Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Column generation based approaches for a tour scheduling problem with a multi-skill heterogeneous workforce

A B S T R A C T

In this paper, we address a multi-activity tour scheduling problem with time varying demand. The objective is to compute a team schedule for a fixed roster of employees in order to minimize the over-coverage and the under-coverage of different parallel activity demands along a planning horizon of one week. Numerous complicating constraints are present in our problem: all employees are different and can perform several different activities during the same day-shift, lunch breaks and pauses are flexible, demand is given for 15 minutes periods. Employees have feasibility and legality rules to be satisfied, but the objective function does not account for any quality measure associated with each individual's schedule. More precisely, the problem mixes simultaneously days-off scheduling, shift scheduling, shift assignment, activity assignment, pause and lunch break assignment.

To solve this problem, we developed four methods: a compact Mixed Integer Linear Programming model, a branch-and-price like approach with a nested dynamic program to solve heuristically the subproblems, a diving heuristic and a greedy heuristic based on our subproblem solver. The computational results, based on both real cases and instances derived from real cases, demonstrate that our methods are able to provide good quality solutions in a short computing time. Our algorithms are now embedded in a commercial software, which is already in use in a mini-mart company.

Keywords:
Employee scheduling
Integer programming
Branch-and-price
Heuristics

1. Introduction

Employee scheduling is an important issue in retail
, as personnel wages account
for a large part of their operational costs. This problem raises considerable computational difficulties, especially when certain factors are considered, such as employee availability, fairness, strict labor rules, highly variable work demand, mixed full and part-time contracts, etc. Since the seminal work of Dantzig , a large quantity of research papers have developed models and methods to assist managers and planners in their employee scheduling tasks (more than 300 papers published between 2004 and 2012 were surveyed in Van Den Bergh, Beliën, De Bruecker, Demeulemeester, & De Boeck). For a comprehensive literature review of classical studies on this problem, we refer to.

In this paper, we study a real-life multi-activity tour scheduling problem with highly heterogeneous employees and flexible working hours. Given a fixed set of employees, the objective is to construct their work schedule or planning that minimizes the distance to the ideal coverage of the demand. Numerous complicating factors described in the literature are taken into account and, to the best of our knowledge, this paper is one of the first attempts (in parallel with Restrepo, Gendron, & Rousseau, 2015) to combine days-off scheduling, shift scheduling, shift assignment, activity assignment, pause and lunch break assignment.

Several features of our problem are still considered as major issues in the recent literature :
individual constraints and flexibility of employees, integrated days-off, shift scheduling and assignment and multi-activity

assignment.
Although
the *lunch break assignment* between two timeslots is taken into account in most research papers, *pause assignment* during activities themselves remains a gap in the academic literature
To our knowledge, only deals with both types of breaks at the same time.

Although integer linear programming (ILP) models exist for this family of problems, they cannot be used directly to solve large scale problems with many constraints. Therefore, several works propose heuristics based on those ILP models to reduce their computational burden. Heuristic methods can be obtained by applying a hierarchical decomposition. First,
good shifts are computed, and then employees are assigned to the shifts in a second phase. Unfortunately, this technique cannot be applied directly to our problem, where each employee can change activity during his shift and has his very specific features such as availabilities, skills and pre-assignments. When the time horizon is large, and the problem can be solved for a smaller time horizon (typically one week) without risking infeasibilities for the planning, an interesting approach is
to use a rolling horizon heuristic, where the problems related to smaller time horizons are solved in an iterative manner. In our problem, the total number of worked hours for each employee is fixed, which may lead such method to unfeasible schedules.

Many algorithms for solving such employee scheduling problems are based on the column-generation approach.
. Recent papers address shift or
tour scheduling problems with branch-and-price methods. Boyer, Restrepo et al. use branch-and-price to solve
very general multi-activity shift scheduling problems. Their approaches rely on the description of shifts using a context-free grammar. Another recent work on the subject was realized by Brunner and Stolletz . They use an ad-hoc branch-and-price
method to solve a tour scheduling problem. The main ingredients of their approach are the use of variables related to day-shifts, which are recombined in the master problem, and stabilization strategies to reduce the number of column generation iterations. Another recent work uses branch-and-price
in the context of employee-scheduling. They use a nested dynamic programming approach, which is well-suited to the structure of their problem.

Our approach is also based on a branch-and-price algorithm. However, the problem settings do not allow us to use directly the algorithms from
In our problem, each employee is different, the time
horizon is much larger than the ones in Boyer et al.
, and many constraints restrict the construction of the
shifts. This leads to a prohibitively large pricing problem solution time. Since our aim is to handle real-life instances, we had to use a heuristic version of the branch-and-price, where some constraints are treated heuristically in the subproblem. The hierarchical structure of our shifts called for an ad-hoc specific nested dynamic program , which proves to be much more
efficient than a straightforward dynamic programming approach.

An important practical requirement is to find a good solution in a short amount of time (a few seconds for 100 employees). To respect this time limit, we designed a greedy algorithm based on our dynamic program. Also, a diving heuristic is proposed for cases when we have several minutes of computational time. Our algorithms have been implemented and are now embedded in a commercial software. They are able to find feasible solutions with good quality in a small or reasonable time for all test cases that were provided by our industrial partner. Our algorithms are now in use in a mini-mart company.

In Section 2, we describe formally our problem. Our column generation framework is presented in Section 3, followed by the nested dynamic program used to solve the pricing problem in Section 4. Our heuristic algorithms based on column generation are presented in Section 5, while computational experiments on real and generated instances are reported in Section 6.

2. Problem description

The problem consists in scheduling a fixed workforce to maximize the fit to a given time-varying demand. The planning horizon consists of $\mathcal{D}$ consecutive days. Each day is divided into the same number of successive time periods of equal length (15 minutes in this paper). Set $\mathcal{T}$ represents the different time periods in the discrete planning horizon. The set of heterogeneous employees is denoted by $\mathcal{E}$.

The whole set of activities that employees can carry out is divided into two distinct groups: *production activities* $\mathcal{A}$, related to work demands, and *pause activities* $\mathcal{P}$, related to non-productive activities. In our retail context, a production activity can represent, for example, the welcome desk, a cash desks line or a meat counter. Each employee $e \in \mathcal{E}$ has a set of production activities $\mathcal{A}(e,t)$ that he/she can perform at time period t. Set $\mathcal{P}(e,t)$ contains a pause if employee can take it at time period t; this set is empty otherwise. The beginning and the length of a pause are strictly constrained by the personalized pause policy of the company agreement. An employee e is unavailable at time period t if $\mathcal{A}(e,t) \cup \mathcal{P}(e,t) = \emptyset$. In this case, the planning computed for employee e cannot contain any activity at time t. Note that if an employee is unavailable the entire day, then a day-off has to be scheduled. Some employees may be pre-assigned to activities for certain time periods. In this case, finding a schedule that respects this pre-assigned tasks is a part of the problem.

The work demand $\mathrm{DE}_{a,t}$ represents the ideal number of employees needed to realize production activity a in the best possible conditions during time period t (see the representation given in Fig. 1). Satisfying exactly the demand is not mandatory : in most cases it is not possible. In this case, either an under-coverage, or an over-coverage is produced. Furthermore, if over-coverage (respectively under-coverage) exceeds the given threshold $\mathrm{OV}_{a,t}$ (respectively $\mathrm{UN}_{a,t}$), then it becomes critical and indicates that too many (respectively too few) employees have been assigned to activity a during time period t.

Our objective is to construct a feasible team schedule that minimizes the sum of the over-coverage and under-coverage costs for the whole planning horizon and all production activities.

2.1. A hierarchical structure of a team schedule

A feasible solution follows a hierarchical structure (see Fig. 2). For each level of the hierarchy, there is an associated set of constraints. This flexible structure does not rely on the use of a precomputed day-shift or individual planning library, since the number of possibilities is far too large.

- A *team schedule* consists of a set of $|\mathcal{E}|$ valid employee plannings.
- An *individual planning* for employee e is a set of successive *day-shifts* and *days-off* over a week. Two consecutive day-shifts are separated by a *rest break*.
- A *day-off* represents a special day when employee e does not participate in any activity. Deciding whether or not an employee takes a day-off is part of the optimization process (but some days-off are mandatory if the employee is unavailable).

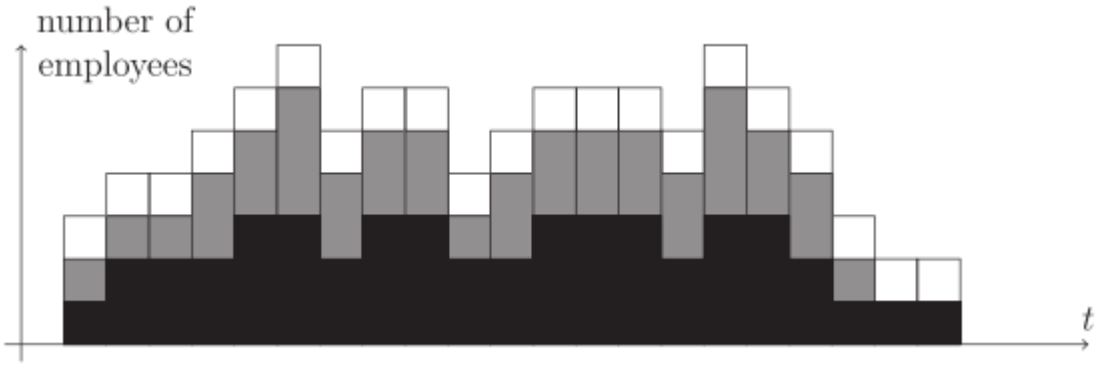

Fig. 1. Representation of the workload for a production activity : the ideal number of employees required to cover the demand is in gray, the thresholds of critical undercoverage and overcoverage are given respectively in black and white.

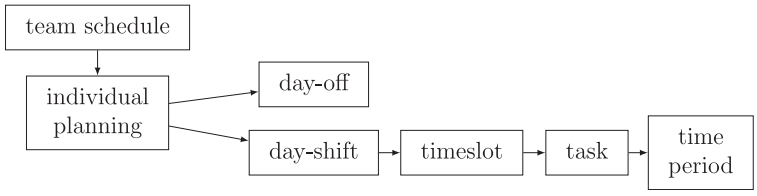

Fig. 2. Hierarchical structure of a team schedule.

- A *day-shift* consists of one *timeslot* or two timeslots separated by a *lunch break*.
- A *timeslot* is a non-empty sequence of *tasks* where different activities are carried out successively and continuously. Two consecutive tasks cannot be related to the same activity. The set of possible beginning times of all timeslots of employee e is denoted as $\mathcal{B}_e$. This set contains disjoint intervals, some of them are for the first timeslot of a day, others are for the second.
- A *task* is a time interval where a single activity a is performed over contiguous time periods. Activity a can be either a production activity or a pause.

Example 1. For a given day, an employee works from 8.00 AM to 12.30 AM during his first timeslot, then takes a one-hour lunch break, and finally does his second timeslot from 2.00 PM to 5.00 PM. During the first timeslot, three tasks are performed : from 8.00 to 9.00 in activity a, then from 9.00 to 11.00 in activity b, and finally from 11.00 to 12.30 in activity c. His second timeslot is devoted to the single task with activity b. According to the pause policy, a single pause is assigned from 9:00 AM to 9.15 AM during the first timeslot.

2.2. Planning constraints

In this paper, we take into account constraints that we have encountered in real-life customer contexts. Each employee has his own set of planning constraints and each constraint has its own parameters.

At each level of the team schedule hierarchy, duration and numerical constraints have to be satisfied. In Table 1, we list these constraints grouped by levels of the hierarchy. Note that *duration* of entities possibly include breaks (pauses and lunches), whereas *working time* equals to the "net duration" that excludes the breaks. Furthermore, an important feature in this problem is that each employee has a target of weekly working time LE_e that must be met exactly.

We stress the fact that each employee is different: he/she has his own skills, potential pre-assignment and availability for each time period, etc. A day-shift designed for an employee e is not likely to be valid for another employee e'.

Table 1
Planning constraints over an horizon of one week.

A task of employee $e \in \mathcal{E}$ performing activity $a \in \mathcal{A}$
Duration
A timeslot of employee $e \in \mathcal{E}$ beginning at time b
Beginning time b
Finishing time
Number of tasks
A day-shift of employee $e \in \mathcal{E}$ on day d
Beginning time
Finishing time
Working time
Duration
Number of timeslots
Rest (lunch) duration
Between timeslots
Minimum working time of
At least one timeslot
A weekly individual planning of employee $e \in \mathcal{E}$
Target working time
Number of day-shifts
Number of consecutive
day-shifts
Rest duration between
Consecutive day-shifts

2.3. Pause assignment policy

There are numerous pause assignment policies in practice. In this work, we use the following rules. First, pauses are not included in the working time. There is at most one pause assigned per timeslot. The pause is assigned if and only if the duration of the timeslot is at least four hours (including the pause duration). A pause must be located in the second third of its timeslot, and its duration is exactly one time period. Some pauses can be initially set at some time periods as pre-assignment constraints.

In our settings, each pause is positioned inside an existing task k. The two parts of task k before and after the pause are considered as a unique task, *i.e.* the two constitute a single task with one beginning and one end. Note that pauses are different from lunch breaks in our models: a lunch break separates the day-shift into two timeslots.

3. Our column generation approach

The Dantzig–Wolfe decomposition is well adapted to our scheduling problem, since it consists of disjoint subproblems (one per employee) that are linked by demand constraints. Similar to Dantzig , the *subproblem* for employee e consists in designing a *valid individual planning* respecting the

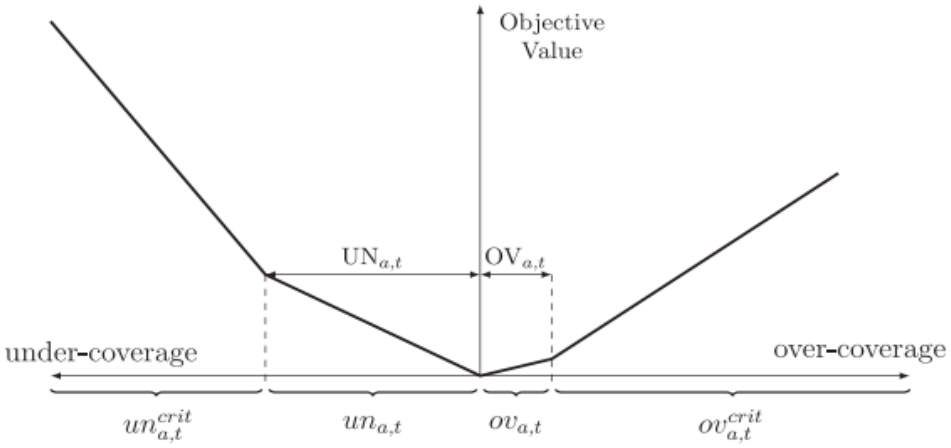

Fig. 3. Piecewise-linear objective function for a given production activity and time period.

specific set of constraints of employee e, but disregarding the requirements dealing with the others plannings. The *master problem* combines the employee plannings (columns) to minimize the total cost of over-coverage and under-coverage.

Another version of the set-covering model for the tour scheduling was proposed by Stolletz. Instead of using variables representing plannings, the author uses (day-)shift variables that are combined in the master problem to form valid plannings. Our problem settings do not allow easy recombinations of shifts: all employees are different and therefore each planning is associated to exactly one employee, and the total number of working periods in a planning is a fixed parameter. In our model, we keep the original planning variables, similar to what is done in Dantzig .

3.1. Master problem

Let $\mathcal{X}(e)$ denote the set of individual plannings (or columns) for employee e and $\mathcal{C}(e)$ its column index set: $\mathcal{X}(e) = \{X_c\}_{c\in\mathcal{C}(e)}$. Each column X_c is represented by a vector $[x_{c,a,t}]_{t\in\mathcal{T},a\in\mathcal{A}}$ where:

$$x_{c,a,t} = \begin{cases} 1 & \text{if employee is assigned to activity } a \text{ at time} \\ & \text{period } t \text{ in planning } c, \\ 0 & \text{otherwise.} \end{cases}$$

A binary variable q_c, $c \in \mathcal{C}(e)$, $e \in \mathcal{E}$, determines whether individual planning X_c is chosen for employee e. Continuous variables $ov_{a,t}$, $un_{a,t}$, $ov^{crit}_{a,t}$, and $un^{crit}_{a,t}$, $t \in \mathcal{T}$, $a \in \mathcal{A}$, represent, respectively, over-coverage, under-coverage, critical over-coverage, and critical under-coverage of the demand of activity a at time period t.

The cost function is piecewise linear and its structure is represented in Fig. 3. It depends on slack variables related to demand constraints. For a given solution $\{q_c : c \in \mathcal{C}(e), e \in \mathcal{E}\}$, for a given production activity a and a time period t, the coverage of the demand can be computed as $\mathrm{DE}_{a,t} - \sum_c x_{c,a,t} q_c$. We distinguish over-coverage $ov_{a,t}$ (resp. under-coverage $un_{a,t}$) from *critical* over-coverage $ov^{crit}_{a,t}$ (resp. *critical* under-coverage $un^{crit}_{a,t}$) that occurs when the over-coverage (resp. under-coverage) is greater than $\mathrm{OV}_{a,t}$ (resp. $\mathrm{UN}_{a,t}$). When critical over/under-coverage is reached, a larger unit cost has to be paid.

The master problem can be formulated as follows:

The piecewise objective function (1) minimizes the total cost of over-coverage and under-coverage over the planning horizon and production activities. Constant values $\mathrm{CO}_a \in \mathbb{R}_+$ and $\mathrm{CU}_a \in \mathbb{R}_+$ represent, respectively, the unitary costs of over-coverage and under-coverage for production activity a. Constant values $\mathrm{CO}^{\mathrm{crit}}_a \in \mathbb{R}_+$ and $\mathrm{CU}^{\mathrm{crit}}_a \in \mathbb{R}_+$ represent respectively the costs of critical over-coverage and under-coverage for production activity a. Critical over-coverage and critical under-coverage have larger costs: $\mathrm{CU}_a < \mathrm{CU}^{\mathrm{crit}}_a$ and $\mathrm{CO}_a < \mathrm{CO}^{\mathrm{crit}}_a$.

Constraints (2) link the decision variables and calculate the gap between the produced work and the work demand $\mathrm{DE}_{a,t}$ for each time period and each production activity. Constraints (3) assign exactly one individual planning to each employee e.

3.2. Pricing subproblems

The pricing problem decomposes into $|\mathcal{E}|$ independent subproblems (one for each employee). Let $[\pi_{a,t}]_{t\in\mathcal{T},a\in\mathcal{A}}$ be the dual values related to master problem constraints (2) and $[\pi_e]_{e\in\mathcal{E}}$ be the dual values related to master problem constraints (3). The subproblem for employee e consists in finding a feasible individual planning (denoted by vector $X = [x_{a,t}]_{t\in\mathcal{T},a\in\mathcal{A}}$) with the minimum reduced cost. We use variables $x_{a,t}$, which will be used to construct the constant column descriptors $x_{c,a,t}$ in the master problem. Recall that $\mathcal{X}(e)$ denotes the set of individual planning (or columns) for employee e. The subproblem for employee e can be stated as follows.

3.3. Acceleration strategies

Acceleration techniques are key elements for the efficiency of our column generation approach. Several papers list strategies for this purpose, and more specifically for employee scheduling problems in Brunner and Stolletz . We used the following strategies.

1. Instead of adding one column with the best reduced cost, we add to the restricted master problem several negative reduced cost columns at each iteration. Practically speaking, at each iteration of the column generation method, we add the best column found for each employee if it has a negative reduced cost. This means that at most $|\mathcal{E}|$ columns are added at each iteration. This method dramatically decreases the number of column generation iterations.
2. After solving a restricted master problem, if the number of variables exceeds a given threshold (more than 5000 columns in practice), then we delete all variables with a reduced cost exceeding 10^{-12}.
3. The lagrangian lower bound is computed at each iteration to stop the algorithm earlier if this bound and the solution value of the restricted master are equal. The lagrangian lower bound is computed as follows. Let $OPT(RMP)$ be the optimum of the current reduced master problem and $RC(SP_e)$ be the best reduced cost of a variable generated by subproblem e at the current iteration. The lagrangian lower bound is equal to $OPT(RMP) - \sum_{e\in\mathcal{E}} RC(SP_e)$.

We have also tried to apply dual price smoothing stabilization in order to accelerate column generation. However, it did not have a clear positive impact on the solution time. Note that reports very good speed-ups from stabilized column generation in a branch-and-price algorithm for a similar shift scheduling problem. We conjecture that the explanation for this different stabilization impact comes from the presence of the total working time constraint in our variant of the problem. Preliminary experiments showed that the dual price smoothing stabilization improves a lot the column generation solution time once this constraint is removed.

We also tried to solve only one pricing subproblem at each step (by considering only one employee), or to add only the column of best reduced cost among all subproblems. In both cases, the method was less efficient. This can be explained by the fact that solving the master problem takes a large amount of time. Moreover, several subproblems are solved in parallel, which helps reducing the time spent to solve all subproblems.

4. A nested dynamic program for the pricing subproblem

A pricing subproblem corresponds to finding the best individual planning for one employee according to his set of constraints. In this section, we discuss two possible ways from the literature to formulate this problem. Then we present our nested dynamic programming algorithm.

4.1. Limits of the resource constrained shortest path formulation

The pricing subproblem can be formulated as a resource-constrained shortest path problem (RCSPP) in a directed acyclic graph (DAG). In this DAG, each arc is characterized by a cost to use it and a set of resource consumptions while each node is characterized by a position in time and an amount of resource consumption already used for each resource. The objective is to find a path from a source node to a sink node that minimizes the overall cost and satisfies the resource consumption bounds. In Engineer, Nemhauser, and Savelsbergh , the authors present an exact dynamic programming algorithm based on relaxations and alternated forward and backward searches to solve shortest path problems involving a huge number of local resource constraints. This algorithm is much more efficient when only upper bounds are considered. When both lower and upper bounds coexist, the dominance relations, used to reduce enumeration, are weaker. Another recent work propose an exact method capable of handling large-scale networks in a reasonable amount of time.

In our problem, we have a large number of lower and upper bounds for the resource consumption, and some arc costs are negative. This weakens considerably the dominance rules used in the solution methods for the resource-constraint shortest path problem. Preliminary experiments confirmed that this approach was not efficient for our problem.

4.2. Limits of the grammar-based formulation

The structure of individual plannings makes the subproblem suitable for a solution method that uses context-free grammars, like it is done in Boyer et al. for a shift scheduling problem (horizon with a single day) and in Restrepo et al. for a tour scheduling problem (horizon of 7 days). Namely, for each employee $e \in \mathcal{E}$ we can define a grammar which describes the set of all valid plannings for e. Based on this grammar, a directed acyclic hyper-graph (called graph with or-nodes and and-nodes in can be constructed. Every unit flow in this hyper-graph defines a feasible individual planning for e. So the search for an individual planning with the best reduced cost can be done using a dynamic programming algorithm that seeks a min cost unit flow in the hyper-graph.

We have performed preliminary experiments with this approach and obtained the following results. A considerable number of bound constraints and a long time horizon result in a huge hyper-graph. Therefore, the construction of this hyper-graph takes a large amount of time, making it impossible to embed the grammar-based dynamic program in a fast heuristic. Moreover, even if the graph is constructed, this algorithm takes too much time to be called at every column generation iteration to solve the subproblem.

Therefore, we designed a nested dynamic programming algorithm. In order to reduce its running time, we heuristically remove some states, as it is explained below.

4.3. A nested dynamic programming algorithm

The specific structure of our problem leads to the following observations.

- There are a large number of resource constraints, but only a subset of them are active at a given node.
- Many paths share identical subpaths. Due to the hierarchical structure of the planning, the best day-shift for a given day is likely to be used in many non-dominated partial solutions.

This led us to design an alternative approach based on a nested dynamic program. A relevant and similar approach is described by Dohn and Mason to find the best individual plannings in a nurse rostering problem by using 3 levels and 2 segmentations. We call segmentation the phase where levels k and $k-1$ are combined. If the number of levels is z, then the number of segmentations should be $z-1$. In the first segmentation, the method combines day-shifts to design the best feasible sequence, this sequence is completed at the end with days-off to find the best feasible *blocks of workdays*. In the second segmentation, it combines the block of workdays to get the best individual planning.

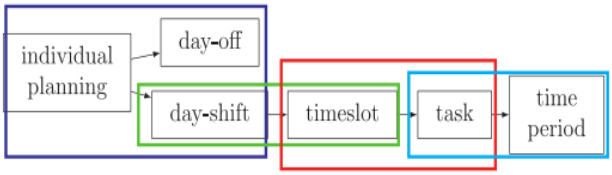

Fig. 4. Nested dynamic programming segmentation.

We have adapted the nested method to the specific features of our problem by using 5 levels and 4 segmentations.
For each employee e, we build an individual planning $X_c \in \mathcal{X}(e)$ by combining day-shifts constituted by one or several timeslots, themselves composed of tasks. To manage easily path dominance rules and symmetries, the dynamic programming algorithm is segmented into several sub-problems according to the hierarchical structure of the planning. At each level, the design of a given entity consists in combining the valid entities of the level immediately below.

The bottom-up presentation of the method consists in calculating the best reduced costs of the following entities: task, timeslot, day-shift and individual planning.

4.3.1. Reduced cost of a task

For an employee $e \in \mathcal{E}$, let $\alpha_e(b, f, a)$ be the reduced cost of the task in which the employee starts activity $a \in \mathcal{A}$ at period b, and finishes it at period f. Note that this task is valid for the employee, if it respects the duration bounds and employee skills and pre-assignments. We set the reduced cost of an invalid task to $+\infty$. Then, the formula for the reduced cost calculation is:

4.3.2. Reduced cost of a timeslot

For an employee $e \in \mathcal{E}$, let $\bar{\beta}_e(b, f, n, a)$ be the best reduced cost of a partial timeslot, which starts at period b, finishes at period f, contains a sequence of n consecutive tasks, the first of which does not perform activity a. The following recursion formula is used for the reduced cost calculation:

We denote $\bar{\beta}_e(b, d, n, -)$ the best reduced cost without imposing the constraint on the first task activity:

Let now $\hat{\beta}_e(b, f)$ be the best reduced cost of a complete timeslot, which starts at period b and finishes at period f. Note that this timeslot is valid for the employee, if its starting time is in $\mathcal{B}_e$, it respects the completion and duration bounds, and the bounds on the number of tasks it contains. We set the reduced cost of an invalid timeslot to $+\infty$. Then, the formula for the reduced cost calculation is:

Note that at this moment the pause policy may not be respected, as until now pauses are not included in timeslots. After calculating values $\hat{\beta}$, every timeslot without a pause and lasting more than four hours is replaced by one timeslot with a pause. For practical purposes, this is done in a greedy manner: we put the pause to a period in the second third of the timeslot such that its reduced cost is minimized. The pause replaces the corresponding work period such that the duration of timeslot is not increased. Note that this greedy approach for inserting pauses makes the whole dynamic programming procedure heuristic (sub-optimal solutions may be generated).

If a pre-assigned pause is contained inside a timeslot, and it is not positioned in the second third of it, such a timeslot is declared invalid, and its cost is set to $+\infty$. The same happens if the employee cannot take any pause ($\mathcal{P}(e, t)$ is empty for all time moments in the second third of the timeslot).

Let $\beta_e(b, f)$ be the best reduced cost of a timeslot, which starts at period b, finishes at period f, and respects the pause policy. Let $\ell(b, f)$ be this timeslot's working time, which can be uniquely determined from its duration $(f - d + 1)$ according to the pause policy.

4.3.3. Reduced cost of a day-shift

For an employee $e \in \mathcal{E}$, let $\delta_e^{\kappa}(d, b, f, \ell)$ be the best reduced cost of a day-shift of day d that starts at period b, completes at period f, contains κ timeslots and ℓ working periods. A valid day-shift should satisfy starting, completion, working time bounds and the daily pre-assignments. Let set Ω_{ed} contain the set of valid triples (b, f, ℓ):

The formula for the day-shift containing one timeslot is:

The formula for the day-shift containing two timeslots separated by a lunch break is:

The best reduced cost $\delta_e(d, b, f, \ell)$ of a day-shift with one or two timeslots can now be computed :

4.3.4. Reduced cost of an individual planning

In this step, we seek the best combination of day-shifts and days-off that designs a valid individual planning for employee e given its total working time LE_e and its number of day-shifts in $[\mathrm{NE}_e^-, \mathrm{NE}_e^+]$. This is also called a *tour scheduling problem* for a single employee.

For an employee $e \in \mathcal{E}$, let $\bar{\eta}_e^0(d, n, \ell)$ be the best reduced cost of a partial employee planning for the first d days, which contains n day-shifts and ℓ working periods, and ends with a day-off. Let also $\bar{\eta}_e^1(d, f, n, \ell)$ be the best reduced cost of a partial employee planning for the first d days, which contains n day-shifts and ℓ working periods, and ends at period f with a day-shift. These reduced costs are calculated using the following recursions. We set

where $\hat{\eta}_e^0(d, f, n, \ell)$ is the best reduced cost with the condition that the employee had a day-off on day $d-1$, and $\hat{\eta}_e^1(d, f, f', n, \ell)$ is the best reduced cost with the condition that the employee had a day-shift of day $d-1$ finishing at time f'

During this step, the algorithm deals also with the maximum number of successive day-shifts without day-off. In recursion (15), $\bar{\eta}_e^1(d, f, n, \ell)$ should be written $\bar{\eta}_e^1(d, f, n, m, \ell)$, where m is the number of consecutive day-shifts ending at day d such that $m \leq \mathrm{ME}_e^+$. However, we decided to omit the full recursion for the sake of simplification.

The best reduced cost η_e of an individual planning for employee e can be computed using the following formula:

4.3.5. Accelerating the algorithm heuristically

Our pricing algorithm is already a heuristic because of the simplified handling of the pauses. We now introduce a slight restriction of the state space, which also makes the method heuristic. As was mentioned above, the nested dynamic programming algorithm takes too much time because of a large number of states. In order to accelerate the algorithm, we heuristically delete some states. Namely, the set of states $\{\delta_e(d, b, f, \ell)\}_{\forall d, b, f, \ell}$ is reduced to the set of states $\{\delta_e(d, b, \ell)\}$ in the following way:

5. Column-generation based algorithms

At the end of the column generation method, the obtained solution may be non-integer. To get a good integer solution, we use an enumerative branch-and-price like method. By abuse of language, we will use the term branch-and-price even if the pricing subproblem is solved heuristically. Our branch-and-price is not always able to terminate within the time limit. Since our algorithms are designed for practical use, we also propose two different heuristics to find good solutions in less time.

5.1. Branch-and-price algorithm

Our branching scheme consists in fixing a variable $x_{c,a,t}$ for all candidate columns X_c related to a given employee e. In the formulation, this branching is accomplished as follows:

- $x_{c,a,t} = 0$ forbids employee e to be assigned to activity a at time period t. We delete all columns X_c, $c \in \mathcal{C}(e)$, in which activity a is performed during period t. In the pricing subproblem for employee e, the corresponding transition is forbidden.
- $x_{c,a,t} = 1$ assigns employee e to production activity a at time period t. We delete all columns X_c, $c \in \mathcal{C}(e)$, in which activity a is not performed at period t. The subproblem for employee e is modified by assigning a very large negative cost to the corresponding transition.

When branching, we choose the triplet (employee e, production activity a and time period t) which is the most fractional, *i.e.* for which $|0.5 - \sum_{c \in \mathcal{C}(e)} x_{c,a,t} q_c|$ is minimum (where q_c is determines whether individual planning X_c is chosen for employee e, as defined above). We use a depth-first strategy to explore the search tree. We tried different strategies (sometimes mixed together): branching on the slack variables (under or over-coverage variables), or branching on entities (forcing/forbidding an employee to work during a given day or time-slot). However, we do not have convincing results which show an advantage of these strategies over the scheme above.

The time allowed at each node of the branch-and-bound tree was limited to one hour. In rare cases, this results in premature termination of column generation at some nodes. In that case, our heuristic branch-and-price continues and carries out its branching strategy.

The heuristic dynamic programming algorithm for the subproblem makes our branch-and-price algorithm also heuristic. This means that theoretically there exist test instances for which the solution found is not optimal even after termination of the branch-and-price. However, for our test instances that are solved both by the branch-and-price and the MIP solver applied to the compact formulation, the obtained solution values are equal.

5.2. Diving heuristic

The diving heuristic is an algorithm in which the branch-and-price tree is searched partially
As usually done in diving heuristics, we use a different branching strategy for the diving, as the goal here is not to have a balanced search tree, but a good feasible solution quickly. As suggested in Joncour et al., at each node, after the termination of column generation, we select and fix a column $X_{c'}$, *i.e.* we select a complete planning for employee e' such that $c' \in \mathcal{C}(e')$. After that, all columns X_c, $c \in \mathcal{C}(e')$, are excluded from the problem, demands $\mathrm{DE}_{a,t}$ are updated according to the fixed partial solution. Then, the subproblem for employee e' is not called in descendant nodes.

As no backtracking occurs, the method stops after at most $|\mathcal{E}|$ nodes. In our algorithm, we select the column related to the variable with the largest value in the solution of the master problem.

To obtain a fast heuristic, we introduce the time limit for column generation at each node of the diving heuristic. When this time limit is reached, we use the current master solution

values for fixing the next column, even if this solution is not optimal.

5.3. Greedy heuristic based on the nested dynamic program

When the time limit is set to a handful of seconds, the diving heuristic may not be able to terminate. We propose a simple heuristic based on our pricing subproblem to find good solutions in a small amount of time. In this heuristic, the employee plannings are still computed by our dynamic program, but the plannings are individually generated one by one and added iteratively to the solution. Here, the objective function of the subproblems is based on the residual work demand $RE_{a,t}$, which corresponds to the remaining work demand, taking into account the individual plannings already in the current partial solution. Each time an individual planning is computed, the residual work demand $RE_{a,t}$ is updated, and the method is run again with the remaining set of employees.

At initialization, $RE_{a,t}$ have the same value as the work demand $DE_{a,t}$ and they are updated each time a planning is added or deleted in the team schedule. In the objective function of the subproblem for employee e, the cost $\hat{\pi}_{a,t}$ of variable $x_{a,t}$, which determines whether activity a is performed during time period t, is calculated by the relation:

The greedy heuristic is presented formally in Algorithm 1. The first iteration is complete when the first complete solution is constructed. Then we perform additional iterations in which, for each employee, the current individual planning is deleted from the solution and another planning is computed based on the updated residual work demand. Initial employees sorting, objective function costs and the number of iterations are parameters of the algorithm.

Algorithm 1:

Input: work demand $DE_{a,t}$;
Best found solution is empty: $\Omega_{best} \leftarrow \emptyset$;
$cost(\Omega_{best}) \leftarrow +\infty$;
Partial solution is empty: $\Omega \leftarrow \emptyset$;
$\mathcal{E}' \leftarrow$ **Sort** employees set $\mathcal{E}$;
for $i = 1, ..., nbIterations$ **do**
  foreach *employee* $e \in \mathcal{E}'$ **do**
    Delete current planning for employee e in partial schedule (if exists): $\Omega \leftarrow \Omega \setminus \{X_c\}$, $c \in \mathcal{C}(e)$;
    Compute the residual work demands $\forall a, t$: $RE_{a,t} \leftarrow DE_{a,t} - \sum_{X \in \Omega} x_{a,t}$;
    Compute $\hat{\pi}_{a,t}$ $\forall a, t$, according to (17);
    Solve subproblem with costs $\hat{\pi}_{a,t}$ for employee e to obtain a planning X_c, $c \in \mathcal{C}(e)$;
    Assign X_c to partial solution: $\Omega \leftarrow \Omega \cup \{X_c\}$;
  end
  if *solution* Ω *is complete* **and** $cost(\Omega) < cost(\Omega_{best})$ **then**
    $\Omega_{best} \leftarrow \Omega$;
  end
end
return *best found solution* Ω_{best} ;

Despite our efforts, we did not find a particular sorting algorithm for employees that gave better results than others on average. So we use Algorithm 1 several times with different random orders on the employees, and keep the best result found. Empirical tests suggest that after three iterations, the solution is usually not improved anymore.

6. Computational experiments

Our four methods, solving MIP compact model by a commercial solver, the branch-and-price algorithm, the diving heuristic, and the greedy heuristic have been tested on both real data coming from a customer and randomly generated data. The MIP compact model is described in the electronic supplement.

6.1. Customer data

Our customer data comes from a company of mini-marts. All instances are defined over one week divided into 15 minutes periods. In the customer data, almost 10% of employees work only on Saturday, while the others may work at most five days. Around 70% of the employees can only perform one type of production activity. Most of the employees have a small flexibility in their schedule: general beginning and finishing time of timeslots can be shifted by one hour (for instance, the first timeslot of a day-shift starts between 8.00 AM and 9.00 AM for a given employee, but this range can be different for another day).

The cost coefficients in the objective function are the following: over-coverage $CO_a = 1$, critical over-coverage $CO_a^{crit} = 2$, under-coverage $CU_a = 2$, critical under-coverage $CU_a^{crit} = 5$. For a work demand of $DE_{a,t}$, critical over-coverage occurs when strictly more than $OV_{a,t} = DE_{a,t} + 1$ employees are assigned to production activity a at time period t, while critical under-coverage occurs when strictly less than $UN_{a,t} = \lceil DE_{a,t}/2 \rceil$ employees are assigned to the activity. We use a representative set of twelve customer data. They have a different number of employees $|\mathcal{E}|$, and a different number of production activities $|\mathcal{A}|$.

We ran different methods on the customer data: the greedy heuristic , dive120, dive600 and dive1800 (the diving of Section 5.2 with the cumulated column generation time limit of 120, 600 and 1800 seconds, respectively) and the branch-and-price algorithm. In the method dive$T$, we limit the column generation time at each node to $T/|\mathcal{E}|$ seconds. As the diving requires at most $|\mathcal{E}|$ nodes for a new solution, dive$T$ lasts at most T seconds (plus additional time for initialization of the different nodes). The best solution found by heuristics dive120, dive600 and dive1800 is used for the initialization of the branch-and-price. The time reported for the branch-and-price does not include the time spent by the heuristic. All tests were run using a standard PC of the experimental platform "Plafrim" (see Acknowledgment) with 4 gigabits of memory over four cores (four subproblems are solved in parallel). All methods were implemented in Java and IBM Cplex 12.6 was used for solving the MIP and the linear master problems.

Tables 2 and 3 summarize the results obtained (respectively the objective function value of the solution found, and the execution time) with our customer data. In column "LB_{Lagr}", we report the Lagrangian lower bound computed at the root node of the B&P. Recall that $|\mathcal{E}|$ is the number of employees, and $|\mathcal{A}|$ the number of production activities. Column "LB_{Triv}" is a trivial lower bound computed in the following way. Let L be the cumulated working time of the team (measured in time periods): $L = \sum_{e \in \mathcal{E}} LE_e$. Let also D be the cumulated demand (measured in time periods): $D = \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}} DE_{a,t}$. If $D \geq L$ then the solution value cannot be less than $LB_{Triv} = \min_{a \in \mathcal{A}}\{CU_a\} \times (D - L)$. If $D < L$ then the solution value cannot be less than $LB_{Triv} = \min_{a \in \mathcal{A}}\{CO_a\} \times (L - D)$. The trivial lower bound is quite far from the Lagrangian bound for the customer data instances with 5 activities.

The running time of the greedy heuristic is very small, even for instances with 45 employees. The difference between the value of the greedy solution and the optimal one can be large. However, recall that simple constructive heuristics may fail to find feasible shifts, since assigning too much or too few hours at the beginning

Table 2
Customer data: solution values obtained by our methods. "T": branch-and-price method and the MIP solver did not terminate within 24 hours of calculation. In this case, the solution value is the best one found ; "-": the MIP solver did not find any feasible solution within the time limit. The best found dive solution is used for initialization of the B&P.

Data	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{A}\rvert$	LB_{Trir}	Greedy	dive120	dive600	dive1800	B&P	LB_{Lagr}	Compact
A1-7	5	1	204	390	335	335	335	325	321.5	325
A1-9	5	1	288	423	299	299	299	299	299	299
A1-0	10	1	334	528	393	393	393	393	393	393
A1-3	10	1	152	326	228	228	228	228	225.55	228
A3-5	25	3	680	1077	832	834	832	824 T	819.7	890 T
A3-9	25	3	866	1181	984	960	987	954 T	951	999 T
A3-1	30	3	880	1285	970	962	954	954 T	935.4	1095 T
A3-2	30	3	358	931	592	551	543	529 T	487.4	739 T
A5-5	42	5	140	1111	918	909	852	852 T	804	3179
A5-6	42	5	303	1254	1029	942	925	925 T	883.6	1298
A5-0	45	5	404	1713	1522	1510	1504	1504	1504	-
A5-1	45	5	412	1793	1529	1525	1533	1513 T	1507.7	-

Table 3
Customer data. Running time (in seconds) of our methods (B&P time does not include the time needed to find the initial solution). "T": our branch-and-price method and CPlex for solving the compact model did not terminate within 24 hours of calculation. In this case, the solution value is the best one found; "-": that the MIP solver failed due to memory issues.

of the week may not allow to find a solution that respects all bound constraints. Moreover, the piecewise linear cost function will take a large value even if most of the working demand is fulfilled.

The diving heuristics are much more effective to find near optimal solutions for these instances. The relative gap of the dive120 heuristic with the branch-and-price is greater than 10% only for two instances (A3_2 and A5_6). It may happen that the diving gives better results when a smaller computing time is set (A3_5, A3_9 and A5_1). However, different experiments, not reported here, showed that giving more time to the diving heuristic at each node generally improves the result.

The branch-and-price method terminates for five instances out of twelve within 24 hours of computation time. For three instances the bounds are tight at the root node, for two instances the root "lower bound" was improved (recall that the pricing is performed heuristically), for one instance the initial upper bound was improved, and for one instance both bounds were improved by branch-and-price. For five instances out of the remaining six, branch-and-price improves the upper bound before hitting the time limit. Note that when the MIP solver is able to find an optimal solution, its value is equal to the solution value found by the heuristic branch-and-price.

Over the twelve instances, the compact method gives optimal solutions for the four smallest ones. Feasible solutions were found within 24 hours for all instances, except for the two largest ones. Actually, the higher the number of employees is, the lower is the quality of the found solution. Furthermore, we note that the MIP solver is outperformed by the heuristic dive120 in terms of execution time and quality of the solution.

6.2. Generated data

Experiments show that our algorithms have a good behavior on confidential customer data. In order to allow a fair comparison with our methods, and push further the analysis, we have designed a random data generator based on our customer data experience. The setting of parameters is performed using four following inputs: the number of employees $|\mathcal{E}|$, the number of production activities $|\mathcal{A}|$, the flexibility index of employees F and the under-coverage index G.

The planning horizon is fixed to 7 days, and time periods have a length of 15 minutes. All employees are multi skilled, *i.e.* they can work in all production activities. Costs of over-coverage (CO_a, CO_a^{crit}) and under-coverage (CU_a, CU_a^{crit}) and related thresholds ($UN_{a,t}$, $OV_{a,t}$) are fixed as in the customer data. Flexibility index F of employees affects all employee planning constraints, limiting the accessible time periods. The higher this index is, the higher is the number of allowed time periods for a given employee. For example, all first timeslots must start between $[\![8.00 - R_1\ ,\ 8.30 + R_2]\!]$ where R_1, R_2 are random values uniformly distributed in {0, 15, ..., F∗15}. Under-coverage index G takes a value between 0 and 4 and determines the work demand $DE_{a,t}$. To create a work demand, a team with $(|\mathcal{E}| + G)$ employees is initialized and planned in a random order with an initial random work demand. Obtained plannings are thus used to define the work demand $DE_{a,t}$ that corresponds exactly to the work capacity of the $(|\mathcal{E}| + G)$ employees. We then randomly remove G employees in the team in such a way that the work demand could not be covered by the final team of employees: results will present under-coverage. Note that there exists a solution of value zero if $G = 0$.

Table 4
Generated data: solution values obtained by our methods. "T": branch-and-price method and MIP solver did not terminate after 24h. In this case, the solution value is the best one found; "-": MIP solver did not find any feasible solution within the time limit ; "L": column generation at the root node did not converge within 1 hour, hovewer our heuristic branch-and-price continues and carry out its branching strategy. "M": branch-and-price algorithm failed due to memory issues. The best found dive solution is used for initialization of the B&P.

Data	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	G	F	LB_{Triv}	Greedy	dive120	dive600	dive1800	B&P	LB_{Lagr}	Compact
dst1	10	1	0	0	0	15	0	0	0	0	0	0
dst2	10	1	0	8	0	187	3	0	0	0	0	0
dst3	10	1	1	0	260	260	260	260	260	260	260	260
dst4	10	1	1	8	260	320	272	272	272	260	260	260
dst5	10	1	2	0	360	360	360	360	360	360	360	360
dst6	10	1	2	8	440	497	440	440	440	440	440	440
dst7	25	3	0	0	0	90	27	6	9	6 T	0 L	0
dst8	25	3	0	8	0	545	274	208	180	180 T	0	5072 T
dst9	25	3	1	0	260	275	260	260	260	260	260	260
dst10	25	3	1	8	220	593	373	328	286	286 T	220	8395 T
dst11	25	3	3	0	540	540	540	540	540	540	540	540
dst12	25	3	3	8	620	857	650	650	650	635 T	623	778 T
dst13	40	5	0	0	0	180	180	99	66	66 T	0	12 T
dst14	40	5	0	8	0	829	589	451	360	360 T	0	-
dst15	40	5	3	0	580	709	586	580	580	580	580	589 T
dst16	40	5	3	8	460	1008	784	646	589	589 M	463	-
dst17	40	5	6	0	1160	1160	1160	1160	1160	1160	1160	1163 T
dst18	40	5	6	8	1200	1515	1350	1227	1227	1215 M	1206	-

Table 5
Running time (in seconds) of our methods (B&P time does not include the time needed to find the initial solution). Symbol "T" indicates that our branch-and-price method and CPlex for solving the compact model did not terminate within 24 hours of calculation. In this case, the solution value is the best one found. Symbol "L" indicates that column generation at the root node did not converge within 1 hour, hovewer our heuristic branch-and-price will still continue and carry out its branching strategy. Symbol "M" indicates that the branch-and-price algorithm failed due to memory issues.

Data	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	G	F	Greedy	dive120	dive600	dive1800	B&P	Col. gen.	Compact
dst1	10	1	0	0	0.3	10.8	10.0	10.0	0.2	0.2	11.1
dst2	10	1	0	8	0.4	90.1	426	802	0.4	0.3	1042
dst3	10	1	1	0	0.3	0.7	0.4	0.3	0.6	0.6	4.3
dst4	10	1	1	8	0.4	5.2	4.5	4.4	0	1.7	49
dst5	10	1	2	0	0.3	0.6	0.3	0.2	0.4	0.4	3.6
dst6	10	1	2	8	0.3	5	4.2	4.1	1.2	1.2	35
dst7	25	3	0	0	0.5	124	582	1666	T	L	8836
dst8	25	3	0	8	1.0	130	629	1773	T	2983	T
dst9	25	3	1	0	0.5	46.1	56	56.2	9	9	2359
dst10	25	3	1	8	0.9	126	606	1674	T	103	T
dst11	25	3	3	0	0.5	1.9	1.2	1.2	1.8	1.7	2493
dst12	25	3	3	8	0.9	123	320	319	T	13.9	T
dst13	40	5	0	0	0.8	3.4	619	1744	T	27943	T
dst14	40	5	0	8	1.5	139	654	1812	T	66011	T
dst15	40	5	3	0	0.9	117.3	119	119	9.3	9.3	T
dst16	40	5	3	8	1.5	140	635	1827	M	936	T
dst17	40	5	6	0	0.9	3.3	2.7	2.7	3.3	3.3	T
dst18	40	5	6	8	1.5	136	608	1385	M	51.3	T

Tables 4–6 sum up the results obtained with the generated data. The results are reported in the same way as those obtained for the customer cases. It transpires from our results that the structure of the data strongly impacts the algorithms behavior. As one would expect, the computing time depends on the number of employees to schedule, and the number of production activities. Flexibility is also a difficulty factor: it increases the number of possible shifts, and makes the dynamic program slower.

The tests on generated data confirm the conclusion drawn on the customer data. We observe that the "lower bound" obtained at the root node of the heuristic branch-and-price still has a very good value, *i.e.* it is often close or equal to the optimal solution or the best solution found, and the final absolute gap is small, except for the instances with a large size, and a large flexibility (dst8, dst10, and dst16).

The greedy heuristic is fast and the quality of the planning is good but rarely optimal. The diving heuristic gives results that are close to those of the branch-and-price in most cases (the results are even optimal for small data). Similar to what happens with the customer instances, giving more time at each node of the diving clearly improves the results on average, although it may happen (dts7) that better results are obtained when less time is allowed.

A specificity of generated data instances is that the trivial lower bound is very close to the Lagrangian one and the optimal solution. Moreover, the bounds are equal for most of the instances. When this is the case, it is quite easy to find an optimal dual solution π (optimal Lagrangian multipliers) for the column generation algorithm. However, even when an optimal dual solution is known, it takes a large number of iterations to find an optimal primal solution of the linear relaxation of the master problem. This fact explains why known stabilization techniques such as dual smoothing do not improve convergence of column generation. They are aimed at stabilizing around the best dual solution. This does not help in our case as the optimal dual solution is already known.

Table 6
Generated data: behavior indicator of the methods according to the data set. Symbol "T" indicates that our branch-and-price method did not terminate within 24 hours of calculation. In this case, the solution value is the best one found.

		nbNodes				nbMP				nbColumns			
Data	$\|\mathcal{E}\|$	dive120	dive600	dive1800	B&P	dive120	dive600	dive1800	B&P	dive120	dive600	dive1800	B&P
dst1	10	10	10	10	1	209	209	209	1	1089	1089	1089	0
dst2	10	10	10	10	1	1116	2891	3234	1	5894	17140	22885	0
dst3	10	2	2	2	1	5	5	5	5	40	40	40	40
dst4	10	10	10	10	48	91	91	91	176	293	293	293	1280
dst5	10	2	2	2	1	3	3	3	3	20	20	20	20
dst6	10	10	10	10	1	90	90	90	7	227	227	227	60
dst7	25	25	25	25	1 T	390	1119	2431	2002 T	3014	8068	17970	50025 T
dst8	25	25	25	25	132 T	383	1388	2301	6870 T	2301	6795	12183	168450 T
dst9	25	25	25	25	802	268	305	22933	42	3280	3332	3332	448335
dst10	25	25	25	25	802 T	339	1057	2076	22933 T	2136	6380	11770	448335 T
dst11	25	2	2	2	1	8	8	8	8	175	175	175	175
dst12	25	25	25	25	920 T	421	826	826	31912 T	3002	4827	4827	617519 T
dst13	40	1	40	40	91 T	9	701	1310	9758 T	320	8426	16423	386680 T
dst14	40	40	40	40	2 T	240	694	1222	1994 T	1869	5522	10348	79680 T
dst15	40	40	40	40	1	313	261	256	27	4095	3579	3434	290
dst16	40	40	40	40	1129	279	718	1358	7167	2135	5817	11470	241470
dst17	40	2	2	2	1	10	10	10	10	354	354	354	580
dst18	40	40	40	40	1151	262	794	1321	13007	2178	8339	11502	397319

7. Conclusion

In this paper, we describe efficient strategies for solving a real-life employee scheduling problem that mixes days-off scheduling, shift scheduling, shift assignment, activity assignment, pause assignment and break assignment. Our approaches are based on the Dantzig–Wolfe decomposition, and we have successfully implemented a heuristic branch-and-price algorithm, from which we derived a diving heuristic and a greedy algorithm. These methods were tested on both customer and randomly generated data with excellent results. The computational experiments show that the proposed approaches yield optimal or near optimal solutions in many cases.

The behavior of our methods raises several questions. Since classical stabilization strategies were not able to improve the convergence of the algorithm, we need a deeper analysis of the structure of the problem to come up with new strategies dedicated to this kind of problems. As explained above, stabilization techniques acting in the dual space have a limited impact on the convergence. Therefore, an effort should be done in developing primal stabilization strategies.

From a customer point of view, it would be interesting to consider the *annualized workforce allocation problem*, in which employees are scheduled over the planning horizon of several weeks (up to a year). This problem includes constraints linking successive weeks of work. Since solving this problem for one week is already challenging for state-of-the-art methodologies, we plan to derive heuristics for these very large scale instances.

In this work, we consider independent employees, *i.e.* they perform activities independently of other employees. Another challenge would be to consider activities or tasks that require simultaneous presence of several employees with different skills.

Acknowledgment

The authors would like to thank the anonymous referees for their useful comments, which helped improving the presentation of the paper.

Experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from Laboratoire Bordelais de Recherche en Informatique and Institut de Mathématique de Bordeaux and other entities