A hybrid MIP/CP approach for multi-activity shift scheduling

Domenico Salvagnin[1] and Toby Walsh[2]

[1] DEI, University of Padova
[2] NICTA and UNSW, Sydney

Abstract. We propose a hybrid MIP/CP approach for solving multi-activity shift scheduling problems, based on regular languages that partially describe the set of feasible shifts. We use an aggregated MIP relaxation to capture the optimization part of the problem and to get rid of symmetry. Whenever the MIP solver generates a integer solution, we use a CP solver to check whether it can be turned into a feasible solution of the original problem. A MIP-based heuristic is also developed. Computational results are reported, showing that the proposed method is a promising alternative compared to the state-of-the-art.

1 Introduction

A shift scheduling problem assigns a feasible working shift to a set of employees, in order to satisfy the demands for a given set of activities at each period in a given time horizon. The set of feasible shifts that can be assigned to employees is often defined by a complex set of work regulation agreements and other rules. Assigning a shift to an employee means specify an activity for each period, which may be a working activity or a rest activity (e.g., lunch). The objective is usually to minimize the cost of the schedule, which is usually a linear combination of working costs plus some penalties for undercovering/overcovering the demands of the activities in each time period. If the set of working activities W is made by a single activity, we talk of *single-activity* shift scheduling, while if there are several working activities we talk of *multi-activity* shift scheduling. In this paper we consider the latter case, with the additional constraint that all employees are identical.

In particular, suppose we are given a planning horizon divided into a set of periods T, a set of activities A, a subset $W \subset A$ of working activities, and a set of employees E. For each period $t \in T$ and for each working activity $a \in W$, we are given a demand d_{at}, an assignment cost c_{at}, an undercovering cost c^-_{at} and an overcovering cost c^+_{at}. Introducing the set of integer variables y_{at}, which count the number of employees assigned to activity a at period t, and integer variables s^-_{at}, s^+_{at} that count the appropriate under/over covering, we can formulate the problem as:

$$\min \sum_a \sum_t c_{at} y_{at} + \sum_a \sum_t c^+_{at} s^+_{at} + \sum_a \sum_t c^-_{at} s^-_{at} \tag{1}$$

$$y_{at} - s^+_{at} + s^-_{at} = d_{at} \quad \forall a \in W, \forall t \in T \tag{2}$$

$$\sum_e x_{eat} = y_{at} \quad \forall a \in W, \forall t \in T \tag{3}$$

$$\langle x \text{ define a feasible shift } \forall e \in E \rangle \tag{4}$$

$$y_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{5}$$

$$x_{eat} \in \{0, 1\} \tag{6}$$

Depending on how we formulate the constraints (4), we may end up with very different models. A convenient way to define the set of feasible shifts that can be assigned to a given employee is to use a regular or a context-free language, i.e., the set of feasible shifts can be viewed as the words accepted by a finite automaton or more generally by a push-down automaton. It has been shown in [14, 3] that it is possible to derive a polyhedron that describes a given regular/context-free language. Such representations are compact (in an appropriate extended space, i.e., introducing additional variables) and thus lead directly to a MIP formulation of the problem. In particular, the extended formulation for a regular language is essentially a network flow formulation based on the expanded graph associated with the accepting automaton (see [13, 3] for details). The extended formulation for the context-free language, on the other hand, is based on an and-or graph built by the standard CYK parser [10] for the corresponding grammar [14, 16].

Note that it is not necessary to describe completely the set of feasible shifts by a regular/context-free language. The formal language may capture only some of the constraints defining a feasible shift, with the remaining ones described as linear inequalities. This may simplify the corresponding automaton considerably. For example, regular languages are notoriously bad at handling counting arguments, and an automaton describing the set of feasible shifts completely in the presence of even a few cardinality constraints may require thousands of states. Such large automaton are not trivial to generate. This also has a direct influence on the size of the model and the efficiency of reasoning about it. The same holds for context free languages. It is true that they can be enriched considerably by adding constraints that limit the applicability of productions rules, without even increasing the size of the model. However, certain cardinality constraints may overly complicate the language. Finally, depending on the application, the model derived using a context free language may be much bigger than an equivalent one derived using a regular language.

However, describing the set of feasible shifts with formal languages alone has some important implications. First of all, it has been proven for both the regular and context-free languages that the derived polyhedron is integral [14], and thus, if the are no other constraints, it is possible to optimize a linear function over the set of feasible shifts by solving just a linear program. Even more importantly, these results have been extended also to polyhedra describing

sets of feasible shifts [4]. It is then possible to consider an aggregated (*implicit*) model and reconstruct an optimal solution of the original one with a polynomial post-processing phase. This gives the current state-of-the-art for solving multi-activity shift scheduling problems. Finally, if the formal language completely describes the set of feasible shifts, then it is possible to apply some very effective large neighborhood search heuristics to find quickly high quality solutions [17].

2 A hybrid MIP/CP approach

The explicit MIP model based on formal languages mentioned in the previous section has two drawbacks. First of all, its size is directly proportional to the number of employees in the instance; given that thousands of variables may be needed to completely describe the set of feasible shifts for a single employee, the linear programming relaxation can quickly become the bottleneck for branch-and-cut algorithm solving the instance. Second, with interchangeable employees, the enumeration itself explodes because of symmetry issues. In other words, the explicit model scales very badly as the number of employees increases.

A recently developed technique in the MIP community to deal with symmetric instances is called *orbital shrinking* [6], which is closely related to the implicit model mentioned earlier. The basic idea behind orbital shrinking is, given an orbit partitioning of the variables of a problem, to aggregate the variables within any orbit and consider the derived shrinked model on these aggregated variables, which is at the same time smaller and symmetry free. In the case of scheduling, where the symmetries are due to the interchangeable employees, this procedure automatically produces the implicit model used in [4]. Note that orbital shrinking is an exact reformulation only for convex optimization problems. On the one hand, this means that the LP relaxation of the shrinked model yields the same dual bound as the LP relaxation of the explicit model. On the other hand, given an arbitrary MIP, such reformulation is in general only a relaxation, although it can be tighter and/or faster to compute than the LP relaxation (more on this in Section 3).

Interestingly, for some special cases, the orbital shrinking reformulation is exact also for the MIP problem. This happens, for example, whenever it can be proven that an optimal solution of the aggregated model can always be turned into a solution of the original model of the same cost (and thus optimal). Examples of this behaviour are the assignment polytope and the regular/context-free language polytopes.

Consider for example the regular polytope in its extended form: the optimal solution is always a flow of integral value, say k, and basic network flow theory guarantees that it can be decomposed into k paths of unitary flow (and since each path in the expanded graph corresponds to a word in the language, this is a feasible solution for the original explicit problem). Similar reasoning applies to the grammar polytope (although it is not a flow model), as successfully shown in [4].

Unfortunately, it is not always reasonable to describe the set of feasible shifts completely with a formal language. While it is true that formal languages can be extended without changing the complexity of the corresponding MIP encoding (this is particularly true for context-free languages [16]), still some cardinality constraints may be very awkward to express, as shown in the following example:

Example 1. Let's consider a time horizon of 18 hours, divided into 18 periods. A feasible shift is a word of length 18 build from the alphabet $\Sigma = \{a, b, r\}$ (where a denotes the only working activity, r is a rest period, while b is a break period) that follows the pattern *rest-work-break-work-break-work-break-work-rest*. Suppose that the breaks are constrained to be one period long, and the number of working periods must be between 6 and 8. Then, a very simple grammar encoding the set of feasible shifts, ignoring the cardinality constraint, is:

$$S \to RFR \quad F \to PBP \quad P \to WBW$$

$$R \to Rr|r \quad W \to Wa|a \quad B \to b$$

In this particular case, since the number of breaks in the shift is fixed (3), it is very easy to extend the grammar to deal with the cardinality constraint by restricting the production rule $F \to PBP$ to be applied only with substring of length between 9 and 11. This can be handled very well by the CYK parser, and thus the cardinality constraint can be added essentially at no cost.

However, let's consider a slightly more complicated case. The pattern of a feasible shift is the same, but now the length of breaks is *not* fixed to one. In particular, the number of break periods is constrained to be between 4 and 6. The best we can do keeping approximately the same grammar as before is the following (we use the notation of [4] to indicate restrictions on production rules):

$$S \to RFR \quad F_{[10,14]} \to PBP \quad P_{[3,10]} \to WBW$$

$$R \to Rr|r \quad W \to Wa|a \quad B \to Bb|b$$

It is easy to see that the restrictions cannot be tightened any further, otherwise we may lose feasible shifts. However, the grammar also accepts the substring $rrababab\mathit{bbbbbbaarr}$, which violates both cardinality constraints. □

Of course the issues of the previous example are not theoretical. As the set of feasible shifts is finite, there always exists a regular/context-free language that describes that set. However, the corresponding automaton may be unreasonably large in practice.

In order to turn the orbital shrinking approach into a complete method for the multi-activity shift scheduling problem when the formal language does *not* completely describe the set of feasible shifts, we propose a hybrid MIP/CP approach based on decomposition. In particular, whenever the MIP solver generates an integer feasible solution of the aggregated model, we must check whether it can be turned into a feasible solution of the explicit model. Because orbital shrinking always aggregates variables with the same costs (otherwise they would

not be on the same orbit), this is indeed a pure feasibility problem. As such, we propose to formulate the check as a CSP problem, to be solved with a CP solver. In this way, we not only avoid solving the LP relaxations (that would provide no meaningful bounds), but we can explicitly state symmetry breaking constraints. Note that this is essentially a master/slave decomposition similar to a generalized Benders method, where the master problem is a MIP model, while the slave is a CP model.

2.1 MIP model

The MIP model that we use is a simple modification of the general model hinted at in Section 1. The main difference is that we partition the set of feasible shifts Ω into k subsets Ω_k, each of which is described by a potentially different deterministic finite automaton (DFA) and cardinality constraints. This partition can simplify a lot the structure of the DFAs, and in general makes the implicit model more accurate, since the cardinality constraints are aggregated only within employees of the same "kind". This of course increases the size of the relaxation, but since the aggregated model is quite compact, this is usually well worth it. For each shift type Ω_k, the MIP model decides how many employees are assigned a shift in Ω_k, and then computes an aggregated integer flow of the same value. In details:

$$\min \sum_k \sum_a \sum_t c_{at} y^k_{at} + \sum_a \sum_t c^+_{at} s^+_{at} + \sum_a \sum_t c^-_{at} s^-_{at} \tag{7}$$

$$\sum_k y^k_{at} - s^+_{at} + s^-_{at} = d_{at} \quad \forall a \in W, \forall t \in T \tag{8}$$

$$\text{regular}(y^k, w^k, \text{DFA}^k) \quad \forall k \in K \tag{9}$$

$$\langle \text{cardinality constraints for } y^k \rangle \quad \forall k \in K \tag{10}$$

$$\sum_k w^k \leq E \tag{11}$$

$$w^k, y^k_{at}, s^+_{at}, s^-_{at} \in \mathbb{Z}^+ \tag{12}$$

Note that we use the notation of constraint (9) to refer to the extended MIP formulation of the regular constraint involving flow variables. The constraint ensures that variables y^k can be decomposed into w^k words accepted by the automaton DFA^k. Constraints (10) refers to the cardinality constraints expressed as linear constraints that complete the description of sets Ω_k. Finally, if an upper bound E is given on the number of employees that can scheduled, it can be imposed in constraint (11).

2.2 CP checker

The decision to partition the set of feasible shifts into k subsets Ω_k has an important consequence on the structure of the CP checker: the model actually

decomposes into k separate CP models, one for each type of shift. Given an index k, suppose the master (MIP) model assigns $\overline{w}^k$ employees, with their aggregated shifts described by $\overline{y}_{at}$. Then the corresponding CP model, which is similar to the one proposed in [5], reads:

$$\text{gcc}(x^e, \sigma^e, A) \quad \forall e \in 1, \ldots, \overline{w}^k \tag{13}$$

$$\tau^e = \sum_{a \in W} \sigma^e_a \quad \forall e \in 1, \ldots, \overline{w}^k \tag{14}$$

$$\langle \text{cardinality constraints for } \sigma^e, \tau^e \rangle \quad \forall e \in 1, \ldots, \overline{w}^k \tag{15}$$

$$\text{regular}(x^e, \text{DFA}^k) \quad \forall e \in 1, \ldots, \overline{w}^k \tag{16}$$

$$\text{gcc}(x_t, \overline{y}_t, A) \quad \forall t \in T \tag{17}$$

$$x^e \preceq x^{e+1} \quad \forall e \in 1, \ldots, \overline{w}^k - 1 \tag{18}$$

Variables x^e_t denote the activity assigned to employee e at time t. Variables σ^e_a count the number of periods assigned to each activity for employee e, while τ^e gives the sum over all working activities. Both are needed to specify the cardinality constraints (15). Constraints (17) link the variables in the CP model to the master solution $\overline{y}_{at}$. Finally, we impose a lexicographic order among the shifts of the employees with constraints (18).

The CP model above is usually extremely fast in proving whether the aggregated solution can be turned into a solution of the original. However, as the number of activities and employees increases, it can occasionally become very time consuming. The main reason for this behavior is the weak interaction between the cardinality constraints (17) and the symmetry breaking constraints (18). The issue can be easily explained with an example:

Example 2. Let's consider a vector of 5 binary variables $x_1, \ldots, x_5$, each with initial domain $\{0, 1\}$, and aggregate variables $y_0 = 2$ and $y_1 = 3$, linked with the x by a cardinality constraint. In addition, there are symmetry breaking constraints of the form $x_1 \leq x_2 \leq \cdots \leq x_5$. From the cardinality constraint point of view, any permutation of the solution $(0, 0, 1, 1, 1)$ is feasible, so no reductions are possible. The same happens from the symmetry breaking point of view, because symmetry breaking alone cannot exclude the assignments where all x variables take the same value (0 or 1). However, it is clear that the only solution feasible for both constraints together is indeed $(0, 0, 1, 1, 1)$. □

To overcome this issue, we implemented an ad-hoc propagator that implements a custom symmetry breaking strategy based on the cardinality constraints. This propagator handles the case in which there is a matrix of variables, with cardinality constraints in each column and symmetry on the rows (i.e., any permutation of the rows is feasible). As such, although it is very related to our CP model, it is not specific to our particular instances. The propagator works by partitioning the rows of the matrix into sets of identical rows (given the current domains). Then, for each set, it considers the first column with unassigned

variables. For each possible value v in the domains of this subset of variables, it computes a lower bound l_v and an upper bound u_v on the number of variables that must be assigned to v, by taking into account the cardinality constraints on the column and the domains of the variables. Finally, it assigns, for each value v, l_v variables to v. The complexity of such propagation is linear in the size of the matrix/domains. Note that this symmetry breaking strategy does *not* enforce constraints (18) on the rows of the matrix, and is not guaranteed to remove all possible row symmetries from the model, as shown in the following example.

Example 3. Suppose we have a cell with 5 rows and let $x_1, \ldots, x_5$ be the first 5 unassigned variables (one for each row). Suppose that we have 2 possible values, a and b, and that we can compute the following lower/upper bounds on the number of occurrences of these 2 values in the 5 vars:

$$l_a = 1 \quad u_a = 3 \quad l_b = 2 \quad u_b = 5$$

Before propagation, the domains of the 5 variables are all equal to $\{a, b\}$. After symmetry breaking we can reduce the domains to $\{a\}, \{b\}, \{b\}, \{a, b\}, \{a, b\}$. However, since it is possible to have both *abbab* and *abbba*, the propagator does not enforce a lexicographic order on the rows, and does not eliminates all symmetry. □

Another issue with the CP model above is that the minimum/maximum length of a working shift (i.e., the number of periods, breaks included, between the first and last working period) is constrained only implicitly by the regular constraints. Again, we implemented a custom propagator that deals with that. The combined effect of these propagators is impressive: we often observed reductions of 2−3 orders of magnitude in both the number of nodes and the running times on hard instances. On occasion, we observed even higher savings. For example, on one instance with 9 full-time employees and 3 activities, we reduced the running times from 6 minutes to 10^{-5} seconds, with a number of nodes dropping from 765,026 to 1.

Finally, the same model could be solved repeatedly if the aggregated solutions are too similar, i.e., if the values of the variables y^k coincide for some k. So, we implemented a "caching" mechanism that stores the last CP models and their status, in order to avoid solving the same model twice. According to our computational experience, the custom propagators and the cache were sufficient to keep the time spent within the CP solver negligible.

2.3 MIP repair/improve heuristic

Finding a good quality feasible solution early in the process is often crucial to effectively solve an optimization problem. However, dual decomposition strategies (such as generalized Benders decomposition and the hybrid approach presented here) are usually quite weak at finding (good) feasible solutions. In order to solve this issue, we devised an ad-hoc heuristic procedure for the shift scheduling problem.

The procedure is a simple generalization of the large neighborhood search developed in [17]. Suppose the set Ω can be completely described by a regular language. Then every feasible shift is a path in the expanded graph associated to the DFA and a solution to the problem is just a set of paths in this graph. Given a solution $S = \{s_1, \ldots, s_n\}$, let N_S be the set of solutions that can obtained by replacing a shift s_k with a different shift r. Given a choice s_k for the shift to remove, searching an improving replacement shift r can be formulated as a shortest path problem on the expanded graph, and N_S can be used as a neighborhood in the large neighborhood search heuristic (see [17] for details).

If Ω is not described completely by a regular language (the main assumption in the present work), then the search for an improving shift r cannot be formulated anymore as a shortest path. However, it can still be formulated as a MIP, to be solved by a black box MIP solver. This is the basic step of our heuristic, to be used in a scheme akin to the one in [17]. Note that basically the same MIP can be used to:

- find a feasible shift r to replace another feasible shift s_k.
- find a feasible shift r to replace an *infeasible* shift s_k.
- find a feasible shift r to add to current solution S (if doing so reduces the cost).

As such, the same heuristic can be used to (i) construct an initial feasible solution, (ii) improve a feasible solution, and (iii) repair an infeasible solution.

3 Computational Results

We tested our method on the multi-activity instances used in [3, 4, 17]. This testbed is derived from a real-world store, and contains instances with 1 to 10 working activities (each class has 10 instances). A basic description of the problem is as follows:

- The planning horizon of 1 day is divided into 96 slots of 15 minutes.
- A part-time employee must work a minimum of 3 hours and less than 6 hours, and is entitled to one break of 15 minutes.
- A full-time employee can work between 6 and 8 hours, and is entitled to two breaks of 15 minutes plus a lunch break of 1 hour (in any order).
- When an employee starts working on one activity, it must do it for at least 1 hour. In addition, a break/lunch is needed before changing activity.
- A break cannot be scheduling at the beginning/end of the shift.
- At specific times of the day (e.g., when the store is closed), no employee is allowed to work.
- Overcovering/undercovering is allowed, with an associated cost.
- The cost of a shift is the sum of the costs of all working activities performed in the shift.

We implemented our method in C++, using IBM ILOG Cplex 12.2 [11] as black box MIP solver, and Gecode 3.7.1 [7] as CP solver. All tests have been

performed on a PC with an Intel Core i5 CPU running at 2.66GHz, with 8GB of RAM (only one core was used by each process). Every method was given a time limit of 1 hour per instance. Concerning the set of feasible shifts Ω, we simply partitioned it into full-time and part-time shifts. We could have partitioned the full-time shifts further (depending on the relative order of breaks and lunch), but it seemed overkill because all full-time shifts share the same cardinality constraints (this was confirmed by some preliminary tests). In general, disaggregating shifts depending on the cardinality constraints seems to work well in practice.

From the implementation point of view, our hybrid method is made of the following phases:

- First, the aggregated model is solved with Cplex, using the default settings. The outcome of this (usually fast) first phase is a dual bound (potentially stronger than the LP bound) and the set of aggregated solutions collected by the MIP solver during the solution process (not necessarily feasible for the original model).
- We apply our MIP repair/improve heuristic to each aggregated solution which is within 20% of the aggregated model optimal solution. The outcome of this phase is always a feasible solution for the original model, thus a primal bound. Note that if the gap between the two is already below the 1% threshold, we are done.
- We solve the aggregated model again, this time implementing the hybrid MIP/CP approach. This means that we disable dual reductions (otherwise the decomposition would not be correct) and use Cplex callbacks framework to implement the decomposition.

Here is a more detailed description of the last phase. Whenever the MIP solver finds an integer solution, either with its own heuristics or because the LP relaxation happens to be integer, we build the corresponding CP models and solve them with Gecode DFS algorithm. As far as the branching strategy of the CP solver is concerned, after some trial-and-error we found that ranking the variables by increasing time period was the most successful policy. If the check is successful, then we update the incumbent, otherwise the solution is rejected. In both cases, we apply the MIP repair/improve heuristic on it to try to find a new incumbent. If the solution was the optimal solution of an LP relaxation, then we force a branching on a integer variable and keep going. As for branching inside the MIP solver, we let Cplex apply its own powerful strategy whenever the relaxation has some fractional variables. If this is not the case, we branch first on the w variables and then, if all w variables are already fixed, on the y, again ranking them by increasing time period. The rationale behind this strategy is that if the w variables are not fixed to some value, then we cannot even formulate the CP checking model, so the sooner we fix them the better. Note that as soon as the w variables are fixed, we can build a CP model akin to (13)-(18) where the y variables are not necessarily fixed but just take the domains of the current node. In this case, we let the CP solver run with a strict fail limit (1000 in our code) and, if it detects infeasibility, then we prune the node.

Table 1 reports a comparison between the proposed method and others in the literature, for a number of activities from 1 to 10. As far as the number of employees is concerned, we put an upper bound of 12 for instances with up to 2 activities, of 24 for instances with 3 to 8 activities and of 30 for instances with 9 or 10 activities. `cpx-reg` refers to the explicit model based on the regular constraint in [3], while `grammar` refers to the implicit model based on the grammar constraint in [4]. Note that for `grammar` we are reporting the results from [4], which were obtained on a different machine and, more importantly, with an older version of Cplex, so the numbers are meant to give just a reference. All methods were run to solve the instances to near-optimality, stopping when the final integrality gap dropped below 1%.

According to Table 1, `hybrid` outperforms significantly the explicit model `cpx-reg`, which, as already noted, scales very poorly because of symmetry issues and slow LPs. When compared to `grammar`, `hybrid` is very competitive only for up to 2 activities, while after that threshold `grammar` clearly takes the lead. This is somewhat expected: the set of feasible shifts in these instances can indeed be described without too much effort with an extended grammar, and it is no surprise that the pure implicit MIP model outperforms our decomposition approach. However, `hybrid` is likely to be the best approach if the extended grammar is not a viable option.

Table 1. Average computing times between the different methods to solve to near-optimality (gap $\leq 1\%$) the instances with up to 10 activities.

	# solved (10)			time(s)		
# act.	`cpx-reg`	`hybrid`	`grammar`	`cpx-reg`	`hybrid`	`grammar`
1	10	10	10	41.3	9.1	283.7
2	9	10	9	707.9	194.5	379.9
3	4	5	9	2957.3	1996.4	205.4
4	3	6	10	2970.2	1827.9	300.5
5	0	8	10	3600.0	1438.4	146.2
6	1	4	10	3530.6	2340.6	213.8
7	1	6	10	3438.7	2399.0	230.9
8	0	5	10	3600.0	2201.5	257.1
9	0	4	10	3600.0	2444.0	289.1
10	0	2	10	3600.0	3275.6	516.7

Table 2 reports a closer comparison between `cpx-reg` and `hybrid`, reporting the average final gap, average number of variables in the model and average node throughput for each category. According to the table, `hybrid` consistently yields very small gaps (always below 3% on average), while for `cpx-reg` is always above 60% with more than 4 activities. As far as the number of variables of the models is concerned, `hybrid` needs approximately 1/10 of the number of variables of `cpx-reg`, which promptly turns into a much faster node throughput: `hybrid` is

more than two order of magnitude faster in exploring nodes than `cpx-reg`. Note that, according to [4], `grammar` models range from 70,000 variables for instances with 1 activity to 96,000 for instances with 10 activities, so the hybrid model based on regular languages is significantly smaller.

Table 2. Comparison of average final gap between `cpx-reg and hybrid`.

	gap(%)		#vars		node/sec	
# act.	`cpx-reg`	`hybrid`	`cpx-reg`	`hybrid`	`cpx-reg`	`hybrid`
1	0.72	0.24	9,956	1,908	21.99	10.36
2	0.78	0.61	13,608	2,925	3.52	20.60
3	3.74	3.00	34,903	4,152	0.59	8.42
4	25.18	1.39	43,005	5,291	0.20	3.92
5	62.55	1.01	52,979	6,828	0.05	3.32
6	75.89	1.59	62,442	8,364	0.03	1.86
7	90.00	0.90	73,693	9,936	0.01	1.64
8	100.00	1.92	78,809	10,603	0.01	1.22
9	100.00	1.52	104,561	11,509	0.01	1.05
10	100.00	2.76	120,049	13,302	0.01	0.86

Finally, Table 3 shows the gap just before the beginning of the last phase (but after the aggregated model has been solved and its solutions have been used to feed the MIP repair/improve heuristic). On almost all categories the average final gap is below 10%, with an average running time of 1 minute. This heuristic alone significantly outperforms `cpx-reg` for a number of activities greater than 3. It is also clear from the table that solving the orbital shrinking relaxations with a black box MIP solver is usually very fast. Interestingly, solving these MIPs turn out to be often faster than solving the LP relaxations of the original models, while providing better or equal dual bounds. For example, on one instance with 1 activity, the LP relaxation of the original model that 0.26 seconds to solve, yielding a dual bound of 142.48, while the shrinked MIP takes 0.12 seconds and yields a dual bound of 182.54 (in this case, equal to the value of the optimal solution). On another instance with 10 activities, the LP relaxation takes 269.55, while the shrinked MIP takes only 52.77 seconds, both yielding the same dual bound in this case.

4 Related work

We divide the related work into two parts: previous work on using regular and context-free languages to specify constraints, especially those constraints occuring in shift scheduling problems, and previous work on hybridizations of CP and MIP solving.

Pesant introduced the global regular constraint in which constraints are specified by regular constraints [13]. He gave a complete propagation algorithm based

Table 3. MIP repair/improve heuristics standalone results.

# act.	time(s)	gap(%)
1	6.2	1.5
2	46.5	6.5
3	24.7	20.3
4	30.3	7.1
5	34.5	5.9
6	33.5	10.5
7	63.2	7.1
8	69.3	7.7
9	89.8	6.7
10	65.9	8.0

on dynamic programming. Coincidently Beldiceanu, Carlsson and Petit proposed specifying global constraints by means of finite automata augmented with counters [1]. Regular languages are precisely those accepted by (deterministic) finite automata. Propagators for such an automaton are constructed automatically from the specification of the automaton by means of a decomposition into simpler constraints. Quimper and Walsh proposed a closely related decomposition of the regular constraint based on transition constraints and variables introduced to represent the states of the unfolded automaton which recognizes the language [15]. They showed that such decomposition was effective and efficient in practice. Demassay et al. [5] used a column generation technique to solve a shift scheduling problem in which the columns are generated with a CP solver using the cost regular constraint, a variation of the regular constraint, whilst the optimization process is driven by the simplex method. Côté et al. [2] encoded the regular constraint into a MIP and efficiently solved some instances of the shift scheduling problem using the same automaton as Demassay et al.

Quimper and Walsh proposed the context-free grammar constraint in which constraints are specified by a context free grammar [15]. They gave two different propagators, one based on the CYK and the other on the Earley parser. At the same time and independently, Sellmann proposed the same global constraint and gave a similar propagator based on the CYK parser [18]. In [16, 3], context-free grammar constraints have been used to model complex shift-scheduling problems. More recently, Côté, Gendron, Quimper and Rousseau have proposed mixed-integer programming (MIP) encodings of the regular and context-free grammar constraints [3]. The MIP encoding of the regular constraint introduces linear inequalities to model the flow constructed by unfolding the automaton into a layered transition graph. When this is the only constraint in a problem, this can be solved with a specialized path finding algorithm. However, when there are other constraints in the problem, it needs to be solved with a more general 0/1 MIP solver. The MIP encoding has one significant difference with the CYK propagator. If there is more than one parsing for a sequence, it picks

one arbitrarily whilst the CYK propagator keeps all. This simplifies the MIP encoding without changing the set of solutions since only one parsing is needed to show membership in a context-free grammar. Experiments on a shift scheduling problem show that such MIP encodings are highly effective.

The last couple of decades have seen many hybrid approaches to solving optimization and decision problems that exploit both MIP and CP technqiues [12, 20]. There are several different approaches to such hybridization including:

Double modeling: We use both CP and MIP models and exchange information while solving. This may include bounds, infeasibility, nogoods, etc.

Search-inference duality [8]: We view CP and MIP methods as special cases of a search/inference duality. Search methods can be complete methods like branching or incomplete methods like local search. Inference methods can be CP based methods like domain reduction or MIP based methods like cutting plane inference.

Decomposition [9]: We decompose problems into a CP part and a MIP part using, for example, a Benders style scheme. The master problem searches over some of these variables. Given an instantiation for these variable, we get a subproblem which we can, for instance, solve using CP. The MIP/CP hybridization proposed here has this form.

Relaxation [19]: We combine CP based search methods like branching with relaxation techniques from MIP which solve a simpler approximated form of the problem like Langrangian relaxation.

5 Conclusions

We have proposed a hybrid MIP/CP approach for solving multi-activity shift scheduling problems, based on regular languages that *partially* describe the set of feasible shifts. This choice is justified by the fact that it may be much easier from the modeling point of view to define the appropriate formal language, and the corresponding MIP models may be much smaller. Computational results show that the method is a promising alternative compared to the state-of-the-art, being the fastest method when there are not too many activities. When the number of activities increases, the method cannot compete, in its present state, with an implicit formulation where the formal language completely describes the set of feasible shifts. However, it outperforms significantly explicit MIP based formulations, thus becoming the method of choice when it is not practical to describe the set of feasible shifts with a formal language alone. Future work may address the interesting question of how to derive Benders style cutting planes from the CP model (when infeasible), in order to speed up the enumeration of the MIP.

References

1. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) Proceedings of 10th International Confer-

ence on Principles and Practice of Constraint Programming (CP2004). pp. 107–122. Springer (2004)
2. Côté, M.C., Gendron, B., Rousseau, L.M.: The regular constraint for integer programming modeling. In: Proceedings of the Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming (CP-AI-OR 07) (2007)
3. Côté, M.C., Gendron, B., Quimper, C.G., Rousseau, L.M.: Formal languages for integer programming modeling of shift scheduling problems. Constraints 16(1), 54–76 (2011)
4. Côté, M.C., Gendron, B., Rousseau, L.M.: Grammar-based integer programming models for multiactivity shift scheduling. Management Science 57(1), 151–163 (2011)
5. Demassey, S., Pesant, G., Rousseau, L.M.: A cost-regular based hybrid column generation approach. Constraints 11(4), 315–333 (2006)
6. Fischetti, M., Liberti, L.: Orbital shrinking. In: Proceedings of ISCO (2012)
7. Gecode Team: Gecode: Generic constraint development environment (2012), available at `http://www.gecode.org`
8. Hooker, J.N.: Integrated Methods for Optimization. Springer (2006)
9. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming 96(1), 33–60 (2003)
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison Wesley (1979)
11. IBM ILOG: CPLEX 12.2 User's Manual (2011)
12. Milano, M.: Constraint and Integer Programming: Toward a Unified Methodology. Kluwer Academic Publishers (2003)
13. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3258, pp. 482–495. Springer (2004)
14. Pesant, G., Quimper, C.G., Rousseau, L.M., Sellmann, M.: The polytope of context-free grammar constraints. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems pp. 223–232 (2009)
15. Quimper, C.G., Walsh, T.: Global grammar constraints. In: 12th International Conference on Principles and Practices of Const raint Programming (CP-2006). Springer-Verlag (2006)
16. Quimper, C.G., Walsh, T.: Decomposing global grammar constraints. In: 13th International Conference on Principles and Practices of Const raint Programming (CP-2007). Springer-Verlag (2007)
17. Quimper, C.G., Rousseau, L.M.: A large neighbourhood search approach to the multi-activity shift scheduling problem. Journal of Heuristics 16, 373–392 (2010)
18. Sellmann, M.: The theory of grammar constraints. In: Proceedings of 12th International Conference on Principles and Practice of Constraint Programming (CP2006). pp. 530–544. Springer (2006)
19. Sellmann, M.: Theoretical foundations of CP-based Lagrangian relaxation. In: Wallace, M. (ed.) CP. vol. 3258, pp. 634–647 (2004)
20. Van Hentenryck, P., Milano, M.: Hybrid Optimization: The Ten Years of Cpaior. Springer Optimization and Its Applications, Springer (2010)