Grammar-Based Column Generation for Personalized Multi-Activity Shift Scheduling

Marie-Claude Côté
Département de mathématiques appliquées et génie industriel,
École Polytechnique de Montréal, Montréal, Canada
Interuniversity Research Center on Enterprise Networks, Logistics and Transportation,
Montréal, Canada {macote@cirrelt.ca}

Bernard Gendron
Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada
Interuniversity Research Center on Enterprise Networks, Logistics and Transportation,
Montréal, Canada {bernard@cirrelt.ca}

Louis-Martin Rousseau
Département de mathématiques appliquées et génie industriel,
École Polytechnique de Montréal, Montréal, Canada
Interuniversity Research Center on Enterprise Networks, Logistics and Transportation,
Montréal, Canada {louism@cirrelt.ca}

We present a branch-and-price algorithm to solve personalized multi-activity shift scheduling problems. The subproblems in the column generation method are formulated using grammars and solved with dynamic programming. The expressiveness of context-free grammars is exploited to easily model restrictions over shifts, allowing the branch-and-price algorithm to solve large-scale problem instances. We present computational experiments on two types of multi-activity shift scheduling problems and compare our approach with existing methods in the literature. These experiments show that our approach can solve efficiently large-scale instances and is flexible enough to model different classes of problems.

Key words: shift scheduling, context-free grammars, column generation, branch-and-price.

---

1. Introduction

The task of scheduling personnel is, for many organizations, significant and complex. The size of the organization and the variety of the constraints arising from different aspects of the problem make it challenging to create an optimal schedule by hand. Several papers in the literature have addressed personnel scheduling problems. Some of these papers are dedicated to shift scheduling, a particular class of problems that deal with the assignment of work-activities, interspersed with breaks and meals, to a set of employees. Very few papers study shift scheduling problems where several work-activities must be scheduled and where

employees have different characteristics, notably, in the work-activities they can perform and in their availability. In most real-world shift scheduling problems, these aspects are present. In this paper, we study a class of shift scheduling problems that allow multiple work-activities and deal with non identical employees.

We define the *personalized multi-activity shift scheduling problem* as follows. Given a planning horizon I divided into n periods of equal length and a set of work-activities J, a *shift* is defined by its starting and ending times and by an assignment of work and rest-activities, such as breaks, to each period. Given a set of employees E, a set of feasible shifts Ω^e for each employee $e \in E$, and a number of employees b_{ij} required at each period $i \in I$ for each work-activity $j \in J$, one must select for each employee $e \in E$ a feasible shift in Ω^e to cover the required number of employees at minimum cost, given that each feasible shift $s \in \Omega^e$ has an associated cost $c_s^e \geq 0$. We assume that the employees have different characteristics from one another, such as skills that allow them to perform only a subset of work-activities, or restrictions regarding their availability during some periods of the planning horizon. The set of feasible shifts for each employee is therefore determined by the skills, preferences and availability of each employee, but is also constrained by other rules arising from work regulation agreements and ergonomic considerations. This problem differs from the multi-activity shift scheduling problem studied in Côté et al. (2010), where all employees are assumed to be identical.

The classical set covering model, proposed in Dantzig (1954) for the shift scheduling problem first described in Edie (1954), can easily be adapted to the personalized multi-activity shift scheduling problem to obtain model D:

$$
\begin{aligned}
f(D) = \min & \sum_{e \in E} \sum_{s \in \Omega^e} c_s^e x_s^e & & \\
& \sum_{e \in E} \sum_{s \in \Omega^e} \delta_{ijs}^e x_s^e \geq b_{ij}, & \forall i \in I, j \in J, & (1) \\
& \sum_{s \in \Omega^e} x_s^e = 1, & \forall e \in E, & (2) \\
& x_s^e \in \{0,1\} & \forall e \in E, s \in \Omega^e, & (3)
\end{aligned}
$$

where $\delta_{ijs}^e = 1$ if work-activity $j \in J$ is assigned to period $i \in I$ in shift $s \in \Omega^e$, and variable $x_s^e = 1$ if employee e is assigned to shift $s \in \Omega^e$. The number of variables in model D grows rapidly with the number of employees and the number of feasible shifts per employee, which makes impractical to solve the model by generating all feasible shifts *a priori*.

In this paper, we present a column generation approach for solving model D, where the pricing subproblem is modeled by using a context-free grammar that allows to extract the set of feasible shifts for each employee. The pricing subproblem is solved efficiently by a dynamic programming algorithm performed on a directed acyclic graph obtained from the context-free grammar. This column generation method is embedded within a branch-and-price (B&P) algorithm that can solve large-scale problem instances to near optimality. We show that this algorithm is competitive with other methods from the literature, while allowing enough flexibility to address a variety of multi-activity shift scheduling problems.

The paper is organized as follows. In Section 2, we present a literature review on shift scheduling problems and we introduce grammar theory. In Section 3, we present our solution approach, including the grammar-based pricing subproblem and the branching rule used in the B&P algorithm. In Section 4, we present comparative computational results on two types of multi-activity shift scheduling problems presented in the literature.

2. Background Material

In this section, we present a literature review on models and methods for shift scheduling problems, in particular multi-activity shift scheduling problems. We then provide a brief introduction to context-free grammars, describing only the concepts that are relevant to the present work.

2.1. Literature Review

The literature on personnel scheduling distinguishes two problems: *shift scheduling* and *tour scheduling*. In shift scheduling problems, one is interested in determining the assignment of employees to one or more work-activities, interspersed with breaks and meals, typically over a one-day planning horizon. In tour scheduling problems, complete schedules over several days have to be determined. Loucks and Jacobs (1991) and Ritzman et al. (1976) model the tour scheduling problem with assignment variables specifying the number of employees assigned to a given activity at any given time. Since such modeling approaches yield very large integer programming (IP) formulations, both papers propose heuristic methods. These models do not allow to place breaks or meals during the shifts, nor do they handle regulations concerning the transition between activities. The rest of this section focuses on shift scheduling problems and distinguishes between mathematical programming approaches and

constraint programming methods based on formal languages.

Mathematical programming models and approaches. For shift scheduling problems, two types of IP models, explicit and implicit, are considered in the literature. In an *explicit* model, one obtains the schedule for each employee simply by scanning the optimal solution, i.e., in a time linear to the model size; the classical set covering formulation D is an example of an explicit model. In an *implicit* model, a post-processing algorithm, typically efficient, but not linear in the model size, must be called upon in order to derive the schedule for each employee. The literature also distinguishes whether the problem has only one work-activity or multiple work-activities. In a *single-activity* shift scheduling problem, one only specifies, at each period, if an employee is working or not. In a *multi-activity* shift scheduling problem, there are several work-activities and whenever an employee is working at a given period, it is further necessary to specify which work-activity is assigned to that employee.

Problems involving a single work-activity and identical employees have been studied and efficiently solved to optimality using implicit models such as the ones suggested in Bechtolds and Jacobs (1990), Aykin (1996) and Rekik et al. (2004). Implicit models have much less variables than the explicit set covering model (i.e., model D with $|J| = 1$), since the latter counts one variable per feasible shift, while the former uses variables for types of shifts and types of breaks. Implicit models are limited in terms of the rules over shifts they can handle; in particular, we are aware of only one implicit model, the grammar-based model proposed in Côté et al. (2010), that can represent multi-activity shift scheduling problems. By contrast, the explicit set covering model (i.e., model D with an arbitrary set J) includes that case as well.

As mentioned above, the set covering model also has its limitations, since when the number of employees and the number of feasible shifts grow, the model can rapidly become intractable, unless a column generation approach, first introduced in Dantzig and Wolfe (1960), is used. Column generation is based on the idea that optimal solutions to large linear programs can be obtained without explicitly including all the columns (variables). The relevant columns can be generated dynamically by solving a so-called pricing subproblem (see for instance Desaulniers et al. (2005) and Lübbecke and Desrosiers (2005) and references therein for more details about column generation).

Up until now, very few papers addressed personalized multi-activity shift scheduling problems. Demassey et al. (2006) study a multi-activity shift scheduling problem on a 24-

hour planning horizon involving up to ten work-activities. The set covering model is handled with a column generation approach in which the pricing subproblem is solved with constraint programming. This method succeeds in finding optimal integer solutions for some instances involving up to three work-activities, but not for larger instances. Lequy et al. (2009) consider a personalized muti-activity shift scheduling problem where shifts and breaks are fixed *a priori*. This paper describes two IP models and a column generation approach based on multicommodity flow formulations. These approaches can deal with small instances. To solve large-scale instances, a rolling horizon heuristic based on column generation (first studied in Omari (2002), Vatri (2001) and Bouchard (2004)) is used. In our computational experiments reported in Section 4, we will use the problem definitions and the instances from Demassey et al. (2006) and Lequy et al. (2009).

Formal languages have been used to derive IP models for multi-activity shift scheduling problems. Côté et al. (2011) propose two explicit IP models based on formal languages. Both models make use of assignment variables y_{ij}^e indicating whether or not employee e is assigned to activity $j \in J$ at period $i \in I$. One IP model is based on a regular language represented by a finite deterministic automaton that allows to encapsulate the constraints defining each feasible shift using a network flow formulation. Another IP model makes use of a context-free grammar that describes all feasible shifts for each employee. These models can handle personalized shift scheduling instances, but since they generate a large number of variables, they can only solve instances with few work-activities. In addition, in the case where many employees are alike, these models have symmetry issues. To deal with these performance issues, an implicit IP model based on context-free grammars was suggested in Côté et al. (2010) for multi-activity shift scheduling problems where all employees are identical. This model makes use of assignment variables y_{ij} indicating the number of employees assigned to activity $j \in J$ at period $i \in I$. To the best of our knowledge, this model is the first to solve efficiently large-scale multi-activity problem instances with up to ten work-activities, but it cannot be extended to personalized multi-activity problem instances. The present work is an attempt to fill this gap. Although we also use context-free grammars as in Côté et al. (2011) and Côté et al. (2010), our work differs significantly since we use the set covering model instead of models based on assignment variables. Moreover, we solve the pricing subproblem with a specialized dynamic programming algorithm; an alternative, tested in preliminary experiments and shown to be inferior, would consist in solving the pricing subproblem as an IP model similar to the one presented in Côté et al. (2011). Finally, instead of using a

state-of-the-art IP software package to derive integer solutions, as in Côté et al. (2011) and Côté et al. (2010), we develop our own B&P implementation.

Constraint programming approaches based on formal languages. The idea of using formal languages to derive IP formulations was first inspired by constraint programming. As mentionned above, Demassey et al. (2006) use a constraint programming column generation subproblem to generate feasible shifts for a shift scheduling problem. The subproblem is modeled in part using an automaton and is solved by constraint programming techniques. Menana and Demassey (2009) use an automaton with additional constraints ensuring cumulative costs and succeed in finding the lowest cost schedule for one employee for up to 50 work-activities. Kadioglu and Sellmann (2010) use a context-free grammar to model a simplified version of the problem presented in Demassey et al. (2006), where the objective is to minimize the number of employees. They present an incremental arc-consistency algorithm for context-free grammars and find optimal solutions for instances with one and two work-activities in an average time of 9 seconds. Quimper and Rousseau (2010) suggest two ways to model the rules of the problem presented in Demassey et al. (2006): an expanded automaton and a context-free grammar, both including all the constraints over the composition of a shift. They then present neighborhood operators for both approaches for use in large neighborhood search heuristic methods. These approaches return near optimal solutions for one-activity instances and scale well on large instances containing up to ten work-activities.

Quimper and Rousseau (2010) also compared the interest of using either an expanded automaton or a context-free grammar to model the constraints in the composition of multi-activity shifts. As a general rule, they observed that the graph associated with the context-free grammar is of smaller size than the associated expanded automaton. They also point out that the scheduling rules are generally easier to model with a context-free grammar.

2.2. Definitions

Grammars, words and languages. A *context-free grammar* G is characterized by a tuple (Σ, N, P, S) where:

- Σ is an alphabet containing letters $(a, b, c, \ldots)$, also called terminal symbols;
- N is a set of non-terminal symbols $(A, B, C, \ldots)$;

- P is a set of productions of the form $X \rightarrow \alpha$, where $X \in N$ and α is a sequence of terminal and/or non-terminal symbols.
- S is the starting non-terminal.

A sequence of letters from alphabet Σ, called a *word*, is *recognized* by grammar G if it can be generated by successive applications of productions from G, starting with non-terminal S. The set of words recognized by a grammar is called a *language*.

In the following, we will use the term grammar to refer to a context-free grammar and we will assume that, except when specified otherwise, all grammars are in Chomsky normal form, meaning that all productions are of the form $X \rightarrow \alpha$ where $X \in N$ and $\alpha \in (N \times N) \cup \Sigma$. Note that this assumption is not restrictive since any context-free grammar can be converted to Chomsky normal form; we note, however, that such a conversion increases the size of the grammar, although in our case studies, reported in Section 4, we observe moderate increases in the size of the grammars when performing the conversion to Chomsky normal form. We refer to Hopcroft et al. (2001) for more information on formal languages.

Example 1 *The following grammar G defines all feasible shifts for a simple multi-activity shift scheduling problem. A shift must have a duration equal to the planning horizon and contain one break of one period anywhere during the shift except at the first or the last period. The problem contains two work-activities represented by letters j_1 and j_2 and break periods are represented by letter b. A break is mandatory to change from one work-activity to another.*

$G = (\Sigma = (j_1, j_2, b), N = (S, X, W, B, J_1, J_2), P, S)$, *where P is:*

$S \rightarrow WX, \qquad X \rightarrow BW,$

$W \rightarrow J_1J_1 \mid J_2J_2 \mid j_1 \mid j_2,$

$J_1 \rightarrow J_1J_1 \mid j_1,$

$J_2 \rightarrow J_2J_2 \mid j_2,$

$B \rightarrow b,$

*where the symbol $\mid$ specifies a choice of production. Assuming that the planning horizon has a duration of 4 periods, then the shifts $j_1bj_1j_1$, $j_2j_2bj_1$ and $j_1bj_2j_2$, among others, are recognized by G, while $j_1bj_1j_2$ is not. Word $j_1bj_2j_2$ is obtained by the derivation shown in Table 1, where **P** is the production used and **CS** is the current sequence, obtained from the previous sequence by applying the production on the left side.*

P	**CS**
–	S
$S \to WX$	WX
$X \to BW$	WBW
$W \to J_2J_2$	WBJ_2J_2
$W \to j_1$	$j_1BJ_2J_2$
$B \to b$	$j_1bJ_2J_2$
$J_2 \to j_2$	$j_1bj_2J_2$
$J_2 \to j_2$	$j_1bj_2j_2$

Table 1: Derivation of word $j_1bj_2j_2$ from grammar G of Example 1

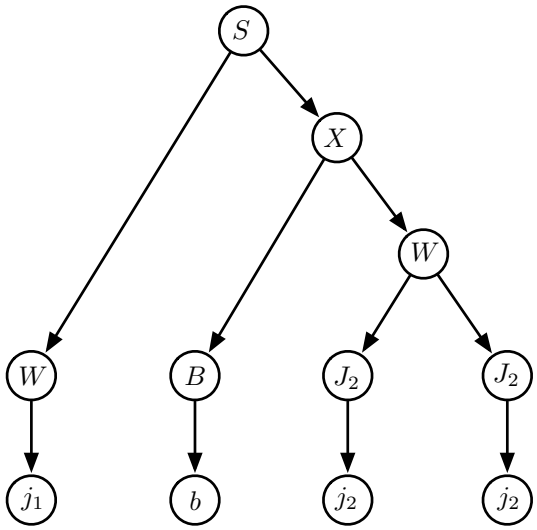

Figure 1: Parse tree for word $j_1bj_2j_2$ derived from grammar G of Example 1

Parse Trees. Any given word recognized by a grammar can be represented by a so-called *parse tree* for which the root node is associated with the starting non-terminal S and the leaves correspond to letters from Σ that form a word when listed from left to right. In a parse tree, the productions are represented as follows: an interior node represents a non-terminal on the left hand side of a production and its children are the non-terminals or the letters on the right hand side of the production. Figure 1 shows the parse tree associated with word $j_1bj_2j_2$ derived from grammar G of Example 1.

Grammar DAG. Quimper and Walsh (2007) suggest a way to generate a directed acyclic graph (DAG) embedding every parse tree corresponding to a word of length n recognized by a given grammar. The resulting DAG has two types of nodes: the or-nodes and the and-nodes. An or-node, associated with a non-terminal X, a position i and a length l, is the root of all the parse trees derived from X and giving a subsequence of length l starting at position i. An and-node represents a production starting with the non-terminal associated with its parent and resulting in a subsequence of length l starting at position i. Thus, any path in this DAG from the root-node to the leaves alternates between or-nodes and and-nodes. To derive any parse tree from the DAG, we start at the root-node. We visit an or-node by selecting exactly one child, which is necessarily an and-node. We visit an and-node by choosing all its children (exactly two if $l > 1$, one otherwise). By traversing the DAG in this way until the only remaining unvisited nodes are leaves, we obtain a parse tree associated to the word defined by the leaves. Conversely, starting from a given word ω, we can traverse the DAG backwards in a straightforward way to derive the parse tree associated to ω. In practice, the DAG is built by a procedure suggested in Quimper and Walsh (2007) inspired by an algorithm from Cooke, Younger, and Kasami (see Hopcroft et al. (2001)).

Figure 2 is the DAG derived from grammar G from Example 1 on words of length 4. The or-nodes are labeled O_{il}^{π} where π is a non-terminal or a letter associated with the node, while i and l are, respectively, the starting position and the length of the subsequence it produces. Usually, an and-node is labeled $A_{il}^{\Pi,t}$, where Π is a production $X \to \alpha$, i and l are, respectively, the starting position and the length of the subsequence generated from this production, and there is an index t for each such possible subsequence. To avoid overloading the figure, we did not label the and-nodes, which are simply illustrated with black dots, since it is easy to deduce the productions they represent from their parent and children nodes. For instance, the and-node having O_{12}^{W} as parent and $O_{11}^{J_1}$ and $O_{21}^{J_1}$ as children would be labeled $A_{12}^{W \to J_1 J_1,1}$.

3. Grammar-Based Column Generation Approach

In this section, we present the main ingredients of our grammar-based B&P algorithm, namely the restricted master problem and the pricing subproblem solved at each iteration of the column generation approach, as well as the branching rule used to produce integer solutions.

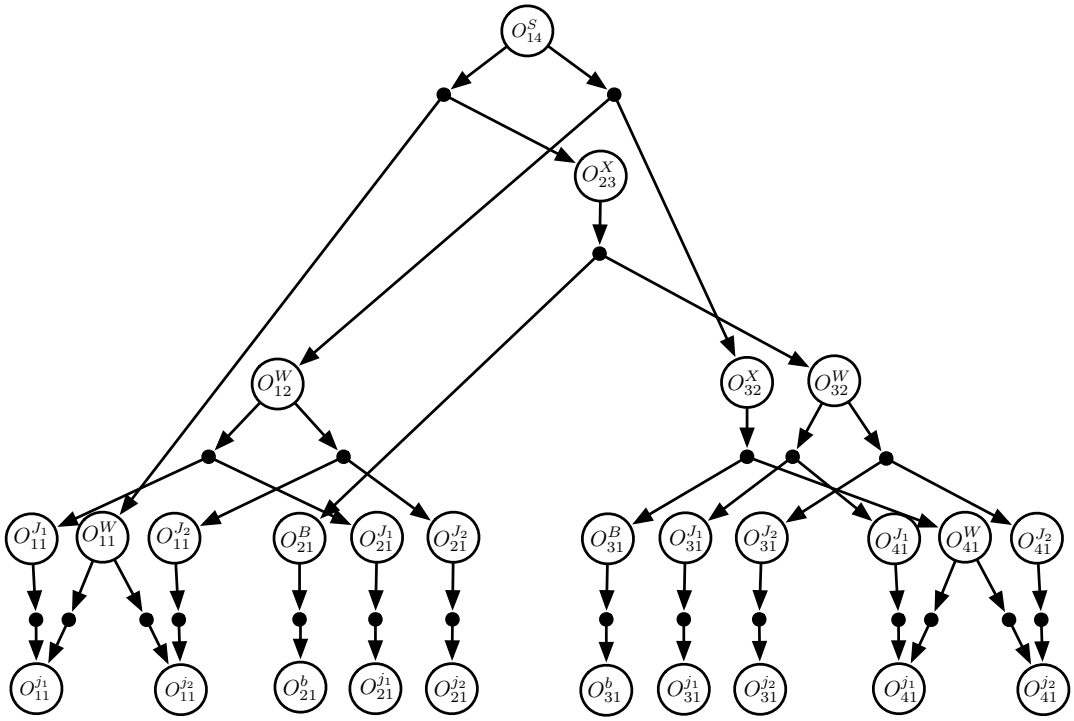

Figure 2: DAG for grammar G from Example 1 on words of length 4

3.1. Restricted Master Problem

At every node of the B&P tree, we solve the linear programming (LP) relaxations of a sequence of restrictions of model D, called the *restricted master problems*. Conceptually, each restriction is defined by allowing only a subset of the feasible shifts $\widetilde{\Omega}^e \subseteq \Omega^e$ for each employee $e \in E$. The sequence of restrictions is obtained by gradually enlarging the subsets of feasible shifts, which yields a sequence of non-increasing values that converges towards the optimal LP relaxation of the node. More precisely, at each iteration of the column generation method, we solve the current restricted master problem and then look for negative reduced cost columns, i.e., shifts in $\Omega^e \backslash \widetilde{\Omega}^e$ such that $\bar{c}^e_s < 0$ for each employee $e \in E$; these columns can be obtained by solving the pricing subproblem, which is described in the next subsection. If no negative reduced cost columns can be generated, the current restricted master problem is optimal and we have computed the LP relaxation of the node by generating only a (typically small) subset of the feasible shifts.

The restricted master problem, called RMP, solved at every iteration of the column

generation method performed at each node of the B&P tree, takes the following form:

$$f(RMP) = \min \sum_{e \in E} \sum_{s \in \widetilde{\Omega}^e} c_s^e x_s^e$$

$$\sum_{e \in E} \sum_{s \in \widetilde{\Omega}^e} \delta_{ijs}^e x_s^e \geq b_{ij}, \qquad \forall i \in I, j \in J, \tag{4}$$

$$\sum_{s \in \widetilde{\Omega}^e} x_s^e = 1, \qquad \forall e \in E, \tag{5}$$

$$x_s^e \geq 0 \qquad \forall e \in E, s \in \widetilde{\Omega}^e. \tag{6}$$

To define each node of the B&P tree, we forbid some shifts to be assigned to a particular employee, as described in Subsection 3.3. Conceptually, this simply amounts to define the restricted subsets associated to that employee in such a way as to remove the corresponding forbidden shifts (in practice, however, we will associate a very large cost to these forbidden shifts, as seen in Subsection 3.3). Thus, the restricted master problems at every node of the B&P tree always take the above form and no additional constraints are needed to capture the branching rules that define the node.

3.2. Pricing Subproblem

To generate new columns to be included to the current RMP, we solve one subproblem for each employee, based on the DAG described in Section 2.2. For each employee $e \in E$, we generate a grammar G_e that represents the shifts employee e can perform, according to the employee's skills, preferences and availability, and considering also general work regulations that apply to all employees. From this grammar, we generate the associated DAG that will be used to solve the pricing subproblem for employee e.

For the current RMP, let $\lambda_{ij} \geq 0$ be the dual variable associated with each constraint of type (4) and σ^e represents the (unrestricted) dual variable associated with each constraint of type (5). For the sake of clarity, we assume that the cost per shift can be decomposed by period and by work-activity as follows: $c_s^e = \sum_{i \in I} \sum_{j \in J_i^e} \delta_{ijs}^e c_{ij}^e$, where J_i^e is the set of work-activities employee e can perform at period i and c_{ij}^e is the cost for employee e to perform work-activity j at period i. At the end of this subsection, we discuss how we can generalize our approach to other, more realistic, cost structures. The reduced cost of column $s \in \Omega^e$ is then:

$$\bar{c}_s^e = \sum_{i \in I} \sum_{j \in J_i^e} (c_{ij}^e - \lambda_{ij}) \delta_{ijs}^e - \sigma^e \qquad \forall e \in E, s \in \Omega^e. \tag{7}$$

To solve the pricing subproblem for employee e, we associate a cost to each node of the DAG. To each leaf corresponding to a work-activity j at period i, we initialize its cost k_{ij} to $c^e_{ij} - \lambda_{ij}$. The other nodes of the graph have their cost initialized to zero. We solve each subproblem by a dynamic programming algorithm suggested in Katsirelos et al. (2008), Quimper and Rousseau (2010) and Kadioglu and Sellmann (2010) to find a minimum cost parse tree in a grammar-based DAG. The algorithm traverses the DAG from the leaves to the root by summing up the children of the and-nodes and by choosing the lowest cost children of an or-node (see the updating formulae below). As a result, every child of the root node with a negative value represents a negative reduced cost column that can be added to RMP. If no such child exists in any of the employee subproblems, no negative reduced cost column can be generated and the current RMP solution is the optimal LP solution.

To present the dynamic programming updating formulae, we use the notations introduced in Section 2.2 to refer to the or-nodes and the and-nodes of the DAG. Let $cost_O(N)$ and $cost_A(N)$ be, respectively, the costs associated to or-node $O(N)$ and to and-node $A(N)$; also, let $ch(N)$ be the children of or-node N. We then update the costs according to the following formulae:

$$cost_O(O^j_{i1}) = \begin{cases} k_{ij}, & \text{if } j \in J^e_i, \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

$$cost_O(O^\pi_{il}) = \min_{A^{\Pi,k}_{il} \in ch(O^\pi_{il})} \left\{ cost_A(A^{\Pi,k}_{il}) \right\}, \qquad l > 1, \tag{9}$$

$$cost_A(A^{\Pi:B \to j,1}_{i1}) = cost_O(O^j_{i1}), \tag{10}$$

$$cost_A(A^{\Pi:B \to CD,k}_{il}) = cost_O(O^C_{ik}) + cost_O(O^D_{i+k,l-k}), \qquad l > 1, \tag{11}$$

where B, C, D are non-terminal symbols of the grammar, C corresponding to a subsequence of length $k < l$.

Note that the assumption restricting the cost to be $c^e_s = \sum_{i \in I} \sum_{j \in J^e_i} \delta^e_{ijs} c^e_{ij}$ is not necessary for our algorithm to work, as we could have cost on any node of the DAG associated to the grammar. For instance, a cost could be assigned to every and-node associated with a production, representing a transition cost between different work-activities. In the dynamic programming algorithm, these and-nodes would be initialized to this transition cost and this cost would be added to the total cost of the node, when processed.

3.3. Branching Rule

Since the optimal solution to the final RMP at any node of the B&P tree is likely to be fractional, we need to perform branching in order to find an optimal IP solution to model D. Branching on individual x_s^e variables by generating the two nodes $x_s^e = 1$ and $x_s^e = 0$ is not a good idea: while the former node is easily dealt with, the latter cannot be easily handled in the pricing subproblem. Therefore, we must develop another type of branching rule that not only eliminates the current fractional solution, but that can also be easily processed by the dynamic programming algorithm used to solve the pricing subproblem.

We suggest the following rule, adapted from the B&P algorithm of Barnhart et al. (2000) for solving integer multicommodity flow problems. First, we select an employee e' such that there exists at least two associated variables having fractional values in the optimal LP solution. For employee e', we select the two shifts $s^{e'}(1)$ and $s^{e'}(2)$ corresponding to the associated variables with the highest fractional values. We then identify the first divergent position, meaning the first period at which shifts $s^{e'}(1)$ and $s^{e'}(2)$ differ in terms of their work-activities. If we denote by i' the first divergent position and by $j(1)$ and $j(2)$, respectively, the two work-activities assigned to $s^{e'}(1)$ and $s^{e'}(2)$ at period i', we then generate a partition of $J_{i'}^{e'}$ into two subsets $J_{i'}^{e'}(1)$ and $J_{i'}^{e'}(2)$ such that $j(l) \in J_{i'}^{e'}(l)$, for $l = 1, 2$. Apart from this rule involving $j(1)$ and $j(2)$, the other work-activities in $J_{i'}^{e'}$ are included arbitrarily in one of the two subsets, but in such a way that both subsets have the same number of elements (up to a difference of one). Finally, we generate two nodes in the B&P tree: each one forbids solutions where employee e' performs a work-activity in $J_{i'}^{e'}(l)$ at period i', for $l = 1, 2$. Note that $J_{i'}^{e'}$, the set of work-activities employee e' can perform at period i', can vary from one B&P node to another according to branching decisions taken higher up in the tree.

This rule ensures a well-balanced tree and can be easily handled in both the restricted master problems and the pricing subproblems. Indeed, it suffices to assign a very large value to the cost $c_{i'j}^{e'}$ associated to forbidden work-activities $j \in J_{i'}^{e'}(l)$, for $l = 1, 2$. This will effectively remove the corresponding shifts from RMP, thus eliminating the current fractional solution. When solving the pricing subproblem for employee e', this modification corresponds to assigning a very large cost to each leaf representing position i' and work-activities $j \in J_{i'}^{e'}(l)$, for $l = 1, 2$, which ensures that these leaves will not be selected by the dynamic programming algorithm.

In many problem instances, the beginning of the shifts, the shift lengths and the breaks

are not fixed *a priori*. In this case, the activities at the divergent position are not necessarily work-activities, but can also be rest-activities (rest, breaks or meals). The branching scheme must then be adapted by simply adding the rest-activities to the set of work-activities, so that large cost values can be assigned to the corresponding forbidden rest-activities, in both the restricted master problem and the pricing subproblem.

4. Computational Experiments

In this section, we compare our grammar-based B&P algorithm with existing methods, using the problem definitions and the instances from Demassey et al. (2006) and Lequy et al. (2009). For the two classes of problems, we describe the problem and the associated grammars, and we provide computational results comparing our approach with available results in the literature. The experiments on our B&P code were performed (in sequential) on a two-processor quad-core intel Xeon 2.4GHz with 48 GB RAM. To solve the restricted master problems, we use the barrier method in CPLEX 11.2, with all parameters kept at their default values. The branching rule and the column generation method at each node of the B&P tree, including the dynamic programming algorithm, are implemented in C++ and embedded in a B&P algorithm using the OOBB framework from Crainic et al. (2009).

4.1. Problem Instances From Demassey et al. (2006)

This section presents a shift scheduling problem for a retail store, allowing up to ten different work-activities. For each number of work-activities (1 to 10), ten instances are available. They differ in their demand curves, number of employees and costs. All employees are assumed to be identical; although our algorithm is specifically designed for personalized shift scheduling problems, it is important to show that it can be easily adapted to the case of identical employees, and still remain competitive with existing methods even in that case. We present the specifications of the problem and then compare our approach to the column generation method from Demassey et al. (2006) and to the formal language based models from Côté et al. (2011) and Côté et al. (2010) tested on these problem instances.

Problem Definition

Given:

- a 24-hour planning horizon divided into 15-minute periods;

- for each work-activity and each period:
  - the required number of employees;
  - unit undercovering and overcovering costs;
  - a cost to perform the work-activity at the given period;
- the number of employees;

the problem is to assign one shift to each employee such that:

- a shift may start at any period of the day allowing enough time to complete its duration during the planning horizon;
- a shift must cover between 3 hours and 8 hours of work-activities;
- if a shift covers at least 6 hours of work-activities, it must have two 15-minute breaks and a lunch break of 1 hour;
- if a shift covers less than 6 hours of work-activities, it must have one 15-minute break, but no lunch;
- if performed, the duration of a work-activity is at least 1 hour (4 consecutive periods);
- a break (or a lunch) is necessary between two different work-activities;
- work-activities must be inserted between breaks, lunch and rest stretches;

while minimizing:

- the total cost of the assigned shifts (the cost of a shift is the sum over all periods of the costs of all work-activities performed in this shift) and
- the total overcovering and undercovering costs.

Adapting the Solution Approach. Although all employees are identical, we could solve the above problem directly as a personalized multi-activity shift scheduling problem with our B&P algorithm. The performance of the algorithm would be seriously impaired for instances with a large number of employees. We therefore adapt our solution approach to address this issue. At the root node, we define a different restricted master problem, RMP',

that aggregates employees instead of considering each of them individually. More specifically, we define Ω to be the set of feasible shifts for any employee and $\delta_{ij}^s = 1$ if work-activity j is assigned to period i in shift $s \in \Omega$. Each variable x_s of the aggregated model represents the number of employees assigned to shift $s \in \Omega$. Finally, we introduce $\widetilde{\Omega} \subseteq \Omega$ the subset of allowed feasible shifts in RMP', which is defined as follows:

$$f(RMP') = \min \sum_{s \in \widetilde{\Omega}} (\sum_{i \in I} \sum_{j \in J} \delta_{ij}^s c_{ij}) x_s + \sum_{i \in I} \sum_{j \in J} (c_{ij}^+ s_{ij}^+ + c_{ij}^- s_{ij}^-)$$

$$\sum_{s \in \widetilde{\Omega}} \delta_{ij}^s x_s - s_{ij}^+ + s_{ij}^- = b_{ij}, \qquad \forall i \in I, j \in J, \quad (12)$$

$$\sum_{s \in \widetilde{\Omega}} x_s = \mid E \mid, \quad (13)$$

$$x_s \geq 0, \qquad \forall s \in \widetilde{\Omega}, \quad (14)$$

$$s_{ij}^+, s_{ij}^- \geq 0, \qquad \forall i \in I, j \in J, \quad (15)$$

where $\mid E \mid$ is the number of employees and c_{ij}, c_{ij}^+ and c_{ij}^- are, respectively, the cost, the unit overcovering cost and the unit undercovering cost associated to work-activity j at period i. Variables s_{ij}^+ and s_{ij}^- correspond, respectively, to the number of employees overcovering or undercovering the demand for work-activity j at period i. A similar model is used in the column generation approach of Demassey et al. (2006).

At the root node of the B&P tree, every iteration of the column generation method first consists in solving RMP' and then in searching for negative reduced cost columns by a single application of the dynamic programming algorithm of Section 3.2. At other nodes of the B&P tree, every iteration of the column generation approach proceeds as described in Section 3, using the restricted master problems of the form RMP, which are initialized by simply copying for each employee the set of columns generated at the root node. Observe that this approach differs significantly from the column generation algorithm in Demassey et al. (2006) which uses aggregated restricted master problems throughout the whole solution process.

Note that another way to deal with identical employees is to use symmetry breaking constraints in the model. However, this approach is difficult to track within a column generation framework. Also, to our knowledge, symmetry breaking during search (Gent et al. (2006)) was never adapted to IP solvers.

Definition of the Grammar. The following presents the grammar used for this problem.

Table 2: Average grammar ($\mid G \mid$) and DAG ($NbNodes$) sizes for the problem instances from Demassey et al. (2006) (10 per class defined by the number of activities, NbA)

NbA	$\mid G \mid$	$NbNodes$
1	28	24529
2	31	42968
3	34	67021
4	37	80006
5	40	96308
6	43	117019
7	46	129754
8	49	136466
9	52	148792
10	55	170208

For the sake of clarity, the grammar is not stated in Chomsky normal form.
$G = (\Sigma = (a_j \ \forall j \in J, r), N = (S, F, H, W, A_j \ \forall j \in J, B, L, R), P, S)$,
where a_j is a period assigned to work-activity $j \in J$ and r is a period assigned to any rest-activity (rest, lunch or break). In P, defined as follows, the notation $\rightarrow_{[min,max]}$ is used to restrict the subsequences generated with a given production to have a length between min and max periods:

$$
\begin{array}{ll}
S \rightarrow RFR \mid FR \mid RF \mid RHR \mid HR \mid HP & B \rightarrow r \\
F \rightarrow_{[30,38]} WBWLWBW \mid WLWBWBW \mid WBWBWLW & L \rightarrow rrrr \\
H \rightarrow_{[13,24]} WBW & R \rightarrow Rr \mid r \\
W \rightarrow_{[4,\infty)} A_j \quad \forall j \in J & \\
A_j \rightarrow A_j a_j \mid a_j \quad \forall j \in J &
\end{array}
$$

For each class of ten instances characterized by the number of activities, NbA, Table 2 presents the average size of the grammar in Chomsky normal form, namely $\mid G \mid$, the number of productions in a grammar G and $NbNodes$, the number of nodes in the corresponding DAG.

Computational Results. We first compare our approach on the instances described above with the results reported in Demassey et al. (2006). Experiments in Demassey et al. (2006) were run on an Opteron 250. Table 3 presents average statistics on the ten classes of instances (a class of instances contains the ten instances with the same number of work-activities).

NbA is the number of work-activities in the class of instances. The *Root node* columns display *Time(s)*, *NbIts* and *NbCols*, respectively, the CPU time in seconds, the number of column generation iterations and the number of columns generated to solve the LP relaxation at the root node. In bold are the average times for classes of instances for which our approach is strictly faster at the root node. The *Branch-and-Price* columns show the following statistics: $NbS(0.01\%)$, the number of instances (out of ten) solved within a 0.01% relative gap (i.e., $Gap = 100(Z^u - Z^l)/Z^u$ where Z^u and Z^l are, respectively, the best upper and lower bounds) within a CPU time limit of 2 hours; *Time(s)*, the average CPU times for the instances solved within a 0.01% relative gap; $NbS(1\%)$, the number of instances solved within a 1% relative gap within a CPU time limit of 2 hours. Note that Demassey et al. (2006) do not report any gaps. In bold are the classes of instances for which the number of instances solved within a 0.01% relative gap is strictly higher with our approach.

The *Root node* columns are a good indication of the performance of the subproblems. These results clearly show that our algorithm can solve the root node LP relaxation faster, in fewer iterations and with less columns than the method of Demassey et al. (2006). In addition, as demonstrated by the Brnach-and-Price columns, our branching rule also performs better on these instances, allowing our B&P algorithm to solve to near optimality instances with up to ten work-activities within 2 hours of CPU time. We observe that for the majority of the 46 instances that are not solved to the 0.01% gap tolerance, the gaps are quite small at the end of the 2h CPU time limit. Indeed, 36 instances have a gap between 0.01% and 1% at the end of the time limit.

The next table compares our B&P algorithm to other approaches based on formal languages. Table 4 displays the CPU times for the one- and two-activity instances (ten in each class, *No.* being the instance number) for the explicit formulations from Côté et al. (2011) based on automata (*IP R M*) and grammars (*IP G M*), for the implicit grammar-based model (*IG M*) from Côté et al. (2010), and for our B&P algorithm (*G-B CG*). The CPU times (in seconds) are the times to reach a 1% relative gap. The notation $> 1h$ means that the instances could not be solved to the gap tolerance within the 1-hour time limit. In bold are the times for which our approach is strictly faster then the three other approaches. Experiments in Côté et al. (2011) were run on a 2.4 GHz Dual AMD Opteron Processor 250 with 3 GB of RAM using CPLEX 10.0. Experiments in Côté et al. (2010) were performed on a 2.3GHz AMD Opteron with 3GB of memory using CPLEX 10.1.1.

These results show that the grammar-based column generation method is competitive

Table 3: Comparison with Demassey et al. (2006) approach

	Root node			Branch-and-Price		
NbA	$Time(s)$	$NbIts$	$NbCols$	$NbS(0.01\%)$	$Time(s)$	$NbS(1\%)$
Grammar-based CG						
1	**0.1**	10	257	5	62	10
2	**0.2**	16	601	6	100	9
3	**0.6**	20	975	**6**	2074	8
4	**1.1**	25	1321	**5**	2096	9
5	**3.0**	46	2321	0	$> 2h$	10
6	**4.0**	47	2457	**9**	915	10
7	**6.6**	47	3117	**5**	2426	9
8	**8.3**	62	3459	**7**	2163	10
9	**9.6**	62	3626	**5**	1886	7
10	**9.9**	43	3405	**6**	3754	8
Demassey et al. (2006) CG						
1	0.4	19	889	8	144	–
2	3.7	48	2340	8	394	–
3	2.0	52	2550	4	1592	–
4	12.5	103	5063	0	$> 2h$	–
5	6.2	86	4288	0	$> 2h$	–
6	13.8	130	6493	0	$> 2h$	–
7	18.4	137	6839	0	$> 2h$	–
8	25.4	155	7736	0	$> 2h$	–
9	25.9	155	7741	0	$> 2h$	–
10	42.0	179	8974	0	$> 2h$	–

Table 4: Comparison between approaches based on formal languages on instances with one and two work-activities-CPU time(s)

No.	G-B CG	IP R M	IP G M	IG M
One work-activity				
1	**0.02**	1.03	7.42	0.26
2	373.01	40.09	$> 1h$	110.88
3	**3.32**	64.64	$> 1h$	75.25
4	**1.78**	46.39	1850.38	2.75
5	**0.13**	14.03	322.57	0.48
6	**0.03**	3.28	130.21	0.34
7	**1.53**	5.99	1662.75	2.71
8	**30.75**	131.77	$> 1h$	2642.12
9	1.87	16.14	1015.10	1.18
10	0.84	20.22	1313.28	0.80
Two work-activities				
1	**0.27**	228.07	2826.4	1.27
2	**3.51**	2870.20	1952.58	4.12
3	**13.63**	1541.15	$> 1h$	81.91
4	25.66	169.96	$> 1h$	16.27
5	**0.32**	$> 1h$	$> 1h$	2.59
6	**6.98**	1288.56	$> 1h$	51.16
7	1.88	29.94	$> 1h$	0.60
8	**23.30**	$> 1h$	325.08	36.20
9	$> 1h$	$> 1h$	$> 1h$	$> 1h$
10	**0.97**	1108.23	$> 1h$	4.99

with the existing approaches based on formal languages, at least for problem instances with up to two work-activities. While it can solve to near optimality many instances involving up to ten work-activities, as shown in Table 3, our B&P algorithm is generally outperformed by the implicit grammar-based model from Côté et al. (2010) on instances from three to ten work-activities. This is only mildly surprising, since the implicit grammar-based model from Côté et al. (2010) totally avoids symmetry issues and does not suffer from the growth in the number of employees, contrary to our B&P algorithm. The implicit grammar-based model cannot, however, deal with *personalized* multi-activity shift scheduling problems. We now present computational results on such problems.

4.2. Problem Instances From Lequy et al. (2009)

This section presents computational results on a personalized multi-activity shift scheduling problem introduced by Lequy et al. (2009) under the name *multi-activity assignment problem*. Two sets of instances are available for this problem: results on the first set are published in Lequy et al. (2009), while results on the second set are still unpublished, but were made available to us by the authors. These results are obtained from experiments performed on an $IntelCore^{TM}2\ CPU$ 6700 clocked at 2.66GHz with 4GB RAM using the Xpress-MP solver.

Problem Definition

Given:

- a planning horizon divided into 15-minute periods;
- for each work-activity;
  - the required number of employees at each period;
  - the undercovering and overcovering costs;
  - its minimum and maximum durations;
- for each available employee;
  - the list of pre-assigned work-pieces (a work-piece is defined by a starting time and a duration);
  - the list of work-activities for which the employee is qualified;

the problem is to fill each work-piece with a sequence of activities such that:

- each employee can only be assigned to work-activities for which she is qualified;
- the minimum and maximum work-activity durations are satisfied;

while minimizing:

- the total undercovering and overcovering costs and
- the total transition costs (a cost is associated to every transition from one work-activity to another within a work-piece).

Definition of the Grammar. The following presents the grammar used for this problem for a given employee e and a given work-piece p. For the sake of clarity, as before, the grammar is not stated in Chomsky normal form.
$G^{e,p} = (\Sigma = (a_j \ \ \forall j \in J_e), N = (S, \left\{A_j, A_j^n, A_j'\right\} \ \ \forall j \in J_e), P, S)$,
where J_e is the set of work-activities for employee e and a_j is a period assigned to work-activity $j \in J_e$. To define P, we use the following notations: $\rightarrow_{[min,max]}$ restricts the subsequences generated with a given production to have a length between min and max periods; l_{ep} is the length of work-piece p for employee e; min_j and max_j are, respectively, the minimum and maximum durations of work-activity j. P is then defined as follows:

$$
\begin{array}{lll}
\forall j \in J_e: & & \\
\quad & S \rightarrow_{[l_{ep},l_{ep}]} A_j A_j^n & \\
\quad & S \rightarrow_{[l_{ep},l_{ep}]} A_j & \text{if } l_{ep} \leq max_j \\
\quad & A_j \rightarrow_{[min_j,max_j]} A_j' & \\
\quad & A_j^n \rightarrow A_{j'} A_{j'}^n \mid A_{j'} & \forall j' \in J_e \setminus \{j\} \\
\quad & A_j' \rightarrow A_j' a_j \mid a_j &
\end{array}
$$

For each class of instances characterized by the triplet $D/E/A$ representing, respectively, the number of days (D), employees (E) and activities (A), Table 5 presents $\mid G^{e,p} \mid$, the average number of productions in a grammar $G^{e,p}$ in Chomsky normal form and $NbNodes$, the average number of nodes in the corresponding DAG.

Computational Results. On the first set of instances, we compare our grammar-based column generation (*G-B CG*) method with three approaches from Lequy et al. (2009): two models solved exactly by the IP solver Xpress-MP and the rolling horizon heuristic method

Table 5: Average grammar ($\mid G^{e,p} \mid$) and DAG ($NbNodes$) sizes for the problem instances from Lequy et al. (2009) (each class counts 5 instances characterized by $D/E/A$, D = number of days, E = number of employees, A = number of activities)

$D/E/A$	$\mid G^{e,p} \mid$	$NbNodes$
First set		
7/20/5	60	7716
1/50/10	50	5471
7/50/7	202	19310
2/75/12	112	11722
Second set		
7/20/5	24	2528
1/50/10	31	2501
7/50/7	40	3956
7/100/15	127	9158

based on column generation (*Horizon CG*). The first model is a multicommodity network flow model (*MC model*), while the second is a reformulation of the first that yields fewer variables (*Block model*). The heuristic method is based on a rolling horizon framework where each time slice is solved with a column generation approach based on a shortest path subproblem. Integer solutions are found with a rounding procedure that iteratively fixes variables to integer values, each time reoptimizing the resulting LP relaxation by column generation.

Table 6 presents the comparative times and solution values for our B&P algorithm and the two exact approaches from Lequy et al. (2009) on the smallest instances, where $No.$ is the instance number and $Value$ is the value of the solution found by the associated approach in $Time(s)$ seconds. The notation $> 1h$ means that the optimality could not be proved within the 1-hour CPU time limit. In these cases, $Value$ is the best integer solution found within this time limit. In bold are the times for which our approach is strictly faster than the two other approaches. For the largest instances, Lequy et al. (2009) only reports times in seconds to find the first integer solution with the *Block model*. Table 7 compares these times with the time necessary for the B&P algorithm to find the first integer solution. For our approach, we also present the value of the first integer solution (Val) found and the relative gap between this solution and the best solution found for this instance ($Gap(\%) = 100(Z_F - Z_B)/Z_F$, where Z_F is the first integer solution found and Z_B is the best solution reported in Table

Table 6: Comparison with exact methods on the smallest instances of the first set of instances of Lequy et al. (2009) problem

	G-B CG		MC model		Block model	
No.	$Value$	$Time(s)$	$Value$	$Time(s)$	$Value$	$Time(s)$
7 days, 20 employees, and 5 activities						
1024	7220	**14**	7220	114	7220	17
1773	6345	**9**	6345	50	6345	17
2732	7420	94	7420	200	7420	21
4657	6400	2591	6400	129	6400	135
5553	7535	**21**	7535	92	7535	34
1 day, 50 employees, and 10 activities						
1808	3270	$> 1h$	3250	2681	3250	1190
5066	2440	947	2440	716	2440	294
5135	2580	103	2580	155	2580	98
5226	2725	**34**	2725	107	2725	93
8854	2800	$> 1h$	2740	839	2740	321

8). The value of the first integer solution found with the *Block model* is not reported in Lequy et al. (2009). In bold are the times for which our approach succeed in finding the first integer solution faster. Table 8 presents the comparative results between our approach and the *Horizon CG* method on all the instances. To perform a fair comparison with the heuristic method, we stopped the B&P algorithm when a solution within a 1% relative gap was found. $> 1h$ means that this gap could not be achieved within the 1-hour CPU time limit. In these cases, $Value$ is the best integer solution found within the time limit. In bold we highlight the times and values for the instances where our approach is strictly better than *Horizon CG*. Note that the values in Lequy et al. (2009) differ from the ones reported here, because Lequy et al. (2009) multiply the undercovering and overcovering costs for a period by 15, the number of minutes in a period. The solutions obtained are, however, exactly the same.

The results presented in Table 6 show that our approach is generally competitive with the *MC model* and the *Block model* on smaller instances, except for a few instances, where it is clearly outperformed. For larger instances, however, our method finds a first integer solution much faster than the *Block model*, as shown in Table 7. The first integer solution found by our method is also of extremely good quality. In Table 8, when compared to the

Table 7: Comparison with the *Block model* to find the first integer solution on the largest instances of the first set of instances of Lequy et al. (2009) problem

No.	G-B CG			Block model
	$Time(s)$	Val	$Gap(\%)$	$Time(s)$
7 days, 50 employees, and 7 activities				
1007	**1313**	14115	0.00	16108
156	**1787**	13420	0.00	21271
237	**1439**	13610	0.15	16050
4369	**1530**	13675	0.00	25029
5216	**1811**	14800	0.00	13020
2 days, 75 employees, and 12 activities				
1855	**1504**	6325	2.21	18685
2106	**947**	6525	0.00	15400
2435	**734**	6050	0.00	3511
4225	**864**	6270	0.24	18584
9863	**624**	5870	0.00	19216

heuristic method *Horizon CG*, our method is not competitive in terms of CPU times, but for 16 out of 20 instances, it finds a better solution, still in reasonable time.

Table 9 compares our approach with the *Horizon CG* heuristic method on the second set of instances. As in Table 8, $No.$ is the instance number and $Value$ is the value of the solution found by the associated approach in $Time(s)$ seconds. For the *G-B CG* approach, we stopped when a solution within a 1% relative gap was found. No time limit was applied for these instances. As before, we highlight in bold the times and values for the instances where our approach is strictly better than *Horizon CG*.

On the second class of instances, as the results in Table 9 show, our B&P algorithm generally outperforms *Horizon CG*, not only in solution quality, but also in CPU times. Indeed, for all instances, the B&P algorithm finds solution values equal or better than those obtained by method *Horizon CG* and it does so in less time for 17 out of the 20 instances.

When comparing the last two tables, we observe that our results on the two sets of instances are very contrasting. Indeed, we observe two main differences between the two sets of instances. First, each employee has more skills in the first set than in the second set, so each of them is allowed to perform almost all work-activities. Second, the work-pieces are, on average, of larger size in the first set than in the second set. These two characteristics yield a greater number of feasible shifts for each employee in the first set than in the second set.

Table 8: Comparison with the *Horizon CG* approach on the first set of instances of Lequy et al. (2009) problem

	G-B CG		Horizon CG	
No.	$Value$	$Time(s)$	$Value$	$Time(s)$
7 days, 20 employees, and 5 activities				
1024	**7220**	11	7265	10
1773	**6360**	9	6440	9
2732	7420	19	7420	17
4657	**6400**	2003	6415	11
5553	7600	15	7550	15
1 day, 50 employees, and 10 activities				
1808	**3270**	$> 1h$	3315	86
5066	**2440**	935	2565	237
5135	**2580**	**8**	2595	18
5226	**2725**	**8**	2740	19
8854	2800	$> 1h$	2770	70
7 days, 50 employees, and 7 activities				
1007	**14115**	1321	14265	363
156	**13420**	1793	13570	612
237	13610	$> 1h$	13590	333
4369	**13675**	1536	13890	473
5216	**14800**	1824	15040	543
2 days, 75 employees, and 12 activities				
1855	6265	$> 1h$	6185	1068
2106	**6525**	$> 1h$	6630	878
2435	**6050**	$> 1h$	6090	252
4225	**6255**	$> 1h$	6410	655
9863	**5870**	$> 1h$	6035	388

Table 9: Comparison with the *Horizon CG* approach on the second set of instances of Lequy et al. (2009) problem

	G-B CG		Horizon CG	
No.	$Value$	$Time(s)$	$Value$	$Time(s)$
7 days, 20 employees, and 5 activities				
1024	2940	**0.80**	2940	1.99
1773	2770	**0.76**	2770	2.17
2732	3820	**0.54**	3820	1.37
4657	3210	**0.61**	3210	2.21
5553	3270	**0.59**	3270	1.51
1 day, 50 employees, and 10 activities				
342	**1875**	**26.70**	1950	55.52
369	**2315**	146.08	2360	23.59
71	2050	**1.26**	2050	5.37
737	**2065**	53.23	2105	16.43
869	**1875**	**41.67**	1890	107.33
7 days, 50 employees, and 7 activities				
5600	**8440**	**59.49**	8500	98.05
592	**7345**	**33.03**	7375	130.09
8597	**7645**	**31.58**	7705	126.58
9445	7900	**14.67**	7900	42.42
949	**8155**	**44.19**	8185	132.59
7 days, 100 employees, and 15 activities				
530	**15200**	**5818.36**	15275	6667.41
1024	**15420**	**4602.99**	15690	6889.64
2596	15855	10806.10	15855	5805.99
6384	**15250**	**2064.81**	15400	4461.14
7862	**15940**	**1391.95**	16030	3545.54

Our B&P is clearly sensitive to the number of feasible shifts per employee. Also, since each employee can perform almost all work-activities in the first set of instances, the employees are almost identical, which yields symmetry issues.

4.3. Summary

The previous sections presented computational results comparing our B&P algorithm with different approaches from Demassey et al. (2006), Côté et al. (2011), Lequy et al. (2009) and Côté et al. (2010).

From a modeling point of view, the grammar-based column generation approach is generic, as it can handle a variety of multi-activity shift scheduling problems, in particular personalized instances. By contrast, the column generation method of Demassey et al. (2006) and the implicit grammar-based model of Côté et al. (2010) can also easily model multi-activity instances, but the former was not extended to the personalized case, while the latter simply cannot be extended to that case. We have also shown that the use of grammars can easily adapt to the context introduced in Lequy et al. (2009), where the breaks, the shift beginnings and the shift lengths are known *a priori*. The models and methods in Lequy et al. (2009) were developped precisely for this problem and cannot be adapted easily to another context.

From a computational point of view, the grammar-based column generation approach is flexible enough to solve efficiently a variety of problem instances. Indeed, the B&P algorithm provides integer solutions of good quality in reasonable computational time for almost all tested instances. In particular, on personalized problem instances from Lequy et al. (2009), it is competitive with a specialized heuristic method, showing superior performance on some classes of instances.

Grammar-based modeling can, however, yield a very large DAG. Indeed, the number of nodes in the graph associated with a grammar G can be, in the worst case, $O(n^3 \mid G \mid)$ where n is the sequence length and $\mid G \mid$ is the number of productions in grammar G. Tables 2 and 5 show that the DAG size in our test cases are moderate and that performances do not suffer from this aspect.

5. Conclusion

In this paper, we presented a B&P algorithm to solve different types of personalized multi-activity shift scheduling problems. The algorithm solves the LP relaxation of the classical set covering formulation with column generation. The pricing subproblem is modeled with a context-free grammar and solved with a dynamic programming algorithm based on traversing the DAG associated to the grammar. The B&P algorithm also integrates a branching rule that preserves the structure of the pricing subproblem. Our computational experiments show that the B&P algorithm is competitive with existing approaches from the literature. Furthermore, it is flexible enough to address different types of problems. This characteristic is mostly due to the expressiveness of grammars that enables to encode a large set of rules over shifts.

There are, however, some limitations in the rules that can be expressed with a grammar. For instance, in a case where it is possible, within a shift, to work for a consecutive number of periods, rest for an unspecified number of periods and then work again, it would be difficult with a context-free grammar to limit the total number of working periods. This situation also occurs when planning a complete week of work where beginnings and ends of shifts are not fixed *a priori*. To deal with this issue, one could use side constraints with a grammar-based IP subproblem (see Côté et al. (2011)) to capture, in this case, the total number of working periods. Another possibility is to model the subproblem as a constraint programming satisfaction problem, such as in Fahle et al. (2002), to consider constraints that cannot be dealt with by a grammar alone. As future work, it would be interesting to study the introduction of resources within the subproblem dynamic programming algorithm; such resources would allow, for instance, to count the number of working periods.

Acknowledgments

We wish to thank three anonymous referees whose comments have helped us to improve the paper. This work was supported by a grant from the Fond québécois de recherche sur la nature et les technologies. We would like to thank the following individuals: François Guertin for allowing us to use the OOBB framework and for his help during the developpement of the algorithm; Serge Bisaillon for his support on many technical aspects throughout the progress of this work; Quentin Lequy for providing us with his two sets of instances and his related

results.

References

Aykin, T. 1996. Optimal shift scheduling with multiple break windows. *Management Science* **42** 591–602.

Barnhart, C., C.A. Hane, P.H. Vance. 2000. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research* **48** 318–326.

Bechtolds, S., L. Jacobs. 1990. Implicit optimal modeling of flexible break assigments. *Management Science* **36** 1339–1351.

Bouchard, M. 2004. Attribution des activités aux employés travaillant sur des quarts. M.Sc. Thesis, Ecole Polytechnique de Montréal.

Côté, M.-C., B. Gendron, C.-G. Quimper L.-M. Rousseau. 2011. Formal languages for integer programming modeling of shift scheduling problems. *Constraints* **16** 54–76.

Côté, M.-C., B. Gendron, L.-M. Rousseau. 2010. Grammar-based integer programming models for multi-activity shift scheduling. Tech. rep., Publication CIRRELT-2010-01.

Crainic, T.G., A. Frangioni, B. Gendron, F. Guertin. 2009. Oobb: An object-oriented library for parallel branch-and-bound. Presented at the CORS/INFORMS International Conference, Toronto, Canada, June 14-17 2009.

Dantzig, G. 1954. A comment on Edie's traffic delay at toll booths. *Journal of the Operations Research Society of America* **2** 339–341.

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

Demassey, S., G. Pesant, L.-M. Rousseau. 2006. A cost-regular based hybrid column generation approach. *Constraints* **11** 315–333.

Desaulniers, G., J. Desrosiers, M. M. Solomon. 2005. *Column Generation.*. New York, NY : Springer.

Edie, L. 1954. Traffic delays at toll booths. *Journal of the Operations Research Society of America* **2** 107–138.

Fahle, T., U. Junker, S. E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. 2002. Constraint programming based column generation for crew assignment. *Journal of Heuristics* **8** 59–81.

Gent, I., W. Harvey, T. Kelsey. 2006. Groups and constraints: Symmetry breaking during search. *Principles and Practice of Constraint Programming - CP 2002*, vol. 2470. 233–240.

Hopcroft, J., R. Motwani, J. D. Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley.

Kadioglu, S., M. Sellmann. 2010. Grammar constraints. *Constraints* **15** 117–144.

Katsirelos, G., N. Narodytska, T. Walsh. 2008. The weighted cfg constraint. *Proc. 5th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - CPAIOR'08, LNCS 5015*. 323–327.

Lequy, Q., M. Bouchard, G. Desaulniers, F. Soumis. 2009. Assigning multiple activities to work shifts. Tech. rep., Les Cahiers du GERAD G-2009-86.

Loucks, J.S., F.R. Jacobs. 1991. Tour scheduling and task assignment of a heterogeneous work force: a heuristic approach. *Decision Sciences* **22** 719–739.

Lübbecke, M. E., J. Desrosiers. 2005. Selected topics un column generation. *Operations Research* **53** 1007–1023.

Menana, J., S. Demassey. 2009. Sequencing and counting with the multicost-regular constraint. *Proc. 6th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - CPAIOR'09, Springer-Verlag LNCS 5547*. 178–192.

Omari, Z. 2002. Optimisation des pauses dans le problème de fabrication des horaires avec quarts de travail. M.Sc. Thesis, Ecole Polytechnique de Montréal.

Quimper, C.-G., L.-M. Rousseau. 2010. A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics* **16** 373–392.

Quimper, C.-G., T. Walsh. 2007. Decomposing global grammar constraint. *Proc. of CP'07, Springer-Verlag LNCS* **4741** 590–604.

Rekik, M., J.-F. Cordeau, F. Soumis. 2004. Using benders decomposition to implicitly model tour scheduling. *Annals of Operations Research* **128** 111–133.

Ritzman, L., L.J. Krajewski, M.J. Showalter. 1976. The disaggregation of aggregate manpower plans. *Management Science* **22** 1204–1214.

Vatri, E. 2001. Integration de la génération de quart de travail et de l'attribution d'activités. M.Sc. Thesis, Ecole Polytechnique de Montréal.