availability costs			
	Trainee 1	Trainee 2	Trainee 3	Trainee 4
1				
2	7			
3	3		5	
4				
5				
6				9
7			5	
8				
9		6		
10		4		1

Table 2 A solution for the problem instance

Period	Trainee schedule			
	Trainee 1	Trainee 2	Trainee 3	Trainee 4
1		act 1	act 3	act 2
2		act 1	act 3	act 2
3	**act 3**	act 1		act 2
4	act 3	act 2	act 1	
5	act 3	act 2	act 1	
6	act 2	act 3	act 1	
7	act 2	act 3		act 1
8	act 2	act 3		act 1
9	act 1		act 2	act 3
10	act 1		act 2	**act 3**

Observe that each trainee has divided in total ten points over the ten periods. In real-life, often these points are concentrated in a small number of periods.

A possible solution for this problem is represented in Table 2. In this solution, trainees 1 and 4 are both scheduled during a period in which they actually prefer not to be scheduled, respectively period 3 and period 10 (indicated in bold), making up for a total cost of $3 + 1 = 4$. 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. The first results in an unsatisfied trainee, the latter causes a discontinuation in the educational program, which at its turn leads to a decrease in the quality of care. In Table 2, there are in total 10 periods in which no activity is scheduled (2 periods for trainee 1, 2 periods for trainee 2, 3 periods for trainee 3 and 3 periods for trainee 4). However, during 6 of these 10 periods, trainees have requested weeks-off (see Table 1). Hence, only 4 periods remain in which neither an activity is scheduled nor a week-off has been requested. During these periods, the trainees will perform activities for which no specific skills are required and for which the experience level as well as the minimal formation requirements are less important. An example of such an activity is consultation. The schedule is thus constructed using a two-phase approach. In the first phase, the difficult activities, i.e., the activities that certainly have to be performed, are scheduled. In the second phase, the partially constructed schedule is completed with the easy activities. This last task is straightforward and can easily be done manually. Therefore, we will only concentrate on the scheduling of the difficult activities in this paper.

2 Decomposition of the problem

In the two excellent bibliographic surveys on medical staff rostering problems that recently appeared (Cheang et al. 2003; Burke et al. 2004), a distinction is made between constraint- or period-based models on the one hand and column- or shift-oriented models on the other hand. In this section the constraint-based model of ILP (1–9) is converted to a column-oriented formulation, which entails a decomposition of the problem. Decomposition involves the division of the problem into several subproblems. Another way of seeing this is to introduce new decision variables, each one representing a subset of the old decision variables, that implicitly satisfy a number of constraints. Solving a subproblem is then analogous to finding a new decision variable or column for the ILP. The constraints that remain in the

ILP can be seen as the *linking* constraints. The advantage of such an approach is that the LP bound of the new formulation is usually much stronger than that of the original formulation and consequently the branch-and-bound tree remains smaller. The drawback is however that the new formulation can have far more variables than can be reasonably attacked directly. Fortunately, column generation can help to overcome this difficulty. Column generation is based on the observation that it is not necessary to enumerate all possible columns in order 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. Hence, column generation alternates LP solving with subproblem solving. The LP containing only a subset of all columns is often called the restricted linear program (RLP) or master problem. Each time the RLP is solved it provides the information (via the dual prices of the constraints) needed to determine whether the LP is solved to optimality and, if not, to provide information in order to determine which columns could further improve the LP objective value. The first use of column generation to solve the master linear program arising from an integer programming problem is probably the work on the cutting stock problem by Gilmore and Gomory (1961). A branch-and-bound algorithm in which the lower bound is calculated by LP relaxation and the LP is solved through column generation, is called a branch-and-price algorithm (Barnhart et al. 1998).

2.1 Decomposition on the activities

Observe that constraint (2) is the only constraint in ILP (1–9) that links the different activities. All other constraints apply on the individual activity level and hence can be satisfied without taking the other activities into account. As a matter of fact, without constraint (2), ILP (1–9) could be divided into p subproblems, that could be solved separately. Each subproblem corresponds to the scheduling of a particular activity so that (3–9) is satisfied. Such an activity schedule is referred to as an activity pattern. An activity pattern includes the scheduling of all trainees having to perform the activity. An example of such an activity pattern for activity 1 can be found in Table 3.

The task is now to find for each activity a pattern so that constraint (2) is not violated and objective (1) is minimized. Hence, to decompose (1–9) on the activities, we introduce decision variables that represent activity patterns. As we will use column generation, the activity patterns are referred to as columns. Let binary decision variable z_{kt} be defined as

Table 3 A column for activity 1

Period	Activity schedule			
	Trainee 1	Trainee 2	Trainee 3	Trainee 4
1		act 1		
2		act 1		
3		act 1		
4			act 1	
5			act 1	
6			act 1	
7				act 1
8				act 1
9	act 1			
10	act 1			

follows:

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

Let a_{ijkt} equal 1 if trainee j is scheduled during period i in column t for activity k. Let c_{kt} be the total cost of column t for activity k (i.e., $c_{kt} = \sum_{i=1}^{n} \sum_{j \in S^k} a_{ijkt} p_{ij}$) and NC_k the total number of different columns for activity k. The model can then be formulated as follows:

$$\text{Minimize} \quad \sum_{k=1}^{p} \sum_{t=1}^{NC_k} c_{kt} z_{kt} \tag{10}$$

subject to:

$$\sum_{k \in A^j} \sum_{t=1}^{NC_k} a_{ijkt} z_{kt} \leq 1 \quad \forall i = 1, \ldots, n \text{ and } \forall j = 1, \ldots, m, \tag{11}$$

$$\sum_{t=1}^{NC_k} z_{kt} = 1 \quad \forall k = 1, \ldots, p, \tag{12}$$

$$z_{kt} \in \{0, 1\} \quad \forall k = 1, \ldots, p, \text{ and } \forall t = 1, \ldots, NC_k. \tag{13}$$

The objective function (10) is again the minimization of costs, but now expressed in terms of the new z_{kt} variables. Constraint set (11) states that each trainee can perform no more than one activity at the same time. Constraint set (12) ensures that exactly one column has to be selected for each activity. Finally, (13) define z_{kt} as a binary variable. The master problem (10–13) is in fact a 0–1 capacitated multicommodity flow problem in which each activity corresponds to a commodity and all arc capacities and commodity requirements are equal to 1. Tests revealed that the LP relaxation of this formulation provides a much stronger lower bound than that from the original formulation of (1–9) (see Sect. 5.1). The reason is that in the new formulation (10–12) the optimization is performed over the convex hull of feasible points of the subproblems, and not just over the relaxed feasible region of (1–8). It can easily be shown that each feasible solution of the new formulation (10–12) is also feasible in the original formulation (1–8). To see this, observe that each z_{kt} solution can be reformed to an x_{ijk} solution, which is feasible to (1–8). Therefore, simply set each x_{ijk} equal to $\sum_{t}^{NC_k} a_{ijkt} z_{kt}$. The reverse is not true when the convex hull of the subproblem constraints is not integral. The feasible solution space of (1–8) is much larger than that of (10–12). Although a smaller feasible region is no guarantee to obtain a better bound, it is quite likely that the optimal solution of (1–8) is not feasible in the new formulation (10–12).

The drawback of the new formulation is that it can have far more variables than can be reasonably attacked directly. 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 a negative reduced cost can be found. Let λ_{ij} represent the dual prices of restrictions (11) and let γ_k represent the dual prices of restrictions (12). The reduced cost of a new column t for activity k is given by:

$$-\gamma_k + c_{kt} + \sum_{i=1}^{n} \sum_{j \in S^k} -\lambda_{ij} a_{ijkt} = -\gamma_k + \sum_{i=1}^{n} \sum_{j \in S^k} (p_{ij} - \lambda_{ij}) a_{ijkt}. \tag{14}$$

In this expression $-\gamma_k$ 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 (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 $-\lambda_{ij}$. Consequently, the LP is solved to optimality when no more columns can be found with a negative reduced cost. The search is started by solving the master problem, stated in (10–12) for a limited number of columns. The master returns an objective value (which is an upper bound for the LP solution) and dual prices λ_{ij} and γ_k. While λ_{ij} serves as a direct input for the objective function of the pricing problem (see Sect. 3), 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. The objective function of each pricing problem is the minimization of the reduced cost (14) of the new column. Columns with a negative reduced cost are added to the master problem and the master is re-optimized. This process continues until no columns price out any more.

2.2 Decomposition on the staff members

In the literature, staff scheduling problems are usually decomposed on the staff members, in this case the trainees (see, e.g., Caprara et al. 2003; Mason and Smith 1998; Jaumard et al. 1998; Bard and Purnomo 2005; Mehrotra et al. 2000). An example of such a column for trainee 2 can be found in Table 4.

To decompose (1–9) on the trainees, we introduce decision variables that represent individual trainee schedules. Let binary decision variable z_{jt} be defined as follows:

$$z_{jt} = \begin{cases} 1, & \text{if column } t \text{ was chosen for trainee } j; \\ 0, & \text{otherwise.} \end{cases}$$

Let a_{ijkt} equal 1 if trainee j is scheduled during period i in column t to perform activity k. Let c_{jt} be the total cost of column t for trainee j (i.e., $c_{jt} = \sum_{i=1}^{n} \sum_{k \in A^j} a_{ijkt} p_{ij}$) and NC_j the total number of different columns for trainee j. The model can then be formulated as

Table 4 A column for trainee 2

Period	Activity schedule			
	Trainee 1	Trainee 2	Trainee 3	Trainee 4
1		act 1		
2		act 1		
3		act 1		
4				
5		act 3		
6		act 3		
7		act 3		
8				
9		act 2		
10		act 2		

follows:

$$\text{Minimize} \quad \sum_{j=1}^{m} \sum_{t=1}^{NC_j} c_{jt} z_{jt} \tag{15}$$

subject to:

$$\sum_{j \in S^k} \sum_{t=1}^{NC_j} a_{ijkt} z_{jt} = 1 \quad \forall i = 1, \ldots, n \text{ and } \forall k = 1, \ldots, p, \tag{16}$$

$$\sum_{t=1}^{NC_j} z_{jt} = 1 \quad \forall j = 1, \ldots, m, \tag{17}$$

$$z_{jt} \in \{0, 1\} \quad \forall j = 1, \ldots, m \text{ and } \forall t = 1, \ldots, NC_j. \tag{18}$$

The objective function (15) is again the minimization of costs, but now expressed as the sum of the trainee schedules. Constraint set (16) states that each activity has to be performed by exactly one trainee at each time period. Constraint set (17) ensures that exactly one column has to be selected for each trainee. The master problem (15–18) is now a 0-1 set partitioning problem. Let π_{ik} represent the dual prices of restrictions (16) and let μ_j represent the dual prices of restrictions (17). The reduced cost of a new column t for trainee j is now given by:

$$-\mu_j + c_{jt} + \sum_{i=1}^{n} \sum_{k \in A^j} -\pi_{ik} a_{ijkt} = -\mu_j + \sum_{i=1}^{n} \sum_{k \in A^j} (p_{ij} - \pi_{ik}) a_{ijkt}. \tag{19}$$

3 The pricing problems

First, we state the pricing problem when decomposing on the activities. The pricing problem for activity k can be stated as follows. Let x_{ij} equal 1 if trainee j is scheduled to perform activity k during period i and y_{ij} be 1 if trainee j starts activity k at period i.

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j \in S^k} (p_{ij} - \lambda_{ij}) x_{ij} \tag{20}$$

subject to:

$$\sum_{j \in S^k} x_{ij} = 1 \quad \forall i = 1, \ldots, n, \tag{21}$$

$$\sum_{i=1}^{n} x_{ij} \geq l_{jk} \quad \forall j \in S^k, \tag{22}$$

$$\sum_{i=1}^{n} x_{ij} \leq u_{jk} \quad \forall j \in S^k, \tag{23}$$

$$y_{1j} = x_{1j} \quad \forall j \in S^k, \tag{24}$$

$$y_{ij} \geq x_{ij} - x_{(i-1)j} \quad \forall i = 2, \ldots, n \text{ and } \forall j \in S^k, \tag{25}$$

$$\sum_{i=1}^{n} y_{ij} \leq 1 \quad \forall j \in S^k, \tag{26}$$

$$x_{ij}, y_{ij} \in \{0, 1\} \quad \forall i = 1, \ldots, n \text{ and } \forall j \in S^k. \tag{27}$$

Objective (20) simply entails the minimization of the (variable part of) the reduced cost (14). Constraints (21–27) are just a repetition of the constraints (3–9) for activity k.

When decomposing on the trainees the pricing problem only differs with respect to the first constraint. Since constraint (2) applies at the individual trainee level, this constraint is included in the pricing problem. Since the linking constraint is now constraint (3), this constraint is left out of the pricing problem. The pricing problem for trainee j can be stated as follows. Let x_{ik} equal 1 if trainee j is scheduled to perform activity k during period i and y_{ik} be 1 if trainee j starts activity k at period i.

$$\text{Minimize} \quad \sum_{i=1}^{n} \sum_{k \in A^j} (p_{ij} - \pi_{ik}) x_{ik} \tag{28}$$

subject to:

$$\sum_{k \in A^j} x_{ik} \leq 1 \quad \forall i = 1, \ldots, n, \tag{29}$$

$$\sum_{i=1}^{n} x_{ik} \geq l_{jk} \quad \forall k \in A^j, \tag{30}$$

$$\sum_{i=1}^{n} x_{ik} \leq u_{jk} \quad \forall k \in A^j, \tag{31}$$

$$y_{1k} = x_{1k} \quad \forall k \in A^j, \tag{32}$$

$$y_{ik} \geq x_{ik} - x_{(i-1)k} \quad \forall i = 2, \ldots, n \text{ and } \forall k \in A^j, \tag{33}$$

$$\sum_{i=1}^{n} y_{ik} \leq 1 \quad \forall k \in A^j, \tag{34}$$

$$x_{ik}, y_{ik} \in \{0, 1\} \quad \forall i = 1, \ldots, n \text{ and } \forall k \in A^j. \tag{35}$$

Objective (28) simply entails the minimization of the (variable part of) the reduced cost (19). Constraints (29–35) are just a repetition of the constraint (2) and constraints (4–9) for activity k.

The pricing problems in both decomposition approaches can be solved in a similar way using a forward dynamic programming approach. Consider first the case in which we decompose on the activities. Suppose we are searching a new column for activity k. This problem can be visualized by a matrix. The columns in this matrix represent the trainees $j \in S^k$ and the rows represent the time horizon. Each cell of the matrix has a cost g_{ij} which

Table 5 Pricing problem[a] for activity k: optimal solution in bold

Period i	g_{ij} = non-availability cost p_{ij} - dual price λ_{ij}			
	Trainee $j=1$	Trainee $j=2$	Trainee $j=3$	Trainee $j=4$
1	**0**	1	1	2
2	**0**	1	1	1
3	4	2	**1**	4
4	2	1	**0**	0
5	0	**0**	4	0
6	1	5	3	**1**

[a]For ease of explanation all cost values are integer. Note however that during column generation these cost values are usually fractional due to the dual prices

equals the corresponding non-availability cost p_{ij} minus the corresponding dual price λ_{ij}. This matrix has to be traversed from top to bottom in the cheapest way possible, while visiting each column exactly once between a minimum and a maximum number of rows. Table 5 represents an instance of such a pricing problem for an activity performed by four trainees that all have to be scheduled between one and two periods in a time horizon of six periods. The optimal solution has a cost of 2 and is indicated in bold. It first schedules trainee 1 for two periods, followed by trainee 3 for two periods, then trainee 2 for one period and finally trainee 4 for one period.

The pricing problem can be solved with a 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. If T denotes a set of trainees, the subproblem can be described as finding the cheapest way to reach a period i with all trainees in T scheduled. Let $cost(i, T)$ represent this cost. Consider the pricing problem for activity k. Recall that l_{jk} and u_{jk} are the respective minimum and maximum number of periods assistant j has to perform activity k. The recursive formulation of $cost(i, T)$ is as follows:

$$cost(i, T) = \mathrm{MIN}_{j \in T} \left\{ \mathrm{MIN}_{l_{jk} \le d \le u_{jk}} \left\{ cost(i-d, T \backslash \{j\}) + \sum_{b=1}^{d} g_{(i-d+b)j} \right\} \right\}. \tag{36}$$

To solve the pricing problem, we must calculate $cost(n, S^k)$ using (36) and make sure we know which schedule it represents. Algorithm 1 (that uses a recursive subprocedure outlined in Algorithm 2) can be used to achieve this purpose. In this pseudo-code, $sch\_cost(i_1, i_2, j)$ equals the cost to schedule trainee j from period i_1 until i_2 and *current* represents the cost of the partial schedule (or path) during construction.

Algorithm 1 FIND-NEW-ACTIVITY-COLUMN(k)

```
for (i = 1 to n) do
  for (all possible subsets T of trainee set S^k) do
    cost(i, T) ← 999999999;
  end for
end for
RECURSION(n, S^k);
```

Algorithm 2 RECURSION(i,T)

if ($i = 0$) **AND** ($T = \emptyset$) **then**
  return 0; {beginning of time horizon reached}
else if ($\sum_{j \in T} l_{jk} > i$) **OR** ($\sum_{j \in T} u_{jk} < i$) **then**
  $cost(i, T) \leftarrow +\infty$;{state cannot lead to a feasible solution}
  return $cost(i, T)$;
else if ($cost(i, T) \neq 999999999$) **then**
  return $cost(i, T)$; {state already visited, can be pruned}
else
  $min\_current \leftarrow +\infty$;
  for (all trainees $j \in T$) **do**
    for ($d = l_{jk}$ to u_{jk}) **do**
      $current \leftarrow sch\_cost(i - d + 1, i, j) +$ RECURSION$(i - d, T \backslash \{j\})$;
      if ($current < min\_current$) **then**
        $min\_current \leftarrow current$;
      end if
    end for
  end for
  $cost(i, T) \leftarrow min\_current$;
  return $cost(i, T)$;
end if

Recursive algorithm 2 starts with a number of checks in order to be able to stop the current recursion. The first one checks if the beginning of the time horizon is reached. If this is not the case, the second step checks whether it is still possible to find a feasible schedule starting from the current state. This is not the case if the sum of all minimum requirements of the not yet scheduled trainees exceeds the remaining time horizon or, the reverse, if the sum of all maximum requirements does not suffice to reach the beginning of the time horizon. In the third step it is checked if the current state has already been visited. If all checks are passed the partial path is extended with the assignment of a next trainee. Therefore, all possible assignments of all not yet scheduled trainees are investigated. Once all the calculations are done, the best new column can easily be constructed backwards. We use binary encoding to represent the (sub)sets T. For more details on the solution of the pricing problem we refer to Beliën and Demeulemeester (2006).

Suppose m trainees have to perform a particular activity. For each time instance we need to store at most all possible subsets of m trainees. Hence, the space complexity is given by $O(n \cdot 2^m)$. In the recursive algorithm each state (i, S) leads to at most $R * m$ other states. The complexity of this recursion is thus $O(n \cdot 2^m \cdot R \cdot m)$.

If we are decomposing on the trainees, a similar approach can be used. This time the columns of the cost matrix represent the activities. Since it is possible that a trainee performs no activity during certain time periods, an extra column has to be added that represents 'performing no activity' and can be visited more than once. Obviously, all rows of this column have a cost equal to 0. Table 6 visualizes the pricing problem for a particular trainee j that has to perform three activities all between a minimum of one and a maximum of two periods. Each cell of the matrix has a cost h_{ik} which is the difference between the

Table 6 Pricing problem[a] for trainee j: optimal solution in bold

Period i	h_{ik} = non-availability cost p_{ij} - dual price π_{ik}			
	Activity $k=1$	Activity $k=2$	Activity $k=3$	No activity
1	**−2**	1	1	0
2	**−2**	1	1	0
3	4	2	1	**0**
4	2	1	**−1**	0
5	0	**−3**	4	0
6	1	5	3	**0**

[a]For ease of explanation all cost values are integer. Note however that during column generation these cost values are usually fractional due to the dual prices

corresponding non-availability cost p_{ij} and the corresponding dual price π_{ik}. Note that cost values can be negative due to possible positive values for the dual prices π_{ik}. We also applied dynamic programming to solve this pricing problem. The recursive algorithm is very similar as the one outlined above.

Suppose a trainee has to perform p activities. For each time instance we need to store at most all possible subsets of p activities. Hence, the space complexity is given by $O(n \cdot 2^p)$. In the recursive algorithm each state (i, S) leads to at most $R * p * n$ other states. The added factor n, in contrast with the previous subproblem, is due to the fact that the activities do not necessarily immediately follow each other, but instead some periods may be left blanc. The complexity of this recursion is thus $O(n^2 \cdot 2^p \cdot R \cdot p)$.

4 Branching

If the LP relaxation of the master problem does not have an integral optimal solution, a branching scheme is needed to drive the solution into integrality. Vanderbeck (2000) describes several branching schemes that are appropriate in branch-and-price algorithms. Beliën and Demeulemeester (2006) describe three possible branching schemes for the trainee scheduling problem. Since their computational results indicated that branching on the timetable cells provides the best and most robust results, we will use this branching scheme.

Branching on the timetable cells corresponds to branching on the original x_{ijk} variables. The main advantage of this branching scheme is that it preserves the structure of the pricing problem, in the activity-based decomposition as well as in the trainee-based decomposition. Let us first consider the case in which we are decomposing on the activities. The next x_{ijk} to branch on is found by selecting the largest fractional column. Suppose this is a column for activity k. Then, we search the first timetable cell (i, j) for which there exists another fractional column that schedules a different activity at timetable cell (i, j). In the left branch x_{ijk} is set to 1, while in the right branch it is set to 0. If x_{ijk} is set to 1, $g_{ij'}$ is set to $+\infty$ for all $j' \neq j$ in the pricing problem of activity k. For the pricing problems for activities $k' \neq k$ only g_{ij} is set to $+\infty$. Else if x_{ijk} is set to 0, g_{ij} is set to $+\infty$ in the pricing problem of activity k. A second advantage is the fact that this branching scheme yields a more balanced branch-and-bound tree compared to standard column branching.

The same branching scheme can be applied if we are decomposing on the trainees. Here, the next x_{ijk} to branch on is also found by selecting the largest fractional column. Suppose

this is a column for trainee j. Then, we search for this column the first time period i for which there exists a second fractional column that schedules a different activity than the first column during time period i. Suppose that the first fractional column schedules activity k during the conflicting time period i. Again, x_{ijk} is set to 1 in the left branch and to 0 in the right branch. The timetable costs in the pricing problems are modified as follows. If x_{ijk} is set to 1, $h_{ik'}$ is set to $+\infty$ for all activities $k' \neq k$ in the pricing problem of trainee j. Furthermore, h_{ik} is set to $+\infty$ in the pricing problems of all trainees $j' \neq j$. Else if x_{ijk} is set to 0, h_{ik} is set to $+\infty$ in the pricing problem of trainee j.

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. Therefore, the column generation proceeds after each new branching.

Since we have a method to generate columns and a branching scheme to cut away fractional solutions, our branch-and-price algorithm is complete. The initial set of columns is composed by a heuristic that repeatedly adds columns until a feasible solution is found. The branch-and-price algorithm is extended with a number of speed-up techniques the most important being a lower bound calculation based on the Lagrange dual. For more details on the initial heuristic and on the speed-up techniques we refer to Beliën and Demeulemeester (2006). After each master optimization exactly one pricing problem is solved for each decomposed item. Hence, the dual prices are updated after the addition of at most p or m columns depending on the decomposition technique.

5 Computational results

5.1 Real-life data sets

The first computational experiment entails the application of the original ILP formulation (1–9) and both decomposition approaches on two real-life problems. In both problems 8 trainees are scheduled for a time horizon of 35 periods. All trainees have to perform 6 activities in the first problem and 7 activities in the second problem. The differences between the maximum and minimum number of periods trainee have to perform an activity vary between 1 and 4 for both problems. The algorithms are coded in Visual C++.NET. The CPLEX 8.1 LP solver has been used to solve the master problems. For the original ILP formulation of (1–9) we used the CPLEX 8.1 MIP solver with standard settings. The experiments have been performed on a 2.4 GHz Pentium 4 PC with the Windows XP operating system. Table 7 displays the computational results.

The best results are obtained when decomposition takes place on the activities. As can be seen in Table 7, using ILP (1–9) the number of nodes left was still growing after 1800

Table 7 Results on two real-life problems

	Problem 1			Problem 2		
	ILP (1–9)	Decomp. on activities	Decomp. on trainees	ILP (1–9)	Decomp. on activities	Decomp. on trainees
Best solution found	–	16	20	–	10	10
LP relaxation	5.00	15.09	15.00	6.00	9.37	9.30
Explored nodes	453	100	71	390	24	42
Comp. time (s)	1800	78.73	1800	1800	20.89	1036.06

seconds of computation time. Whereas through ILP (1–9) even though no feasible solution could be found within this time limit, the branch-and-price algorithm in which decomposition takes place on the activities could solve the first problem in 78.73 seconds and the second problem in 20.89 seconds. When one is decomposing on the trainees only the second problem could be solved to optimality within the given time limit. These differences are mainly caused by the huge difference between the LP relaxations of ILP (1–9) on the one hand and the LP relaxations of both decomposition approaches on the other hand: 5.00 versus 15.09 and 15.00 for the first problem and 6.00 versus 9.37 and 9.30 for the second problem. Comparing these bounds with the optimal solutions, 16 and 10, we conclude that the LP relaxation of ILP (1–9) is dramatically weak, whereas the LP relaxations of the decomposition approaches, particularly when decomposing on the activities, are extremely strong.

Both real-life problems have indicated that a decomposition approach outperforms standard integer programming for solving this kind of scheduling problems. In order to be able to state more founded conclusions with respect to the differences between both decomposition approaches, we had to run them on a larger number of problems. Therefore, we have composed a test set.

5.2 Test set

The computational performance of both decomposition approaches was compared using the problem set introduced by Beliën and Demeulemeester (2006). This test set has been composed by a guided random procedure in which the six factors that could have an influence on the problem complexity were varied: 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 periods (further referred to as the range) and finally the magnitude of the costs. In Table 8, a number of settings for these six factors are displayed.

This table requires some further information. First of all, the number of activities and the number of trainees having to perform an activity are expressed as a percentage of the number of trainees. For instance, a test problem with 12 trainees and 75% activities contains 8 activities. Observe that the number of activities never surpasses the number of trainees, as otherwise not all activities can be performed. In other words, the schedule is not totally occupied after solving the scheduling problem. Recall that the remaining part of the schedule has to be filled up with activities for which the consecutiveness is less important. The ratio number of activities over number of trainees is the total schedule occupation percentage. The factor setting random(x) indicates that the data is generated randomly, but in such a way that

Table 8 Design of the experiment

Factor setting	Nr. of periods	Nr. of trainees	Nr. of activities	Nr. of trainees per activity	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)	

the average is equal to x. A magnitude of the costs equal to 1 indicates that the non-available time periods, that are generated randomly for each trainee, all 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 trivial and non-feasible problems as much as possible, the trainee occupations were kept more or less at the same level. The total number of periods containing positive costs equals 3, 4 or 5 per trainee for instances with 18, 35 and 52 periods. For each factor setting three problem instances have been generated. The total number of test problems was bounded by subsequently fixing the first three factors and the next two factors at an intermediate level. This resulted in $3 * (4 * 6 * 2) + 3 * (3 * 4 * 3 * 2) = 360$ problem instances.

5.3 Discussion of results

In this section, we summarize the most important findings from our computational experiments. Detailed figures of the results for the activity-based decomposition technique can also be found in Beliën and Demeulemeester (2006). Here we compare these results with those for the trainee-based decomposition approach. Table 9 and following contain subsets of these results that are representative for all obtained results. For each problem the LP relaxation, best found solution, computation time, number of generated columns and number of nodes is given for both decomposition techniques. The problem name in the second column refers to the different factor settings for generating the problem (see Table 8). The first number stands for the first factor, the second for the second factor etc. The last number (after '_') indicates the replication number. If the algorithm fails to find an optimal solution within 1800 seconds, the computation time in Table 9 and following is indicated with '1800'. The reason why the computation time was limited to 1800 seconds is that the algorithms could not solve the problem to optimality within a reasonable time limit for the larger problem instances in our test set. Obviously, without this time limit, it would be much easier to make a fair comparison between both decomposition approaches. We could, for instance, look to the total number of nodes or to the total computation time required by each decomposition technique to obtain the optimum for all the instances. Unfortunately, some problems may require days (or even weeks) of computation time, particularly for the decomposition on the trainees. Therefore, we had no option but to set a time limit within which a reasonable amount of the problem instances could be solved by at least one of the decomposition approaches. Fortunately, a time limit of 1800 seconds leads already to important performance differences and hence provides us with a suitable database for comparing both decomposition approaches.

Table 9 gives a first indication of how the two decomposition approaches compare to each other. This table contains the summarized results for 36 of the easy instances in our test set. These are problems with 35 periods, 8 trainees and 5 activities in which each activity is performed by 4 trainees. For these dimensions both decompositions manage to find the optimal solution for all problem instances within the time limit. However, the computation times tend to be higher when decomposing on the trainees. Note that one of the explanations of this performance difference can be found in the difference between the LP relaxations. The LP relaxations of the decomposition on the activities approach tend to be higher than those of the decomposition on the trainees approach.

Table 10 contains the same information, but now for 36 problem instances in which the number of trainees having to perform each activity increases from 4 to 6. If we compare this table with the previous one, it becomes clear that the number of trainees having to perform

Table 9 Results for problems with 35 periods, 8 trainees and 6 activities in which each activity is performed by 4 trainees

Nr.	Problem	Decomposition on activities					Decomposition on trainees				
		LP relax.	Sol.	Time	Columns	Nodes	LP relax.	Sol.	Time	Columns	Nodes
1	222111_1	13	13	0.20	63	1	13	13	6.84	4095	10
2	222111_2	16	16	0.28	80	2	16	16	1.25	960	1
3	222111_3	16	16	0.25	69	4	16	16	5.23	3126	7
4	222112_1	36	36	0.25	73	2	36	36	3.17	2283	4
5	222112_2	50	50	0.28	84	2	50	50	2.33	1696	4
6	222112_3	45	45	0.11	39	0	45	45	0.56	328	0
...											
30	222152_3	37	37	0.33	94	0	37	37	4.05	1629	1
31	222161_1	12	12	0.67	230	2	12	12	19.73	4184	4
32	222161_2	10.5	11	1.05	358	4	10.5	11	54.33	7331	11
33	222161_3	12.5	13	0.91	338	6	12.4	13	39.44	5006	7
34	222162_1	32	32	0.95	303	1	32	32	37.73	4213	2
35	222162_2	37	37	0.47	134	0	37	37	4.05	992	0
36	222162_3	33.56	35	6.47	1393	67	33.51	35	28.56	5772	9
	Average	24.35	24.86	0.91	279.17	6.86	24.13	24.86	80.60	7837.08	13.83
		Nr. Solved to optimality: 36					Nr. Solved to optimality: 36				

Table 10 Results for problems with 35 periods, 8 trainees and 6 activities in which each activity is performed by 6 trainees

Nr.	Problem	Decomposition on activities					Decomposition on trainees				
		LP relax.	Sol.	Time	Columns	Nodes	LP relax.	Sol.	Time	Columns	Nodes
37	222211_1	13	13	1.34	479	10	13	13	333.24	13760	14
38	222211_2	14	14	0.25	78	0	14	14	5.88	800	0
39	222211_3	13	13	1.23	445	6	13	13	1248.62	37182	56
40	222212_1	33	33	1.49	580	10	33	33	223.49	8902	6
41	222212_2	31	31	1.25	458	11	31	31	196.35	12470	14
42	222212_3	26	26	1.39	518	10	26	26	340.72	15642	16
...											
67	222261_1	6	6	18.61	1635	27	6	7	1800	28618	88
68	222261_2	7	8	121.47	3804	127	6.33	9	1800	33093	101
69	222261_3	4.01	5	35.36	2033	108	4	6	1800	32430	75
70	222262_1	14.13	15	9.73	860	7	14	18	1800	30918	70
71	222262_2	20.29	21	14.25	1254	15	20	23	1800	31414	90
72	222262_3	13.98	16	344.36	7687	1198	13.5	20	1800	32367	76
	Average	13.68	14.28	121.20	5083.08	119.61	13.47	15.61	1352.02	27244.61	62.03
		Nr. Solved to optimality: 35					Nr. Solved to optimality: 13				

each activity is an important factor for the difficulty of our problem. When decomposing on the trainees, only 13 problems could be solved to optimality within 1800 seconds, compared to 35 problems when decomposing on the activities. When the algorithm failed to solve the problem to optimality, a computation time of 1800 seconds was accounted for the calculation of the average. Even with this underestimation of the computation times for the non-solved problems, there is a clear difference between the average computation times of both decomposition approaches. If we compare the average solution quality, we can conclude that the trainees decomposition, although frequently not capable of detecting the optimal solution, succeeds in finding close to optimal solutions.

When we look at the problem instances with only 18 periods instead of 36 (Table 11), we see that all 72 problems could be solved to optimality within the time limit of 1800 seconds when decomposing on the activities. When decomposing on the trainees, the optimum was not found for one problem instance.

If we compare these figures with the results for the problem instances with 52 periods (Table 12), we can conclude that also the number of periods is an important factor for the difficulty of the problem. For the largest problems often even no feasible solution could be obtained. In that case the column containing the best found solution (Sol.) reports a '–' and the average solution could not be calculated.

The complexity of the problem also grows with an increasing number of trainees, an increasing number of activities and an increasing magnitude of the range. Compare, for instance, the first six lines with the last six lines in Table 11 and Table 12. More details on the contribution of each factor to the problem difficulty falls beyond the scope of this paper, but can be found in Beliën and Demeulemeester (2006).

An overall summary of the computational results is given in Table 13. The first row indicates the number of problems that could be solved to optimality within 1800 seconds using each decomposition approach. The second row contains the number of problems for which the decomposition was faster. For the remaining problems, either the computation time was the same or none of both approaches succeeded in solving the problem within the time limit of 1800 seconds. The fourth row indicates the average solution quality for the 340 problems for which both decompositions found at least a feasible solution. For the required computation time, the number of columns and the number of nodes, a distinction is made between the results for all instances and the results for only those problem instances for which both decompositions found an optimal solution within the time limit.

These results clearly indicate that decomposition on the activities outperforms decomposition on the trainees. When decomposing on the activities, more problems could be solved to optimality, average computation times are lower, less columns are needed to prove optimality and more nodes could be evaluated. Moreover, only for two instances no feasible solution was found compared to 20 instances in the trainee-based decomposition approach.

If we only look at those instances for which both decompositions found a feasible solution, the average solution quality of the trainee-based decomposition exceeds that of the activity-based decomposition by more than 10%.

If we only look at the problems for which both decompositions found an optimal solution, the number of nodes evaluated in the activity-based decomposition still exceeds those of the trainee-based decomposition. The higher number of nodes in the activity-based decomposition is contradictory with the higher LP relaxations. It turns out that the average is misleading at this point. The last-but-one row in Table 13 indicates that the activity-based decomposition could solve more problems in less nodes than the trainee-based decomposition. Hence, there is only a small number of problems for which the number of nodes of the activity-based decomposition dramatically exceeds those of the trainee-based decomposition. As can be expected, this mainly occurs in those few problems for which the LP

Table 11 Results for problems with 18 periods

Nr.	Problem	Decomposition on activities					Decomposition on trainees				
		LP relax.	Sol.	Time	Columns	Nodes	LP relax.	Sol.	Time	Columns	Nodes
145	111231_1	1	1	0.13	7	0	1	1	0.22	216	0
146	111231_2	6	6	0.01	6	0	6	6	0.09	150	0
147	111231_3	0	0	0	3	0	0	0	0.23	542	2
148	111232_1	1	1	0.03	15	0	1	1	0.14	254	0
149	111232_2	1	2	0.05	55	1	0.8	2	0.39	718	6
150	111232_3	8	8	0	10	0	8	8	0.11	199	0
...											
169	122231_1	3	3	1.84	867	16	3	3	6.36	4510	23
170	122231_2	2	2	1.72	818	14	2	2	83.41	19134	75
171	122231_3	1	1	1.67	798	10	1	1	7.08	4830	20
172	122232_1	9	9	2.47	1191	19	9	9	9.36	5596	25
173	122232_2	8	8	1.81	911	18	8	8	5.94	3622	13
174	122232_3	7	7	1.98	1001	12	7	7	19.42	8956	29
175	123231_1	11	11	8.38	3082	50	11	11	11.63	5646	24
176	123231_2	11	11	4.38	1880	28	11	11	10.48	5362	19
177	123231_3	9	9	5.06	2126	18	9	9	38.38	13570	43
...											
211	143231_1	6	6	51.28	5006	83	6	6	233.32	35572	100
212	143231_2	8	8	51.48	4377	95	8	8	382.77	57272	143
213	143231_3	12	12	38.91	3381	89	12	12	151.41	28600	101
214	143232_1	21	21	65.3	6084	92	21	21	145.31	21854	86
215	143232_2	16	16	44.14	3968	86	16	16	130.83	20002	81
216	143232_3	23	23	55.98	4912	97	23	23	127.49	21238	87
	Average	8.10	8.13	9.60	1434.72	27.00	8.08	8.17	89.50	10686.51	34.29
		Nr. Solved to optimality: 72					Nr. Solved to optimality: 71				

Table 12 Results for problems with 52 periods

Nr.	Problem	Decomposition on activities					Decomposition on trainees				
		LP relax.	Sol.	Time	Columns	Nodes	LP relax.	Sol.	Time	Columns	Nodes
289	311231_1	9	9	0.03	9	0	9	9	0.89	374	0
290	311231_2	6	6	0.03	10	0	6	6	0.34	318	0
291	311231_3	4	4	0.03	12	0	4	4	0.5	420	0
292	311232_1	23	23	0.03	8	0	23	23	0.36	310	0
293	311232_2	12	12	0.05	13	0	12	12	0.81	660	2
294	311232_3	14	14	0.05	13	0	14	14	0.56	383	0
...											
313	322231_1	8.15	9	6.53	993	8	8	16	1800	23190	48
314	322231_2	14	14	2.81	407	0	14	18	1800	10879	12
315	322231_3	9.33	10	7.54	1551	19	9.33	13	1800	15588	40
316	322232_1	23	23	3.98	808	4	23	46	1800	8160	7
317	322232_2	20	22	449.42	17250	245	20	46	1800	10262	11
318	322232_3	22.79	25	1800	16469	1102	22	36	1800	10182	14
319	323231_1	22	22	36.31	3953	21	22	100	1800	11799	19
320	323231_2	24	24	14.06	2246	17	24	31	1800	13675	31
321	323231_3	23	23	377.08	15067	89	23	100	1800	7897	11
...											
355	343231_1	13.78	26	1800	21392	206	13.7	–	1800	7326	25
356	343231_2	11	19	1800	20591	193	11	–	1800	7660	25
357	343231_3	13.17	20	1800	15259	82	13	–	1800	7987	21
358	343232_1	26.45	66	1800	21696	242	26.4	–	1800	7108	10
359	343232_2	26.37	–	1800	13151	16	26	–	1800	6958	14
360	343232_3	31.62	53	1800	15114	31	31	–	1800	6632	15
	Average	18.19	–	504.55	6393.38	95.00	18.14	–	1186.15	10000.60	20.01
		Nr. Solved to optimality: 54					Nr. Solved to optimality: 27				

Table 13 Overall summary computational results

	Decomposition on	
	Activities	Trainees
Nr. solved to optimality	329	220
Nr. times faster	315	10
Avg. solution value[a]	13.14	14.84
Nr. times feasible solution	358	340
Avg. comp. time (s)[b]	218.32	790.90
Avg. comp. time (s)[c]	19.92	149.24
Avg. nr. columns	4543.21	15591.94
Avg. nr. columns[c]	335.38	607.14
Avg. nr. nodes	120.05	39.82
Avg. nr. nodes[c]	29.16	22.42
Nr. times nr. nodes is lower[c]	126	40
Nr. times LP relaxation is lower	12	76

[a]For only the 340 problems in which both decompositions found at least a feasible solution within 1800 seconds

[b]A computation time of 1800 seconds was accounted if the algorithm failed to find the optimal solution within the time limit

[c]For only those 219 problems in which both decompositions found an optimal solution within 1800 seconds

relaxation is lower. The last row contains the number of instances for which the LP relaxation is lower. For only 12 instances the LP relaxation of the activity-based decomposition is lower compared to 76 for the reverse case. For the other instances, both LP relaxations were equal. 8 out of the 12 instances in which the LP relaxation of the activity-based decomposition is lower have a random number of trainees per activity (setting 4 for factor 4 in Table 8) and a small range (setting 1 for factor 5 in Table 8).

Decomposition on the trainees resulted in a smaller computation time for only 10 instances. How can we explain this difference? First of all, as already mentioned, the LP relaxation of the root node (thus before branching) tends to be higher if one decomposes on the activities. Since the problem structure does not change after branching, LP relaxations may be expected to exceed those in the trainee-based decomposition approach throughout all nodes of the branch-and-bound tree. Consequently, nodes can be pruned earlier in the activity-based decomposition approach. A second reason why decomposition on the activities is faster than decomposition on the trainees lies in the difference between the master problems. The first master contains m times n 'lower than or equal to' constraints, whereas the second master contains p times n 'equal to' constraints. Note that all equality constraints are translated into two inequality constraints. Since $m * n$ is smaller than $2 * p * n$ for all our problems, the master is often solved faster for the activity-based decomposition approach. Third, also the networks in the pricing problems tend to be smaller and thus can be solved faster if one decomposes on the activities.

6 Modeling power

For the problem we have described in Sect. 1, both decomposition techniques could be applied. Would this still be the case if the problem statement slightly changes? In this section

it will be shown that decomposing on the activities can only be applied if the problem has specific characteristics. On the contrary, decomposing on the staff members is a more general approach, since it can be used in a much larger range of staff scheduling problems. This nice property of staff-based decomposition is probably the reason why decomposing on the activities has never been applied in the staff scheduling literature so far.

In order to successfully decompose a problem, the question one has to ask is which constraints will be taken care of in the subproblem or similarly, which constraints will remain in the master. The information provided by the dual prices of this last set of constraints should be easily carried over to the subproblem without complicating it too much. To make this point clear, suppose that there are precedence constraints on the order in which the staff members perform their respective activities, i.e., a trainee can only perform a certain activity after (s)he has already performed another activity. This constraint would make it considerably harder to decompose on the activities. Similarly, suppose that the *holes* in the individual trainee schedules (with holes we mean periods in which no activity is scheduled), in one way or another, contribute to the objective function (e.g., one hole of two periods is preferred to two holes of one period). Whereas this extension could be perfectly addressed in the trainee-based decomposition scheme, it produces serious problems in the activity-based decomposition scheme. As a third example, observe that in our problem the requirement constraints (4) and (5) for each trainee are automatically satisfied if one selects a schedule (column) for each activity. If it would, however, not be possible to select, for each activity, a set of trainees so that the individual requirement constraints are implicitly satisfied (and thus can be left out of the master), decomposition on the activities would not be suitable. Summarizing, decomposition on the activities is only appropriate if either no constraints (or few) apply at the individual staff member level or, alternatively, if these constraints are automatically satisfied when scheduling activity patterns.

7 Conclusions and future research

In this paper, a comparison was made between two decomposition approaches for a particular staff scheduling problem in which a number of trainees has to be scheduled. In the first part the trainee scheduling problem was introduced and stated as an integer program. Next, two decomposition approaches have been developed. First, the problem was reformulated as a zero-one multi-commodity flow problem in which we decompose on the activities. Second, the problem was reformulated as a set partitioning problem in which we decompose on the staff members. The pricing problems in both decomposition approaches could be solved in a similar way using a dynamic programming approach. A branching scheme was developed in order to cut away fractional solutions. Also the branching scheme could be applied in both decomposition approaches. An experiment was set up in which the computational efficiency of both decomposition approaches has been compared.

The computational tests revealed that decomposition on the activities clearly outperforms decomposition on the staff members. In the next section the modeling power of both decomposition techniques has been discussed. Since most staff scheduling problems have a lot of constraints that apply at the level of individual staff members, decomposition on staff members could be used in a wider range of problems. In the rare case that most constraints apply at the level of the activity schedules, decomposition on the activities is more suitable. Activity-based decomposition is also appropriate if each combination of activity schedules automatically satisfies all individual staff member constraints. This was the case for the considered trainee scheduling problem.

Concerning future research it would be interesting to identify other staff scheduling problems for which decomposition on the activities could be applied. Given the interesting computational properties, this approach could also be suitable to calculate lower bounds for a number of staff scheduling problems for which the above mentioned conditions do not hold. To that purpose a part of the individual staff member constraints is relaxed so that the resulting problem could be optimized using activity-based column generation.

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) at the Katholieke Universiteit Leuven 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. We are indebted to the three anonymous referees whose suggestions have greatly improved the presentation of this paper.

References

Bard, J. F., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operational Research*, *164*, 510–534.

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

Beliën, J., & Demeulemeester, E. (2006). Scheduling trainees at a hospital department using a branch-and-price approach. *European Journal of Operational Research*, *175*, 258–278.

Bellman, R. (1957). *Dynamic programming*. Princeton: Princeton University Press.

Billionnet, A. (1999). Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research*, *114*, 105–114.

Burke, E. K., De Causmaecker, P., Vanden Berghe, G., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *The Journal of Scheduling*, *7*, 441–499.

Caprara, A., Monaci, M., & Toth, P. (2003). Models and algorithms for a staff scheduling problem. *Mathematical Programming*, *98*, 445–476.

Cheang, B., Li, H., Lim, A., & Rodrigues, B. (2003). Nurse rostering problems—a bibliographic survey. *European Journal of Operational Research*, *151*, 447–460.

Dantzig, G. B., & Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, *8*, 101–111.

Dreyfus, S. E., & Law, A. M. (1977). *The art and theory of dynamic programming*. New York: Academic.

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting stock problem. *Operations Research*, *9*, 849–859.

Jaumard, B., Demet, F., & Vovor, T. (1998). A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, *107*, 1–18.

Mason, A. J., & Smith, M. C. (1998). A nested column generator for solving rostering problems with integer programming. In *International conference on optimisation: techniques and applications* (pp. 827–834).

Mehrotra, A., Murphy, K. E., & Trick, M. A. (2000). Optimal shift scheduling: a branch-and-price approach. *Naval Research Logistics*, *47*, 185–200.

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