The Shift Scheduling Problem using a branch-and-price approach

Mathematics bachelor thesis
Svitlana A. Titiyevska

Vrije Universiteit Amsterdam
Faculteit der Exacte Wetenschappen
De Boelelaan 1081a
1081 HV Amsterdam

Amsterdam, 25 september 2006

Supervisor: Sandjai Bhulai

ABSTRACT

In this thesis, the shift scheduling problem is discussed and its solution by different methods. A number of models and algorithms that have been reported in the literature are reviewed. Further, the modification of the branch-and-price algorithm for optimal staff scheduling with multiple breaks is discussed. The proposed method solves the set covering formulation implicitly, using column generation. Specialized branching rules suitable for the given problem and subproblem routine are proposed. Also direct comparison with the alternative methods was examined, which shows the wide applicability and computationally superiority of this method. An example application of the method is presented that illustrates how the introduced technique can be used for solving the no cyclic scheduling problem to optimality.

TABLE OF CONTENTS

1. INTRODUCTION

The optimal employee scheduling problem arises in one or another form in a variety of service delivery settings, such as telephone companies, airlines, hospitals, police departments, banks, transportation companies, fire departments, etc. This process of matching the number of working employees to the number of employees required is frequently a large determinant of service organization efficiency. The construction of these schedules is often a complicated task. In an organization where customer demand fluctuates greatly and workforce availability is equally variable, inadequate employee scheduling can lead to costly over- or understaffing. Overstaffing results in inflated payroll expenditures. Understaffing is an even greater concern, inadequate staffing leads to poor customer service, causing reduced customer conversion rates and a potential loss of revenues. Inadequate employee scheduling can also lead to lower quality of life for employees, resulting in high turnover rates and increased recruitment and training costs. It is clear, that for any service organization it is important to schedule its manpower in an efficient manner to minimize labour costs and at the same time provide the desired service level.

Employee scheduling, or rostering, is the process of constructing work timetables for its staff so that organizations can satisfy the demand for its services or goods. This process can be seen as sequence of six steps, which are:

1) determining how many staff members are needed at different times over some planning period (planning horizon). Depending on the organization, this can be based on fixed requirements or on the estimated demand, which is based on collected data for required service levels;
2) determining days of scheduling, that involve the assignment of rest and work days for different lines of work;
3) shift scheduling, which deals with the problem of selecting from a potentially huge pool of possible candidates and assigning the number of employees to each of these shifts;
4) creating lines of work, or spanning the planning horizon for each staff member. This process depends on the basic building blocks, typically shifts, duties or constraints that are used;

5) task assignment, that includes assignment of one or more tasks to different shifts, on the basis of the start time and the length of available shifts;
6) staff assignment, which involves the assignment of individual staff to the lines of work. This step sometimes can be done during construction of the work lines.

These operations provide a general procedure for employee scheduling within which different rostering models and algorithms may be placed. Often the shift scheduling is the most relevant part of this problem because the high flexibility in shifts creates too many possible variants and the rules that make such assignments feasible are quite complicated.

In shift scheduling, the scheduler works with part-time as well as full-time employees, and shift types contrast with each other in their features: duration (length); start time; the number of breaks; the placement of the breaks. The scheduler must determine the shift types (and the number of each type to employ). The union rules, company policy restrictions and other specific characteristics of the company create large differences in difficulty and size of these problems, which, in their own turn, lead to a variety of used approaches and mathematical models.

Scheduling employee shifts involves planning in time intervals. Estimates of customer demand can vary greatly across time intervals. Employees can be assigned to various shifts specified by length, start time, and the number and length of relief and meal breaks. Thus the general shift scheduling problem involves determining the number of employees to be assigned to each shift and specifying the timing of their relief and meal breaks. It is known that the flexibility provided in shift types (length, start time, and break placements) lowers the total staffing costs and the number of required employees. A set-covering approach for this problem, on which traditionally integer programming approaches have been based, was originally developed by Dantzig (1954). This approach treats every possible shift type, shift start time, and break placement as a separate shift and therefore requires a separate integer variable for it. Thus, a large number of alternative work shifts can rapidly increase the number of decision variables and make this model inefficient and even unapplicable. This difficulty has lead researchers to study other solution methods to this problem. A number of implicit integer programming models and heuristic approaches have been proposed. Since real organizations commonly exhibit a high level of scheduling flexibility, most published shift scheduling procedures are

heuristics, which use little running time but do not guarantee optimality of the solution. However, a few important integer approaches to shift scheduling offered improvements over Dantzig's model. These integer programming formulations offer flexibility in scheduling but have some disadvantages like, for example, complex structure or undesired inherent model related assumptions. The branch-and-price approach introduced by Mehrotra, Murphy, and Trick (1999) has as goal to exploit the advantages of ease of modeling that the set covering type model offers and to overcome the problems in solving it because of its large size.

The thesis is organized as follows. Section 2 introduces the set covering formulation, a number of implicit formulations, and the generalized set covering model for a shift scheduling problem. Section 3 discusses the basic algorithm for the column generation and presents the new modification of the branch-and-price approach. The pricing scheme and the various branching strategies are examined. Section 4 describes the computational experiments that have been conducted by the discussed methods. Section 5 presents an example that illustrates the new modification of the branch-and-price approach based on the general set covering formulation and considers the computational results. Finally, some conclusions in Section 6 complete the thesis.

2. MODELS OF SHIFT SCHEDULING

Since the publication of Eddie`s “Traffic delays at toll booth’s” (1954) [4] the shift scheduling problem has received a lot of attention in the literature. Both heuristic and integer programming based approaches have been proposed.

2.1. Set covering formulation

Integer programming approaches to the shift scheduling problem have traditionally been based upon the set covering formulation which was introduced by Dantzig in “A comment on Eddie`s ‘Traffic delays at toll booth’s’ ” (1954). He states the model as follows:

$$\min \sum_{j \in J} c_j x_j$$

$$\text{subject to } \sum_{j \in J} a_{tj} x_j \geq b_t, \text{ for } t \in T,$$

$$x_j \geq 0 \text{ and integer for } j \in J,$$

where J represents the set of all shifts, T represents the planning periods that the shift schedule covers, b_t is the estimated demand for period t, and c_j is the cost of assigning an employee to shift j. Further, $a_{tj} = 1$ if period t is a work period for shift j, and $a_{tj} = 0$ otherwise, and the non-negative integer x_j variable is defined as the number of employees assigned to shift j, for $j \in J$.

Basically, a solution of this system can be easily obtained by a special adaptation of the dual form of the simplex method, which does not require knowledge of the inverse of the matrix of basis variables. Unfortunately, including different shift lengths, start times, and patterns of breaks cause a dramatic increase in the number of variables in this formulation. Given different shift types, the set covering formulation requires that all possible combinations of these features be included as different shifts in J. Thus, standard integer programming tools fail to be useful. To overcome this difficulty, researchers have studied other possible formulations. This resulted in a number of models where shifts were modeled implicitly rather than explicitly.

A first such example of implicitly representing shifts was presented by Moondra in 1976. He considered full-time shifts with a lunch window allowing two break start times and part-time shifts working between four to eight hours without a meal break. The length of the part-time shifts was represented implicitly. Also the formulation assumes that half of the full-time shifts have a meal break in the first break period and the remaining half in the second break period.

Bechtold and Jacobs in 1990 [3] introduced an implicit integer programming approach to model break-placement flexibility combined with explicit shift representation in the traditional set covering formulation. A main advantage of that model is a substantial reduction in model size compared to Dantzig`s model. This was realized by implicit representation of break assignments for all shifts which was done by associating break variables with planning periods as opposed to shifts. The considered model assumed an identical duration of the single break in each shift. A particular break variable represents the total number of employees starting a break in its associated planning period. Thus, break assignments are not made until the optimal solution to the implicit model has been obtained. The largest problem that was solved by this model had 10,237 alternate shifts.

Thompson in 1995 [8] presented a doubly-implicit IP shift scheduling model which integrated the implicit shift modeling of Moondra with the implicit meal-break modeling of Bechtold and Jacobs. This model not only reduces model size compared to the set covering model, but also more compactly represents greater scheduling flexibility than the model of Bechtold and Jacobs. The three groups of decision variables that were used: starting and finishing variables, originate in Moondra’s model and the break variables originate in Bechtold and Jacobs model. However, all the sets were newly defined with the subscript representing shift types. For solving this model the branch and bound IP procedure is proposed. After the optimal solution is found, the explicit shifts are reconstructed from the information on the shift starting and finishing times, and the matching meal and relief breaks to the reconstructed shifts on a first-in-first-out basis: each shift is constructed as follows: the first available start period (given by the start variables found) is matched with the first available finish and breaks periods. The sets of these periods are given by values of the finish and the breaks variables, respectively. This model solved a large set of diverse shift scheduling problems, the largest of which had 15,885 alternate shifts.

Another interesting model for optimal shift scheduling with multiple breaks and break windows was proposed by Aykin in 1996 [1]. This approach also leads to an IP model requiring a substantially smaller number of variables than the equivalent set-covering model. The proposed model is based on the implicit representation of the break placements and on the introduction of different sets of break variables for different shift-break combinations to determine the break placements. Also the proposed formulation is not restricted to a particular type of shift scheduling problem. Shifts may involve multiple rest and mail breaks and multiple disjoint break windows. Breaks and break windows associated with different shifts may be different in number and length, and there may be break window overlaps. Like the set-covering formulation, this approach is applicable to problems involving 24-hour continuous operations as well as less than 24-hour continuous operations. The largest problem which was solved with this approach involved 32,928 shift variations.

Compared to the set covering model all these approaches are more efficient and sometimes solve problems that can not be solved by the explicit set covering formulation. On the other hand, implicit models use much more complex rules to determine feasible shifts than the set covering type model that has been proposed. These models have some inherent assumptions which make them unsuitable in some cases (for example, no disjoint break windows, or no possibility to deal with costs of a shift that depend on the starting time, duration etc). These difficulties lead Mehrotra, Murphy, and Trick [7] to develop an approach which exploits the advantages of the set covering model and overcomes the problems in solving it because of its large size by developing an implicit optimization technique. They employed a branch-and-price approach for optimally solving the set covering formulation efficiently. They also modified Dantzig's model and introduced a generalized set covering type formulation that include both upper and lower bounds on the number of employees needed and the number of employees who can be on break.

2.2. The generalized set covering model

The generalized set covering model is an extension of Dantzig's model which includes both upper and lower bounds on the number of employees needed in each time

period and bounds on the number of employees who can be on break in any time period. The formulation (GF) is the following:

$$\min \sum_{j \in J} c_j x_j$$

$$\text{subject to } p_t \le \sum_{j \in J} a_{tj} x_j \le q_t, \text{ for } t \in T,$$

$$t_i \le \sum_{j \in J} b_{tj} x_j \le s_t, \text{ for } t \in T,$$

$$x_j \ge 0 \text{ and integer for } j \in J.$$

Here J represents the set of all shifts, T represents the planning periods that the shift schedule covers, p_t and q_t are the minimum and maximum number of employees who can be working during period t, r_t and s_t are the minimum and maximum number of employees who can use period $t \in T$ as a break period, c_j is the cost of assigning an employee to shift j. Moreover, $a_{tj} = 1$ if period t is a work period for shift j and $a_{tj} = 0$ otherwise, b_{tj} is equal to 1 if period t is a break period for shift j and zero otherwise, and x_j is an integer variable defined as the number of employees assigned to shift j, for $j \in J$.

Obviously, for a problem of any reasonable size there are too many variables in order to solve this IP directly. Fortunately, column generation can help to overcome this difficulty. The LP is solved by using only a subset of the columns and more columns are generated as needed.

2.3. Applicability of model

The introduced general set covering formulation assumes a high degree of flexibility in shifts start times, shift length, and numbers and placement of breaks. In practice, usually the company, union and legal requirements such as work law, for example, give some restrictions on the shifts lengths and break assignments. The start time and the length of shifts determine the number and type of breaks (rest, lunch, idle period for a split-shift) that employees receive. Because of the explicit representation of shifts it is not possible within the general set covering model to provide these kind of restrictions.

Thus, for shift scheduling problems that permit only a particular type of shifts, a different formulation could be more efficient.

3. SOLUTION METHODS

Mehrotra, Murphy, and Trick used a branch-and-price approach for solving staff scheduling problems. Now, branch-and-price is an increasingly important technique for treating large integer programming models. This method is based on column generation methods, implicit pricing of non-basic variables to generate new columns and is used to prove LP optimality at a node of the branch-and-bound tree. Since introduction by Barnhart et al. in 1998, several specialized branch-and-price algorithms have appeared in the literature. 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 (Mason and Smith (1998), Mehrotra et al. (1999), Caprara et al. (2003)). Alternatively, 0-1 multicommodity flow formulations have been proposed (Cappanera and Gallo (2001), Beliën and Demeulemeester (2004)).

Column generation, which is the basis of the introduced branch-and-price approach, is a powerful tool in solving LP problems with a huge number of variables. Dantzig and Wolfe (1960) proposed the column generation technique as a scheme for solving large linear programming problems that have many variables. The first and best known application of the approach to the cutting stock problem was formulated by Gilmore and Gomory in 1961. Since then several different types of this approach were proposed in the literature. Generic algorithms of this method were presented by Barhart et al. (1998) and Vanderbeck and Wolsey (1996).

3.1. The branch-and-price approach

The branch-and-price approach is an important technique for solving LP problems that have too many columns so that efficient handling with traditional methods is not possible. This approach is based on column generation which is used together with the branch-and-bound algorithm.

The LP relaxation is solved to optimality by first considering a restricted LP relaxation (RLP, of the restricted master problem), where many columns are left out. The solution of the RLP is optimal for the LP relaxation if all variables of the LP relaxation have non-negative reduced costs. Since only a subset of the columns of the LP relaxation is

explicitly available, this can not be checked explicitly. For verifying optimality a pricing algorithm is used. If the solution is not optimal, this algorithm identifies at least one column with negative reduced costs. This column is added to the RLP, and the procedure continues. The column generation scheme terminates when no columns with negative reduced costs can be found. The optimal solution to the RLP is then also optimal for the LP relaxation. Generally the optimal solution found for the LP relaxation is non-integer and hence not feasible for the integer LP, so it is necessary to use branching to find a feasible solution to the ILP. Thus, each realization of a branch-and-price approach requires two major pieces: a method for generating the "best" improving column relative to the current set (a pricing algorithm) and a method for branching away from fractional solutions. It is known that standard branching schemes do not work in case of generation of improved shifts. Thus, Mehrotra, Murphy, and Trick specially developed branching rules that required only minimal modification to the subproblem routine. In addition, they also proposed the pricing algorithm.

3.2 Pricing strategy (the column generation)

In the branch-and-price algorithm, sets of columns are left out of the LP relaxation because there are too many of them. Such a problem can not be solved directly. However, a restricted master problem that considers only a subset of the columns can be solved directly using, for instance, the simplex method. Additional columns for the restricted problem can be generated as needed by solving the subproblem of the pricing problem, which is a separation problem for the dual LP. Using optimal dual variables from the restricted LP relaxation, a pricing subproblem can be solved exactly or heuristically to generate a column with negative reduced costs. If such columns are found, the LP is reoptimized. Otherwise the current optimal solution for the RLP is also optimal for the unrestricted LP. In case the optimal solution does not satisfy the integrality conditions the branching procedure is applied.

Let us write the given GF problem in vector form and replace the integrality constraints "$x_j \geq 0$ and integer for $j \in J$" by "$x_j \geq 0$, $j \in J$". Hence we have:

$$\min \quad c^T x$$

$$\text{subject to} \quad p \leq Ax \leq q, \qquad (1)$$

$$r \le Bx \le s,$$
$$x_j \ge 0 \quad \text{for} \quad j \in J.$$

The column generation scheme starts with a restricted LP (RLP) relaxation that contains a subset of the columns in coefficient matrices. For this purpose, a set of columns is required that forms a feasible basis for the LP. For the moment we assume that such a set of columns is available. These determine the initial RLP, which can be formulated as follows:

$$\begin{aligned} \min \quad & \bar{c}^T \bar{x} \\ \text{subject to} \quad & p \le \bar{A}\bar{x} \le q, \qquad (2) \\ & r \le \bar{B}\bar{x} \le s, \\ & \bar{x}_j \ge 0 \ \text{for} \ j \in \bar{J} \subseteq J. \end{aligned}$$

The RLP has the same form as the original LP but has a subset $\bar{J}$ of the variables with corresponding submatrices $\bar{A}$ and $\bar{B}$, and cost vector $\bar{c}$. The corresponding dual problem is then given by

$$\begin{aligned} \max \quad & q^T u_1 + s^T w_1 + p^T u_2 + r^T w_2 \\ \text{subject to} \quad & \bar{A}^T u_1 + \bar{B}^T w_1 + \bar{A}^T u_2 + \bar{B}^T w_2 \le \bar{c}, \qquad (3) \\ & u_1 \ge 0, w_1 \ge 0, u_2 \le 0, w_2 \le 0. \end{aligned}$$

The column generating scheme starts by solving the RLP and the dual problems to optimality. Since RLP (2) has the same constraints as the LP (1), any basic feasible solution of the RLP forms a basic feasible solution of the LP (by letting $x_j = 0$ for $j \in J \setminus \bar{J}$). In general, an optimal solution to the RLP is not necessarily optimal for the LP. The strong duality theorem helps to determine if an optimal solution of the RLP is also optimal for the LP:

Strong Duality Theorem*: For a linear program (P) and its dual (D), there are only the following possibilities:*

1) (P) is bounded and feasible and (D) is bounded and feasible. In this case opt(P)=opt(D),
2) (P) is infeasible, (D) unbounded,
3) (P) unbounded, (D) infeasible,
4) (P) infeasible, (D) infeasible.

Since any feasible solution of the RLP is also feasible for the LP, the primal is feasible. From the other side, any optimal solution $\bar{x}^*$ of (2) corresponds to the optimal solution u_1^*, w_1^*, u_2^*, and w_2^* of (3) and also

$$\bar{c}^T \bar{x}^* = u_1^{*T} q + w_1^{*T} s + u_2^{*T} p + w_2^{*T} r$$

Thus, for a feasible solution x^* to the LP we have that

$$c^T x^* = u_1^{*T} q + w_1^{*T} s + u_2^{*T} p + w_2^{*T} r .$$

Hence, the optimal solution $\bar{x}^*$ of the RLP is optimal for the LP if the corresponding dual solution u_1^*, w_1^*, u_2^*, and w_2^* of the dual RLP is feasible for the dual of the LP. This is true when

$$A_j^T u_1^* + B_j^T w_1^* + A_j^T u_2^* + B_j^T w_2^* \le c_j \text{ for all } j \in J ,$$

or

$$c_j - A_j^T (u_1^* + u_2^*) - B_j^T (w_1^* + w_2^*) \ge 0 \text{ for all } j \in J .$$

The last inequality can be written as

$$c_j - \sum_i a_{ij} \left[(u_1^*)_i + (u_2^*)_i \right] - \sum_i b_{ij} \left[(w_1^*)_i + (w_2^*)_i \right] \ge 0 \quad \text{for all } j \in J .$$

Here the term $c_j - \sum_i a_{ij} \left[(u_1^*)_i + (u_2^*)_i \right] - \sum_i b_{ij} \left[(w_1^*)_i + (w_2^*)_i \right]$ is in fact the reduced costs $\hat{c}_j$ of the

variable x_j. Evidently, $\hat{c}_j \ge 0$ for all $j \in \bar{J}$.

Therefore, determining if an optimal solution of (2) is also optimal for (1) boils down to determining if the reduced costs $\hat{c}_j$ of all variables x_j for all $j \in J \setminus \bar{J}$ is non-negative. Thus, for any shift $j \in J \setminus \bar{J}$ such that the costs c_j of the shift is less than the sum

$$\sum_i a_{ij} \left[(u_1^*)_i + (u_2^*)_i \right] + \sum_i b_{ij} \left[(w_1^*)_i + (w_2^*)_i \right]$$

provides an improved shift. Obviously, the most improving solution corresponds to the shift that satisfies

$$\max_{j \in J} \left\{ \sum_i a_{ij} \left[(u_1^*)_i + (u_2^*)_i \right] + \sum_i b_{ij} \left[(w_1^*)_i + (w_2^*)_i \right] - c_j \right\}.$$

The column generation scheme thus solves the RLP and applies the pricing algorithm successively in each iteration, and terminates when no negative reduced cost columns can be found in the pricing algorithm. Classical LP theory insures that this procedure can be done in a finite number of iterations. But in general, the obtained optimal solution of the LP is not integer, thus one needs to perform branching to enforce integrality.

Cleary, one of the basic aspects of any branch-and-price implementation is the algorithm used to solve the subproblem, which can be formulated as follows:

Given weights $\alpha_i, \beta_i, \gamma_i, \delta_i$ for each time period $i \in T$, determine a shift such that the cost c_j of shift j is less than the sum $\sum_i a_{ij}(\alpha_i + \beta_i) + \sum_i b_{ij}(\gamma_i + \delta_i)$.

The efficiency of column generation procedures is to a large extent determined by the complexity of this subproblem. This problem needs to be solved at each iteration of column generation and further affects not only the effectiveness of the system but also the branching routine. In many applications of column generation the subproblem is relatively easy, but there also exist applications, where the subproblem turns out to be hard. For employee scheduling defining the subproblem itself is not straightforward, some of them can be very complicated.

To overcome this difficulty and to provide generality, simply enumerating all shifts was proposed using precalculation of common values as a method for speeding up calculations.

Given the T time periods, with unrestricted sign values α_t (for the value of having a work period in period t) and γ_t (the value of having a break in period t), the question is to find a maximum value shift. Here the value of the shift is the total value of the work and break values minus the shift costs. Each shift can have a fixed number of alternating work and break series. Let now $W[i, j] = \sum_{t=i}^{j} \alpha_t$ and $B[i, j] = \sum_{t=i}^{j} \gamma_t$. Now, the value of any shift can be calculated in constant time (assuming a constant number of work and break series). Then finding the best assignment is then just enumerating over all feasible patterns.

Of course, using specially structured shifts may have a more efficient pricing routine, but the proposed general technique is reasonably efficient provided the number of shifts is not too large.

3.3. Branching rules

Another difficulty in using column generation for the solution of integer programs is the development of branching rules to ensure integrality. After termination of the column generation scheme on the RLP, the resulting optimal LP solution is found. If it is integral, this solution also forms an optimal solution to the shift scheduling problem. If the solution is fractional, some branching algorithm is needed to transform the fractional solution into an integral solution. It is known that standard techniques generally cannot be used directly for restricted integer programs where the columns are generated implicitly since they can interfere with the column generation algorithm.

A general branching rule for solving ILP's by branch-and-price has been developed by F. Vanderbeck in 1994. This branching rule is applicable when the coefficients of variables are 0 or 1 and is based on the observation that there always exists a subset of rows such that the sum of variables that cover each row in that subset sums up to be a fractional basic feasible solution. Since then a large number of specific branching schemes for different problems appeared in literature.

Mehrotra, Murphy, and Trick describe several possible branching schemes for the shift scheduling problem that require only minimal modifications to the subproblem routine.

Let J' be the set of all shifts that represented in the final matrix A constructed by the column generation procedure.

Described below, work-based and break-based strategies, like traditional schemes, separate the feasible space into two parts and the current fractional solution is not feasible in either branch:

Work period based branching: Suppose that in an optimal fractional solution there exist a time period t such that $\sum_{j \in J'} a_{tj} x_j = f_t$ with a fractional f_t. Create two branches, one in which $\sum_{j \in J'} a_{tj} x_j \le \lfloor f_t \rfloor$ and $\sum_{j \in J'} a_{tj} x_j \ge \lceil f_t \rceil$ in another.

Break period based branching: Suppose that in an optimal fractional solution there exist a time period t such that $\sum_{j \in J'} b_{tj} x_j = g_t$ with a fractional g_t. Create two branches, one in which $\sum_{j \in J'} b_{tj} x_j \le \lfloor g_t \rfloor$ and $\sum_{j \in J'} b_{tj} x_j \ge \lceil g_t \rceil$ in another.

Obviously, here the problem in each branch is another instance of the general set covering formulation, and the subproblem for the column generation stays the same. Thus, these branching rules do not change either the master or the subproblem. It is well known, that under some restrictions, these branching rules are applicable in any noninteger solution to the GF relaxation. For example, both of the above described rules are applicable in case when all shifts have the same length, number and placement of breaks but as well as in case of different start times. Suppose now, that there is a single shift length with a single period floating break. Then, for a fractional optimal solution either the work-based branching rule or the break-based branching rule (or both) can be used. Also the same is true in our case when there is a single shift length with a single floating break of fixed length.

In the following branching scheme a duty period for a shift is used, that starts with the first work period and ends with the last work period (here the break periods are considered to be part of it). Let D be the set of all possible duty periods. Set now $d_{ij} = 1$ if shift j has duty period $i \in D$, and $d_{ij} = 0$ otherwise. Then the branching rule, based on extending the general set covering formulation to include duty period constraints, is:

$$t_i < \sum_{j \in J} d_{ij} \le u_i \ , \ i \in D .$$

Here t_i and u_i are the minimum and maximum number of employees that can be allocated to a duty period i. This value is based on the total number of employees who can either work or be on a break in a time period that is part of this duty period.

Duty period based branching: Suppose that in an optimal fractional solution there exists a duty period i such that $\sum_{j \in J'} d_{ij} x_j = h_t$, where h_t is fractional. Create two branches, one which enforces $\sum_{j \in J'} d_{tj} x_j \le \lfloor h_t \rfloor$, and the other which enforces $\sum_{j \in J'} d_{tj} x_j \ge \lceil h_t \rceil$.

This branching scheme needs just a simple modification of the subproblem routine and can be useful in many cases, when the number of duty periods is small enough. The routine for the subproblem must be modified to apply to dual variables when the duty period is determined. From theory it is well known, that in case when all shifts corresponding to the fractional shifts have the same length and neither the work- nor the break-based rules are applicable, the duty period based branching rule also cannot be used.

Authors have also introduced a more general rule for branching, which can be used not only for the shift scheduling problem, but for all set partitioning type problems. This rule is based simply on finding a set of (fractional) columns covering the two time periods and summing up to be fractional. Let $r_{ijk} = 1$ denote that time periods i and j are both work periods for shift k, and $r_{ijk} = 0$ otherwise. The following rule is based on the general set covering formulation extended by constraints, which represent all feasible combinations of two time periods that can be simultaneously covered by any shift:

$$l_{ij} < \sum_{k \in J} r_{ijk} x_k \leq u_{ij}, \text{ for } i, j \in T .$$

Here the numbers l_{ij} and u_{ij} reflect the minimum and maximum number of employees that can be allocated to a shift that have work periods in both the time periods i and j.

Two period branching rule: Suppose that in an optimal fractional solution there exists a two period combination (i, j) such that $\sum_{k \in J'} r_{tjk} x_k = h$ with a fractional h. Create two branches, one in which $\sum_{k \in J'} r_{tjk} x_k \leq \lfloor h \rfloor$ and $\sum_{k \in J'} r_{tjk} x_k \geq \lceil h \rceil$ in the other.

The last of the introduced branching schemes can be applied to any fractional solution and is based simply on branching on a single decision variable:

Specific shift branching rule: In case when a variable x_k takes on a fractional value h, simply create two subproblems: one with constraint $x_k \leq \lfloor h \rfloor$ and the other with constraint $x_k \geq \lceil h \rceil$.

Obviously, this branching rule can be used to any fractional solution, but may make the subproblem solution more complicated.

Other branching strategies are still possible, but they are more complicated. It is clear, that choosing between alternative branching rules depends on the used pricing algorithm and the master problem. In some scheduling environments there is no need to use the complicated branching rules. For example, in cyclic scheduling (this is 24-hours scheduling) the specific shift branching can be fruitfully employed. In noncyclic scheduling, under some restrictions, the break based and the work based branching schemes are sufficient.

The proposed method above for solving the subproblem is making clear the advantages of work and break period branching strategies. Such a branching method only changes the α and γ values and therefore does not change the structure of subproblem. Two-period branching is almost as easy, since the duals for that problem can be applied to the $W[i, j]$ and $B[i, j]$ values. Duty period branching might be difficult to apply, but it is usually it is possible to determine the duty period for a shift a-priori and therefore its dual can be easily applied. It is only the specific shift branching rule that can be a really problem. In this case, a feasible shift must be compared against a list of shifts for which the dual values apply.

4. COMPUTATIONAL RESULTS

The proposed method was tested on a variety of scheduling instances from the literature. The test problems were taken from Aykin (24-hours cyclical scheduling) and Thompson (15-hours and 20-hours noncyclical scheduling) studies. These problems differ in the demand patterns, the planning horizons, the allowable shift characteristics, the length of duty periods, the number and placement of breaks. In the Aykin study all used shifts have the same costs, in the Thompson study the costs are proportional to the number of work hours in a shift. They also used a set that permits multiple length duty periods of shifts in combination with multiple breaks. This created problems involving up to 86,400 shifts, which is more than five times as large as the problem solved in the literature.

Problems in the Aykin study were solved using the integer programming option of LINDO running on a VAX8550 computer, Thompson implemented his method using SAS-OR on a 486DX33 based personal computer.

The branch-and-price algorithm has been implemented using MINTO, a Mixed INTeger Optimizer. MINTO is a system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxation. It is also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. The system was used with all system functions, such as preprocessing and cut generation, turned off. Computational experiments have been conducted with CPLEX 4.0 and have been run on a DEC ALPHA 3000 (model 900). In order to provide a general code, only work-, break and the two-period branching rules were used. All subproblems were solved by the enumeration and precalculation approach presented by Mehrotra, Murphy, and Trick.

The comparison across different software packages and computing systems is very difficult and can be misleading. Hence, for a more realistic comparison of the studied approaches, also the Aykin formulation was implemented used CPLEX 4.0 on a DEC ALPHA 3000 (model 900).

4.1. The Thompson study

In his study Thompson used 2 different sets of problems: 15-hour and 20-hour demand patterns, which vary greatly along the requirements pattern, the length of planning intervals, the length of the operational day, and the restrictions defining acceptable shifts. He allowed 3 hours of CPU time for solving each problem. Under that condition, 16 problems (from 588) failed to solve within the given time, and the problems that could be solved took approximately on average one hour. Because of the size of the test set, only the largest of the problems were used for the comparison study. They were solved by Aykin`s method and the branch-and-price approach (BPA).

For the set of problems with 15-hours demand patterns the maximum CPU time required by the BPA was 2.2 sec and on average only about 7% of total possible shifts were generated. Aykin`s method took on average about 25 times more CPU time. However, both methods solved all the problems to optimality within the time limit (4 min CPU time for Aykin`s method and 2 min CPU time for BPA).

From 80 tested problems of 20-demand patterns Aykin`s formulation could solve only 55 in the given 4 min of CPU time, and BPA found optimal solutions for 69 problems. In the problems that were solved by both methods, Aykin`s formulation took between 10 to 40 times more CPU time than BPA. In all of the instances of the problems for both formulations fewer part time shift costs per period were used than regular time shifts. Furthermore, in each of the 11 problems for which BPA exceeded the time limit before confirming optimality, the solution was within the cost of one time period of the lower bound on the problem. Thus, it is very much likely that some or even all of these solutions were in fact optimal, but this can not be verified using the branch-and-price algorithm.

4.2. The Aykin study

For comparison, 40 problems were studied in Aykin [1]. The BPA confirmed optimality in 27 problems within the given time (2 min), for the rest of the 13 problems the set covering formulation was solved by using only the shifts generated in the branch-and-price scheme. This procedure is a heuristic, but in many cases the optimality of solutions can be proved by comparing to the generated linear programming lower bound. The time

limit for this procedure was stated as 2 min. Within this time 8 of the 13 problems were solved to provable optimality. Aykin`s formulation also could not solve to optimality 5 out of the 40 problems. The computational study shows that in this set of problems Aykin`s formulation gives better CPU time than the BPA.

4.3. Multiple duty periods and multiple breaks

The branch-and-price approach was tested on one additional set of problems, which the authors named as “multiple spans and multiple breaks”. For these instances the same demand patterns as in Aykin`s study was considered. In these problems a set of rules was used that permits multiple length duty periods (spans) of shifts along with including multiple breaks. Also cyclic shifts starting in every time period ware permitted. These rules give problems involving up to 86,400 alternative shifts, and here the BPA was superior. This method was able to solve 10 more problems than Aykin`s formulation and the total CPU time used by BPA was less than used by Aykin`s.

These rules were also used on the demand patterns from Thompson`s study. Most of the problems were easily solved by both methods (Aykin`s formulation and BPA), CPU time for 15-hours demand patterns was less than 1s, and less than 10s on the 20-h demand patterns for instances which could be solved within the given time. However, neither approach found an optimal solution for 11 of the 64 problems in the time limit of 2 min.

5. AN EXAMPLE

Shift scheduling is very important for banks in the staffing of tellers as well as some back office staff. This has the same features like other customer service organizations. The main difficulty is the variability of demand over the day. Hence, using not regular part-time shifts can be very useful to cope with this variability. The branch-and-price approach which provides the high level of flexibility in used shifts, seems to be a good technique for solving this problem.

We illustrate the proposed modification of the branch-and-price approach for a shift scheduling problem (see [7]) for the “Rabobank” bank and present computational results with implementation issues.

Problem formulation

The problem involves the construction of 12-hours operational day schedule for bank employees. Employee requirements are determined for each of the 48 15-min planning periods. On basis of the CAO (the Collective labor agreement, an agreement between one or more categories of employers and employees) for service employees and common sense, the following assumptions are used:

1) the work time must be not more than 9 and not less than 4 hours;
2) in case the work time is more than 5.5 hours, the break time is 30 minutes either in one block or divided into 2 breaks of each 15 minutes;
3) in case the work time is more than 8 hours, the break time is 45 minutes, and 30 minutes of them should be in one block;
4) in case the work time is less than 5.5 hours, the break time is 15 minutes;
5) a work time without breaks must be not less than 1.75 and not more than 3.25 hours.

In order to state the given problem in the general set covering formulation, the set of decision variables, constraints and the objective function are defined as follows:

Let t be a time period, $t \in T = \{1,2,\ldots,48\}$, and j the shift index, $j \in J$, where J is a set of all possible shifts, which are a-priori not defined explicitly. Then, the non-

negative decision variable x_i is the number of employees that are assigned to a shift j. Further, each shift j is given by variables a_{tj} and b_{tj}, where

$$a_{tj} = \begin{cases} 1, & \text{if period } t \text{ is a work period for shift } j \\ 0, & \text{otherwise} \end{cases}$$

and

$$b_{tj} = \begin{cases} 1, & \text{if period } t \text{ is a break period for shift } j \\ 0, & \text{otherwise} \end{cases}$$

In this model two sets of constraints are used:

$$p_t \le \sum_{j \in J} a_{tj} x_j \le q_t, \; t \in T$$

and

$$r_t \le \sum_{j \in J} b_{tj} x_j \le s_t, \; t \in T,$$

The first set involves the work period constraints. Here p_t is the minimum number of employees who can be working during time period t, that value is equal to the employee requirement in the given period; q_t represents the maximum number of employees who can be working at time period $t \in T$. Thus, these constraints ensure that demand requirements are met.

The second set involves the break period constraints. Here r_t is the minimum number of employees who can be using period $t \in T$ as a break period, in our case that value is equal to zero, and s_t represents the maximum number of employees who can have a break. These restrictions could be useful when there is limited space available for employees on break (due to limited accommodation, fire regulation, or company policy). In the studied instance, we let $s_t = 9$ for all $t \in T$.

The objective function is given by

$$\min \sum_{j \in J} c_j x_j,$$

where c_j the cost of assigning an employee to shift j. We assume that all shifts have the same costs per working period, and then these values are equal to number of work periods for the given shift.

At the start of the algorithm, the matrices $A = (a_{tj})_{t \in T, j \in J}$ and $B = (b_{tj})_{t \in T, j \in J}$ are not defined, and the solving procedure begins with a small subset of possible shifts, which

gives a feasible solution for the stated problem. Furthermore, shifts are generated during the column generation procedure.

Decomposition of the problem into a master and subproblem

The optimal solution of the stated ILP problem is found by first solving the LP by column generation, and then using the branching algorithm, proposed in [7].

The idea of the column generation is to work only with a small subset of variables that forms a restricted master problem; more variables are added only when needed. Then each iteration of this algorithm consists of two parts: first – optimizing the restricted master problem in order to determine the current optimal objective function value and dual values and second – finding, if there still is one, a new legal shift with maximal negative reduced cost. This second step (which is often named as the subproblem) can be done by solving the problem

$$\max_{j \in J} \left\{ \sum_i a_{ij} \left[(u_1^*)_i + (u_2^*)_i \right] + \sum_i b_{ij} \left[(w_1^*)_i + (w_2^*)_i \right] - c_j \right\},$$

where u_1^*, u_2^* and w_1^*, w_2^* are the dual values for each of the work and the break period constraints, respectively.

An other approach for solving the subproblem was proposed in [7]. The pricing is done by simply enumerating all legal shifts, and for speeding up by computation, precalculation of common values is used. In case when at least one shift with negative reduced cost is found, the new shift with maximal negative cost is added to the restricted master problem, and the next iteration of column generation starts with new matrices $A = (a_{ij})_{i \in T,\, j \in J}$ and $B = (b_{ij})_{i \in T,\, j \in J}$. If there are no legal shifts with negative reduced costs, then there are no improving shifts and we are done with the linear relaxation of the original problem. The algorithm can be summarized as follows.

Step 0. Start with a basic feasible solution, construct a set of columns, say $A_0 = \{a_{ij} : i \in T, j \in J'\}$, such that given LP problem has a feasible solution. Set $A = A_0$. Then the matrix A represents a set of legal shifts and define a matrix with break values $B = (b_{ij})_{i \in T,\, j \in J'}$, and the cost vector $c = (c_j)_{j \in J'}$.

Step 1. Using the simplex method, solve the restricted master problem

$$\begin{aligned} &\min \quad c^T x \\ &\text{subject to} \quad p \le Ax \le q \\ &\qquad\qquad r \le Bx \le s \\ &\qquad\qquad x_j \ge 0 \quad \text{for} \quad j \in J' . \end{aligned}$$

Step 2. Using the found dual variables u_1^*, u_2^*, w_1^* and w_2^*, calculate the common values

$Wp[i, j] = \sum_{t=i}^{j} \left[(u_1^*)_t + (u_2^*)_t \right]$ and $Bp[i, j] = \sum_{t=i}^{j} \left[(w_1^*)_t + (w_2^*)_t \right]$

for all $i, j \in T$.

Step 3. Solve the subproblem: find a column N that represents a legal shift with maximal negative reduced costs. If there are no legal shift with negative reduced costs, go to Step 4. Otherwise, add column N to matrix A. Thus, set $A = (A, N)$, $J' = J' \cup \{\text{new shift}\}$ and construct on basis of it a new matrix B and cost vector c. Go to Step 1.

Step 4. Solve the final master problem

$$\begin{aligned} &\min \quad c^T x \\ &\text{subject to} \quad p \le Ax \le q \\ &\qquad\qquad r \le Bx \le s \\ &\qquad\qquad x_j \ge 0 \quad \text{for} \quad j \in J' ; \end{aligned}$$

The found solution is an optimal solution of the original LP problem.

Solving the subproblem is a major issue of any column generation approach. The details of this procedure are discussed in the next part of this section.

For generating the “best” improving shift relative to the current matrices A and B, the reduced costs of all allowable shifts must be examined. Thus, at the first step the rules for constructing these shifts must be formulated.

All legal shifts depend on the break duration and can be divided into three groups:

- shift type 1: duty period (duration of a shift including breaks) between 4.25 and 5.75 hours, with one break of 15 min;
- shift type 2: duty period between 5.75 and 8.5 hours, with one break of 30 min or two breaks of each 15 min;
- shift type 3: duty period between 8.5 and 9.75 hours, with two breaks (30 min and 15 min).

We will illustrate the creation of a new allowable shift in the next example.

Let t_{start} and t_{end} be the first and the last periods of the generated shift and suppose that duty period of it has a value between 5.75 and 7 hours. Then, 30 minutes of break can be assigned in two different ways: one break of 30 min, or 2 breaks of each 15 min. In order to check all allowable shifts with these start end finish periods, we define a break window, that is a set for periods that can be a start period for a break. Thus, in the first case there is one break window and in the second there are two break windows. The first and the last period of these windows are defined by the restrictions given above.

In case of one 30-min break, the break window can not start earlier then period $(t_{start} + 7)$ (the work time without break must not be less than 1.75 hours, or 7 periods). From another side, the work time after break can not be longer than 3.25 hours, or 13 periods, and that means that the break can not start earlier than period $(t_{end} - 14)$, with respect to 2 break periods. Because both conditions must be satisfied, the start period of the break window is $w_{start} = \max(t_{start} + 7, t_{end} - 14)$. For the same reasons, the last period of the break window is given by $w_{end} = \min(t_{start} + 13, t_{end} - 8)$. For shifts with a length between 5.75 and 7 hours (23 and 28 periods) $t_{end} - 8 > t_{start} + 13$, and thus $w_{end} = t_{start} + 13$. Now, each period from the interval $[w_{start}, w_{end}]$ can be the start period for the break.

In case of two 15-min breaks, the first break cannot be started earlier than $w1_{start} = \max(t_{start} + 7, t_{end} - 27) = t_{start} + 7$, and later than $w1_{end} = \min(t_{start} + 13, t_{end} - 14)$ for the same reasons: after the break, the employee can not work longer than 27 periods (3.25 hours work + 0.25 hours second break + 3.25 hours work) and shorter than 14 periods (1.75 hours work + 0.25 hours break + 1.75 hours work). Suppose now, that the first break started at period t_{break1}. Then the window for the second break is given by $[w2_{start}, w2_{end}]$, where

$$w2_{start} = \max(t_{break1} + 8, t_{end} - 14)$$

and $$w2_{end} = \min(t_{break1} + 14, t_{end} - 7).$$

All legal shifts must be checked, thus we must construct all allowable shifts with one 30-min break:

- first work block: from t_{start} until $t_{break} - 1$,

- break: from t_{break} until $t_{break} + 1$,

- second work block: from $t_{break}+2$ until t_{end} for $t_{break}=w_{start},w_{start}+1,...,w_{end}$;

and all allowable shifts with two 15-min breaks:

- first work block: from t_{start} until $t_{break1}-1$,

- first break: t_{break1} for $t_{break1}=w1_{start},w1_{start}+1,...,w1_{end}$,

- second work block: from $t_{break1}+1$ until $t_{break2}-1$ for $t_{break2}=w2_{start},w2_{start}+1,...,w2_{end}$,

- second break t_{break2},

- third work block: from $t_{break2}+1$ until t_{end}.

In order to solve the subproblem, the reduced costs of all legal shifts for each pair (t_{start},t_{end}) must be checked, where t_{start} takes a values 1, 2,…, 32 (no shifts can start later; the minimum duty time is 4.25 hours) and t_{end} takes values $t_{start}+16$, $t_{start}+17$, …, $t_{start}+\min(38,48-t_{star})$. The shift with maximal negative reduced costs (in case if there at least one of such shifts) is constructed and the corresponding column is added to the matrix A. This process is repeated until there are no improving shifts.

5.3. Branching algorithm

The solution found by column generation is not necessarily integer and applying a standard branch-and-bound procedure will not guarantee an optimal (or feasible) solution. Therefore, a sequence of specific branching rules (work-, break-, duty-, two- period branching rules) were proposed in [7]. From theory is well known that for non-cyclic problems under some simple restrictions the first two branching rules (work- and break-periods) are applicable. For the given problem, the depth-first search strategy (descend deeply into the solution tree along one branch until a target node is found, or until it is pruned out) and a work-period branching rule are chosen.

The search tree is developed dynamically during the search and consists initially of only the root node. Then, on each iteration of the branching procedure the current node becomes pruned or two new child nodes are constructed by the addition of constraints to

the problem of the current (parent) node. This decomposition of the solution space is given by the next branching rule: let t be a time period such that $\sum_{j\in J'} a_{tj}x_j = f_t$ with a fractional value f_t. Then, create two branches by adding to the current problem the constraint $\sum_{j\in J'} a_{tj}x_j \le \lfloor f_t \rfloor$ on one node and $\sum_{j\in J'} a_{tj}x_j \ge \lceil f_t \rceil$ on the other. This branching rule does not change the original problem

$$\min \sum_{j\in J'} c_j x_j$$
$$\text{subject to } p_t \le \sum_{j\in J'} a_{tj}x_j \le q_t, \text{ for } t \in T,$$
$$t_i \le \sum_{j\in J'} b_{tj}x_j \le s_t, \text{ for } t \in T,$$
$$x_j \ge 0 \text{ for } j \in J'.$$

Adding of the first constraint ($\sum_{j\in J'} a_{tj}x_j \le \lfloor f_t \rfloor$) simply changes the value of q_t, and the second gives a new value for p_t. Thus, for a chosen time period t on the first node we have the constraint $p_t \le \sum_{j\in J'} a_{tj}x_j \le \lfloor f_t \rfloor$ and on the second node $\lceil f_t \rceil \le \sum_{j\in J'} a_{tj}x_j \le q_t$.

Further, in order to reduce the size of the branching tree, the next pruning criteria are used :

- pruning by optimality: the solution found on the current node is integer;
- pruning by infeasibility: the constructed problem is infeasible;
- pruning by bound: the lower bound of the solution found is greater than the upper bound for all previously examined solutions. In that case, no optimal solution can lie on the current branch.

It is well known, that branching algorithms give the integer solutions for the ILP in a finite number of iterations, thus, theoretically, the procedure stops when all nodes of the search tree are either pruned or solved. But in reality there are many problems in which the branching tree is too large to be checked in a reasonable execution time. The proposed branching procedure allows 500 iterations; after that the best feasible solution found (incumbent solution) is used as an approximation to the optimal solution of the original ILP, if there is at least one illuminated. One of the advantages of the depth-first search strategy is that most likely a feasible solution is quickly found.

Now, the proposed branching procedure can be summarized as follows:

Step 0. (Initialization) Set the current best objective $zopt = \infty$, the incumbent solution $incumb = (0)_{j, j \in J'}$ (the zero vector). Solve the original problem; if the solution is integer, go to Step 8; otherwise if there exists a time period $t*$ such that $\sum_{j \in J'} a_{t*j} x_j = f_{t*}$ with fractional f_{t*}, go to Step 1, otherwise go to Step 7.

Step 1. Create a stack for the branching tree; store the current node (time period $t*$, value $\lfloor f_{t*} \rfloor$, and a direction indicator that is equal to 0 if we taking a node with the extra constraint $\sum_{j \in J} a_{tj} x_j \le \lfloor f_{t*} \rfloor$, and 1 otherwise). Set *Level=1*, the number of iterations equal to 0, and *indicator= 0*. Go to Step 2.

Step 2. If *Level=0* , or the number of iterations is equal to 501, go to Step 7. Otherwise, if *indicator=0*, store the current value of q_{t*} and set $q_{t*} = \lfloor f_{t*} \rfloor$; otherwise store the value of p_{t*} and set $p_{t*} = \lceil f_{t*} \rceil$. Increase the number of iterations by 1 and solve the created problem. If the problem has no feasible solution go to Step 6, otherwise go to Step 3.

Step 3. (pruning by bound) If the objective value of the solution $z(x^*)$ found is greater than *zopt* (thus, the optimal solution can not be at any of the child nodes from the current node and hence we can prune the tree at this node) go to Step 6 (pruning the current node), otherwise go to Step 4.

Step 4. If the solution x^* to the solved problem satisfies the integer constraints, set $zopt = z(x^*)$, $incumb = x*$. Since this branch has been fathomed, go to Step 6 (pruning the current node). If the integer condition is not satisfied, go to Step 5.

Step 5. (creating new nodes) Take a new time period $t*$ such that $\sum_{j \in J'} a_{t*j} x_j = f_{t*}$ with fractional f_{t*}, and store the new node (time period $t*$, value $\lfloor f_{t*} \rfloor$, *indicator = 0*). Set *Level=Level+1,* and go to Step 2.

Step 6. (prune the current node). While *indicator = 1* for the current node, delete it from the stack, remove the corresponding constraint from the model (restore the previously value of p_t for this time period); go to the next node and set *Level =Level - 1*. Set *indicator=1,* restore the value of q_{t*} for the new current time period $t*$, and go to Step 2.

Step 7. Print current (incumbent) solution, Stop.

Step 8. Print "The solution is not integer; there is no work period with fractional value; another branching rule must be used. ", Stop.

5.4. Computational results

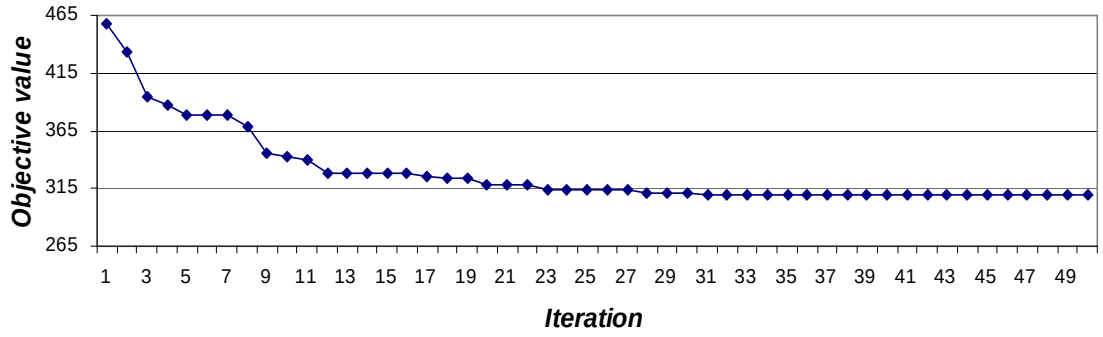
The column generation and the branching algorithm were implemented using GAMS IDE rev 145, the LP solver used is CPLEX 10.0.1.

The column generation procedure starts with a reduced *A*-matrix that has 4 columns. This initial restricted master problem has a feasible LP relaxation, the corresponding optimal solution has objective value 458.00. When the subproblem is solved, many columns with equal negative reduced costs are available. It is not possible to point a-priori the one of them that will give the most improvement. Different strategies can be used for choosing the column (or few of them) that must be added to the restricted problem. For instance, we can generate columns which maximally improve the objective function value. But practical usefulness of such proposals is a very disputable subject due to required execution time and resources. The used procedure generates the first column with maximum negative value of the reduced costs. After solving the subproblem on the first iteration, one of these columns (with value of the reduced costs equal to - 55.00) is chosen. Including this column rapidly reduces the objective value to 434.00. On the next iterations algorithm converges fast, for example, objective value of the solution found on 15th iteration is 320.00. Unfortunately, simplex-based column generation is known for its poor convergence. A typical phenomena in column generation is a tailing-off effect, that is at the end of the procedure columns with negative reduced costs are still found but the value of the objective function is not or almost not improving. In such an event after the column with negative reduced costs is added to reduced LP, it enters the basis with a value zero.

In the solved instance such a situation appears on the iteration 33, since that moment adding of new improving columns to the matrix does not change the value of the objective function, and it stays equal to 309.00 during next 247 iterations, after that number of steps the procedure stops. The method still converges but listed above the tailing-off effect makes the further process not interesting and we can conclude that without a principal correction of the basic column generation algorithm it is not possible for the given problem to get the proved optimal solution within a reasonable time.

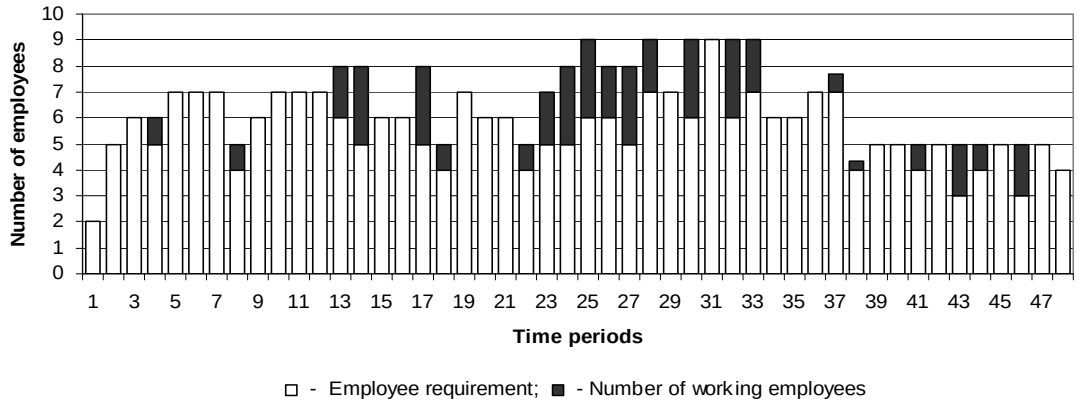
Next graphic illustrated converge process of the column generation for the given instance:

Values of the objective function on first 50 iterations of the column generation procedure



The figure below demonstrates the difference between required number of employees per one time period (15 min) and given by the optimal solution of the restricted LP found on 33th iteration of the column generation algorithm (*A*-matrix has 37 columns):

Number of required and planned employees



Two different approaches to deal with the tailing-off effect in different forms were introduced in the literature: lower bound approximation (Freling R., 1997, Models and techniques for integrating vehicle and crew scheduling, PhD Thesis, Tinbergen Institute, Erasmus University, Rotterdam) and stabilized column generation (du Merle et al., Stabilized column generation, Discrete Mathematics 194 (1999), 229-237). The first method terminates column generation when the difference between the solution of the restricted LP and a lower bound to the LP relaxation solution is sufficiently small. The idea of second one is the bounding value of the dual variables which is done by merging the

perturbation and exact penalty methods. The reason of it lays in fact that the dual variable values do not smoothly converge to their respective optimum, but change rapidly, without any regular pattern. This kind approaches base on Boxtep method introduced by Marsten in his work “The use of the boxstep method in discrete optimization” (1975). By including lower and upper bounds dual variables are forced to lie in a sort of “box” around previous dual solution. However, implementation of these techniques falls outside the scope of this thesis.

The GAMS code of the column generation procedure and the computational results can be found in Appendix A and B respectively.

The branching algorithm is tested on one of the restricted problems, that appeared on 14th iteration on the column generation procedure. This problem has 17 columns *A*-matrix; the objective function has a value 324.5 and found optimal solution is fractional. After 0.016 sec executing time 51 branching nodes were created and the integer solution is found. The objective value of it is 332, with integrality gap lesser then 2.5 percent. The GAMS code and the results can be found in Appendix C and D respectively.

6. CONCLUSIONS

There is great variety in scheduling problems and their solution techniques. Hence, there are a lot of approaches and model formulations to solve these problems which are, in general, very difficult to solve. Computational experiments described in [7] clearly show that the modification of the branch-and price approach proposed by the authors gives superior results on a large number of shift scheduling problems (especially in case of a high degree flexibility in the shifts) and is very competitive with alternative methods on shift scheduling problems described in the literature. This method gives the proved optimal solution even for huge problems and hence can be successfully used in cases when the number of potential columns is large and they cannot be generated explicitly. However, some disadvantages of this method can sufficiently restrict the area of its applicability. Here, two elements are extremely important: the subproblem (the pricing algorithm) and the branching strategy. Pricing rules are sensitive to the dual variable values in case of a non-unique dual solution. In some cases, the subproblem can be so complicated and hard to solve, that this computational effort cancels the theoretical benefits. Also, the branching procedure can create serious problems, so that heuristic methods are more preferable.

Still, the previously mentioned advantages of the proposed branch-and-price methodology make this approach a very attractive research area for future studies.

REFERENCES

1. Aykin, T.: Optimal shift scheduling with multiple break windows. Management Science 42, 591-602 (1996)

2. Barnhart, C., Jonson, E.L., Nemnauser, G.L., Salvelsbergh, M.W.P., Vance, P.H.: Branch-and-price: column generation for solving huge integer programs. Operation Research 46, 316-329 (1998)

3. Bechtold, S.E., Jacobs, L.W.: Implicit modeling of flexible break assignments in optimal shift scheduling. Management Science 36, 1339-1351 (1990)

4. Dantzig, G.B.: A comment on Eddie`s traffic delays at toll booths. Operation Research 2, 339-341 (1954)

5. Ernst A.T., Jiang H., Krishnamoorthy M., Sier D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153, 3-27 (2004)

6. Kalvelagen, E.: Column generation with GAMS. http://www.gams.com/~erwin/colgen/colgen.pdf (2002)

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

8. Thompson, G.M.: Improved implicit optimal modeling of the labor shift scheduling problem. Management Science 41, 595-607 (1995)

9. Vanderbeck, F., Wolsey, L.A.: An exact algorithm for IP column generation. Operations Research Letters 19, 151-160 (1996)

10. Wilhelm, W.E.: A technical review of column generation in integer programming. Optimization and engineering 2, 159-200 (2001)

Appendix A

GAMS source code of the column generation procedure Colgen.gms

```
*-----------------------------------------------------------
*  Column generation procedure,
*  the start A-matrix has 4 columns
*  and the first objective is 458.0
*-----------------------------------------------------------

Sets i planning periods   /1*48/,
     n possible shifts /n1*n400/,
     iter  maximum iteration  /  1*300  /,
     j(n) dynamic subset,
     pi(n) dynamic set;

Scalar k help variable;

*---------------------
*   Data
*---------------------

Parameters  p(i)  minimum working in period i employees
                / 1   2,  2   5,  3   6,  4   5,  5   7,  6   7,
                  7   7,  8   4,  9   6,  10  7,  11  7,  12  7,
                  13  6,  14  5,  15  6,  16  6,  17  5,  18  4,
                  19  7,  20  6,  21  6,  22  4,  23  5,  24  5,
                  25  6,  26  6,  27  5,  28  7,  29  7,  30  6,
                  31  9,  32  6,  33  7,  34  6,  35  6,  36  7,
                  37  7,  38  4,  39  5,  40  5,  41  4,  42  5,
                  43  3,  44  4,  45  5,  46  3,  47  5,  48  4   /,

            q(i)  maximum working in period i employees
                  / 1*48     30 /,

            r(i)  minimum having break in period i
                  / 1*48     0/,

            s(i)  maximum having break in period i
                  / 1*48     9 /         ;

 Table  ainit(i,n) work periods
                      n1      n2      n3      n4
             1        1
             2        1
             3        1       1
             4        1       1
             5        1       1
             6        1       1
             7                1
             8        1       1
             9        1       1
             10       1       1
             11       1
             12       1       1
             13       1       1
```

```
14       1       1
15       1       1
16       1       1
17       1       1
18               1               1
19               1               1
20                               1
21                               1
22                               1
23                       1       1
24                       1       1
25                       1       1
26                       1       1
27                       1
28                       1       1
29                       1       1
30                       1       1
31                       1       1
32                       1       1
33                       1       1
34                               1
35                               1
36                       1       1
37                       1       1
38                       1
39                       1
40                       1
41                       1
42                       1
43                       1
44                       1
45                       1
46                       1
47                       1
48                       1    ;
```

```
*---------------------
* Initialization
*---------------------

Parameter a(i,n) matrix work periods growing in dimension n,
          b(i,n) matrix break periods growing in dimension n,
          c(n) cost vector growing in dimension n;

a(i,'n1')=ainit(i,'n1');
a(i,'n2')=ainit(i,'n2');
a(i,'n3')=ainit(i,'n3');
a(i,'n4')=ainit(i,'n4');
b('7','n1')  = 1;
b('11','n2') = 1;
*b('17','n2') = 1;
b('34','n3') = 1;
b('35','n3') = 1;
b('27','n4') = 1;
c(n) = sum(i,a(i,n));
```

```
pi('n1') = yes;
for (k=1 to 4, j(pi)=yes; pi(n)=pi(n-1));

*---------------------
* Master problem
*---------------------

Positive variable x(n) shifts used;

Variable z objective variable;

Equations
         cost      'define objective function'
         lwork(i)  'minimum werking employee'
         uwork(i)  'maximum werking employee'
         lbreak(i) 'minimum employee on pause'
         ubreak(i) 'maximum employee on pause';

cost.. z =e= sum(j, c(j)*x(j));
uwork(i)  .. sum(j, a(i,j)*x(j)) =l= q(i);
lwork(i)  .. sum(j, a(i,j)*x(j)) =g= p(i);
ubreak(i) .. sum(j, b(i,j)*x(j)) =l= s(i);
lbreak(i) .. sum(j, b(i,j)*x(j)) =g= r(i);

model master /all/;

*reduce amount of information written to the listing file
master.solprint = 2;
master.limrow = 0;
master.limcol = 0;


*----------------
* the branch-and-price algorithm
*---------------

Set row /1*48/,
    col /1*48/;

Scalar sp start period voor the genereting shif,
       sht duty time for the generating shift,
       rc reduced costs of the generated shift / 1 /,
       costs costs of the new shift /  0  /,

       sbw1 start period of the first break window,
       ebw1 end period of the first break window,

       sbw2 start period of the second break window,
       ebw2 end period of the second break window,
       iteration,t, maxrc  / 0.001  / ,
       sht / 0 /,
       done / 0 /;

Parameter wshift(i) column of matrix A for new shift (work periods),
          bshift(i) column of matrix B for new shift (break periods),
          zero(i)   / 1*48        0 / ,
          Bp(row,col),
```

```
          Wp(row,col),
          tw1, tb1, tw2, tb2, tw3, duty;


loop (iter$(not done),

      Solve master using LP minimizing z;

      Wp(row,col) = sum(i$(ord(i) GE ord(row) and ord(i) LE ord(col)),
                    lwork.m(i)+uwork.m(i));

      Bp(row,col) = sum(i$(ord(i) GE ord(row) and ord(i) LE ord(col)),
                    lbreak.m(i)+ubreak.m(i));


*---------------------
* Subproblem
*---------------------

sp=1;
wshift(i) = zero(i);
bshift(i) = zero(i);
maxrc = 0.001;
iteration = 1;

while (sp le 31,

    sht = min(38,48-sp);
*------
*  shifts with length (duty time) between 9 and 9,75 (8,25 and 9 work hours)
*------
    while (sht ge 35,

        sbw1 = max(sp+sht-28,sp+7);
        ebw1 = sp + 13;

        while (sbw1 LE ebw1,
*-----------
*         first break (30 min) starts in period sbw1,
*         second break (15 min) starts in period sbw2;
*----------
            sbw2 = max(sbw1+9,sp+sht-13);
            ebw2 = min(sbw1+15,sp+sht-7);

            while (sbw2 LE ebw2,

               rc = sum((row,col)$(ord(row) EQ sp and ord(col) EQ sbw1-
1),Wp(row,col))+
                    sum((row,col)$(ord(row) EQ sbw1+2 and ord(col) EQ
sbw2-1),Wp(row,col))+
                    sum((row,col)$(ord(row) EQ sbw2+1 and ord(col) EQ
sp+sht),Wp(row,col))+
                    sum((row,col)$(ord(row) EQ sbw1 and ord(col) EQ
sbw1+1),Bp(row,col))+
                    sum((row,col)$(ord(row) EQ sbw2 and ord(col) EQ
sbw2),Bp(row,col))-
                    sht + 2;
```

```
               if (rc > maxrc,
*          create a new shift:
                  tw1 = sp;
                  tb1 = sbw1;
                  tw2 = sbw1 + 2;
                  tb2 = sbw2;
                  tw3 = sbw2 + 1;
                  duty = sht;
                  maxrc = rc;
                  );
               sbw2 = sbw2 + 1;
               );
*--------
*          first break (15 min) starts in period  sbw1,
*          second break (30 min) starts in period sbw2
*--------
        if (sbw1 < 40,  sbw2 = max(sbw1+8,sp+sht-14);
                        ebw2 = min(sbw1+14,sp+sht-7););

        while(sbw2 LE ebw2,
           rc = sum((row,col)$(ord(row) EQ sp and ord(col) EQ sbw1-
1),Wp(row,col))+
                 sum((row,col)$(ord(row) EQ sbw1+1 and ord(col) EQ sbw2-
1),Wp(row,col))+
                 sum((row,col)$(ord(row) EQ sbw2+2 and ord(col) EQ
sp+sht),Wp(row,col))+
                 sum((row,col)$(ord(row) EQ sbw1 and ord(col) EQ
sbw1),Bp(row,col))+
                 sum((row,col)$(ord(row) EQ sbw2 and ord(col) EQ
sbw2+1),Bp(row,col))-
                 sht + 2;

           if (rc > maxrc,
              tw1 = sp;
              tb1 = sbw1;
              tw2 = sbw1 + 1;
              tb2 = sbw2;
              tw3 = sbw2 + 2;
              duty = sht;
              maxrc = rc;
           );
           sbw2 = sbw2 + 1;
        );
        sbw1 = sbw1 + 1;
        );
        sht = sht - 1;
    );

    sht = sht - 1;
* there can not be a shift with duty length 35 periods

*------
*  shifts with length (duty time) between 7,25 and 8,5 hours
* (6,75 and 8 work hours); only two breaks (each 15 min) possible
*------
    while (sht ge 28,

        sbw1 = sp + 7;
```

```
        ebw1 = sp + 13;

        while (sbw1 LE ebw1,
*-----------
*            first break (15 min) starts in period sbw1,
*            second break (15 min) starts in period sbw2;
*----------
           sbw2 = max(sbw1+8,sp+sht-13);
           ebw2 = min(sbw1+14,sp+sht-7);

           while(sbw2 LE ebw2,

              rc = sum((row,col)$(ord(row) EQ sp and ord(col) EQ sbw1-
1),Wp(row,col))+
                   sum((row,col)$(ord(row) EQ sbw1+1 and ord(col) EQ
sbw2-1),Wp(row,col))+
                   sum((row,col)$(ord(row) EQ sbw2+1 and ord(col) EQ
sp+sht),Wp(row,col))+
                   sum((row,col)$(ord(row) EQ sbw1 and ord(col) EQ
sbw1),Bp(row,col))+
                   sum((row,col)$(ord(row) EQ sbw2 and ord(col) EQ
sbw2),Bp(row,col))-
                   sht + 1;

              if (rc > maxrc,
                  tw1 = sp;
                  tb1 = sbw1;
                  tb2 = sbw2;
                  duty = sht;
                  maxrc = rc;
              );
              sbw2 = sbw2 + 1;
           );
        sbw1 = sbw1 + 1;
        );
    sht = sht - 1;
    );

*------
* shifts with length between 6,25 and 7 hours (5,75 and 6,5 work hours)
* 1 break (30 min) of 2 x 15 min possible
*-----
    while (sht ge 24,

       sbw1 = max(sp+7,sp+sht-14);
       ebw1 = sp + 13;

       while (sbw1 LE ebw1,
*-----------
*        one break (30 min) starts in period sbw1
*----------
          rc = sum((row,col)$(ord(row) EQ sp and ord(col) EQ sbw1-
1),Wp(row,col))+
               sum((row,col)$(ord(row) EQ sbw1+2 and ord(col) EQ
sp+sht),Wp(row,col))+
               sum((row,col)$(ord(row) EQ sbw1 and ord(col) EQ
sbw1+1),Bp(row,col))-
               sht + 1;
```

```
          if (rc > maxrc,
                  tw1 = sp;
                  tb1 = sbw1;
                  tb2 = 0;
                  duty = sht;
                  maxrc = rc;
          );
          sbw1 = sbw1 + 1;
       );

       if (sbw1 < 40, sbw1 = sp + 7; ebw1 = min(sp+13,sp+sht-14););

       while (sbw1 LE ebw1,
*-----------
*   2 breaks: first break (15 min) starts in period sbw1,
*             second break (15 min) starts in period sbw2;
*----------
          sbw2 = max(sbw1+8,sp+sht-14);
          ebw2 = min(sbw1+14,sp+sht-7);

          while (sbw2 LE ebw2,

             rc = sum((row,col)$(ord(row) EQ sp and ord(col) EQ sbw1-
1),Wp(row,col))+
                  sum((row,col)$(ord(row) EQ sbw1+1 and ord(col) EQ sbw2-
1),Wp(row,col))+
                  sum((row,col)$(ord(row) EQ sbw2+1 and ord(col) EQ
sp+sht),Wp(row,col))+
                  sum((row,col)$(ord(row) EQ sbw1 and ord(col) EQ
sbw1),Bp(row,col))+
                  sum((row,col)$(ord(row) EQ sbw2 and ord(col) EQ
sbw2),Bp(row,col))-
                  sht + 1;

             if (rc > maxrc,
                 tw1 = sp;
                 tb1 = sbw1;
                 tb2 = sbw2;
                 duty = sht;
                 maxrc = rc;
             );
             sbw2 = sbw2 + 1
          );
          sbw1 = sbw1 + 1;
       );
       sht = sht - 1;
    );

    sht = sht - 1;
*------
* shifts with length between 4,25 and 5,75 (4 and 5,5 work hours);
* one break (15 min) is possible
*------
    while (sht ge 16,

       sbw1 = max(sp+sht-13,sp+7);
       ebw1 = min(sp+13,sp+sht-7);
```

```
       while (sbw1 LE ebw1,

          rc = sum((row,col)$(ord(row) EQ sp and ord(col) EQ sbw1-
1),Wp(row,col))+
               sum((row,col)$(ord(row) EQ sbw1+1 and ord(col) EQ
sp+sht),Wp(row,col))+
               sum((row,col)$(ord(row) EQ sbw1 and ord(col) EQ
sbw1+1),Bp(row,col))-
               sht;

          if (rc > maxrc,
              tw1 = sp;
              tb1 = sbw1;
              duty = sht;
              maxrc = rc;
          );
          sbw1 = sbw1 + 1;
          );
       sht = sht - 1;
       sbw1 = sbw1 + 1;
       );
    sp = sp+1;
    );
wshift(i) = zero(i);
bshift(i) = zero(i);


if (duty ge 35,
    for (k=tw1 to tb1-1, wshift(i)$(ORD(i) EQ k)=1);
    for (k=tw2 to tb2-1, wshift(i)$(ORD(i) EQ k)=1);
    for (k=tw3 to tw1+duty, wshift(i)$(ORD(i) EQ k)=1);
    for (k=tb1 to tw2-1, bshift(i)$(ORD(i) EQ k)=1);
    for (k=tb2 to tw3-1, bshift(i)$(ORD(i) EQ k)=1);
    costs = sht - 2;
else
   if (duty ge 24,
       if (tb2 = 0,
*   one break (30 min)
           for (k=tw1 to tb1-1, wshift(i)$(ORD(i) EQ k)=1);
           for (k=tb1+2 to tw1+duty, wshift(i)$(ORD(i) EQ k)=1);
           bshift(i)$(ord(i) EQ tb1 OR ord(i) EQ tb1+1) = 1;
       else
* two breaks each of 15 min
           for (k=tw1 to tb1-1, wshift(i)$(ORD(i) EQ k)=1);
           for (k=tb1+1 to tb2-1, wshift(i)$(ORD(i) EQ k)=1);
           for (k=tb2+1 to tw1+duty, wshift(i)$(ORD(i) EQ k)=1);
           bshift(i)$(ord(i)EQ tb1 OR ord(i) EQ tb2)=1;
           costs = sht - 1;
       );
   else
       if (duty ge 16,
          for (k=tw1 to tb1-1, wshift(i)$(ORD(i) EQ k)=1);
          for (k=tb1+1 to tw1+duty, wshift(i)$(ORD(i) EQ k)=1);
          bshift(i)$(ord(i) EQ tb1) = 1;
          costs = duty;
       );
   );
);
```

```
*---------------------
* end of pricing scheme
*---------------------


* new shift with negatief reduced costs found?

      if (maxrc GE 0.001,

*---------------------
* new schift with negatief reduced costs found
*---------------------
         a(i,pi) = wshift(i);
         b(i,pi) = bshift(i);
         c(pi) = sum(i,a(i,pi));
         j(pi) = yes;
         pi(n) = pi(n-1);
         iteration = ord(iter);
         display iteration, z.l, x.l;

      else
         done = 1 );
);

abort$(not done) "Too many iteration;";

*---------------------
* Solve final master problem
*---------------------

Solve master using LP minimizing z;
Display z.l, x.l, a, c;
display 'The best found solution (LP):',z.l, x.l;
```

Appendix B

The output of the program Colgen.gms (first 35 iterations)

```
COMPILATION TIME     =        0.046 SECONDS      3 Mb  WIN222-145 Apr 21, 2006
GAMS Rev 145  x86/MS Windows                          09/24/06 18:42:35 Page 3
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----    454 PARAMETER iteration            =        1.000
            VARIABLE z.L                   =      458.000  objective variable
----    454 VARIABLE x.L  shifts used
n1 7.000,    n2 7.000,    n3 5.000,    n4 6.000


----    454 PARAMETER iteration            =        2.000
            VARIABLE z.L                   =      434.000  objective variable
----    454 VARIABLE x.L  shifts used
n1 5.000,    n2 5.000,    n3 5.000,    n4 6.000,    n5 2.000
```

Appendix C

The GAMS source code of the branching procedure Branching.gms

```
* ---------------------------------------------
*    branching procedure;
*    work-period branching rule,
*    depth-first branching strategy
*---------------------------------------------


Sets i periods /1*48/,
     n possible shifts /1*17/,
     nc possible nodes /1*10000/,
     ind(nc) dynamic subset,
     pti(nc) dynamic subset,
     m / 1*3 /;

Parameters t, k, zopt, Level,
           cp / 1 /,  cq / 1 /,
           nod  / 1 / ;

Parameters f(i), backupp(nc), incumb(n), backupq(nc),
           nodes(nc,m);
* nodes(*,1): branching period t;
* nodes(*,2): branching value [f(t)];
* nodes(*,3): 0 voor LE, 1 voor G



*---------------------
*   Data
*---------------------

Parameters  p(i)  minimum working in period i employees
                / 1   2,  2   5,  3   6,  4   5,  5   7,  6   7,
                  7   7,  8   4,  9   6,  10  7,  11  7,  12  7,
                  13  6,  14  5,  15  6,  16  6,  17  5,  18  4,
                  19  7,  20  6,  21  6,  22  4,  23  5,  24  5,
                  25  6,  26  6,  27  5,  28  7,  29  7,  30  6,
                  31  9,  32  6,  33  7,  34  6,  35  6,  36  7,
                  37  7,  38  4,  39  5,  40  5,  41  4,  42  5,
                  43  3,  44  4,  45  5,  46  3,  47  5,  48  4   /,


            q(i)  maximum working in period i employees
                  / 1*48     30 /,


            r(i)  minimum having break in period i
                  / 1*48     0/,


            s(i)  maximum having break in period i
                  / 1*48     9 /         ;
```

Table a(i,n) work periods

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1										1		1			1	1
2	1					1	1				1		1			1	1
3	1	1				1	1				1		1			1	1
4	1	1				1	1				1		1			1	1
5	1	1				1	1				1		1		1	1	1
6	1	1				1	1				1		1	1	1	1	1
7		1			1	1	1	1	1		1		1	1	1	1	1
8	1	1			1	1	1	1	1					1	1		1
9	1	1			1	1	1	1	1		1			1	1	1	1
10	1	1			1	1	1	1	1		1		1	1	1	1	
11	1				1	1	1	1	1		1	1	1	1	1	1	
12	1	1			1		1	1	1		1	1	1	1	1	1	1
13	1	1			1			1	1		1	1	1	1		1	1
14	1	1				1			1		1	1	1	1	1	1	1
15	1	1			1	1	1		1		1	1	1	1	1	1	1
16	1	1			1	1	1	1	1		1	1	1	1	1	1	1
17	1	1			1	1	1	1			1	1	1		1	1	1
18		1		1	1	1	1	1			1	1	1		1	1	1
19		1		1	1	1	1	1	1		1	1	1	1	1	1	1
20				1	1	1	1	1	1		1	1	1	1	1		1
21				1	1	1	1	1	1	1			1	1	1		1
22				1	1	1	1	1	1	1	1		1	1			1
23			1	1	1	1	1	1	1	1	1	1		1			
24			1	1	1	1	1	1	1	1	1	1	1	1			1
25			1	1	1	1	1	1	1	1	1	1	1	1			1
26			1	1	1		1	1	1	1	1	1	1	1			1
27			1		1	1		1	1	1	1	1	1	1			1
28			1	1		1	1	1	1		1	1	1	1			1
29			1	1		1	1		1	1	1	1	1	1			1
30			1	1		1	1	1	1	1	1	1	1	1			1
31			1	1		1	1	1	1	1	1	1	1	1			1
32			1	1		1	1	1		1	1	1	1				1
33			1	1		1	1	1	1	1	1	1	1				1
34				1		1	1	1	1	1	1	1	1				1
35				1		1	1	1	1	1		1	1				1
36			1	1		1	1	1	1				1				1
37			1	1		1	1	1	1	1		1					
38			1			1	1	1	1	1		1					
39			1			1	1	1	1	1		1					
40			1				1	1	1	1		1					
41			1					1	1	1		1					
42			1					1	1	1		1					
43			1						1	1		1					
44			1						1	1		1					
45			1						1	1		1					
46			1							1		1					
47			1							1		1					
48			1									1				;	

```
*---------------------
* Initialization
*---------------------


Parameter b(i,n) matrix break periods growing in dimension n,
          c(n) cost vector growing in dimension n;


b('7','1')  = 1;
b('11','2') = 1;
b('34','3') = 1;
b('35','3') = 1;
b('27','4') = 1;
b('14','5') = 1;
b('12','6') = 1;
b('13','6') = 1;
b('26','6') = 1;
b('13','7') = 1;
b('14','7') = 1;
b('27','7') = 1;
b('14','8') = 1;
b('15','8') = 1;
b('29','8') = 1;
b('17','9') = 1;
b('18','9') = 1;
b('32','9') = 1;
b('28','10')= 1;
b('36','10')= 1;
b('8','11') = 1;
b('21','11')= 1;
b('21','12')= 1;
b('22','12')= 1;
b('36','12')= 1;
b('8','13') = 1;
b('9','13') = 1;
b('23','13')= 1;
b('17','14')= 1;
b('18','14')= 1;
b('13','15')= 1;
b('8','16') = 1;
b('10','17')= 1;
b('11','17')= 1;
b('23','17')= 1;

c(n) = sum(i,a(i,n));
```

```
*---------------------------------
*  Model definition
*---------------------------------

Positive variable x(n) shifts used;

Variable z objective variable;

Equations
         cost      'define objective function'
         lwork(i)  'minimum werking employee'
         uwork(i)  'maximum werking employee'
         lbreak(i) 'minimum employee on pause'
         ubreak(i) 'maximum employee on pause';

cost.. z =e= sum(n, c(n)*x(n));
uwork(i)  .. sum(n, a(i,n)*x(n)) =l= q(i);
lwork(i)  .. sum(n, a(i,n)*x(n)) =g= p(i);
ubreak(i) .. sum(n, b(i,n)*x(n)) =l= s(i);
lbreak(i) .. sum(n, b(i,n)*x(n)) =g= r(i);

model master /all/;

*reduce amount of information written to the listing file
master.solprint = 2;
master.limrow = 0;
master.limcol = 0;



*-----------------------------------
* branching procedure
*-----------------------------------

Solve master using LP minimizing z;

display  'Starting solution for given ILP: ',
         '    Objective function: ', z.l,
         '    Dessision varaibles: ', x.l;

f(i) = sum(n, a(i,n)*x.l(n));
Loop(i$(Floor(f(i)) < f(i)), t = Ord(i); Level = 1 );
Loop(n$(Floor(x.l(n)) < x.l(n)), k = Ord(n));

if (t=0 and k > 0,
   display 'On one of nodes found solution is not integer,',
           'there is no work period with fractional value,',
           'another branching scheme must be used.';
   Level = -1;
);

pti('1') = yes;
ind('1') = yes;
nodes(ind,'1') = t;
nodes(ind,'2') = sum((i)$(ORD(i) EQ t),Floor(f(i)));
nodes(ind,'3') = 0;

zopt = inf;
```

```
while ( Level > 0 AND nod < 501,

   nod = nod + 1;
   t = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));

   if ( sum((ind,m)$(ORD(m)EQ 3),nodes(ind,m)) = 0,
        backupq(nc)$(ORD(nc) EQ cq) = sum((i)$(ORD(i) EQ t),q(i));
        q(i)$(ORD(i) EQ t)= sum((ind,m)$(ORD(m)EQ 2),nodes(ind,m));
     else
        backupp(nc)$(ORD(nc) EQ cp) = sum((i)$(ORD(i) EQ t),p(i));
        cp = cp + 1;
        p(i)$(ORD(i) EQ t)= sum((ind,m)$(ORD(m)EQ 2),nodes(ind,m)) + 1;
   );

   Solve master using LP minimizing z;

   if (master.ModelStat = 1,
*  optimal solution is found

         if (z.l > zopt,
*  prune by bound

             k=t;
             while (sum((ind,m)$(ORD(m)EQ 3),nodes(ind,m)) = 1,
                cp = cp - 1;
                p(i)$(ORD(i) EQ k) = sum(nc$(ord(nc) EQ cp),backupp(nc));
                pti(ind) = no;
                ind(nc-1) = ind(nc);
                k = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));
                Level = Level - 1;
             );
             nodes(ind,'3') = 1;
             k = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));
             q(i)$(ORD(i) EQ k) = sum(nc$(ord(nc) EQ cq),backupq(nc));
             cq = cq - 1;

         else

            k = t;
            t = 0;
            f(i) = sum(n, a(i,n)*x.l(n));
            Loop(i$(Floor(f(i)) < f(i)), t = Ord(i));

            if (t = 0,
* solution is integer

               if (z.l < zopt, zopt = z.l; incumb(n) = x.l(n););

               while (sum((ind,m)$(ORD(m)EQ 3),nodes(ind,m)) = 1,
                  cp = cp - 1;
                  p(i)$(ORD(i) EQ k) = sum(nc$(ord(nc) EQ cp),backupp(nc));
                  pti(ind) = no;
                  ind(nc-1) = ind(nc);
                  k = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));
                  Level = Level - 1;
               );
```

```
                nodes(ind,'3') = 1;
                k = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));
                q(i)$(ORD(i) EQ k) = sum(nc$(ord(nc) EQ cq),backupq(nc));
                cq = cq - 1;

            else
* solution not integer;
* create new branching node:

                Level = Level + 1;
                ind(nc) = ind(nc-1);
                pti(ind) = yes;
                nodes(ind,'1') = t;
                nodes(ind,'2') = sum((i)$(ORD(i) EQ t),Floor(f(i)));
                nodes(ind,'3') = 0;
                cq = cq + 1;
            );
          );

    else
* prune by unfeasibeality, unbounding, etc.

       while (sum((ind,m)$(ORD(m)EQ 3),nodes(ind,m)) = 1,
          cp = cp - 1;
          p(i)$(ORD(i) EQ k) = sum(nc$(ord(nc) EQ cp),backupp(nc));
          pti(ind) = no;
          ind(nc-1) = ind(nc);
          k = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));
          Level = Level - 1;
       );
       nodes(ind,'3') = 1;
       k = sum((ind,m)$(ORD(m)EQ 1),nodes(ind,m));
       q(i)$(ORD(i) EQ k) = sum(nc$(ord(nc) EQ cq),backupq(nc));
       cq = cq - 1;
    );
);

if( Level > 0,
    display  'Too many iteration (500); ',
             'The best integer aproximation of the optimal solution is:',
             '    Objective function: ', zopt,
             '    Dessision varaibles: ', incumb;
else
  if (Level = 0,
       display  'Integer optimal sioltion is found;'
                '    Nodes: ' , nod,
                '    Objective function: ', zopt,
                '    Dessision varaibles: ', incumb;
  );
);
```

Appendix D

The output of the program Branching.gms

```
COMPILATION TIME     =        0.015 SECONDS      3 Mb  WIN222-145 Apr 21, 2006
GAMS Rev 145  x86/MS Windows                            09/22/06 12:55:06
Page 3
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----    182 Starting solution for given ILP:
                Objective function:
            VARIABLE z.L                =      324.500  objective variable
                Dessision varaibles:

----    182 VARIABLE x.L  shifts used

1  0.250,    2  1.000,    3  3.000,    4  1.000,    6  1.250,    8  0.250
10 1.000,    12 1.000,    13 1.250,    15 1.000,    16 2.000,    17 0.250




GAMS Rev 145  x86/MS Windows                            09/22/06 12:55:06
Page 4
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----    306 Integer optimal sioltion is found;
                Nodes:
            PARAMETER nod                   =       51.000
                Objective function:
            PARAMETER zopt                  =      332.000
                Dessision varaibles:

----    306 PARAMETER incumb

1  1.000,    2  2.000,    3  3.000,    6  2.000,    10 1.000,    12 1.000
13 2.000,    15 1.000



EXECUTION TIME       =        0.000 SECONDS      3 Mb  WIN222-145 Apr 21, 2006
```