Solving Resource Allocation/Scheduling Problems with Constraint Integer Programming

Stefan Heinz*
Zuse Institute Berlin
Berlin, Germany
heinz@zib.de

J. Christopher Beck
Department of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
jcb@mie.utoronto.ca

Abstract

Constraint Integer Programming (CIP) is a generalization of mixed-integer programming (MIP) in the direction of constraint programming (CP) allowing the inference techniques that have traditionally been the core of CP to be integrated with the problem solving techniques that form the core of complete MIP solvers. In this paper, we investigate the application of CIP to scheduling problems that require resource and start-time assignments to satisfy resource capacities. The best current approach to such problems is logic-based Benders decomposition, a manual decomposition method. We present a CIP model and demonstrate that it achieves performance competitive to the decomposition while outperforming the standard MIP and CP formulations.

Introduction

Constraint Integer Programming (CIP) (Achterberg 2007b; 2009) is a generalization of both finite domain constraint programming (CP) and mixed integer programming (MIP) that allows the native integration of core problem solving techniques from each area. With a CIP solver, such as SCIP (Achterberg 2009), it is possible to combine the traditional strengths of MIP such as strong relaxations and cutting planes with global constraint propagation and conflict analysis. Indeed, SCIP can be seen as a solver framework that integrates much of the core of MIP, CP, and SAT solving methodologies.

Our primary goal in this paper is to start a broad investigation of CIP as a general approach to scheduling problems. While CP tends to be very successful on a variety of scheduling problems, it is challenged by problems that exhibit weak propagation either due to their objective functions (Kovács and Beck 2011) or to the need for a cascading series of interdependent decisions such as encountered in combined resource allocation and scheduling problems (Hooker 2005). We are interested to see if CIP techniques can address these challenges.

Our investigations in this paper focus on problems which combine resource allocation and scheduling. Given a set of jobs that require the use one out of a set of alternative resources, a solution will assign each job to a resource and schedule the jobs such that the capacity of each resource is respected at all time points. We address this problem in CIP by the `optcumulative` global constraint in SCIP, extending the `cumulative` constraint to allow each job to be optional. That is, a binary decision variable is associated with each job and resource pair and the corresponding cumulative constraint includes these variables in its scope. This approach is not novel in CP, dating back to at least (Beck and Fox 2000). We handle the requirement that exactly one resource must be chosen for a job in standard MIP fashion by specifying that the sum of the binary variables for a given job, across all resources, must be one.

Our experimental results demonstrate that our preliminary implementation of the `optcumulative` constraint is sufficient to allow a CIP model to be competitive with the state-of-the-art logic-based Benders decomposition (LBBD) on two problems sets with unary and discrete resource capacity, respectively.

In the next section, we formally define the scheduling problems and provide necessary background on CIP, LBBD, and the `cumulative` global constraint which forms the basis for the `optcumulative` constraint. We then present four models of our scheduling problems: MIP, LBBD, CP, and CIP. The following section contains our empirical investigations, both initial experiments comparing the four models and a detailed attempt to develop an understanding of the impact of primal heuristics for the CIP performance. In the final section, we conclude.

Background

In this section we introduce the scheduling problem under investigation and give background on CIP, LBBD, and the `cumulative` constraint.

Problem Definition

We study two classes of scheduling problems referred to as UNARY and MULTI (Hooker 2005; Beck 2010). Both problems are defined by a set of jobs $\mathcal{J}$, and a set of resources $\mathcal{K}$. Each job has a release date, $\mathcal{R}_j$, a deadline, $\mathcal{D}_j$, a resource-specific processing time, p_{jk}, a resource assignment cost, c_{jk}, and a resource requirement, r_{jk}. Each job, j, must be assigned to one resource, k, and scheduled to start at or after

*Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

its release date, end at or before its due date, and execute for p_{jk} time units. Each resource, $k \in \mathcal{K}$, has a capacity, C_k, and an associated resource constraint which states that for each time point, the sum of the resource requirements of the executing jobs must not exceed the resource capacity. A feasible solution is an assignment where each job is placed on exactly one resource and no resource is over capacity. The goal is to find an optimal solution, a feasible solution which minimizes the total resource assignment cost.

In the UNARY problem, the capacity of each resource and the requirement of each job is one. In the MULTI problem, capacities and requirements may be non-unary.

Constraint Integer Programming

Mixed integer programming (MIP) and *satisfiability testing (SAT)* are special cases of the general idea of *constraint programming (CP)*. The power of CP arises from the possibility to model a given problem directly with a large variety of different, expressive constraints. In contrast, SAT and MIP only allow for very specific constraints: Boolean clauses for SAT and linear and integrality constraints for MIP. Their advantage, however, lies in the sophisticated techniques available to exploit the structure provided by these constraint types.

The goal of *constraint integer programming (CIP)* is to combine the advantages and compensate for the weaknesses of CP, MIP, and SAT. It was introduced by Achterberg and implemented in the framework SCIP (Achterberg 2009). Formally a constraint integer program can be defined as follows.

Definition 1. A *constraint integer program (CIP)* $(\mathfrak{C}, I, c)$ consists of solving

$$c^\star = \min\{c^T x \mid \mathfrak{C}(x),\ x \in \mathbb{R}^n,\ x_j \in \mathbb{Z} \text{ for all } j \in I\}$$

with a finite set $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ of constraints $\mathcal{C}_i : \mathbb{R}^n \to \{0,1\}, i = \{1, \ldots, m\}$, a subset $I \subseteq N = \{1, \ldots, n\}$ of the variable index set, and an objective function vector $c \in \mathbb{R}^n$. A CIP must fulfill the following additional condition:

$$\forall \hat{x}_I \in \mathbb{Z}^I\ \exists (A', b') : \quad (1)$$
$$\{x_C \in \mathbb{R}^C \mid \mathfrak{C}(\hat{x}_I, x_C)\} = \{x_C \in \mathbb{R}^C \mid A' x_C \leq b'\}$$

with $C := N \setminus I$, $A' \in \mathbb{R}^{k \times C}$, and $b' \in \mathbb{R}^k$ for some $k \in \mathbb{Z}_{\geq 0}$.

Restriction (1) ensures that the sub-problem remaining after fixing all integer variables is a linear program. Note that this does not forbid quadratic or other nonlinear, and more involved expressions – as long as the nonlinearity only refers to the integer variables.

The central solving approach for CIP as implemented in the SCIP framework is branch-and-cut-and-propagate: as in SAT, CP, and MIP-solvers, SCIP performs a branch-and-bound search to decompose the problem into sub-problems. Also as in MIP, a linear relaxation, strengthened by additional inequalities/cutting planes if possible, is solved at each search node and used to guide and bound search. Similar to CP solvers, inference in the form of constraint propagation is used at each node to further restrict search and detect dead-ends. Moreover, as in SAT solving, SCIP uses conflict analysis and restarts. In more detail, CIP solving includes the following techniques

- *Presolving.* The purpose of presolving, which takes place before the tree search is started, is threefold: first, it reduces the size of the model by removing irrelevant information such as redundant constraints or fixed variables. Second, it strengthens the linear programming relaxation of the model by exploiting integrality information, e.g., to tighten the bounds of the variables or to improve coefficients in the constraints. Third, it extracts information from the model such as implications or cliques which can be used later for branching and cutting plane separation.
- *Propagation.* Propagation is used in the same fashion as in CP for pruning variable domains during the search.
- *Linear Relaxation.* A generic problem relaxation can be defined that includes only linear constraints. The relaxation can be solved efficiently to optimality and used in two primary ways: first to provide a guiding information for the search and second as the source of the "dual bound" a valid lower (upper) bound on the objective function for a minimization (maximization) problem.
- *Conflict Analysis.* The idea of conflict analysis is to reason about infeasible sub-problems which arise during the search in order to generate *conflict clauses* (Marques-Silva and Sakallah 1999; Achterberg 2007a) also known as *no-goods*. These conflict clauses are used to detect similar infeasible sub-problems later in the search. In order to perform conflict analysis, a bound change which was performed during the search, due to a propagation algorithm for example, needs to be explained. Such an *explanation* is a set of bounds which imply the performed bound change. The explanations are used to build up so-called conflict graph which lead to derivation of valid conflict clauses.

A global constraint in the framework of CIP can, but does not have too, contribute to all of these techniques. For example, as in CP, it can provide propagation algorithms for shrinking variable domains while also adding linear constraints to the linear programming relaxation, and supplying explanations to the conflict analysis reasoning. The minimal function of a global constraint is to "check" candidate solutions returning whether it is satisfied or not by a given variable assignment.

CIP has been applied to MIP (Achterberg 2009), mixed-integer nonlinear programming (Berthold, Heinz, and Vigerske 2009), nonlinear pseudo-Boolean programming (Berthold, Heinz, and Pfetsch 2009), the verification of chip designs (Achterberg, Brinkmann, and Wedler 2007), and scheduling (Berthold et al. 2010). The final paper is most relevant to the work here. Berthold et al. applied SCIP to resource-constrained project scheduling problems and demonstrated that CIP is competitive with the state-of-the-art in terms of finding both high quality solutions and in proving lower bounds on optimal solutions. This work forms one of our motivations for a broader investigation of CIP for scheduling problems.

Logic-based Benders Decomposition

Logic-based Benders decomposition (LBBD) (Hooker and Ottosson 2003) is a manual decomposition technique that generalizes classical Benders decomposition. A problem is modeled as a master problem (MP) and a set of sub-problems (SPs) where the MP is a relaxation of the global problem designed such that a solution to the MP induces one or more SPs. Each SP is an inference dual problem (Hooker 2005) that derives the tightest bound on the MP cost function that can be inferred from the current MP solution and the constraints and variables of the SP.

Solving a problem by LBBD is done by iteratively solving the MP to optimality and then solving each SP. If the MP solution satisfies all the bounds generated by the SPs, the MP solution is globally optimal. If not, a *Benders cut* is added to the MP by at least one violated SP and the MP is resolved. For models where the SPs are feasibility problems, it is sufficient for correctness to solve the SPs to feasibility or generate a cut that removes the current MP solution.

In order for the MP search to be more than just a blind enumeration of its solution space, some relaxation of the SPs should be present in the MP model and the Benders cuts should do more than just remove the current MP solution.

LBBD has been successfully applied to a wide range of problems including scheduling (Beck 2010; Bajestani and Beck 2011), facility and vehicle allocation (Fazel-Zarandi and Beck 2011), and queue design and control problems (Terekhov, Beck, and Brown 2009).

Models & Solution Approaches

In this section, we define the models used for the UNARY and MULTI problems for MIP, LBBD, CP, and CIP.

Mixed Integer Programming

One of the standard MIP models for scheduling problems is the so-called *time-indexed* formulation (Queyranne and Schulz 1994). A decision variable, x_{jkt}, is defined, which is equal to 1 if and only if job j, starts at time t, on resource k. Summing over appropriate subsets of these variables can then enforce the resource capacity requirement. The model we use, taken from (Hooker 2005), is as follows:

$$\min \quad \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}}\sum_{t=\mathcal{R}_j}^{\mathcal{D}_j-p_{jk}} c_{jk}\,x_{jkt}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}}\sum_{t=\mathcal{R}_j}^{\mathcal{D}_j-p_{jk}} x_{kjt} = 1 \qquad \forall j\in\mathcal{J} \tag{2}$$

$$\sum_{j\in\mathcal{J}}\sum_{t'\in T_{jkt}} r_{jk}\,x_{jkt'} \le C_k \quad \forall k\in\mathcal{K},\ \forall t \tag{3}$$

$$x_{jkt}\in\{0,1\} \qquad \forall k\in\mathcal{K},\ \forall j\in\mathcal{J},\ \forall t,$$

with $T_{jkt} = \{t-p_{jk},\ldots,t\}$.

The objective function minimizes the weighted resource assignment cost. Constraints (2) ensure that each job starts exactly once on one resource while Constraints (3) enforce the resource capacities on each resource at each time-point.

To solve this model, we rely on the default branch-and-bound search in the SCIP solver (Achterberg 2009). The default search has been tuned for solving MIP models and consists of a variety of modern algorithm techniques including: primal heuristics for finding feasible solutions, reliability-based branching heuristics, conflict analysis, and cutting planes. Details can be found in Achterberg and Berthold (2009).

Logic-based Benders Decomposition

The LBBD model (Hooker 2005; Beck 2010) defines two sets of decision variables: binary resource assignment variables, x_{jk}, which are assigned to 1 if and only if job j is assigned to resource k, and integer start time variables, S_j, which are assigned to the start-time of job j. The former variables are in the master problem while the latter are in sub-problems, one for each resource.

Formally, the LBBD master problem (MP) is defined as follows:

$$\min \quad \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}} c_{jk}\,x_{jk}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} x_{jk} = 1 \qquad \forall j\in\mathcal{J} \tag{4}$$

$$\sum_{j\in\mathcal{J}} x_{jk}\,p_{jk}r_{jk} \le \hat{C}_k \quad \forall k\in\mathcal{K} \tag{5}$$

$$\sum_{j\in\mathcal{J}_{hk}} (1-x_{jk}) \ge 1 \qquad \forall k\in\mathcal{K},\ h\in[H-1]^1 \tag{6}$$

$$x_{kj}\in\{0,1\} \qquad \forall k\in\mathcal{K},\ \forall j\in\mathcal{J},$$

with $\hat{C}_k = C_k\cdot(\max_{j\in\mathcal{J}}\{\mathcal{D}_j\} - \min_{j\in\mathcal{J}}\{\mathcal{R}_j\})$.

As in the global MIP model, the objective function minimizes the total resource allocation costs. Constraints (4) ensure that each job is assigned to exactly one resource. Constraints (5) are a linear relaxation of each resource capacity constraint. They state that the area of the rectangle with height C_k and width from the smallest release date to the largest deadline must be greater than the sum of the areas of the jobs assigned to the resource.

Constraints (6) are the Benders cuts. Let H indicate the index of the current iteration and $\mathcal{J}_{hk}$ denote the set of jobs that resulted in an infeasible sub-problem for resource k in iteration $h < H$. The Benders cut, then, simply states that the set of jobs assigned to resource k in iteration h should not be reassigned to the same resource. This is a form of *no-good cut* (Hooker 2005).

Because the MP assigns each job to a resource and there are no inter-job constraints, the SPs are independent, single-machine scheduling problems where it is necessary to assign each job a start time such that its time window and the capacity of its resource are respected. The SP for resource k can be formulated as a constraint program as follows, where $\mathcal{J}_k$ denotes the set of jobs assigned to resource k:

[1] For an $n\in\mathbb{N}$ we define $[n] := \{1,\ldots,n\}$ and $[0] := \varnothing$.

$$
\begin{aligned}
&\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p_{\cdot k}}, \boldsymbol{r_{\cdot k}}, C_k) \\
&\mathcal{R}_j \leq S_j \leq \mathcal{D}_j - p_{jk} && \forall j \in \mathcal{J}_k \qquad (7)\\
&S_j \in \mathbb{Z} && \forall j \in \mathcal{J}_k.
\end{aligned}
$$

$\boldsymbol{S}$, $\boldsymbol{p_{\cdot k}}$, and $\boldsymbol{r_{\cdot k}}$ are the vectors containing the start time variables and the processing times and demands for resource k with respect to subsets of jobs $\mathcal{J}_k$.

The global constraint `cumulative` (Baptiste, Pape, and Nuijten 2001) enforces the resource capacity constraint over all time-points at which a job may run. Constraints (7) enforce the time-windows for each job.

The MP and SPs are solved using the default search of SCIP. As CIP models admit global constraints, the subproblems are example of where the CIP model is solved primarily with CP technology.

Constraint Programming

As with MIP, we use the standard CP model for our problem (Hooker 2005). The start time of job j on resource k, is represented by the integer variable, S_{jk}. The model is as follows:

$$
\begin{aligned}
\min \quad & \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}} c_{jk}\, x_{jk} \\
\text{s.t.} \quad & \sum_{k \in \mathcal{K}} x_{jk} = 1 && \forall j \in \mathcal{J} \\
& \texttt{optcumulative}(\boldsymbol{S_{\cdot k}}, \boldsymbol{x_{\cdot k}}, \boldsymbol{p_{\cdot k}}, \boldsymbol{r_{\cdot k}}, C_k) \\
& && \forall k \in \mathcal{K} \qquad (8)\\
& \mathcal{R}_j \leq S_j \leq \mathcal{D}_j - p_{jk} && \forall j \in \mathcal{J}_k \\
& x_{jk} \in \{0,1\} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{K} \\
& S_{jk} \in \mathbb{Z} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{K}.
\end{aligned}
$$

Except for Constraints (8), the model components are analogous to those previously defined in the MIP and LBBD models. The `optcumulative` constraint is equivalent to the standard `cumulative` constraint with the addition that the jobs are optional: the jobs do not necessarily have to execute on this resource. The x_{jk} variable is used to indicate if job j executes on resource k and the `optcumulative` constraint include these variables in its scope. Formally, an assignment to the start time variables S_{jk} and binary choice variables x_{jk} for each job j and resource k is feasible if and only if the following condition holds at each time-point t:

$$
\sum_{j \in \mathcal{J}: t \in [S_{jk}, S_{jk} + p_{jk})} x_{jk} r_{jk} \leq C_k.
$$

We implement this model in IBM ILOG Solver and IBM ILOG Scheduler version 6.7. The `optcumulative` constraint is implemented by placing the set of machines in an *alternative resource set* and having each job require one resource from the set. The x_{jk} variable is implemented via the reification of the constraint stating that job j requires resource k.

To solve the problem, we use two pre-defined goals in the following order: `IloAssignAlternatives`, `IloSetTimesForward`. The first goal assigns jobs to resources in arbitrary order. When all jobs are assigned (and no dead-ends have been found via constraint propagation), the second goal implements the *schedule-or-postpone* heuristic (Pape et al. 1994) to assign start times to each job. Chronological backtracking is done when a dead-end is encountered.

Constraint Integer Programming

The CP model above is also the CIP model we use. Unlike the CP model, we implement and solve the CIP model using SCIP. As noted above, in CIP a global constraint such as `optcumulative` can contribute to the search in a number of ways. The current implementation of the `optcumulative` global constraint provides the following:

- *Presolving.* A number of problem reductions can be made in presolving, including normalization of the demands and the resource capacity and a detection of irrelevant jobs that do not influence the assignment/feasibility of remaining jobs on that resource. For example, if a job has a latest completion time which is smaller than the earliest start time of all remaining jobs then this job is irrelevant and can be ignored.
- *Propagation.* Following (Beck and Fox 2000), we adapt the standard bounds-based cumulative propagation (Baptiste, Pape, and Nuijten 2001) in a somewhat naive manner: we propagate all jobs that are known to execute on the resource. For each job j that is still optional, we perform singleton arc-consistency (SAC) (Debruyne and Bessière 1997): we assume that the job will execute on the resource and trigger propagation. If the propagation derives a dead-end, we can soundly conclude that the job cannot execute on the resource and appropriately set the x_{jk} variable. Otherwise, we retain the pruned domains for S_{jk}. In either case, the domains of all other variables are restored to their states before SAC. This propagation is stronger, but more costly, than the standard propagation of cumulative constraints with optional jobs due to Vilím, Barták, and Čepek (2005).
- *Linear Relaxation.* Each `optcumulative` constraint adds Constraint (5) as in the LBBD model to the linear programming relaxation.
- *Conflict Analysis.* Each time the `optcumulative` has to explain a bound change it first uses the `cumulative` explanation algorithm to derive an initial explanation. The explanation is extended with the the bounds of all choice variables which are (locally) fixed to one. In case of the SAC propagation, a valid explanation is the bounds of all choice variables which are fixed to one in the moment of the propagation.

The default parameters of SCIP are used to solve the CIP model. As these settings are tuned for pure MIP problems, it is likely that future work will be able to find more appropriate settings for CIPs.

Experiments

In this section we present first experimental results which indicate that the CIP model performs competitively with the LBBD model while out-performing MIP and CP.

Experimental Set-up

For all computational experiments except for the CP model we used the constraint integer programming solver SCIP (Achterberg 2009) that includes an implementation of the `cumulative` constraint (Berthold et al. 2010). As described above, we have extended SCIP by implementing the `optcumulative` global constraint. Using the same solver helps to focus on the impact of the models and algorithms used, controlling somewhat for different software engineering decisions made across different solvers. However, for the CP model, we choose to use IBM ILOG Solver and IBM ILOG Scheduler version 6.7 as they represent the commercial state-of-the-art and we did not want to unfairly penalize the CP approach due to using SCIP which has primarily been developed to this point as a MIP solver.

We used the scheduling instances introduced by Hooker (2005). Each set contains 195 problem instances. For both problem sets the number of resources ranges is from 2 to 4 and the number of jobs ranges from 10 to 38 in steps of 2. The maximum number of jobs for the instances with three and four resources is 32 while for two resources the number of maximum number of jobs is 38. For each problem size, we have five instances. For the MULTI problems the resource capacity is 10 and the job demands are generated with uniform probability on the integer interval $[1, 10]$. See Hooker (2005) for further details.

All computations reported were obtained on Intel Xeon E5420 2.50 GHz computers (in 64 bit mode) with 6 MB cache, running Linux, and 6 GB of main memory. We enforced a time limit of 7200 seconds. For all models other than CP, we use version 2.1.0.3 of SCIP integrated with SoPlex version 1.5.0.3 as the underlying linear programming solver (Wunderling 1996). Thus, we only used non-commercial software, with available source code.

Results

Tables 1 and 2 present the results for the UNARY test set and MULTI test set, respectively. The first two columns define the instance size in terms of the number of resources $|\mathcal{K}|$ and the number of jobs $|\mathcal{J}|$. For each model (for now we ignore the last four columns), we report the number of instance solved to proven optimality "opt" and the number instances for which we found a feasible solution "feas", which, of course, include the instances which are solved to optimality. We use the shifted geometric mean[2] for the number of "nodes" and for the running "time" in seconds.

The *shifted geometric mean* has the advantage that it reduces the influence of outliers. The geometric mean ensures that hard instances, at or close to the time limit, are prevented of having a huge impact on the measures. Similar shifting reduces the bias of easy instances, those solved in less than 10 seconds and/or less than 100 nodes. For a detailed discussion about different measures we refer to Achterberg (2007b).

For each resource-job combination, we display the best running time over all four models in bold face. In case one model could not solve any of the 5 instances for a particular resource-job combination, we omitted to display the shifted geometric mean of 7200.0 for the running time (instead we state "–").

UNARY On the UNARY problems, all four models are able to find feasible solutions for each instance. The CIP model finds and proves optimality for 194 out of 195 problem instances (timing-out on an instance with 30 jobs and 4 resources) followed by LBBD with 175, MIP at 161, and CP with 143. The CIP model is about three times faster than LBBD, using about half the number of nodes. However, note that the LBBD statistic includes only the nodes in the master problem search not the sub-problems. The time, however, includes both master and sub-problem solving.

The LBBD results are consistent with those of an existing implementation (not using SCIP) (Beck 2010). We found 20 time-outs for LBBD while Beck's results had 14 time-outs. We believe that this relatively small difference can be attributed to the use of different solvers and different computers.

Overall the CIP model dominates all other model for the UNARY case.

MULTI The MULTI results are somewhat different. CIP is not the dominant approach anymore. The performance of LBBD and CIP are very similar with respect to number of solved instances and overall running time. CIP manages 123 instances while LBBD solves 119. Both approach, however, are superior to the MIP and CP model using the measure of overall running time and number of solved instances to proven optimality. MIP solves 98 instances and CP solves only 62 instances.

This time, the LBBD results are not consistent with the previous implementation of Beck. He solved 175 instances which are 51 instances more than our LBBD model. We assume that using SCIP for solving the sub-problems instead of IBM ILOG Solver and IBM ILOG Scheduler leads to this differences. We plan to further investigate this issue.

The results of the CP model coincide with those of Hooker (2005) where it was shown that instances with 18 jobs or more could not be solved. The MIP results, in contrast, are substantially better than those reported by Hooker. The reason is not clear but we tentatively attribute the difference to the advance in MIP solving technology in the past six years.

Feasibility vs. Optimality It is of particular note that the MIP model was able to find feasible solutions for all instances in both problem sets. In fact, the optimality gap for MIP was usually very low: on the MULTI set the largest gap was 5.3% and 55 of the 97 instances not solved to optimality

[2] The shifted geometric mean of values $t_1, \ldots, t_n$ is defined as $\left(\prod(t_i + s)\right)^{1/n} - s$ with shift s. We use a shift $s = 10$ for time and $s = 100$.for nodes

Table 1: Results for the UNARY test set. Each resource-job combination consists of 5 instances. This adds up to a total of 195.

		MIP				LBBD				CP				CIP				CIP (optimality proof)			
$\|\mathcal{K}\|$	$\|\mathcal{J}\|$	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	inst	proved	nodes	time
2	10	5	5	1	**0.0**	5	5	61	0.2	5	5	96	**0.0**	5	5	25	**0.0**	5	5	1	0.0
	12	5	5	3	0.6	5	5	115	0.5	5	5	256	**0.0**	5	5	58	**0.0**	5	5	4	0.0
	14	5	5	93	2.4	5	5	566	1.5	5	5	741	**0.1**	5	5	130	**0.1**	5	5	4	0.0
	16	5	5	249	5.0	5	5	80	0.3	5	5	2 433	0.2	5	5	139	**0.1**	5	5	4	0.0
	18	5	5	221	11.0	5	5	75	0.3	5	5	6 767	0.5	5	5	216	**0.1**	5	5	8	0.0
	20	5	5	222	10.4	5	5	440	2.1	5	5	11 174	1.1	5	5	269	**0.2**	5	5	2	0.0
	22	5	5	16 898	139.5	5	5	195	1.6	5	5	47 460	4.8	5	5	117	**0.1**	5	5	1	0.0
	24	3	5	5 273	191.5	5	5	22	15.9	5	5	105 056	10.8	5	5	162	**0.1**	5	5	1	0.0
	26	5	5	13 359	174.8	4	4	300	33.7	5	5	326 528	34.8	5	5	439	**0.5**	5	5	18	0.0
	28	2	5	115 839	1813.9	5	5	510	28.9	5	5	497 455	60.0	5	5	346	**0.3**	5	5	1	0.0
	30	1	5	151 780	2432.7	4	4	1 836	74.2	5	5	4 231 229	490.2	5	5	2 139	**8.9**	5	5	114	1.2
	32	3	5	3 596	285.8	5	5	287	2.3	5	5	6 786 880	832.5	5	5	706	**0.8**	5	5	8	0.0
	34	1	5	125 588	2030.5	4	4	274	43.8	5	5	17 724 154	2239.7	5	5	897	**1.2**	5	5	9	0.0
	36	3	5	46 420	1322.9	1	1	656	2011.8	3	5	35 832 546	4939.0	5	5	1 014	**1.4**	5	5	7	0.0
	38	3	5	9 506	569.6	3	3	983	425.0	3	5	35 021 851	5151.3	5	5	1 076	**0.8**	5	5	1	0.0
3	10	5	5	1	0.1	5	5	356	0.6	5	5	584	**0.0**	5	5	73	0.1	5	5	15	0.0
	12	5	5	1	0.2	5	5	191	0.4	5	5	2 801	0.2	5	5	119	**0.1**	5	5	13	0.0
	14	5	5	7	1.6	5	5	2 759	5.0	5	5	13 431	1.0	5	5	314	**0.5**	5	5	68	0.2
	16	5	5	9	3.1	5	5	223	0.9	5	5	58 688	4.4	5	5	322	**0.4**	5	5	4	0.0
	18	5	5	7	2.6	5	5	444	**0.8**	5	5	236 227	20.3	5	5	632	1.4	5	5	24	0.1
	20	5	5	2 949	28.4	5	5	1 898	9.0	5	5	1 277 898	106.5	5	5	956	**3.0**	5	5	50	0.2
	22	5	5	602	29.6	5	5	1 106	12.2	5	5	9 746 557	873.9	5	5	1 217	**3.5**	5	5	51	0.2
	24	5	5	2 555	47.6	5	5	1 745	**5.7**	4	5	47 841 477	4432.3	5	5	1 641	6.7	5	5	48	0.3
	26	4	5	18 023	190.8	5	5	17 639	57.5	0	5	70 242 148	–	5	5	5 647	**22.6**	5	5	259	1.2
	28	4	5	2 715	100.4	5	5	3 721	**11.6**	0	5	70 336 374	–	5	5	4 591	18.5	5	5	72	0.3
	30	2	5	327 576	2904.1	3	3	11 963	133.8	0	5	68 045 013	–	5	5	18 901	**103.9**	5	5	605	12.1
	32	3	5	44 141	1209.6	4	4	6 228	95.6	0	5	70 408 953	–	5	5	10 677	**52.0**	5	5	268	7.8
4	10	5	5	1	0.1	5	5	262	0.7	5	5	1 166	0.1	5	5	29	**0.0**	5	5	4	0.0
	12	5	5	1	**0.1**	5	5	589	1.3	5	5	9 889	0.6	5	5	79	**0.1**	5	5	13	0.0
	14	5	5	1	**0.2**	5	5	2 390	5.8	5	5	44 482	3.4	5	5	211	0.4	5	5	15	0.1
	16	5	5	17	**2.4**	5	5	22 922	41.2	5	5	155 441	13.0	5	5	768	2.5	5	5	130	0.7
	18	5	5	1	**0.8**	4	4	9 857	61.4	5	5	2 006 248	163.8	5	5	904	2.9	5	5	73	0.4
	20	5	5	967	18.9	5	5	19 750	23.2	5	5	11 029 872	952.8	5	5	2 525	**8.7**	5	5	118	0.7
	22	5	5	10 364	95.9	3	3	222 873	245.1	3	5	52 404 067	5026.7	5	5	8 912	**44.4**	5	5	603	4.8
	24	5	5	2 343	55.4	4	4	43 687	70.5	0	5	81 261 130	–	5	5	7 355	**38.5**	5	5	209	2.4
	26	4	5	4 426	**119.3**	4	4	151 494	256.6	0	5	76 244 443	–	5	5	37 140	179.3	5	5	919	9.8
	28	3	5	137 997	1281.6	4	4	242 117	375.1	0	5	79 742 951	–	5	5	33 422	**175.1**	5	5	663	9.6
	30	3	5	187 957	2226.1	4	4	129 582	**129.4**	0	5	73 680 622	–	4	5	63 047	378.3	5	5	2 236	42.1
	32	2	5	15 488	831.6	4	4	526 447	**487.3**	0	5	71 281 227	–	5	5	73 517	491.8	5	5	642	5.5
		161	195	2 059	74.1	175	175	2 177	27.9	143	195	620 577	204.9	194	195	1 111	**9.9**	195	195	84	1.7

had gaps of less than 2%. From another perspective, therefore, there is an argument that MIP is performing best on the MULTI problems.

In general MIP solving, it is know that "good" primal solutions which are detected early in the search can significantly increase performance. A more detailed examination of the MIP runs shows that the primary reason for the ability of the MIP model to find feasible solutions is the use of such primal heuristics. In case of CIP, the default primal heuristics in SCIP are able to occasionally find feasible solution for the UNARY instances but never for MULTI.

To evaluate the viability of future research to develop CIP primal heuristics, we ran an additional experiment by providing the CIP model with the optimal value for all the instances in which it is known: all 195 instances for the UNARY problems and 176 instances for MULTI. The solver, then, has only to prove that the instance has no better solutions. If we are able to show that the proofs are short, we have an indication that, if we were able to develop strong primal heuristics for CIP, we would be able to substantially improve the problem solving performance. The final four columns in Tables 1 and 2 display the results. The "inst" columns is the number of instances where an optimal solution is known and the column "proved" is the number of instances that the CIP model was able to prove optimality having been given the optimal cost as an upper bound. The other two columns state again the shifted geometric mean of the number of search "nodes" and the running "time". For the UNARY instances, we are able to solve all problems, proving optimality in on average about a quarter of the time required to find and prove optimality. On the MULTI instances, however, the picture is different. Some instances (e.g., two resources and 30 or more jobs) show the same effect as for the UNARY case. On other instances it was possible to find and

Table 2: Results for the MULTI test set. Each resource-job combination consists of 5 instances. This adds up to a total of 195.

		MIP				LBBD				CP				CIP				CIP (optimality proof)			
$\|\mathcal{K}\|$	$\|\mathcal{J}\|$	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	inst	proved	nodes	time
2	10	5	5	99	1.4	5	5	51	0.2	5	5	2 792	0.1	5	5	152	**0.0**	5	5	35	0.0
	12	5	5	362	4.2	5	5	19	0.4	5	5	33 689	0.8	5	5	156	**0.0**	5	5	37	0.0
	14	5	5	613	13.2	5	5	6	2.7	5	5	466 774	11.5	5	5	342	**0.2**	5	5	200	1.7
	16	5	5	10 950	46.4	5	5	3	**17.0**	5	5	22 812 045	328.0	5	5	3 110	18.1	5	4	2 058	30.1
	18	5	5	55 304	153.9	5	5	7	88.1	0	3	494 257 496	–	5	5	9 951	**18.0**	5	5	2 563	29.0
	20	5	5	197 808	621.5	3	3	21	157.2	0	0	453 120 052	–	5	5	4 106	**2.4**	5	5	886	1.3
	22	3	5	971 481	3289.6	2	2	10	**702.9**	0	0	397 205 523	–	2	2	338 309	1324.1	4	2	29 718	371.6
	24	1	5	1 372 460	6912.5	0	0	1	–	0	0	339 570 055	–	3	3	353 667	**706.1**	5	2	14 242	508.7
	26	1	5	561 690	**3903.6**	1	1	1	5192.4	0	0	302 945 001	–	1	1	1 714 440	5439.1	5	2	15 689	508.5
	28	0	5	755 538	–	3	3	10	441.0	0	0	285 893 897	–	3	3	90 382	**159.6**	5	3	2 799	129.1
	30	0	5	537 011	–	1	1	1	**2971.1**	0	0	289 896 816	–	0	0	2 389 029	–	2	2	1	0.0
	32	1	5	447 812	5553.7	1	1	1	5679.1	0	0	285 549 706	–	3	3	494 763	**281.7**	3	3	1	0.0
	34	0	5	378 116	–	1	1	1	3014.7	0	0	291 878 950	–	1	1	2 729 498	**1396.4**	1	1	1	0.0
	36	0	5	459 463	–	1	1	1	2044.0	0	0	271 990 663	–	2	2	1 313 548	**699.3**	2	2	1	0.0
	38	0	5	202 726	–	1	1	1	3368.3	0	0	256 182 585	–	3	3	6 441 053	**1675.6**	3	3	1	0.0
3	10	5	5	7	0.8	5	5	49	0.2	5	5	1 508	**0.0**	5	5	85	**0.0**	5	5	19	0.0
	12	5	5	891	6.0	5	5	267	0.7	5	5	42 886	0.9	5	5	480	**0.3**	5	5	147	0.1
	14	5	5	575	8.8	5	5	94	**0.3**	5	5	536 284	10.0	5	5	1 152	1.1	5	5	234	0.2
	16	5	5	53 582	169.2	5	5	837	**9.9**	5	5	26 881 777	419.2	5	5	14 443	21.7	5	5	8 949	16.6
	18	4	5	312 463	952.5	5	5	3 196	**20.5**	0	2	485 036 163	–	4	4	201 128	138.6	5	5	75 534	53.7
	20	3	5	453 808	1629.4	5	5	1 613	**5.8**	0	1	460 840 309	–	5	5	37 498	34.8	5	5	5 612	38.3
	22	2	5	591 165	3117.1	5	5	2 254	**148.1**	0	0	431 132 614	–	2	2	632 677	1351.4	5	2	793 451	960.1
	24	3	5	577 419	5107.3	1	1	812	**2323.8**	0	0	410 966 318	–	1	1	1 660 912	3164.6	5	1	631 291	4709.2
	26	0	5	578 821	–	4	4	1 340	1350.8	0	0	370 925 295	–	3	3	650 086	**727.0**	5	2	264 430	1469.2
	28	0	5	418 861	–	0	0	8	–	0	0	337 254 114	–	2	3	725 799	**1260.3**	5	2	51 512	624.4
	30	0	5	292 470	–	0	0	49	–	0	0	288 175 928	–	0	0	1 382 511	–	4	1	55 610	1383.1
	32	0	5	95 543	–	0	0	3	–	0	0	265 357 963	–	1	1	1 747 365	**6917.5**	3	1	26 140	795.1
4	10	5	5	1	0.2	5	5	13	**0.1**	5	5	2 055	**0.1**	5	5	105	**0.1**	5	5	13	0.0
	12	5	5	544	5.1	5	5	30	**0.1**	5	5	30 179	1.0	5	5	242	0.2	5	5	32	0.0
	14	5	5	2 004	13.9	5	5	388	**1.1**	5	5	1 136 699	26.9	5	5	1 118	1.7	5	5	149	0.3
	16	5	5	1 539	29.3	5	5	251	**0.6**	5	5	6 907 787	111.8	5	5	1 094	1.3	5	5	290	0.3
	18	4	5	453 646	921.9	5	5	3 296	**3.5**	2	3	496 600 144	6994.2	5	5	28 359	20.1	5	5	2 118	4.0
	20	3	5	739 750	2188.5	5	5	1 298	**26.4**	0	1	479 264 386	–	5	5	34 834	28.6	5	5	9 271	3.9
	22	3	5	432 934	1957.6	5	5	3 363	**45.3**	0	0	471 200 720	–	4	4	77 455	166.3	5	3	7 883	135.6
	24	0	5	712 175	–	2	2	1 979	**1445.7**	0	0	413 825 956	–	2	2	1 796 900	3021.0	5	2	596 525	2190.7
	26	0	5	430 563	–	1	1	15 646	**4069.6**	0	0	399 153 422	–	0	1	2 436 109	–	5	0	2 952 736	7200.0
	28	0	5	479 513	–	1	1	679	**2803.2**	0	0	407 550 307	–	1	1	1 309 160	6575.0	5	1	158 262	1925.8
	30	0	5	237 592	–	1	1	186	**2105.0**	0	0	401 058 271	–	0	0	1 781 515	–	5	1	348 307	1982.5
	32	0	5	218 931	–	0	0	136	–	0	0	353 061 759	–	0	0	1 972 257	–	4	0	1 029 999	7200.0
		98	195	62 568	778.8	119	119	222	227.3	62	70	34 504 645	1311.1	123	125	61 323	**212.0**	176	125	5 656	83.2

prove optimality in our previous experiment, but not here when the optimal cost was given (e.g., an instances with 2 resources and 16 jobs). Clearly, the pattern of remaining open nodes and the conflict clauses learning in finding the optimal solution in the previous experiment helps in also proving it.

Overall, these results indicate that strong primal heuristics would be a clear benefit on the UNARY problems and for some of the MULTI instances. However, it is also clear from the MULTI results that research is also needed to speed-up the proofs of optimality (e.g., to strengthen the linear relaxation to improve bounding).

Summary Overall, the best performing approaches in terms of finding and proving optimality are CIP and LBBD. The former clearly dominates on the UNARY instances while their performance is similar on the MULTI instances. We conclude, therefore, that the CIP model is the best non-decomposition based approach for solving these combined resource allocation/scheduling problems.

The performance of MIP is not as strong as the two top approaches but better than both CP and the MIP results reported by Hooker (2005). However, only MIP is able to find provable good feasible solutions for all instances.

The relatively poor performance of CP is surprising given its usual success in scheduling problems. We plan to investigate more informed CP search heuristics (Beck and Fox 2000).

Conclusion

We studied four models for solving combined resource optimization and scheduling using mixed integer programming, constraint programming, constraint integer programming, and logic-based Benders decomposition. Previous results indicated the logic-based Benders decomposition was

the dominant approach. Our results demonstrated that on problems with unary capacity resources, constraint integer programming is able to solve 194 out of 195 instances to optimality compared to only 175 for logic-based Benders decomposition. However, on problems with non-unary resource capacity, the picture is not as clear as logic-based Benders and constraint integer programming showed similar performance. The results for the logic-based Benders, however, are weaker than previous results (Beck 2010). Interesting, the mixed integer programming model is able to find feasible solutions with a small optimality gap to all problems instances in both sets, unlike all other techniques. The constraint programming model is the worse performing model over both problem sets. Based on these results, we conclude that constraint integer programming represents the state-of-the-art for non-decomposition based approaches to these problems.

There are a number of avenues for future work including the investigation of primal heuristics for constraint integer programming and ways to improve the search done in constraint programming. Most significantly, we believe that our results here demonstrate that constraint integer programming may be a promising technology for scheduling in general and therefore we plan to pursue its application to a variety of scheduling problems.

References

Achterberg, T., and Berthold, T. 2009. Hybrid branching. In van Hoeve, W.-J., and Hooker, J. N., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2009)*, volume 5547 of *LNCS*, 309–311.

Achterberg, T.; Brinkmann, R.; and Wedler, M. 2007. Property checking with constraint integer programming. ZIB-Report 07-37, Zuse Institute Berlin.

Achterberg, T. 2007a. Conflict analysis in mixed integer programming. *Discrete Optimization* 4(1):4–20. Special issue: Mixed Integer Programming.

Achterberg, T. 2007b. *Constraint Integer Programming*. Ph.D. Dissertation, Technische Universität Berlin.

Achterberg, T. 2009. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation* 1(1):1–41.

Bajestani, M. A., and Beck, J. C. 2011. Scheduling an aircraft repair shop. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS2011)*. to appear.

Baptiste, P.; Pape, C. L.; and Nuijten, W. 2001. *Constraint-based Scheduling*. Kluwer Academic Publishers.

Beck, J. C., and Fox, M. S. 2000. Constraint directed techniques for scheduling with alternative activities. *Artificial Intelligence* 121(1–2):211–250.

Beck, J. C. 2010. Checking-up on branch-and-check. In Cohen, D., ed., *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 of *LNCS*, 84–98.

Berthold, T.; Heinz, S.; Lübbecke, M. E.; Möhring, R. H.; and Schulz, J. 2010. A constraint integer programming approach for resource-constrained project scheduling. In Lodi, A.; Milano, M.; and Toth, P., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *LNCS*, 313–317.

Berthold, T.; Heinz, S.; and Pfetsch, M. E. 2009. Nonlinear pseudo-boolean optimization: relaxation or propagation? In Kullmann, O., ed., *Theory and Applications of Satisfiability Testing – SAT 2009*, volume 5584 of *LNCS*, 441–446.

Berthold, T.; Heinz, S.; and Vigerske, S. 2009. Extending a cip framework to solve MIQCPs. ZIB-Report 09-23, Zuse Institute Berlin.

Debruyne, R., and Bessière, C. 1997. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*, 412–417.

Fazel-Zarandi, M. M., and Beck, J. C. 2011. Using logic-based benders decomposition to solve the capacity and distance constrained plant location problem. *INFORMS Journal on Computing*. in press.

Hooker, J. N., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96:33–60.

Hooker, J. N. 2005. Planning and scheduling to minimize tardiness. In van Beek, P., ed., *Principles and Practice of Constraint Programming – CP 2005*, volume 3709 of *LNCS*, 314–327.

Kovács, A., and Beck, J. C. 2011. A global constraint for total weighted completion time for unary resources. *Constraints* 16(1):100–123.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Pape, C. L.; Couronné, P.; Vergamini, D.; and Gosselin, V. 1994. Time-versus-capacity compromises in project scheduling. In *Proceedings of the Thirteenth Workshop of the UK Planning Special Interest Group*.

Queyranne, M., and Schulz, A. S. 1994. Polyhedral approaches to machine scheduling problems. Technical Report 408/1994, Departement of Mathematics, Technische Universitat Berlin, Germany. Revised 1996.

Terekhov, D.; Beck, J. C.; and Brown, K. N. 2009. A constraint programming approach for solving a queueing design and control problem. *INFORMS Journal on Computing* 21(4):549–561.

Vilím, P.; Barták, R.; and Čepek, O. 2005. Extension of $O(n \log n)$ filtering algorithms for the unary resource constraint to optional activities. *Constraints* 10(4):403–425.

Wunderling, R. 1996. *Paralleler und objektorientierter Simplex-Algorithmus*. Ph.D. Dissertation, Technische Universität Berlin.