

A Solution Framework for the Generalised Rostering Problem

Faramroze G Engineer
Department of Engineering Science
University of Auckland
New Zealand
feng003@ec.auckland.ac.nz

Abstract

Over the last two decades industries have significantly benefited from research that has been carried out towards the design of efficient rostering systems. However, the complex nature of rostering problems means that rostering tools are generally specifically tuned for a particular industry or job. As a result, software solutions based on optimal strategies are usually limited to handling problems that are very closely related to each other. Column generation holds particular interest to us, not only because it is capable of achieving optimal solutions, but also because it separates the characteristics that define a particular rostering problem from the optimiser. Hence, by embedding generality at the column generation level, we can achieve overall generality.

1 Formulation of the Generalised Rostering Problem

The formulation of rostering and scheduling problems in many ways resembles the Multi-commodity network flow model with resource constraints (Desrosiers *et al.* (1995)). For the rostering problem, an individual employee or staff member can be considered as a commodity which flows through a sequence of activities. Collectively, this flow represents the assignment of activities to staff in order to meet the overall staffing requirements. An activity can range from a simple task or time-off, to the more complex Tour of Duties described by Ryan (1992). In this paper we use the term shift to describe any working activity.

In a network flow model, a path is deemed to be legal if the resources accumulated over that path is within a specified bound. In the rostering context, we use the term attributes to describe both physical resources and abstract entities that characterise an activity. Paid minutes, weekends off and start time are just three basic examples of attributes that we might be interested in during the construction of a roster. The number of days worked since the last time-off, is an example of a more complex attribute whose value has to be reset to 0 after every time-off activity. The legality of a sequence of activities, is governed by the value of these attributes over the sequence of activities in question.

Let T represent a set of tasks to be performed, and d_t the number of employees required to cover a particular task $t \in T$. Let E be the set of employees for which we are creating the roster. For each employee $e \in E$ we have a corresponding time-space graph $G^e = (N^e, A^e)$, where N^e is the set of nodes that represent possible shifts or time-off that can be assigned to the employee, and A^e are the set of arcs that represents the transitions from one activity to another. $o(e)$ and $d(e)$ are artificial activities that mark the beginning and end of the roster period for each employee. Given an activity $n \in N^e$, $S(n)$ is the set of tasks $t \in T$ that are covered by activity n , (obviously a time-off activity will not cover any tasks). Let x_{ij}^e represent the flow of employee e through arc $(i, j) \in A^e$. If x_{ij}^e is confined to be binary, then

$X^e = \{x_{ij}^e\}$ becomes the set of decision variables that determine the assignment of activities to a particular employee.

Let R^e be the set of attributes tracked for employee e , and d_i^{er} the value of attribute $r \in R^e$ for activity $i \in N^e$. The value of an attribute r for employee e , over a sequence of activities from $o(e)$ to i is given by the variable D_i^{er} . $D_i^e = \{D_i^{er}, \forall r \in R\}$ is the set of all attribute values D_i^{er} for a sequence of activities from $o(e)$ to i . In a network flow model resources are simply accumulated over a path (i.e. $D_j^{er} = D_i^{er} + d_j^{er}, \forall (i, j): x_{ij}^e = 1$), however for a rostering model, there is the possibility of a more complex interaction between attribute values. For example, the paid minutes over a sequence of activities is simply the sum of the paid minutes of individual activities, however, the start time of a sequence of activities is the start time of the first activity. This logic is defined by the function $f^r(D_i^{er}, d_j^{er})$ (i.e. $D_j^{er} = f^r(D_i^{er}, d_j^{er}), \forall (i, j): x_{ij}^e = 1$). Also, in the network flow model, the bounds on resources at a particular node i is generally known a priori (i.e. $l_i^{er} \leq D_i^{er} \leq u_i^{er}$, where l_i^{er} and u_i^{er} are the predefined lower and upper bounds at node i). However, in rostering problems the bounds on attribute values at a particular node/activity i , not only depend on the type of activity that precedes i , but also the attribute values of the activities preceding i . For example, if a sequence of shifts is followed by time-off, we might want to make sure that the employee has had sufficient time-off following his/her sequence of shifts. Furthermore, the sufficient time-off could be governed by the number of days worked since the last time-off. However, we do not need to check the legality of time-off if a sequence of shifts is followed by another shift. Hence, the bound on sufficient time-off is only enforced on time-off activities. The upper bound on attribute r at node j is defined dynamically by the function $u_{ij}^r(D_i^{er})$ (i.e. $D_j^{er} \leq u_{ij}^r(D_i^{er}), \forall (i, j): x_{ij}^e = 1$). Similarly, the lower bound on attribute r at node j is defined by the function $l_{ij}^r(D_i^{er})$.

By placing bounds on the attribute values, we are able to place constraints on individual rosters, however, rostering problems often require more global constraints (i.e. constraints involving several employees). Constraints that restrict the number of part time employees performing a particular task, or constraints requiring a group of employees to perform the same task, are examples of global constraints. Let M be the set of global constraints, and for a given $m \in M$, let B_m^e be a subset of nodes N^e as defined by constraint m . Let $b_{m,i}^e$ be the coefficient assigned to node i and employee e for constraint m , and \underline{b}_m and \bar{b}_m be the lower and upper bound values for that global constraint.

Finally, we need some measure of objective while building our rosters. In a network flow model there are usually predefined costs (usually cost per flow) associated with traversing through an arc and/or node. The total cost in this case is simply

$$\sum_{(i,j) \in A^e} x_{ij}^e (c_{ij}^e + c_j^e), \forall e \in E,$$

where c_{ij}^e is the cost associated with arc (i, j) , and c_j^e is the cost of flow through node j . Similarly, the cost of a roster can be made up of the cost of performing individual activities, and the penalty costs associated with the quality of transition between activities. However, the cost of performing an activity i often additionally depends on the type of activity, and the attribute values of activities preceding i . For example, after working 5 consecutive days, it is legal for an employee to take either 2 or 3 days off. However, taking 3 days off after working 5 days might be considered as a transition of poor quality compared to taking 2 days off. We use a cost function $c_{ij}^e(D_i^e, D_j^e)$ to evaluate the cost of assigning an activity j after a

sequence of activities from $o(e)$ to i . Note that it is possible to have different cost functions for each employee, since perception of quality is unique to an individual.

The generalised rostering problem can then be defined by

$$\min \sum_{e \in E} \sum_{(i,j) \in A^e} x_{ij}^e c_{ij}^e (D_i^e, D_j^e) \quad (1.1)$$

st:

$$\sum_{e \in E} \sum_{(i,j) \in A^e: t \in S(j)} x_{ij}^e = d_t, \quad \forall t \in T \quad (1.2)$$

$$\underline{b}_m \leq \sum_{(i,j) \in A^e: j \in B_m^e} b_{m,j}^e x_{ij}^e \leq \bar{b}_m, \quad \forall m \in M \quad (1.3)$$

$$\sum_{(o(e),j) \in A^e} x_{o(e),j}^e = 1, \quad \forall e \in E \quad (1.4)$$

$$\sum_{(i,d(e)) \in A^e} x_{i,d(e)}^e = 1, \quad \forall e \in E \quad (1.5)$$

$$\sum_{i:(i,j) \in A^e} x_{ij}^e - \sum_{i:(j,i) \in A^e} x_{ji}^e = 0, \quad \forall e \in E, \quad \forall j \in N^e \setminus \{o(e), d(e)\} \quad (1.6)$$

$$x_{ij}^e (f^r(D_i^{er}, d_i^{er}) - D_j^{er}) = 0, \quad \forall e \in E, \quad \forall r \in R, \quad \forall (i,j) \in A^e \quad (1.7)$$

$$x_{ij}^e (u_{ij}^r(D_i^{er}) - D_j^{er}) \geq 0, \quad \forall e \in E, \quad \forall r \in R, \quad \forall (i,j) \in A^e \quad (1.8)$$

$$x_{ij}^e (l_{ij}^r(D_i^{er}) - D_j^{er}) \leq 0, \quad \forall e \in E, \quad \forall r \in R, \quad \forall (i,j) \in A^e \quad (1.9)$$

$$x_{ij}^e \in \{0,1\}, \quad \forall e \in E, \quad \forall (i,j) \in A^e \quad (1.10).$$

Constraints (1.2) specify the number of employees that are required to be assigned to a particular task. (1.3) describes the global constraints of a rostering problem, while constraints (1.3-1.5) conserve the flow through the space-time graphs of each employee. Given a sequence of activities $\{o(e), \dots, i, j\}$, constraint (1.6) ensures that the attribute values D_j^{er} for each $r \in R$ is a function of the attribute value D_i^{er} and d_j^{er} . Similarly, (1.8) and (1.9) places bounds on the attribute values of a sequence of activities based on information from preceding activities.

2 Decomposition of the Generalised Rostering Model

Although (1.1-1.10) gives us a global understanding of a generalised rostering problem, it would be impractical to directly implement into an optimisation framework. The complexity within this model arises mainly due to the reliance on information from preceding activities to calculate the attribute values, costs and bounds. Although we cannot explicitly remove this complexity, we can use Dantzig-Wolfe decomposition to separate this complexity from the optimisation process and embed it within a column generation sub problem.

The following section is a summary of the Dantzig-Wolfe decomposition of (1.1-1.10), based on the decomposition performed by Desrosiers *et al.* (1995) for the vehicle routing problem with time windows. The decomposition separates the rostering problem into a master problem and several subproblems. The master problem is defined by the original objective (1.1), the task covering constraints (1.2), and the global constraints (1.3). The subproblem is defined by a modified objective and the employee specific constraints (1.4-1.10).

2.1 Subproblem Formulation

The employee specific constraints are the path constraints on the original flow network. By examining constraints (1.4-1.10) we can clearly see that they are independent between employees. Hence, the subproblem naturally decomposes into $|E|$ disjoint subproblems, one

for each employee. Furthermore, the subproblem for each employee is a resource constraint shortest path problem, in which we wish to assign a sequence of activities to an employee at minimal cost, while satisfying constraints (1.8-1.9). The solution to such a problem is found on a bounded polyhedron. As a result, the original flow variables x_{ij}^e can be expressed as a nonnegative convex combination of the paths generated from the corresponding subproblem.

Unfortunately, the resource constrained shortest path problem is *NP*-hard for which there are no polynomial or pseudo-polynomial algorithms known. In addition, traditional Dynamic Programming (DP) techniques for resource constrained shortest path problems assume that resources are simply accumulated along a path, and as a result, are unable to cope with the complex behaviour of constraints (1.7-1.9). In section 4 we introduce a dynamic programming framework for the generalised rostering problem that further decomposes each subproblem to handle these constraints.

2.2 Master Problem Formulation

Let P^e be the set of feasible paths of subproblem $e \in E$ (i.e. a feasible sequence of activities from $o(e)$ to $d(e)$). Every $p \in P^e$, corresponds to an elementary path which can be described using the original binary flow variables x_{ij}^e . If x_{ij}^e , for all $(i, j) \in A^e$ represents the flow on path p , then a solution to the master problem can be expressed as a nonnegative convex combination of a finite number of elementary paths, i.e.

$$x_{ij}^e = \sum_{p \in P^e} x_{ijp}^e \theta_p^e, \quad \forall e \in E, \quad \forall (i, j) \in A^e \quad (2.1)$$

$$\sum_{p \in P^e} \theta_p^e = 1, \quad \forall e \in E \quad (2.2)$$

$$\theta_p^e \geq 0, \quad \forall e \in E, \quad \forall p \in P^e \quad (2.3)$$

Let a_{ip}^e be a binary constant that takes the value 1 if a sequence of activities p for employee e covers task t . Let b_{mp}^e be the coefficient of the global constraint m on path p . Finally, let c_p^e be the cost of path p . We can then define a_{ip}^e, b_{mp}^e and c_p^e as follows:

$$a_{ip}^e = \sum_{(i,j) \in A^e: t \in S(j)} x_{ijp}^e, \quad \forall e \in E, \quad \forall t \in T, \quad \forall p \in P^e \quad (2.4)$$

$$b_{mp}^e = \sum_{(i,j) \in A^e: j \in B_m^e} b_{m,j}^e x_{ijp}^e, \quad \forall e \in E, \quad \forall m \in M, \quad \forall p \in P^e \quad (2.5)$$

$$c_p^e = \sum_{(i,j) \in A^e} x_{ijp}^e c_{ij}^e (D_i^e, D_j^e) \quad (2.6).$$

By substituting a_{ip}^e, b_{mp}^e and c_p^e into the master problem and rearranging the summation order we get the revised master problem

$$\min \sum_{e \in E} \sum_{p \in P^e} c_p^e \theta_p^e \quad (2.7)$$

st:

$$\sum_{e \in E} \sum_{p \in P^e} a_{ip}^e \theta_p^e = d_t, \quad \forall t \in T \quad (2.8)$$

$$\underline{b}_m \leq \sum_{(i,j) \in A^e: j \in B_m^e} b_{m,j}^e x_{ij}^e \leq \bar{b}_m, \quad \forall m \in M \quad (2.9)$$

$$\sum_{p \in P^e} \theta_p^e = 1, \quad \forall e \in E \quad (2.10)$$

$$\theta_p^e \geq 0, \quad \forall e \in E, \quad \forall p \in P^e \quad (2.11).$$

On closer examination we can see that the revised master problem (2.7-2.11) is indeed the linear relaxation of a Generalised Set-Partitioning Problem (GSPP). The GSPP has been used extensively and successfully as a solution method for a wide range of rostering and scheduling problems. However, the quality of a solution is only as good as the ability of the subproblem to efficiently and accurately model the intrinsic characteristics of the particular problem. As a result, most implementations of subproblem models are highly tuned for the particular problem and lack the flexibility to model a greater range of rostering problems. In section 4 we describe a dynamic programming framework that aims to generalise the rostering subproblem.

3 Column Generation and Solution Strategy

To solve the rostering problem we could enumerate all feasible paths (i.e. sequence of activities) for each employee, and then use the GSPP to assign a single path to each employee so as to meet the overall staffing requirements and global constraints at minimal cost. However, due to the combinatorial nature of the subproblem there are a large number of feasible paths for each employee. In fact, for large rostering problems such as in the airline industry, this could be in excess of a million paths for each employee. Hence, most rostering problems that make use of the GSPP model use column generation techniques to dynamically generate new paths when required.

Column generation techniques coordinate the interaction between the master problem and subproblems. The master problem optimises the allocation of paths from the current set of paths available. Based on dual information of the master problem, the subproblems create new paths with minimal reduced cost. If a new path with negative reduced cost is found, then it is added to the master problem. Solving the master problem over the current set of paths will yield dual variables $\pi_t^1, \forall t \in T$, $\pi_e^2, \forall e \in E$ and $\pi_m^3, \forall m \in M$ at each simplex iteration. The modified objective of the subproblems is then to find the minimal reduced cost path defined by

$$\min \underline{c}_p^e = \sum_{(i,j) \in A^e} x_{ijp}^e c_{ij}^e (D_i^e, D_j^e) - \sum_{t \in T} a_{tp}^e \pi_t^1 - \sum_{m \in M} b_{mp}^e \pi_m^3 - \pi_e^2 \quad (3.1).$$

Since DP techniques for the subproblem often produce many paths with negative reduced cost, in most cases it is beneficial to add several paths to the master problem at the same time. This strategy is especially useful in accelerating the overall performance when solutions to the subproblem are computationally expensive relative to the pricing of non-basic paths in the master problem. Desaulniers *et al.* (1999) gives a good overview of acceleration strategies within a column generation framework.

The optimal solution of the master problem is almost never integer. The optimal integer solution of the rostering problem can be found by finding the optimal integer solution to the master problem. Using column generation techniques we would have generated only a small subset of all paths to obtain the optimal solution to the relaxation. Performing branch and bound on this small subset of paths might result in a sub-optimal or even infeasible solution. Hence, we have to continue generating paths during the branch and bound procedure. Forcing θ_p^e in the master problem to be binary ensures that exactly one path (i.e. sequence of activities from $o(e)$ to $d(e)$) is assigned to each employee. However, in practice conventional variable branching on θ_p^e is ineffective within a GSPP column generation framework. If we force $\theta_p^e = 0$, then it is possible (and probably very likely) that the subproblem for employee e will return exactly the same path p we have just banned from the master problem. In this case the subproblem for employee e should return the next best sequence of activities. As a result, subproblems need to be able to calculate the k^{th} best

solution of a resource constrained shortest path problem, which for most scenarios is a non-trivial matter. Furthermore, the unbalanced nature of 0-1 variable branching on θ_p^e causes additional problems during branch and bound.

To eliminate some of the difficulties that arise from performing a 0-1 branch on the master problem variables, most successful implementations of rostering and scheduling problems use constraint branching techniques that branch on the original flow variables x_{ij}^e . By setting $x_{ij}^e = 1$ we force employee e to perform activities i and j (i.e. ban all paths that do not assign activities i and j to employee e). Similarly, by setting $x_{ij}^e = 0$ we force employees e not to perform activities i and j (i.e. ban all paths that assign activities i and j to employee e). Branching on x_{ij}^e drives integrality of the master problem by tightening the task covering constraints (1.2). Constraint branching schemes are especially useful within a column generation framework as the flow variable x_{ij}^e is naturally represented within a subproblem, i.e. if we set $x_{ij}^e = 1$ we eliminate all arcs $(i, k) \in A^e : k \neq j$. Similarly, if we set $x_{ij}^e = 0$ then we eliminate arc $(i, j) \in A^e$. Another way to eliminate certain arcs from the optimal solution of a subproblem is by making them very unattractive (i.e. giving them very large costs). The reader is referred to Barnhart *et al.* (1998) for more comprehensive discussions on constraint branching strategies and their implementations within a column generation framework.

4 A DP Framework for the Generalised Rostering Subproblem

As discussed earlier, the objective of a subproblem within a column generation framework is to create paths (i.e. sequence of activities) with minimal reduced cost. (3.1) can be re-written in terms of the subproblem parameters to give the formulation of the subproblem for each employee $e \in E$, i.e.

$$\min \underline{c}_p^e = \sum_{(i,j) \in A^e} x_{ij}^e \underline{c}_{ij}^e(D_i^e, D_j^e) - \pi_e^2 \quad (4.1)$$

st:

$$\sum_{(o(e),j) \in A^e} x_{o(e),j}^e = 1, \quad \forall e \in E \quad (4.2)$$

$$\sum_{(i,d(e)) \in A^e} x_{i,d(e)}^e = 1, \quad \forall e \in E \quad (4.3)$$

$$\sum_{i:(i,j) \in A^e} x_{ij}^e - \sum_{i:(j,i) \in A^e} x_{ji}^e = 0, \quad \forall e \in E, \quad \forall j \in N^e \setminus \{o(e), d(e)\} \quad (4.4)$$

$$x_{ij}^e (f^r(D_i^{er}, d_i^{er}) - D_j^e) = 0, \quad \forall e \in E, \quad \forall r \in R, \quad \forall (i, j) \in A^e \quad (4.5)$$

$$x_{ij}^e (u_{ij}^r(D_i^{er}) - D_j^e) \geq 0, \quad \forall e \in E, \quad \forall r \in R, \quad \forall (i, j) \in A^e \quad (4.6)$$

$$x_{ij}^e (l_{ij}^r(D_i^{er}) - D_j^e) \leq 0, \quad \forall e \in E, \quad \forall r \in R, \quad \forall (i, j) \in A^e \quad (4.7)$$

$$x_{ij}^e \in \{0, 1\} \quad \forall e \in E, \quad \forall (i, j) \in A^e \quad (4.8),$$

where the cost function $\underline{c}_{ij}^e(D_i^e, D_j^e)$ is defined by

$$\underline{c}_{ij}^e(D_i^e, D_j^e) = c_{ij}^e(D_i^e, D_j^e) - \sum_{t \in T: t \in S(j)} \pi_t^1 - \sum_{m \in M: j \in B_m^e} b_{m,j}^e \pi_m^3 \quad (4.9).$$

As mentioned earlier, the complexity within this problem arises due to the dynamic nature of the attributes, costs and bounds. Hence, DP techniques similar to the pulling algorithm used by Desrosiers *et al.* (1995) for the resource constrained shortest path problem are impractical for the subproblem in its current form. Instead, we propose a DP scheme that

exploits the inherent characteristics of a generalised rostering problem, and decomposes the subproblem into a more manageable structure.

Let us first define some rostering terminology:

- On-Stretch: a sequence of consecutive days worked
- Off-Stretch: a sequence of consecutive days off
- Work-Stretch: an on-stretch followed by an off-stretch
- Roster-Line: a sequence of work-stretches
- Line of Work (*LoW*): a roster-line covering activities $o(e)$ and $d(e)$.

We use the term state to describe any shift, on-stretch, off-stretch, work-stretch or roster-line. Let Z define the possible types of states within a *LoW*, i.e. $Z = \{Shift, OffStr, OnStr, WrkStr, RostLn\}$. Let $Shift_{kl}$, $OnStr_{kl}$, $OffStr_{kl}$, $WrkStr_{kl}$ and $RostLn_{kl}$ be the set of all shifts, on-stretches, off-stretches, work-stretches and roster-lines that start on day k and end on day l . We can then define on-stretches, work-stretches and roster-lines by their respective parent states as follows:

$$\begin{array}{l} \text{Child State:} \quad \text{Parent States:} \\ \\ OnStr_{kl} = OnStr_{kn} + Shift_{nl}, \quad \forall n : k \leq n \leq l \end{array} \quad (4.10)$$

$$WrkStr_{kl} = OnStr_{kn} + OffStr_{nl}, \quad \forall n : k \leq n \leq l \quad (4.11)$$

$$RostLn_{kl} = RostLn_{kn} + WrkStr_{nl}, \quad \forall n : k \leq n \leq l \quad (4.12).$$

For each $z \in Z$, let P_z define the possible types of states within a state of type z , i.e.

$$\begin{aligned} P_{Shift} &= \{Shift\}, \\ P_{OffStr} &= \{OffStr\}, \\ P_{OnStr} &= \{Shift, OnStr\}, \\ P_{WrkStr} &= \{Shift, OnStr, OffStr, WrkStr\}, \\ P_{RostLn} &= \{Shift, OnStr, OffStr, WrkStr, RostLn\}. \end{aligned}$$

Figure 1 gives a small example of how on-stretches and work-stretches can be built from shifts and off-stretches.

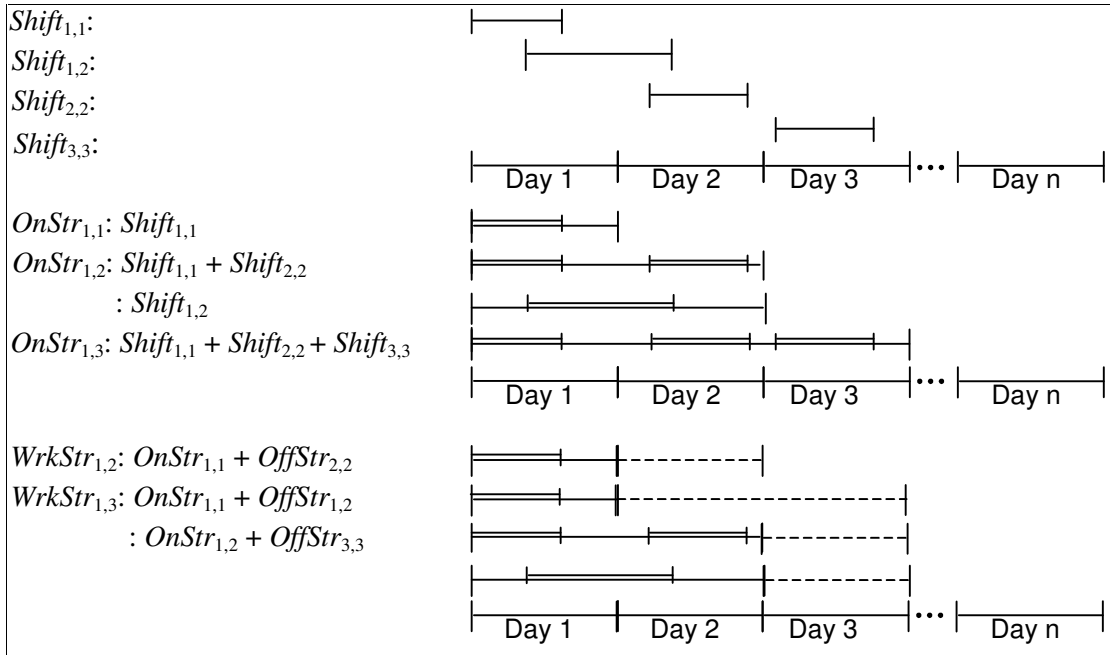


Figure 1 Building On-Stretches and Work-Stretches from Shifts and Off-Stretches

Attributes are characteristics of a state, but only certain types of states can contribute to a particular attribute value. For example, paid minutes is a characteristic of a shift, hence, from

equation (4.10) we can see that an on-stretch will inherit the value of its paid minutes from its shifts. Similarly, from (4.11-4.12) we can see that a work-stretch will inherit the value of paid minutes from its on-stretch, and a roster-line will inherit the value of paid minutes from its work-stretches. In this case paid minutes is an attribute of shifts, on-stretches, work-stretches and roster-lines. However, paid minutes is not an attribute of an off-stretch, as an off-stretch state will never contribute to its value. Furthermore, we might not be interested in the number of paid minutes of a roster line, as long as an employee has worked a certain number of paid minutes in each work-stretch. In this case the roster line does not need to inherit the value of paid minutes from its work-stretches, and hence the paid minutes attribute can be discarded from each work-stretch once we have checked for its legality. In summary

- an attribute is considered as an attribute of a state if that state contributes to that attribute value, or the attribute value can be calculated using inherited information,
- an attribute is only required at a state if there is some child state that checks for the legality of that attribute.

Let us define an attribute by the state which first contributes to its value, i.e. $\forall r \in R$ let $r^s \in Z$ define the type of state that first contributes to r . For example, paid minutes is available for the first time at the shift state, while rest time between shifts is calculated for the first time at an on-stretch state. Every attribute also has a rule associated with it. These rules are constraints (4.6-4.7) of the subproblem. Earlier we discussed how the bounds on an attribute depend on the type of transition between activities. We can enforce rules by imposing bounds at certain activities if the type of transition is merely dependent on the type of activity on either side of that transition. This is equivalent to enforcing a particular rule at a particular type of state. For example, a particular rostering problem requires a constraint on the number of paid minutes between time-off. We can enforce this constraint by imposing bounds on the number of paid minutes at the beginning of every time-off activity and then reset the number of paid minutes to zero. Equivalently, we could simply enforce this constraint by imposing these bounds on all work-stretches.

Let us further define an attribute by the state at which its rule is enforced, i.e. $\forall r \in R$ let $r^g \in Z$ define the type of state at which we place bounds on r . Hence, if we know the state at which an attribute is tracked first, and the state at which the rule is enforced on that attribute, we can exactly define all type of states $Z_r \subseteq Z$ that track attribute $r \in R$, i.e. $Z_r = \{z \in Z : r^s \in P_z, r^g \notin (P_z \setminus z)\}$, $\forall r \in R$. Equivalently, we can define all attributes $R_z \subseteq R$ that are tracked by a particular type of state $z \in Z$, i.e. $R_z = \{r \in R : z \in Z_r\}$.

In summary, we can see that by decomposing a line of work into on-stretches, off-stretches, work-stretches and roster-lines we know exactly which attributes and rules are relevant to each state, where as in the original subproblem formulation (4.1-4.8) we would have to know the types of activities involved in each transition and then apply the relevant rules. Furthermore, by using the suggested decomposed structure, attributes are either passed to the child state or discarded. Hence, we do not have to track every attribute at every state or worry about resetting attributes that have crossed their window of validity.

4.1 The Subproblem Dynamic Program

The current structure enables us to track attributes and enforce rules more efficiently. Using the building block equations (4.10-4.12) we can enumerate all the possible states to build all feasible lines of work for an employee. However, from the combinatorial nature of the problem, it is obvious that such a foolhardy approach will result in an explosion of the number of states. Instead we aim to use a DP approach that discards states if there are

equivalent states at lower cost already available. The aim of the DP is then to find all dominating states.

A state that does not conform to its bounds is considered illegal and hence, cannot be used as a parent of other states. Furthermore, a child state perceives a parent state by the attributes it inherits from its parent. Hence, a state is considered equivalent to other states of the same type, if the values of attributes passed to a child state are the same. Finally, a state can dominate other states of the same type if it is equivalent in the attributes it passes to a child state and it has lower cost.

Let us first define some terminology that will help us understand the structure of the DP. Let $s \in z_{k,l}$ be a state of type $z \in Z$ starting on day k and ending on day l . Let D_s^r be the value of attribute $r \in R_z$. Let D_s be the set of attribute values D_s^r for each $r \in R_z$ of state s . Let $\bar{R}_z \subseteq R_z$ be the set of all attributes $r \in R_z$ being passed to a child state, i.e. $\bar{R}_z = \{r \in R_z : (r \in R_y, \forall y : z \in P_y)\}$. Furthermore, let $\bar{D}_s \subseteq D_s$ be the values of those attributes being passed to a child state, i.e. $\bar{D}_s = \{D_s^r, \forall r \in \bar{R}_z\}$. Let $s^{ParentA}$ and $s^{ParentB}$ be the corresponding parent states that make state s . The attribute values D_s of s are given by $D_s^r = f^r(D_{s^{ParentA}}^r, D_{s^{ParentB}}^r), \forall r \in R_z$. The cost of state s is given by $C_s = c_z^e(D_{s^{ParentA}}, D_{s^{ParentB}}) + C_{s^{ParentA}} + C_{s^{ParentB}}$, where $c_z^e(D_{s^{ParentA}}, D_{s^{ParentB}})$ evaluates the quality of assigning $s^{ParentB}$ after $s^{ParentA}$. However, the objective of the subproblem is to return a sequence of activities with minimal reduced cost and not simply minimal cost. Hence, we have to embed the dual costs of the task covering constraints and global constraints into each state. We can achieve this by defining the cost of a shift state $sh \in Shift_{k,l}$ for employee $e \in E$ as follows

$$C_{sh} = c_{Shift}^e(D_{sh}) - \sum_{t \in T : t \in S(sh)} \pi_t^1 - \sum_{m \in M : sh \in B_m^e} \pi_m^3,$$

where $c_{Shift}^e(D_{sh})$ evaluates the quality of performing shift sh . The dual costs are then inherited by all states made up of shifts.

Earlier we defined $z_{k,l}$ to be the set of all states of type $z \in Z$ starting on day k and ending on day l . We can strengthen our definition by additionally requiring $z_{k,l}$ being the set of all feasible states of type z starting on day k and ending on day l , i.e. $l^r(D_{s^{ParentA}}^r) \leq D_s^r \leq u^r(D_{s^{ParentA}}^r), \forall r \in R_z : r_g = z, \forall s \in z_{k,l}, \forall z \in Z$, where $l^r(D_{s^{ParentA}}^r)$ and $u^r(D_{s^{ParentA}}^r)$ calculate the bounds on attribute r based on the attribute values of $s^{ParentA}$. Note that we only impose bounds on attributes whose rules are enforced at a state of type z . Hence, our bounding functions do not require information on the type of activities involved in the transition as in the original subproblem formulation.

Let $\Phi^{z_{k,l}}$ be the possible sets of attribute values of states in $z_{k,l}$ being passed to a child state, i.e. $\Phi^{z_{k,l}} = \{\bar{D}_s, \forall s \in z_{k,l} : \bar{D}_s \notin \Phi^{z_{k,l}}\}, \forall z \in Z$. Then for each possible attribute set $E \in \Phi^{z_{k,l}}$, let $z_{k,l}(E) \subseteq z_{k,l}$ define the set of all states of type z starting on day k and ending on day l with equivalent attributes being passed to a child state, i.e. $z_{k,l}(E) = \{s \in z_{k,l} : \bar{D}_z = E\}, \forall E \in \Phi^{z_{k,l}}, \forall z \in Z$. The aim of the DP is then to find the minimum cost state within $z_{k,l}(E)$ for each $E \in \Phi^{z_{k,l}}$ and $z \in Z$. We can then define $z_{k,l}^{\min}$ to be the set of dominating states of type z , starting on day k and ending on day l , i.e. $z_{k,l}^{\min} = \{\min z_{k,l}(E), \forall E \in \Phi^{z_{k,l}}\}, \forall z \in Z$. If none of the attributes of a particular type of state $z \in Z$ are passed on to a child state (i.e. $\bar{R}_z = \emptyset$), then the DP for states of type z would be equivalent to finding a resource constrained shortest path for each start and end day combination.

Equations (4.10-4.12) can then be redefined to include DP as follows:

Child State:	Parent States:	
	ParentA:	ParentB:
$OnStr_{kl}$	$= OnStr_{kn}^{\min} + Shift_{nl}^{\min},$	$\forall n : k \leq n \leq l$
$WrkStr_{kl}$	$= OnStr_{kn}^{\min} + OffStr_{nl}^{\min},$	$\forall n : k \leq n \leq l$
$RostLn_{kl}$	$= RostLn_{kn}^{\min} + WrkStr_{nl}^{\min},$	$\forall n : k \leq n \leq l.$

Refer to Figure 2 for a summary of the proposed DP framework.

4.2 State-Space Relaxation

Even with the proposed DP scheme it is possible to have a large number of dominating states. If we have a large number of attributes for each state, it is likely that we will have a large number of possible sets of attribute values being passed to a child state (i.e. $|\Phi^{z_k,l}|$ is large). Hence, in this case we would expect a large number of dominating states. However, we can relax our definition of dominating states by introducing the ideas of dominating attributes.

Certain attributes have a dominating characteristic such that a state is considered of higher quality if that attribute had a larger value. Furthermore, since a larger value for a dominating attribute is considered better, there is no upper bound imposed for that attribute. For example, two roster-lines have exactly the same cost and attribute values except for the number of weekends off. In this case we can say that the roster-line with greater number of weekends off dominates the other roster-line since it is of better quality and is less likely to cause future roster-lines to be infeasible. Hence, we can consider a state to dominate other states if all non-dominating attributes are equivalent, all dominating attributes are equivalent or greater in value, and it has lower cost.

We can further relax our definition of a dominating state by introducing the idea of tolerance. Two states might have different attribute values, however, we could still consider the states equivalent if the values are within a certain tolerance. By varying the tolerance of attributes during the DP we are able to limit the number of dominating states being created by controlling the size of $\Phi^{z_k,l}$. However, using tolerance opens the door to creating sub-optimal lines of work and hence a sub-optimal solution to the master problem.

5 References

- Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., Vance, P. H. 1998. "Branch and Price: column generation for solving huge integer programs." *Operations Research* **46**:316-329.
- Desrosiers, J., Dumas, Y., Solomon, M.M. and Soumis, F. 1995. "Time Constrained Routing and Scheduling." *Handbooks in Operations Research and Management Science*, **8**:35-130.
- Desaulniers, G., Desrosiers, J., Solomon, M.M. 1999. "Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems." Technical report G-99-36, GÉRAD.
- Ryan, D.M. 1992. "The solution of massive generalized set partitioning problems in aircrew rostering." *Operations Research* **43**:459-467.

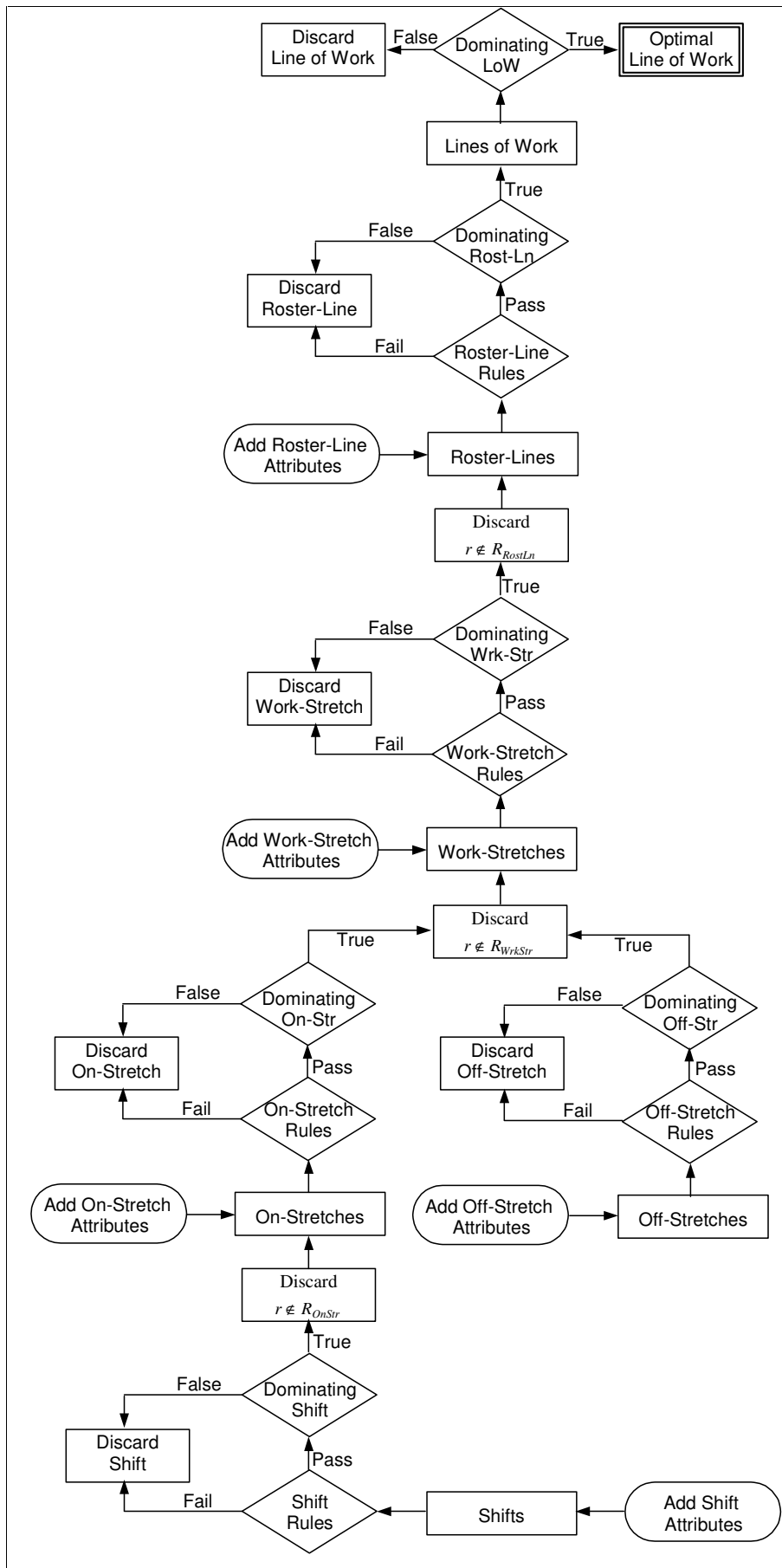


Figure 2 Summary of the proposed DP framework for a Generalised Rostering Problem