Solving Resource Allocation/Scheduling Problems with Constraint Integer Programming

* Supported by the DFG Research Center MATHEON Mathematics for key technologies in Berlin.
** Department of Mechanical & Industrial Engineering, University of Toronto, Toronto, Canada

ZIB-Report 11-14 (May 2011)
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 con-
straint 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 ap-
lication 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 out-
performing 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 solv-
ing techniques from each area. With a CIP solver, such as
SCIP (Achterberg 2009), it is possible to combine the tradi-
tional strengths of MIP such as strong relaxations and cut-
ting 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 solv-
ing methodologies.

Our primary goal in this paper is to start a broad investi-
gation of CIP as a general approach to scheduling problems.
While CP tends to be very successful on a variety of schedul-
ing 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 in-
terdependent decisions such as encountered in combined re-
source 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 re-
sources, 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 \texttt{optcumulative} global constraint in SCIP,
extending the \texttt{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 cumu-
late 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 \texttt{optcumulative} constraint is suf-
ficient 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 capac-
ity, respectively.

In the next section, we formally define the scheduling
problems and provide necessary background on CIP, LBBD,
and the \texttt{cumulative} global constraint which forms the ba-
sis for the \texttt{optcumulative} constraint. We then present
four models of our scheduling problems: MIP, LBBD, CP,
and CIP. The following section contains our empirical inves-
tigations, both initial experiments comparing the four mod-
els 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
\texttt{cumulative} constraint.

Problem Definition
We study two classes of scheduling problems referred to as
\textsc{ Unary} and \textsc{ Multi} (Hooker 2005; Beck 2010). Both
problems are defined by a set of jobs \( J \), and a set of resources \( K \).
Each job has a release date, \( R_j \), a deadline, \( D_j \), 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

\begin{itemize}
\item \( J \) is a set of jobs, \( \{1, \ldots, n\} \);
\item \( K \) is a set of resources, \( \{1, \ldots, m\} \);
\item \( p_{jk} \) is the processing time of job \( j \) on resource \( k \);
\item \( r_{jk} \) is the resource requirement of job \( j \) on resource \( k \);
\item \( c_{jk} \) is the cost of assigning job \( j \) to resource \( k \);
\item \( D_j \) is the deadline of job \( j \);
\item \( R_j \) is the release date of job \( j \).
\end{itemize}

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 out-
performing the standard MIP and CP formulations.
its release date, end at or before its due date, and execute for 
\( p_{jk} \) time units. Each resource, \( k \in K \), 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 \textsc{Unary} problem, the capacity of each resource and 
the requirement of each job is one. In the \textsc{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 \textsc{SCIP} (Achterberg 2009). Formally a constraint integer program can be defined as follows.

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

\[
e^* = \min \{ c^T x \mid \mathcal{C}(x), x \in \mathbb{R}^n, x_j \in \mathbb{Z} \text{ for all } j \in I \}
\]

with a finite set \( \mathcal{C} = \{ C_1, \ldots, C_m \} \) of constraints \( C_i : \mathbb{R}^n \to \{0, 1\}, i = \{1, \ldots, m\} \), 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:

\[
\forall \hat{x}_I \in \mathbb{Z}^I \ \exists (A', b') : \\
\{ x_C \in \mathbb{R}^C \mid \mathcal{C}(\hat{x}_I, x_C) \} = \{ x_C \in \mathbb{R}^C \mid A' x_C \leq b' \}
\]  

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 \textsc{SCIP} framework is branch-and-cut-and-propagate: as in SAT, CP, and MIP-solvers, \textsc{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, \textsc{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 \textsc{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 re-solved. 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:

\[
\begin{align*}
\min & \quad \sum_{k \in K} \sum_{j \in J} \sum_{t \in T_j} d_{jk} x_{jkt} \\
\text{s.t.} & \quad \sum_{k \in K} \sum_{t \in T_j} x_{jkt} = 1 \quad \forall j \in J \quad (2) \\
& \quad \sum_{j \in J} \sum_{t' \in T_{jt}} r_{jk} x_{jkt'} \leq C_k \quad \forall k \in K, \forall t \quad (3) \\
& \quad x_{jkt} \in \{0, 1\} \quad \forall k \in K, \forall j \in J, \forall t,
\end{align*}
\]

with \( T_{jt} = \{ t - p_{jk}, \ldots, t \} \).

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:

\[
\begin{align*}
\min & \quad \sum_{k \in K} \sum_{j \in J} c_{jk} x_{jk} \\
\text{s.t.} & \quad \sum_{k \in K} x_{jk} = 1 \quad \forall j \in J \quad (4) \\
& \quad \sum_{j \in J} x_{jk} p_{jk} r_{jk} \leq \hat{C}_k \quad \forall k \in K \quad (5) \\
& \quad \sum_{j \in J_{hk}} (1 - x_{jk}) \geq 1 \quad \forall k \in K, h \in [H - 1] \quad (6) \\
& \quad x_{jk} \in \{0, 1\} \quad \forall k \in K, \forall j \in J,
\end{align*}
\]

with \( \hat{C}_k = C_k \cdot (\max_{j \in J} \{ D_j \} - \min_{j \in J} \{ R_j \}) \).

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 \( J_{hk} \) 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 \( J_k \) denotes the set of jobs assigned to resource \( k \):

\footnote{For an \( n \in \mathbb{N} \) we define \([n] := \{1, \ldots, n\}\) and \([0] := \emptyset\).}
The start time of job $j$ is represented by the integer variable, $S_j$. The model is as follows:

$$\text{min} \sum_{k \in K} \sum_{j \in J} C_{jk} x_{jk}$$

s.t. \hspace{1em} \sum_{k \in K} x_{jk} = 1 \hspace{1em} \forall j \in J

$$\text{optcumulative}(S_{jk}, x_{jk}, p_{jk}, r_{jk}, C_k) \hspace{1em} \forall k \in K$$

$$R_j \leq S_j \leq D_j - p_{jk} \hspace{1em} \forall j \in J_k$$

$$x_{jk} \in \{0, 1\} \hspace{1em} \forall j \in J, \forall k \in K$$

$$S_{jk} \in \mathbb{Z} \hspace{1em} \forall j \in J, \forall k \in K.$$ 

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 $t$:

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

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 Bessi`ere 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 problem 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 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 |K| and the number of jobs |J|. For each model (for now we ignore the last four columns), we report the number of instances solved to proven optimality “opt” and the number of 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 (time-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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>MIP</th>
<th>LBBD</th>
<th>CP</th>
<th>CIP</th>
<th>CIP (optimality proof)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>opt</td>
<td>feas</td>
<td>nodes</td>
<td>time</td>
<td>opt</td>
</tr>
<tr>
<td>2 5 1 0.0</td>
<td>5 5</td>
<td>61 0.2</td>
<td>5 5</td>
<td>96 0.0</td>
<td>5 5</td>
<td>25 0.0</td>
</tr>
<tr>
<td>2 5 5 0.6</td>
<td>5 5</td>
<td>115 0.5</td>
<td>5 5</td>
<td>256 0.0</td>
<td>5 5</td>
<td>58 0.0</td>
</tr>
<tr>
<td>3 5 5 2.4</td>
<td>5 5</td>
<td>566 1.5</td>
<td>5 5</td>
<td>741 0.1</td>
<td>5 5</td>
<td>130 0.1</td>
</tr>
<tr>
<td>6 5 5 249 5.0</td>
<td>5 5</td>
<td>801 0.3</td>
<td>5 5</td>
<td>2433 0.2</td>
<td>5 5</td>
<td>139 0.1</td>
</tr>
<tr>
<td>8 5 5 221 11.0</td>
<td>5 5</td>
<td>75 0.3</td>
<td>5 5</td>
<td>6767 0.5</td>
<td>5 5</td>
<td>216 0.1</td>
</tr>
<tr>
<td>10 5 5 222 10.4</td>
<td>5 5</td>
<td>440 2.1</td>
<td>5 5</td>
<td>1174 1.1</td>
<td>5 5</td>
<td>269 0.2</td>
</tr>
<tr>
<td>12 5 5 16898 139.5</td>
<td>5 5</td>
<td>195 1.6</td>
<td>5 5</td>
<td>47460 4.8</td>
<td>5 5</td>
<td>117 0.1</td>
</tr>
<tr>
<td>14 3 5 5273 191.5</td>
<td>5 5</td>
<td>22 15.9</td>
<td>5 5</td>
<td>105056 10.8</td>
<td>5 5</td>
<td>162 0.1</td>
</tr>
<tr>
<td>18 3 5 4339 134.7</td>
<td>4 4</td>
<td>300 33.7</td>
<td>5 5</td>
<td>326528 34.8</td>
<td>5 5</td>
<td>439 0.5</td>
</tr>
<tr>
<td>28 2 5 115839 1813.9</td>
<td>5 5</td>
<td>510 28.9</td>
<td>5 5</td>
<td>497455 60.0</td>
<td>5 5</td>
<td>346 0.3</td>
</tr>
<tr>
<td>30 1 5 151780 2432.7</td>
<td>4 4</td>
<td>1836 74.2</td>
<td>5 5</td>
<td>4231299 490.2</td>
<td>5 5</td>
<td>2139 8.9</td>
</tr>
<tr>
<td>32 3 5 3596 285.8</td>
<td>5 5</td>
<td>287 2.3</td>
<td>5 5</td>
<td>678680 832.5</td>
<td>5 5</td>
<td>706 0.8</td>
</tr>
<tr>
<td>34 1 5 125588 2030.5</td>
<td>4 4</td>
<td>274 43.8</td>
<td>5 5</td>
<td>17724154 2239.7</td>
<td>5 5</td>
<td>897 1.2</td>
</tr>
<tr>
<td>36 3 5 46420 1322.9</td>
<td>4 4</td>
<td>1165 2011.8</td>
<td>3 3</td>
<td>35832546 4939.0</td>
<td>5 5</td>
<td>1014 1.4</td>
</tr>
<tr>
<td>38 3 5 9506 569.6</td>
<td>3 3</td>
<td>983 425.0</td>
<td>3 3</td>
<td>35927431 5151.3</td>
<td>5 5</td>
<td>1076 0.8</td>
</tr>
</tbody>
</table>

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 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
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


