Cyclic Rostering Using Branch and Cut

Andrew J Mason
University of Auckland, Private Bag 92019
Auckland, New Zealand, a.mason@auckland.ac.nz
David Panton
School of Mathematics, University of South Australia
The Levels 5095, Australia, david.panton@UniSA.edu.au

Abstract

This paper considers the problem of building cyclic staff rosters. We present a new solution approach using integer programming with 3-way branching and cuts.

1 Introduction

This paper considers the problem of building staff rosters (also called personnel schedules) that repeat in a cyclic fashion. These rosters are found in a wide variety of industries that we have worked with, including ambulance operations, airport operations and hospitals.

In Table 1, we show a typical cyclic roster. This roster is composed of a sequence of activities. These are grouped into lines, where the Monday in a line is considered to follow the Sunday of the previous line. The roster is cyclic in that line 1 is considered to follow the last line, line 3.

For each line, the roster specifies an activity for each day of the week, being either a day off work (denoted by '–'), or one of the two shift types (`A` or `B`) that have to be worked on that day. Each shift type specifies a start and finish time for the work activity; these are detailed in Table 2.

This roster would be worked by 3 staff members, where the staff members follow through the roster lines in Table 1. For example, the roster could begin with staff members 1 to 3 working lines 1 to 3 respectively. That is, on Monday, people 1 and 2 both work an `A` shift type, while person 3 works a `B` shift. On Tuesday, persons 1 and 2 work `A` shifts, while person 3 has a day off. This pattern continues for 7 days, after which the staff members move down to the next line, with the last staff member (person 3) coming back to the top of the roster (line 1). So, in the second week of operation, person 1 would work line 2, person 2 line 3, and person 3 line 1. After 3 weeks of operation, a complete cycle has been worked, and so on week 4, all staff work the same shifts as on week 1.

The lower half of Table 1 shows the total 'coverage' provided by this roster. For example, on every Monday we have 2 staff working `A` shifts, and 1 working a `B` shift.

Line No	M	Tu	W	Th	F	Sa	Su
1	A	A	B	B	B	-	A
2	A	A	A	-	-	B	B
3	B	-	-	A	A	-	-
'A' totals	2	2	1	1	1	0	1
'B' totals	1	0	1	1	1	1	1

Table 1: A 3-week cyclic roster with 2 shift types, A and B.

Shift Type	**Start Time**	**End Time**
A	07:00	15:00
B	15:00	23:00

Table 2: Start and finish times for shift types A and B

This required coverage – often called the 'shift requirements' or 'work requirements' – is typically specified by management. We will assume that the requirement is exact; providing more staff or fewer staff than required on any shift is not acceptable.

It can be useful to consider the cyclic roster as a sequence of 'workstretches', where we define a workstretch to be a continuous sequence of days worked and the associated following days off. The sample roster in Table 1 contains 4 workstretches; these are listed in Table 3. In this table, we have shown the number of 'days-on' (days worked) and 'days-off' (days off work) each workstretch contains. We have also shown the length of the workstretch, being the number of days that are specified by the workstretch.

There are a number of 'quality objectives' that staff seek to optimise when building cyclic rosters. First, staff like to work favourable workstretches. For example, a 5-on, 2-off workstretch will generally be preferred to a 7-on, 1-off workstretch. Second, staff typically dislike split weekends in which one day in the weekend is worked, and the other is a day off. Third, staff will generally express preferences about the content of a workstretch. Workstretches with only one shift type, such as '`A A A A` – –' are typically (but not always) preferred to workstretches such as '`A A B B` – –' which contain a mix of shift types. Where workstretches are of mixed shift type, staff often prefer the shifts to be 'forward rotating' meaning that each successive shift starts at a later (not earlier) time. For example, '`A A B B` – –' is forward rotating, while '`B B A A` – –' is not.

In addition to the quality objectives, a cyclic roster typically has to satisfy a

Work-stretch	Sa	Su	M	Tu	W	Th	F	Sa	Su	Days on	Days off	Length
1			A	A	B	B	B	-		5	1	6
2		A	A	A	A	-	-			4	2	6
3	B	B	B	-	-					3	2	5
4						A	A	-	-	2	2	4

Table 3: The workstretches within the sample roster

number of 'legality constraints'. For example, union regulations may specify that no more than 7 days may be worked consecutively; that at least 2 days off follow 5 or more consecutive days of work; that no more than 10 days be worked in any 2 weeks, and so on.

The 'cyclic roster construction problem' can now be defined as that of generating the highest quality cyclic roster that meets all legality constraints while covering the work requirements with the specified number of roster lines.

2 Previous Work

Much of the previous work on cyclic rostering has focussed on a heuristic decomposition of the problem into distinct steps. As is observed in [2], there is no guarantee that this heuristic decomposition will lead to optimal solutions for the rostering problem. Instead, the emphasis in more recent work has been on finding solution approaches that avoid the need to arbitrarily split the problem into stages and instead treat the complete problem in one process. These approaches are based on both integer programming [2, 3] and shortest path algorithms [1].

Laporte, Nobert and Biron [2] develop an integer programming model in which the columns represent workstretches from which a set is selected that satisfies constraints representing the work requirements and the availability of days off. They also add a constraint that forces the number of weekends off to be maximised. An interesting feature of their model is that while the workstretches they generate satisfy the work requirements, often they cannot be sequenced to form a complete cyclic roster. This is because the solution contains 'disconnected sub-cycles' in the form of cyclic sub-rosters of less than the required length. To avoid such solutions, additional constraints are introduced during the branch and bound that ban solutions with disconnected sub-cycles. The authors report fast solve times on most of their problems, but fail to solve some instances. Other work includes Balakrishnan and Wong [1] who focus on a shortest-path view of the problem with Lagrange multipilers being used to handle the work requirements constraints. Rosenbloom and Goertzen [3] adopt a different decomposition in which the roster is constructed using integer programming to piece together weekly activity sequences. They focus on feasibility, ignoring roster quality issues.

3 Integer Programming Formulation

We will assume that all roster legality and quality issues can be treated at the workstretch level. That is, any sequence of workstretches (assuming the workstetches are themselves legal) is considered to be a legal (partial-) roster, and the cost of the roster is given by the sum of the workstretches it contains. This assumption allows us to fully exploit workstretch decomposition.

We present a model similar to Laporte et al's, but will concentrate on problems with many more workstretch combinations than seen in previous work. In particular, we wish our models to be suitable for subsequent later work on adding column generation to the system.

		Workstretch: 1	2	3	4	Coverage	Columns x_1	x_2	x_3	x_4		RHS
Weekday	M	A	A	B		3	1	1	1			3
	Tu	A	A	-		3	1	1	1			3
	W	B	A	-		3	1	1	1			3
	Th	B	-		A	3	1	1		1		3
	F	B	-		A	3	1	1		1		3
	Sa	-		B	-	3	1		1	1		3
	Su		A	B	-	3		1	1	1		3
A coverage	M	1	1			2	1	1			$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} =$	2
	Tu	1	1			2	1	1				2
	W		1			1		1				1
	Th				1	1				1		1
	F				1	1				1		1
	Su		1			1		1				1
B coverage	M			1		1			1			1
	W	1				1	1					1
	Th	1				1	1					1
	F	1				1	1					1
	Sa			1		1			1			1
	Su			1		1			1			1

Figure 1: A weekly-cyclic model for the cyclic rostering problem.

We motivate our model using Figure 1. The left table in the figure shows our example roster solution, while the right table show this solution within the integer programming model. Notice that the workstretches are defined on weekdays. If we consider the weekday coverage, each weekday appears 3 times in the roster (once in each roster line), and so the workstretches must collectively specify 3 activities for each weekday. The solution specifies workstretches and where they start in the week, but not how they should be sequenced to form the cyclic roster. In this example, it is easy to form a roster by sequencing the workstretches in the order 1, 2, 3, 4. To see this in more detail, we introduce the notation XY to mean day X in line Y of the roster. A sequenced solution (one of several possible) gives workstretch 1 as specifying roster days Mo1→Sa1, workstretch 2 roster days Su1→Fr2, workstretch 3 days Sa2→We3, and finally workstretch 4 as specifying days Th3→Su3. This fully specifies all 21 days of the roster.

This model can be expressed more formally as follows. Let $\mathcal{W}$ be the ordered cyclic set of weekdays: typically $\mathcal{W} = \{M, Tu, W, Th, F, Sa, Su\}$,where $Su{+}1 = M$. We let $\mathcal{P}$ be the set of all legal 'weekday'-dated workstretches, where a workstretch is weekday-dated if the starting and ending weekdays are specified. We define a_{wp} to be the number of times workstretch p specifies an activity that occurs on weekday w. Note that $a_{wp} \in \{0, 1\}$ for each weekday w in a workstretch p of length 7 days or less, but we will have $a_{wp} = 2$ (or greater) for some w in longer workstretches. It is useful to consider a shift to be defined by both its shift type and the weekday it occurs on; we let $\mathcal{S}$ denote the set of these shifts (weekday shift-type pairs) that are required in the roster. In the example above, $\mathcal{S} = \{$M-`A`, Tu-`A`, W-`A`, Th-`A`, F-`A`, Su-`A`, M-`B`, W-`B`, T-`B`, F-`B`, Sa-`B`, Su-`B`$\}$; note that in this example, shifts Sa-`A` and Tu-`B` are not required in the formulation. Let b_s be the number of staff required to work shift $s \in \mathcal{S}$, eg $b_{\text{M-A}} = 2$, $b_{\text{Tu-A}} = 1$ etc. We let $L = 3$ (in this example) be

the number of lines in the roster. We can now formulate our weekly-based cyclic rostering problem:

$$
\begin{array}{llllll}
\text{WbCRP:} & \text{minimise} & \sum_{p \in \mathcal{P}} c_p x_p & & & \\
 & \text{subject to} & \sum_{p \in \mathcal{P}} a_{wp} x_p = L & \forall & w \in \mathcal{W} & \text{`}\textit{weekday coverage}\text{'} \\
 & & \sum_{p \in \mathcal{P}} e_{sp} x_p = b_s & \forall & s \in \mathcal{S} & \text{`}\textit{shift coverage}\text{'} \\
 & & x_p \in \{0, 1, 2, 3, ...\} & \forall & p \in \mathcal{P} & \text{`}\textit{shift coverage}\text{'}
\end{array}
$$

The WbCRP model is equivalent to the Laporte et al model [2]. However, unlike WbCRP, Laporte et al do not include any 'weekday coverage' constraints, but instead require the correct number of days off to occur on each weekday. Laporte et al also include a constraint that forces the maximum number of possible weekends off to be included in the solution. The 'weekends off' constraint is not part of our standard problem formulation. Rather than forcing the number of weekends off to be maximised, we reflect the benefits of having weekends off in the objective function. Our early numerical experiments suggested that this change makes for a more realistic but harder problem.

4 Removing Disconnected Sub-Cycles

The weekday-based formulation WbCRP can allow 'disconnected sub-cycles' to appear in the solution. For example, consider the feasible solution shown in Figure 2. This solution contains 5 workstretches that need to be sequenced to form a 3-week roster. Workstretch 1 determines roster days Mo1→Th1, workstretch 2 roster days F1→Tu2, and workstretch 3 roster days W2→F2. However, neither workstretch 4 nor 5 allows a continuation from this point, as neither of these workstretches commences on a Monday, and hence cannot start on the required next roster day M3. This solution contains 2 disconnected (incompatible) cycles, and is actually two separate cyclic 'sub-rosters' (one with 2 week's worth of lines, the other with just one) that together meet the work requirements. Such 'disconnected' solutions do not meet our requirements for cyclic rosters.

When analysing solutions for disconnected sub-cycles, it is useful to construct a graph where each weekday is represented by a node, and workstretches are represented by arcs. A workstretch of length l starting on weekday i is represented by an arc that starts on weekday i and finishes on weekday $i + l$. Figure 3 shows the graph representation of the disconnected sub-cycle solution of Figure 2. We note that it is not possible to form a eulerian tour on this graph, and so the solution is illegal as there is no ordering of the workstretches that allows them to be worked in succesion by the staff members. We observe that cycles within a roster are not in themselves a problem. The difficulties only arise when the cycles are disconnected in that they do not share any workstretch start days in common. Figure 3 shows a second solution with sub-cycles that share a common node, and hence give connected solution. A eulerian tour can be formed on this graph, and so the workstretches can

		Workstretch: 1	2	3	4	5	Coverage	Columns x_1	x_2	x_3	x_4	x_5		RHS
Weekday	M	A	-			A	3	1	1			1		3
	Tu	A	-			A	3	1	1			1		3
	W	A		A		-	3	1		1		1	x_1	3
	Th	-		A	A		3	1		1	1		x_2	3
	F		A	A	A		3		1	1	1		x_3 =	3
	Sa		A	-	-		3		1	1	1		x_4	3
	Su		A	-		A	3		1	1		1	x_5	3
A coverage	M	1				1	2	1				1		2
	Tu	1				1	2	1				1		2
	W	1		1			2	1		1				2
	Th			1	1		2			1	1			2
	F		1	1	1		3		1	1	1			3
	Sa		1				1		1					1
	Su		1			1	2		1			1		2

Figure 2: An example solution to WbCRP that contains disconnected sub-cycles. Note that this example contains only a single shift type and different work requirements to those shown before.

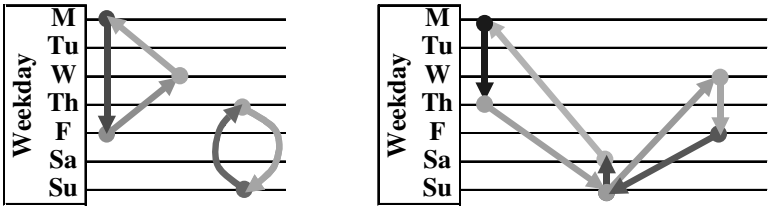

Figure 3: Graph representations of solutions with sub-cycles that are disconnected (left) and connected (right). The nodes in each graph are shown as horizontal lines.

be sequenced to form a legal cyclic roster. This will always be possible if (and only if) the set of weekdays on which workstretches start are 'strongly connected,' i.e. if there is a path (that respects arc directions) between any pair of workstretch-start weekdays.

We will see in our computational results that disconnected sub-cycles often arise in solutions to this model. To explain this, we note that the objective improvement often associated with weekends off can encourage the presence of sub-cycles in optimal solutions.

A possible method for the resolution of cycles in WbCRP solutions is to introduce cuts – extra constraints – that remove integer solutions if the solutions are not connected. This is the approach adopted by Laporte et al. Whenever they find an integer solution during their branch and bound process, they test it for disconnected sub-cycles. If the solution contains these, then they add a constraint (cut) to the problem and continue their branch and bound process. The constraint they add removes only the current integer solution; in this sense, it is the weakest of all possible cuts.

We now consider a new strategy for eliminating solutions that contain disconnected sub-cycles. As in the Laporte et al work, we will use branch and bound to

find integer solutions, and then test these for disconnected sub-cycles. If an integer solution contains disconnected sub-cycles, then we use a combination of branches and cuts to ban the solution. We motivate our strategy by examining in more detail the requirements for a solution to be connected.

Let T be a set of weekdays. For example, we may have $T = \{\text{Tu, Th, Sa}\}$. Now, we define the set $S_{T \leftrightarrow T}$ to be the set of all workstretches that can be combined to form rosters in which each workstretch starts on some weekday $w \in T$. Clearly this is the set of workstretches that start on a weekday $w_1 \in T$ and finish on a weekday $w_2 : w_2 + 1 \in T$. For our example $T = \{\text{Tu, Th, Sa}\}$, $S_{T \leftrightarrow T}$ is the set of all workstretches that start on either Tuesday, Thursday or Saturday, and finish on either Monday, Wednesday or Friday. Of the workstretches shown in Table 3, only the third workstretch ('B B B – –', Saturday to Wednesday) belongs to $S_{\{\text{Tu,Th,Sa}\} \leftrightarrow \{\text{Tu,Th,Sa}\}}$.

If we consider the solution to the weekday model shown in Figure 2, we note that the solution contains two sub-cycles. The first has shifts that start on either Monday, Wednesday or Friday and finish on either Sunday, Tuesday or Thursday. Thus, these workstretches all belong to $S_{T \leftrightarrow T}$, where $T = \{\text{Mo, We, Fr}\}$; we say that the first sub-cycle is contained within $T = \{\text{Mo, We, Fr}\}$. Because we are interested in strict partitionings of the weekdays W, it is convenient to consider the second sub-cycle to be contained within $T' = \{\text{Tu, Th, Sa, Su}\}$. Note that we could, if we preferred, consider the first cycle to be in the larger set $T = \{\text{Mo, Tu, We, Fr}\}$, in which case we would consider the second sub-cycle to be in $T' = \{\text{Th, Sa, Su}\}$. Under either choice of T, the problem with this solution is that the sub-cycles are contained within these disjoint sets of weekdays; the two cycles have no workstretch start days in common, and so the workstretches cannot be sequenced to form a cyclic roster.

Consider now how we might change the solution in Figure 2 so that the workstretches can be sequenced to form a complete roster. If the solution is to contain workstretches from both $S_{T \leftrightarrow T}$ and $S_{T' \leftrightarrow T'}$, then it must necessarily be modified to also contain some workstretch (or sequence of workstretches) that links T and T', that is a workstretch (or workstretch sequence) that starts on some weekday $w_1 \in T$ and finishes on some weekday $w_2 : w_2 + 1 \in T'$. Furthermore, if the roster contains such a link from T into T', then it must necessarily also contain a link from T' back into T.

We define $S_{T \rightarrow T'}$ to be the set of all workstretches that have a start day $w_1 \in T$ and an end day $w_2 : w_2 + 1 \in T'$. These workstretches can be thought of as leaving T and entering T'. Similarly, we define $S_{T \leftarrow T'}$ to be the set of workstretches that have a start day $w_1 \in T'$ and an end day $w_2 : w_2 + 1 \in T$; these workstretches enter T from T'. It is convenient to also define $S_{T \leftrightarrow T'} = S_{T \rightarrow T'} \cup S_{T \leftarrow T'}$. The following results are easy to prove.

Lemma 1 *For any $T \subset W$, a workstretch in $S_{T \leftrightarrow T}$ must be followed in a solution roster by a workstretch belonging to $S_{T \leftrightarrow T}$ or $S_{T \rightarrow T'}$. Similarly, a workstretch in $S_{T' \leftrightarrow T'}$ must be followed by a workstretch belonging to $S_{T' \leftrightarrow T'}$ or $S_{T \leftrightarrow T'}$.*

Theorem 2 *A connected solution with workstretches from both $S_{T \leftrightarrow T}$ and $S_{T' \leftrightarrow T'}$ must necessarily contain at least 1 workstretch from each of $S_{T \rightarrow T'}$ and $S_{T \leftarrow T'}$.*

Lemma 3 *Consider a (possibly fractional) solution* $\mathbf{x}$ *to WbCRP. For any non-empty* T *and* T', $\sum_{p \in S_{T \to T'}} x_p = \sum_{p \in S_{T \leftarrow T'}} x_p$

Corollary 4 *Consider some non-empty* $T \subset W$ *and its non-empty complement* T'. *Now, any cyclic rostering solution which allows a single cycle to be built (i.e. does not contain disconnected sub-cycles) must either: (case 1) contain no workstretches from* $S_{T \leftrightarrow T}$, *or, (case 2) contain no workstretches from* $S_{T' \leftrightarrow T'}$ *or, (case 3) contain at least one workstretch from each of* $S_{T \to T'}$ *and* $S_{T \leftarrow T'}$.

We can use the above results to form a 3-way branch that can be used to remove disconnected subcycles from solutions. Assume we have a solution with workstretches that create (at least) two disconnected subcycles that can be contained within the two non-empty sets $T \subset W$ and its complement $T' \subset W$ respectively. We can form three sub-problems by either (1) banning all columns belonging to $S_{T \leftrightarrow T}$, or (2) banning all columns belonging to $S_{T' \leftrightarrow T'}$, or (3) forcing at least one column from $S_{T \to T'}$ (or $S_{T \leftarrow T'}$) to appear in the solution. This gives a 3-way branch; we term the three branches [BanT], [BanT′], and [Join] respectively. Sub-problems (1) [BanT] and (2) [BanT′] can be formed by reducing the upper bounds on the deleted columns to zero, while sub-problem (3) [Join] requires the addition of an extra constraint requiring $\sum_{p \in S_{T \to T'}} x_p \geq 1$. Clearly the original solution will be infeasible for each of these sub-problems. Note, however, that these branches do not partition the solutions; the same solution may appear under several branches.

Note that this new strategy defines column bans and row cuts in terms of features of the workstretch columns. Therefore, it is easy to respect the branches and cuts within a column generator. This was not the case with the cut of Laporte et al where the participation of a column in a cut depends not on the features of the column but instead on the sequence of integer solutions found so far in the branch and bound process. We now have a new approach that is 'generator friendly', and so we can, in the future, develop column generation strategies for tackling much more complex problems.

5 Aggregate Feature Branching

For these cyclic rostering problems, we will show through numerical experiments that it is useful to branch on aggregate features that are not counted explicitly in the model. The aggregate features that we have identified as useful are: (1) the number of workstretches in the solution, (2) the the number of full weekends (Sat and Sun) off in the solution, and (3) the number of workstretches beginning on each weekday. We add 9 'aggregate features' decision variables (number of workstretches, number of weekends, number of starts on Mon, number of starts on Tues, ..., number of starts on Sun) to the model, along with 9 constraints that link the values of these variables to the sum of the appropriate workstretch features. For example, the workstretch summation constraint is given by $\sum_{p \in \mathcal{P}} x_p - x_{workstretches} = 0$ where as before, $\mathcal{P}$ is the set of workstretches, and x_p gives the number of times workstretch p occurs in the solution. The weekends-off summation constraint is given by $\sum_{p \in \mathcal{P}} q_p x_p - x_{weekends} = 0$

where $q_p = 1$ if workstretch p contains a complete weekend off, and 0 otherwise. The weekday-start variables and associated constraints are of the same form. Clearly, each of these new variables must be integer in any feasible solution. Hence, standard variable branching can be applied to these variables. These branches effectively apply local cuts to the solution. We will see in the numerical results section that the use of aggregate feature branching appears to improve solution times for many problems.

As with the branches and cuts presented in the last section, we note that the coefficients in the new aggregate feature constraints are defined in terms of features of the workstretch columns. Therefore, it is easy to incorporate these new constraints within a column generator.

6 Numerical Results

A range of experiments were performed in which various cyclic rostering problems were solved using the WbCRP model both with and without our new branching strategy. All runs were performed on a 400 MHz Pentium II machine with 128 Megabytes of RAM running Windows NT. The solver was CPLEX 6.5 using 'BB++', a customised C++ branch and bound engine developed by the author. Variable branching was used in all problem runs.

Results for a range of problems solved using WbCRP are given in Figure 4. The problems are similar to those considered by Laporte et al. However, as exact data is not available for these problems (Laporte, personal communication, 1999), they have been 'reconstructed' from summary data provided by Laporte et al in their paper.

First, we note that results are not shown for four of the problems we considered as the LP solutions for these problems were naturally integer and connected. For the remaining problems, we see that where the problem is 'hard', the 'BC+' setting choice – our new branch and cut approach combined with aggregate feature branching – appears to be the best. Indeed, this approach is 300 times faster for one of the problems. For one problem a slight worsening of performance is observed using the new approach. However, this is minor when compared with the benefits provided on the hard problems.

7 Conclusions

We have provided a brief survey of existing and new models for the cyclic rostering problem. We have developed a new branch and cut scheme that is unusual in using a 3-way branch instead of the usual binary branching strategies. We have also defined aggregate variables branches for this problem and shown that when combined with our new branch and cut scheme, big improvements in solution times are obtained for a number of standard problems. We are thus able to recommended that this new approach be adopted for future work in cyclic rostering. In particular, we have laid the foundations for the development of a column generation solution approach for this problem.

Prob	Time	BestSoln	BestSolnTime	Optimal?	Nodes	StillLiveNodes	IntSolns	OptIntSolns	NodesBounded	InfeasibleNodes	Tol	VarBranches	AggVarBranches	Cuts	VarBans
PA_Lap	0.03	400	0.03	Optimal	5		1	1	2		1	4			
PA_Lap+	0.02	400	0.02	Optimal	5		1	1	2		1	4			
PA_BC	0.03	400	0.03	Optimal	5		1	1	2		1	4			
PA_BC+	0.03	400	0.03	Optimal	5		1	1	2		1	4			
PD_Lap	36.04	370	36.04	Optimal	2961		5	1	678	550	1	2464		496	
PD_Lap+	36.41	370	36.41	Optimal	1752		6	1	307	354	1	1248	84	419	
PD_BC	0.29	370	0.29	Optimal	57		2	1	21	8	1	44		4	8
PD_BC+	0.12	370	0.12	Optimal	25		2	1	9	3	1	14	4	2	4
PE_Lap	2.71	640	1.65	Optimal	262		2	1	125	4	1	260		1	
PE_Lap+	5.53	640	5.53	Optimal	587		1	1		293	1	564	22		
PE_BC	3.06	640	1.92	Optimal	280		2	1	136	3	1	276		1	2
PE_BC+	5.47	640	5.47	Optimal	587		1	1		293	1	564	22		
PL1_Lap	0.01	10	0.01	Optimal	5		1	1	2		1	4			
PL1_Lap+	0.02	10	0.02	Optimal	9		1	1	4		1		8		
PL1_BC	0.01	10	0.01	Optimal	5		1	1	2		1	4			
PL1_BC+	0.02	10	0.02	Optimal	9		1	1	4		1		8		
PB_Lap	466.82	190	123.08	Optimal	11830		9	1	5098	582	1	11376		453	
PB_Lap+	3600.12	200	262.51	SubOpt	20972	27	2	1	5435	3697	1	16190	2130	2651	
PB_BC	974.31	190	821.59	Optimal	26478		6	1	11765	1471	1	26462		5	10
PB_BC+	73.34	190	73.08	Optimal	1494		7	1	329	416	1	1400	66	9	18

Key:

Prob	Problem & solution method: Lap=Laporte cuts, BC=new branch/cut,+=aggregate branching
Time	Time to solve Branch and Bound (seconds). LP solve times are negligible.
BestSoln	Objective value of best solution found
BestSolnTime	Time (s) to find the best solution
Optimal?	Is the solution proven optimal, or was the branch and bound terminated early?
Nodes	Number of nodes in the branch and bound tree
StillLiveNodes	Number of nodes not yet fathomed
IntSolns	The number of connected integer solutions founds
OptIntSolns	The number of optimal connected integer solutions founds
NodesBounded	The number of nodes in the tree bounded by a better objective
InfeasibleNodes	The number of nodes that were infeasible
Tol	The bounding tolerance (1% for all problems)
VarBranches	The number of nodes formed by variable branching on workstretch variables
AggVarBranches	The number of nodes formed by variable branching on aggregate feature variables
Cuts	The number of cuts added in the tree (either Laporte cuts or our new cut)
VarBans	The number of nodes formed by banning variables in either S_T or $S_{T'}$

Figure 4: Results for a range of test problems.

References

[1] Nagraj Balakrishnan and Richard T. Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20:25–42, 1990.

[2] Gilbert Laporte, Yves Nobert, and Jean Biron. Rotating schedules. *European Journal of Operational Research*, 4:24–30, 1980.

[3] E. S. Rosenbloom and N.F. Goertzen. Cyclic nurse scheduling. *European Journal of Operational Research*, 31:19–23, 1987.