PROCEEDINGS OF THE

ERCIM Workshop on Constraint Solving and Constraint Logic Programming, 2010

http://www.constraint-programming.de/csclp2010

November 25th-26th, 2010

Editors
Barry O’Sullivan
Cork Constraint Computation Centre
University College Cork, Ireland
b.osullivan@cs.ucc.ie

Armin Wolf
Fraunhofer FIRST, Berlin, Germany
armin.wolf@first.fraunhofer.de
Preface

This volume contains the invited talk abstract, systems demonstration abstract, and technical papers that were presented at the 2010 ERCIM Workshop on Constraint Solving and Constraint Logic Programming held on November 25th–26th 2010 at Fraunhofer FIRST, Berlin, Germany. This event was run on behalf of the ERCIM Working Group on Constraints\(^1\). ERCIM, the European Research Consortium for Informatics and Mathematics, aims to foster collaborative work within the European research community and to increase co-operation with European industry. Leading research institutes from eighteen European countries are members of ERCIM. The ERCIM Constraints working group aims to bring together ERCIM researchers that are involved in research on the subject of constraint programming and related areas.

Constraints have recently emerged as a research area that combines researchers from a number of fields, including Artificial Intelligence, Programming Languages, Symbolic Computing and Computational Logic. Constraint networks and constraint satisfaction problems have been studied in Artificial Intelligence since the 1970s. Systematic use of constraints in programming emerged in the 1980s. The constraint programming process involves the generation of requirements (constraints) and the solution of these requirements, by specialised constraint solvers. Constraint programming has been successfully applied in numerous domains. Recent applications include computer graphics (to express geometric coherence in the case of scene analysis), natural language processing (construction of efficient parsers), database systems (to ensure and/or restore consistency of the data), operations research problems (like optimization problems), molecular biology (DNA sequencing), business applications (option trading), electrical engineering (to locate faults), circuit design (to compute layouts), etc. Current research in this area deals with various foundational issues, with implementation aspects and with new applications of constraint programming. The concept of constraint solving forms the central aspect of this research.

This year’s workshop programme comprised an invited talk from Chris Beck (University of Toronto), the abstract of which is included here. The main technical programme also comprised talks and software demonstrations from many constraints researchers on current aspects of their research agenda.

We would also like to thank our sponsors who provided the support necessary to make this event a success. Finally, we would like to sincerely thank the authors of papers, the speakers, and the attendees, for such an interesting and engaging programme.

Barry O’Sullivan, University College Cork, Ireland
Armin Wolf, Fraunhofer FIRST, Berlin, Germany

\(^1\)http://wiki.ercim.org/wg/Constraints
Workshop Organisation

Workshop Chairs

Barry O’Sullivan, University College Cork, Ireland
Armin Wolf, Fraunhofer FIRST, Berlin, Germany

Invited Speaker

J. Christopher Beck, University of Toronto, Canada

Supporting Institution

Fraunhofer FIRST, Berlin, Germany

Sponsors

Cork Constraint Computation Centre, Ireland
ERCIM Working Group on Constraints
Table of Contents

Invited Talks

CP/MIP Hybridization via Logic-based Benders Decomposition 1
   J. Christopher Beck

Systems Demonstration

CP Applications at Fraunhofer FIRST 2
   Armin Wolf and Hans-Joachim Goltz

Regular Papers

Solving the Rotation Assignment Problem for Airlines Using Constraint Programming 3
   Son Tung Nguyen and Roman Barták

An Adaptation of Path Consistency for Boolean Satisfiability: a Theoretical View of the Concept 16
   Pavel Surynek

A Complete Filtering Algorithm for Cumulative Not-First/Not-Last rule in $O(n^2|H|\log n)$ 31
   Roger Kameugne and Laure Pauline Fotso

Almost Square Packing 43
   Helmut Simonis and Barry O’Sullivan

Creating Tests for a Family of Cost Aware Resource Constraints 58
   Tarik Hadzic and Helmut Simonis

(Co)inductive Semantics for Constraint Handling Rules 74
   Remy Haemmerlé
CP/MIP Hybridization via
Logic-based Benders Decomposition

J. Christopher Beck
University of Toronto, Canada
jcb@mie.utoronto.ca

Abstract. Logic-based Benders decomposition (LBBD) is a general decomposition scheme that has been applied to a number of hard combinatorial problems in the past 10 years. While not inherent to the approach, LBBD provides an excellent architecture for combinations of mixed-integer programming and constraint programming. In this talk, I will present two such hybrid models – for a facility location-truck allocation problem and for a maintenance scheduling problem – that demonstrate orders-of-magnitude improvements over existing approaches. Time permitting, investigations of branch-and-check, a generalization of LBBD, will be presented.
CP Applications at Fraunhofer FIRST

Armin Wolf and Hans-Joachim Goltz

Fraunhofer FIRST, Berlin, Germany
{armin.wolf|hans-joachim.goltz}@first.fraunhofer.de

Abstract. The CP applications realized at Fraunhofer FIRST are ranging from production scheduling, timetabling, interactive appointment and surgery scheduling in medicine over load distribution in maintenance/repair/overhaul and in energy networks to partition scheduling for multi-core/many-core architectures. Some of these systems will be presented in live demonstrations.
Solving the Rotation Assignment Problem for Airlines Using Constraint Programming

Son Tung Nguyen, Roman Barták
Charles University in Prague, Faculty of Mathematics and Physic, Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
n.sontung@gmail.com, bartak@ktiml.mff.cuni.cz

Abstract. This paper deals with one of the most important components of the airlines planning process – a rotation assignment problem. The problem is formulated as a task to determine which aircrafts should operate given flights under the specific sequencing constraints. We propose a constraint model that fully describes the problem, suggest some enhancements to improve the efficiency of the model, and also describe the dedicated search strategy. Finally, we encapsulate the model into a local search procedure in the style of Large Neighborhood Search. The variants of the model are experimentally compared with other techniques using real-life data. The comparison shows that the proposed model achieves significantly better quality of schedules in acceptable time.

Keywords: rotation assignment, airlines, constrained optimization.

1 Introduction

Optimizing resources, namely using of aircrafts, is one of the critical aspects of success in airlines industry. The rotation assignment problem belongs among the base optimization tasks of the airlines planning process. This problem is about allocating aircrafts to flying and non-flying activities (maintenance etc.) respecting some sequencing constraints. A sequence of activities allocated to a single aircraft is called a rotation, hence the name of the problem. The activities are fixed in time and they have starting and ending location. The rotation must satisfy certain constraints about the timing and locations of subsequent activities. Some of these constraints are hard, such as that the ending location of the activity must be identical to the starting location of the subsequent activity in the same rotation, while the other constraints are soft, such as that there are no overlaps of activities in time. The violation of soft constraints is penalized and these penalties together with the cost of using the resources form the objective function.

The rotation assignment problem has been extensively studied in Operation Research [6,8] as utilizing the aircraft use is one of the most important concerns of the airlines industry. One of the first proposals regarding the problem is the work of Levin [10], where he proposed a network flow model for the problem and then solved it by using LP techniques called Dantzig-Wolfe decomposition and Delayed column generation [3]. Another significant contribution [1] regards the combined fleet
assignment and aircraft routing problem which is solved by an approach based on maintenance feasible strings of activities, which are combined to create feasible routes, within the framework of delayed column generation and a technique called Branch-and-Price [2]. Grönkvist [7] used constraint programming (CP) to get a good initial solution for the column generation process to solve the Rotation assignment problem, since a good initial solution makes the convergence to optimum much faster. He built the CP model on a graph to find paths with certain properties in it. Such path models are quite typical for this type of problems and we will also exploit a similar idea. However, Grönkvist’s CP model handles only hard constraints and optimization is realized outside the model. Kilborn [9] extended the model in such a way that the activities violating the required properties are discarded and he then minimized the number of discarded activities.

The common feature of above models is focusing on minimizing the number of used resources or discarded activities. The main difference of our approach is the attempt to allocate all activities to aircrafts and expressing the number of used aircrafts and violated soft constraints explicitly in an objective function whose value is being minimized. We will first formalize the problem to be solved and propose an abstract model based on graphs. Then we will describe the base constraint model and some enhancements including encapsulation to local search. Finally, we will present a preliminary experimental comparison with existing approaches using real-life data.

2 Preliminaries

The Rotation Assignment Problem deals with the allocation of activities (flights, maintenance) to aircrafts. Let \( R = \{ r_i \mid 1 \leq i \leq m \} \) be the set of \( m \) resources representing the aircrafts and \( A = \{ a_i \mid 1 \leq i \leq n \} \) be the set of \( n \) non-preemptive activities. Each activity \( a_i \) is characterized by the following a priori known attributes:

- \( s_i \) defines the start time of the activity
- \( p_i \) defines the duration of the activity
- \( e_i \) defines the finish time of the activity (\( e_i = s_i + p_i \) holds)
- \( bt_i \) defines the non-negative time period during which the activity consumes the allocated resource before it starts
- \( at_i \) defines the non-negative time period during which the activity consumes the allocated resource after it ends
- \( sl_i \) defines the starting location of the activity
- \( el_i \) defines the ending location of the activity
- \( ac_i \) defines a subset of resources \( R \) to which the activity can be allocated

We distinguish between two types of activities: flying activities (flights) and reservation activities (maintenance). The reservation activities have identical starting and ending locations (\( sl_i = el_i \)) and they are pre-allocated to resources (\( ac_i \) is singleton) which differentiates them from the flying activities.

Each activity should be allocated to a resource which can be described by the decision variable \( ar_i \) whose value we are looking for (the resource is known for the reservation activities). The schedule is defined by the instantiation of all decision
variables so each activity is allocated to exactly one resource. To be more precise, the schedule is defined by the sequences of activities allocated to each resource. Let \( S_r = (a_1, \ldots, a_k) \) be a sequence of \( r_k \) activities allocated to resource \( r \) then \( a_i \in S_r \iff a_{ri} = r \). We call \( S_r \) a rotation or a resource schedule.

There are two types of constraints imposed on the schedule. The soft constraints describe the preferences. They can be violated but the violation is penalized by some cost. The hard constraints describe the requirements and they cannot be violated in feasible schedules. There are three types of hard constraints:

(C1) Each activity must be allocated to a compatible resource, \( a_{ri} \in ac_i \).

(C2) Each activity is allocated to exactly one resource, that is, for any pair of resources \( p \neq r \) it holds that \( S_p \cap S_r = \emptyset \) and \( \cup_{r \in R} S_r = A \) (or alternatively \( \forall a_i \in A \exists r \in R: a_{ri} = r \)).

(C3) For each pair of activities \( a_i \) and \( a_{i+1} \) that are subsequent in some resource schedule \( S_r \) the ending location of the earlier activity must be equal to the starting location of the later activity, \( el_{ri} = sl_{r_{i+1}} \).

(C4) Each resource schedule is ordered in the increased starting time of activities, that is, \( \forall a_i, a_{i+1} \in S_r: s_{ri} < s_{r_{i+1}} \).

There are two kinds of soft constraints specifying how much the activities can overlap in time. Formally:

(C5) For each pair of activities \( a_i \) and \( a_j \) appearing in the same resource schedule \( S_r \) such that \( i < j \), it must hold \( e_i \leq s_j \) (activities do not overlap in time).

(C6) For each pair of activities \( a_i \) and \( a_j \) appearing in the same resource schedule \( S_r \) such that \( i < j \), it must hold \( (s_j - e_i) \geq d_{ij} \) where \( d_{ij} = \max\{ a_{ri}, b_{tj} \} \) if both activities are flying activities and \( d_{ij} = \min\{ a_{ri}, b_{tj} \} \) otherwise.

It is easy to observe that there is a relation between the constraints (C5) and (C6). In particular, if (C5) is violated then (C6) is also violated. The reason is that (C5) requires \( (s_j - e_i) \geq 0 \) while (C6) requires \( (s_j - e_i) \geq d_{ij} \geq 0 \).

### 2.1 Schedule Quality Evaluation

The quality of schedules is evaluated based on the level of violation of soft constraints and usage of resources. Let \( S \) be a schedule, that is, a set of resource schedules. Then the objective function \( F \) is a sum of the costs for the resource usage and the costs of violating the soft constraints within the resource schedules:

\[
F(S) = \sum_{S_r \in S} (\pi_r + \sum_{a_i, a_{i+1} \in S_r, i < j} \pi(a_{ri}, a_{r_{i+1}}))
\]

where \( \pi_r \) is the cost for using resource \( r \) and \( \pi(a_i, a_j) \) is a constant describing the violation of the soft constraints between \( a_i \) and \( a_j \) in the following way:

- if (C5) is violated then let \( \tau = | s_j - e_i |, C_a = C_{a,t}, C_b = C_{b,t}, C_c = C_{c,t}, C_d = C_{d,t} \),

\[
LT = Lt_1
\]
if (C6) is violated but not (C5) then let \( \tau = d_{i,j} - (s_j - e_i) \), \( C_a = C_{a,2} \), \( C_b = C_{b,2} \), \( C_c = C_{c,2} \), \( C_d = C_{d,2} \), \( L_I = L_{I_2} \)

\[
\pi(a_i, a_j) = \begin{cases} 
C_a + C_b \cdot \tau + C_c \cdot \tau^2 & \text{if } L_I \leq \tau \\
C_d & \text{if } 0 < \tau < L_I 
\end{cases}
\]

Note that \( \pi, C_{a,1}, C_{b,1}, C_{c,1}, C_{d,1}, L_{I_1}, C_{a,2}, C_{b,2}, C_{c,2}, C_{d,2}, L_{I_2} \) are user defined constants and the above complex formula for computing the cost of constraint violation was given by the user. We include the formula to show the relation with the above defined soft constraints; our solving approach works with any non-negative cost \( \pi(a_i, a_j) \). We call any feasible schedule \( S^\text{optimal} \) if it minimizes the value of function \( F(S) \).

3 An Abstract (Graph) Model

The Rotation Assignment Problem is a type of resource allocation problem with specific sequencing constraints. To capture the nature of the problem we accommodated the weighted connection network [11] describing the sequencing constraints similarly to [12]. The weighted connection network is a directed graph \( G = (V,E,c) \), where each vertex is annotated either by a resource (resource vertices) or by an activity (activity vertices) and there are two special vertices: source and sink \((|V| = |R| + |A| + 2)\). The arcs \( E \) and their costs \( c \) are defined as follows:

- for each resource \( r \in R \) there is an arc from the source vertex to the vertex annotated by \( r \); the weight (cost) of this arc is \( \pi_r \);
- from each vertex annotated by resource \( r \) there is an arc to each vertex annotated by activity \( a_i \) that can be allocated to \( r \), that is, \( r \in ac_i \); the weight (cost) of this arc is 0;
- there is an arc between two different vertices annotated by activities \( a_i \) and \( a_j \) if these two activities can be allocated to the same resource \( (ac_i \cap ac_j \neq \emptyset) \) and the hard constraints (C3) and (C4) between them are not violated; the cost of arc is \( \pi(a_i, a_j) \);
- finally, there is an arc from each activity vertex to the sink vertex; the cost of this arc is 0.

Figure 1 gives an example of the weighted connection network. It is easy to observe that the weighted connection network is acyclic and that each path from source to sink describes a non-empty resource schedule (even if possibly infeasible). To ensure that the path source-sink describes a feasible resource schedule, we impose the following restriction on the path. Let \( G_p \) be the induced graph in \( G \) defined by path \( p \). This graph contains exactly the vertices from \( p \) and all arcs from \( G \) over these vertices. The path \( p \) represents a feasible resource schedule if the resource vertex in \( p \) (there is exactly one such vertex in \( p \)) is connected by arcs in \( G_p \) to all activity vertices in \( p \). This additional restriction represents the constraint (C1) while the constraints (C3) and (C4) are satisfied due to the definition of arcs in \( G \). Moreover, the cost of \( G_p \) (the sum of all arc costs) is the cost of the resource schedule represented by \( p \). In summary, finding a feasible schedule corresponds to finding a set of disjunctive paths from source to sink in \( G \) (the paths share only the source and sink vertices) such that each
vertex representing an activity is included at some path. This ensures that the constraint (C2) is satisfied. If the sum of costs of these paths is minimal among all such sets of paths then we obtain an optimal schedule.

![Fig. 1. The structure of the weighted connected network.](image)

4 A Base CP Approach

We suggest the following constraint model to describe the rotation assignment problem. Recall, that two types of decisions are taken: the allocation of activities to resources and the sequencing of activities allocated to the same resource. Activity sequencing corresponds to finding the paths in the weighted connection network $G$ which can be modeled by using a variable $\text{Successor}(i)$ for each vertex $i$. The domain of this variable contains all vertices that are connected to $i$ in $G$. This model ensures that there is exactly one outgoing arc from each vertex. This is not true for the source vertex so we omit the source vertex from the model and we start the paths in the resource vertices. For the same reason we omit the sink vertex which has no outgoing arcs. There is still an open question to which vertex the last nodes in the path should be connected via the $\text{Successor}$ variables. We use the following method: for each activity vertex we add an arc with zero cost leading to each resource vertex to which the activity can be allocated. Moreover, we add a loop arc with zero cost to each resource vertex. This arc is used when no activity is allocated to the resource. Figure 2 shows how the modified graph looks for the weighted connection network from Figure 1. As a result, instead of paths, we are now looking for the cycles covering all the nodes in the modified graph. To ensure that the cycles are disjoint or in other words, that there is exactly one incoming arc for each vertex, we post the \textit{all-different} constraint \cite{14} among all the $\text{Successor}$ variables.

The next modeling step is ensuring that the cycles describe the feasible resource schedules, namely, that each cycle contains exactly one resource vertex and that all activity vertices correspond to the activities that can be allocated to the resource. It is easy to see that each cycle contains at least one resource vertex (there are no cycles containing only activity nodes due to the construction of the graph). However, it may happen that two or more resources are in some cycle or that an activity incompatible
with the resource is in the cycle. To resolve this problem we introduce \textit{Resource}(i) variable for each vertex \textit{i}. This variable identifies the resource to which the vertex belongs. The domain of this variable is a resource id for the resource vertex, or a set of resource ids to which the activity can be allocated for activity vertices (Figure 2).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{The modified weighted connected network; the sets next to the vertices represent the domains of the Resource variables.}
\end{figure}

The \textit{Resource} variables are connected to the \textit{Successor} variables to ensure that all vertices in the same path are assigned to an identical resource. In particular, the successor of a given vertex is assigned to the same resource as that vertex – \textit{Resource}(\textit{Successor}(\textit{i})) = \textit{Resource} (\textit{i}). Such a constraint can be modeled for example using the well known element constraint as \textit{element}(\textit{Successor}(\textit{i}), \textit{Resource}, \textit{Resource}(\textit{i})), where \textit{Resource} is a sequence of \textit{Resource}(\textit{i}) variables ordered in increasing \textit{i} (we use natural numbers to identify the nodes).

It remains to formalize the objective function. We will use the cost of arcs from the weighted connected network. Just recall that we removed the source vertex and hence also the arcs with the resource cost. We put this cost in the modified graph to the arcs going from the resource vertices to the activity vertices. In particular the cost of arc \((\textit{i}, \textit{j})\), where \textit{i} is a resource vertex and \textit{j} is an activity vertex, will now be \(\pi_i\).

Let us assume that the resource vertices (and resources) are numbered 1 to \textit{m}, the activity vertices \textit{m} + 1 to \textit{m} + \textit{n}, \textit{E} is the set of arcs in the modified graph, and \(c(\textit{i}, \textit{j})\) is the cost of arc \((\textit{i}, \textit{j})\). The base constraint model looks as follows:

\begin{align*}
\text{minimize} & \sum_{\text{all arcs}} c(\text{\textit{i}, \textit{Successor}}(\textit{i})) + \sum_{\text{\textit{i} \in \text{\textit{m} + 1 \text{ to} \text{\textit{m} + \textit{n}}}}} c(\text{\textit{i}, \textit{j}}) \text{ under the constraints:} \\
\text{Successor}(\textit{i}) & \in \{ \textit{j} \mid (\textit{i}, \textit{j}) \in \text{\textit{E}} \} \quad \forall i, 1 \leq i \leq m+n \\
\text{Resource}(\textit{i}) & = \textit{i} \quad \forall i, 1 \leq i \leq m \\
\text{Resource}(\textit{i}) & \in \text{\textit{ac}}_i \quad \forall i, m < i \leq m+n \\
\text{\textit{element}}(\text{\textit{Successor}}(\textit{i}), \text{\textit{Resource}}, \text{\textit{Resource}}(\textit{i})) & \quad \forall i, 1 \leq i \leq m+n \\
\text{\textit{all_different}}(\text{\textit{Successor}}) &
\end{align*}

where \textit{Resource} is a list of \textit{Resource} variables, \textit{Successor} is a list of \textit{Successor} variables and

\begin{align*}
\text{\textit{AllSuccs}}(\textit{i}) = & \{ \text{\textit{Successor}}(\textit{i}) \} \cup \text{\textit{AllSuccs}}(\text{\textit{Successor}}(\textit{i})) \quad \text{if } \text{\textit{Successor}}(\textit{i}) > \textit{m} \\
& = \emptyset \quad \text{otherwise}
\end{align*}
4.1 Search Strategy

To solve the constraint model we use a standard branch-and-bound method where depth-first search interleaves with inference via constraint propagation. First, one should realize that it is enough to instantiate either the Resource variables or the Successor variables as the other set of variables is instantiated by means of constraint propagation. The search space defined by the Resource variables is much smaller than the search space defined by the Successor variables because there is much less resources than activities. Hence we decided to instantiate the Resource variables only (this decision was confirmed experimentally – instantiating the Resource variables was order of magnitude faster than instantiating the Successor variables).

It remains to choose the variable and value ordering heuristics. We tried the traditional variable ordering heuristics such as dom but they did not work well so we decided for a problem dependent heuristic. The idea is to mimic left-to-right scheduling so we always select the activity with the earliest start time first. Regarding to which resource the activity is allocated, we use the cost as the primary decision criterion. The activity is allocated to the resource where the increase of cost is minimal. This includes the cost for using the resource, if not used yet by earlier activities, and the cost for violating the soft constraints with the preceding activities (they are already allocated to resources due to variable ordering heuristics). The ties are broken by some form of look-ahead where the future reservation activities are assumed (they are pre-allocated to resources). Let \( \text{g}_r \) be the gap between the finish time of the currently allocated activity and the start time of the closest reservation activity in resource \( r \). Among the resources with the same cost of allocation, we first select the resource with maximal \( \text{g}_r \) (to leave most flexibility for other allocations to this resource) then the resource with minimal \( \text{g}_r \) (so the available time is exploited and the resource usage is not scattered in time), and finally the remaining resources in some static order defined in the input.

5 Model Improvements

The initial experiments showed that the performance of the base constraint model is not satisfactory. This section describes the improvements we did to obtain better performance. These improvements are based on three ideas: using customized inference rather than the general inference techniques, better integration of search and inference, and using incomplete/repair techniques. Note that the proposed improvements can naturally be realized in modern extensible constraint solvers though the improvements are going beyond the declarative specification of constraint models.

5.1 More Efficient Inference via Tunnelling

Inference in constraint solvers is based on maintenance of local (usually arc) consistency that works as follows. Each time a domain of some variable is changed, the inference procedures for all constraints that contain the affected variable are called
to prune the domains of other variables in these constraints. This process is repeated until a fix point is reached or any domain becomes empty. The potential inefficiency of this approach is that many calls to individual inference procedures may be useless if they do not lead to domain pruning. This was exactly the case of our constraint \( \text{element}(\text{Successor}(i), \text{Resource}, \text{Resource}(i)) \) which is called many times as it involves many variables, but the pruning effect is limited. This has already been observed by Kilborn [9] and Grönkvist [7] so we use a method derived from their approach. We call it *tunneling* as we propagate information through constraints in a specific direction only. This is realized by delayed posting of the constraint until some event happens. We use the construct "if event then constraint" meaning that when some event happens then the constraint is posted (added to the model). In particular, we substituted the above \text{element} constraint by the following rules:

\[
\text{if } \text{Resource}(i) = r \text{ then } \text{element}(\text{Successor}(i), \text{Resource}, r) \\
\text{if } \text{Successor}(i) = j \text{ then } \text{Resource}(i) = \text{Resource}(j)
\]

In a similar style, we can speed up propagation from the \text{Resource} variables that are instantiated during search to the \text{Successor} variables. As we are doing left-to-right scheduling, when the activity \( y \) is allocated to a resource by the search decision then all predecessors are already allocated so we can take the last activity on the same resource and set its successor to \( y \). As the resource can also be allocated by propagation, we need to formulate the tunnel more carefully:

\[
\text{if } \text{Resource}(y) = r \land \forall a, s_x < s_y, \text{Resource}(a) \text{ is assigned} \land \\
\exists x [\text{Resource}(x) = r \land \text{Successor}(x) \text{ not assigned} \land s_x < s_y \land \\
\neg \exists z (\text{Resource}(z) = r \land s_z < s_x < s_y)] \\
\text{then } \text{Successor}(x) = y
\]

Finally, we can use the tunneling principle to implement the procedure for computing the lower bound of the objective function. The idea is that we increase the lower bound of the objective function each time a \text{Successor}(x) variable is instantiated by adding the cost of violated soft constraints between \( x \) and its successors. We explore the successors in the increased order of their start times (downstream the graph) and so we can stop exploring them when we reach a successor not violating the soft constraint with \( x \) (the other successors have even higher start times and they also do not violate the soft constraints). Formally:

\[
\text{if } \text{Successor}(x) = y \land \forall a, s_x < s_y, \text{Successor}(a) \text{ is assigned} \\
\text{then } \min(\text{Cost}) := \min(\text{Cost}) + \sum_{y \in \text{ConflSucc}(x)} c(x,y)
\]

where

\[
\text{ConflSucc}(x) = \{ \text{Successor}(x) \} \cup \text{ConflSucc}(\text{Successor}(x)) \\
= \emptyset \text{ otherwise}
\]

5.2 Integration of Search and Inference

In left-to-right scheduling, we build the resource schedules from the beginning to the end. In other words we are building the loops starting with the resource nodes and
continuing downstream in the graph. When the variable $\text{Successor}(x)$ is instantiated to $y$, $y$ is supposed to be removed from the domains of all other $\text{Successor}$ variables via the all-different constraint. However, due to the nature of the graph (see Figure 2) $y$ rarely appears in the domains of the not-yet instantiated variables as the corresponding activities are downstream (right) from $y$. Hence the pruning effect of the constraint is minimal. Similarly, computing the lower bound of the objective function using rule (4) works only when the $\text{Successor}$ variables are instantiated for the nodes downstream from $x$. In such a case, the set $\text{ConflSuccs}(x)$ is large and the lower bound of $\text{Cost}$ can be better estimated. Again, this is rarely the case when we build the loops from left to right.

Based on above observations we suggest to change the variable ordering heuristics and go from right to left, that is, allocating first the activities with the latest start time. This allows us to better exploit the pruning power of the all-different constraint over the $\text{Successor}$ variables and to better estimate the lower bound of the objective function. On the other hand, then the pruning effect of rule (2) almost vanishes as the resource for $\text{Successor}(i)$ is frequently already known ($\text{Successor}(i)$ represents the activity starting after $i$ or a resource). Nevertheless, we can introduce new $\text{Predecessor}$ variables describing which node precedes a given node in the loop and use the idea of rule (2) in the following way:

$$\text{Predecessor}(j) = i \iff \text{Successor}(i) = j$$  \hspace{1cm} (6)

Finally, we should modify the rule (3) to follow the new search order:

$$\text{if Resource}(x) = r \land \forall a, s_x < s_a \text{ Resource}(a) \text{ is assigned} \land \exists y [\text{Resource}(y) = r \land \text{Predecessor}(y) \text{ not assigned} \land s_x < s_y \land \neg \exists z (\text{Resource}(z) = r \land s_x < s_z < s_y)] \text{ then Predecessor}(y) = x \land \text{Successor}(x) = y$$ \hspace{1cm} (7)

### 5.3 Combining Local Search and CP

Our abstract model of finding cycles covering all vertices recalls the vehicle routing problem where large neighborhood search was successfully applied [13,15]. Hence we decided to adopt a similar technique to improve runtime efficiency. The idea is that after obtaining some initial solution, we use the default allocation provided by the user, some part of the allocation is removed and re-optimized using the proposed constraint model. To be more precise, we split the time horizon into a sequence of non-overlapping time windows and we extend each time window slightly on both its ends to allow some overlap and “interaction” between the time windows. Then we explore the time windows from left to right, for each time window we discard allocation of activities in that window and find a new allocation using the CP approach. When all the time windows are explored, we enlarge the size of the time
window and repeat the process until some maximum size of the window is obtained. The initial size of the time window, the ratio of overlap with the neighboring time windows, and the maximal size time window are given as parameters of the algorithm.

6 Experimental Evaluation

We implemented the solving algorithms using Gecode library [5] and compared variants of the algorithms with three optimization techniques that were tried before on the same problem:

- “LIFO/FIFO” is a method based on proprietary heuristics and backtracking search.
- “Complex Solution” adds some improvements to LIFO/FIFO method.
- “Aircraft Reduction” is a variant of “Complex Solution” with the emphasis on reducing the number of used aircrafts.

We compared these methods with three versions of our models:

- ROTAS Final is a complete model with all the enhancements and search heuristics.
- ROTAS No-Tunnel is the complete model without the tunneling constraints.
- ROTAS No-Local is the complete model without using local search to improve the schedule. This method is based on branch-and-bound and when run for enough time, it guarantees finding an optimal solution.

Recall that all our methods are exploring plans with the incremental improvement of plan quality (smaller cost) so we run the models for a fixed time that is supposed to be acceptable by the customers (from a few seconds to a few minutes). We also tried the base constraint model but it was not able to provide any solution within 1000 seconds even for the easiest problems so it is excluded from the experimental results.

The experiments ran on Intel Xeon 3.2 GHz with 2 GB RAM under Windows Server 2003. We present the experimental results for two types of datasets. First, we used three artificially generated datasets with a small number of reservation activities that are “easy” to optimize due to a small number of conflicts. Second, we used three datasets with real-life data containing many reservation activities that introduce conflicts and hence make optimization more complicated.

Table 1 summarizes the experimental results for the artificial datasets and Table 2 for the real-life datasets. We report the size of the problems, the runtime, and the quality of obtained solutions (the number of used resources and cost).

The existing techniques are optimized for speed and our prototype implementation cannot compete with them yet. Nevertheless, the runtime of our methods is assumed to be acceptable by the customers and the customers focus more on the quality of solutions. This is where our method excels. In all datasets, our complete model found significantly better solutions regarding the cost. Moreover, some of the existing methods failed to find a solution for certain datasets (they left some activities unallocated) which is a typical feature of current techniques. Our models showed that
even for such problems, it is possible to find solutions of good quality. The experiments also confirmed that the idea of using the tunneling rules and embedding the constraint model into local search pays off. The local search was particularly useful for real-life data.

Table 1. Experimental comparison of algorithms on three artificial datasets

<table>
<thead>
<tr>
<th>Period in days</th>
<th>Resources in total</th>
<th>Activities total</th>
<th>Reservation activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>16</td>
<td>262</td>
<td>0</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Run time in seconds</td>
<td>Used resources</td>
<td>Schedule cost</td>
</tr>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>9</td>
<td>61350</td>
</tr>
<tr>
<td>Complex Solution</td>
<td>1</td>
<td>10</td>
<td>67300</td>
</tr>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>9</td>
<td>61350</td>
</tr>
<tr>
<td>ROTAS No-Tunnel</td>
<td>5</td>
<td>8</td>
<td>FAIL</td>
</tr>
<tr>
<td>ROTAS No-Local</td>
<td>5</td>
<td>8</td>
<td>56000</td>
</tr>
<tr>
<td>ROTAS Final</td>
<td>5</td>
<td>8</td>
<td>56000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period in days</th>
<th>Resources in total</th>
<th>Activities total</th>
<th>Reservation activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>16</td>
<td>429</td>
<td>0</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Run time in seconds</td>
<td>Used resources</td>
<td>Schedule cost</td>
</tr>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>9</td>
<td>68200</td>
</tr>
<tr>
<td>Complex Solution</td>
<td>1</td>
<td>10</td>
<td>74150</td>
</tr>
<tr>
<td>Aircraft reduction</td>
<td>1</td>
<td>9</td>
<td>68200</td>
</tr>
<tr>
<td>ROTAS No-Tunnel</td>
<td>26</td>
<td>8</td>
<td>64200</td>
</tr>
<tr>
<td>ROTAS No-Local</td>
<td>26</td>
<td>8</td>
<td>63300</td>
</tr>
<tr>
<td>ROTAS Final</td>
<td>26</td>
<td>8</td>
<td>63300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period in days</th>
<th>Resources in total</th>
<th>Activities total</th>
<th>Reservation activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>16</td>
<td>664</td>
<td>0</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Run time in seconds</td>
<td>Used resources</td>
<td>Schedule cost</td>
</tr>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>9</td>
<td>83450</td>
</tr>
<tr>
<td>Complex Solution</td>
<td>1</td>
<td>10</td>
<td>89400</td>
</tr>
<tr>
<td>Aircraft reduction</td>
<td>1</td>
<td>9</td>
<td>83450</td>
</tr>
<tr>
<td>ROTAS No-Tunnel</td>
<td>70</td>
<td>8</td>
<td>83650</td>
</tr>
<tr>
<td>ROTAS No-Local</td>
<td>70</td>
<td>8</td>
<td>79450</td>
</tr>
<tr>
<td>ROTAS Final</td>
<td>70</td>
<td>8</td>
<td>79900</td>
</tr>
</tbody>
</table>
Table 2. Experimental comparison of algorithms on three real-life datasets

<table>
<thead>
<tr>
<th>Period in days</th>
<th>Resources in total</th>
<th>Activities total</th>
<th>Reservation activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>429</td>
<td>157</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Run time in seconds</th>
<th>Used resources</th>
<th>Schedule cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>-</td>
<td>FAIL</td>
</tr>
<tr>
<td>Complex Solution</td>
<td>1</td>
<td>-</td>
<td>FAIL</td>
</tr>
<tr>
<td>Aircraft reduction</td>
<td>1</td>
<td>8</td>
<td>218427650</td>
</tr>
<tr>
<td>ROTAS No-Tunnel</td>
<td>43</td>
<td>8</td>
<td>238715700</td>
</tr>
<tr>
<td>ROTAS No-Local</td>
<td>43</td>
<td>8</td>
<td>180028950</td>
</tr>
<tr>
<td>ROTAS Final</td>
<td>43</td>
<td>8</td>
<td>180028950</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period in days</th>
<th>Resources in total</th>
<th>Activities total</th>
<th>Reservation activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>8</td>
<td>590</td>
<td>194</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Run time in seconds</th>
<th>Used resources</th>
<th>Schedule cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>-</td>
<td>FAIL</td>
</tr>
<tr>
<td>Complex Solution</td>
<td>1</td>
<td>8</td>
<td>257591700</td>
</tr>
<tr>
<td>Aircraft reduction</td>
<td>1</td>
<td>8</td>
<td>257591700</td>
</tr>
<tr>
<td>ROTAS No-Tunnel</td>
<td>65</td>
<td>8</td>
<td>239835750</td>
</tr>
<tr>
<td>ROTAS No-Local</td>
<td>65</td>
<td>8</td>
<td>256942850</td>
</tr>
<tr>
<td>ROTAS Final</td>
<td>65</td>
<td>8</td>
<td>239835750</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period in days</th>
<th>Resources in total</th>
<th>Activities total</th>
<th>Reservation activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>8</td>
<td>1038</td>
<td>352</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Run time in seconds</th>
<th>Used resources</th>
<th>Schedule cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIFO/FIFO</td>
<td>1</td>
<td>-</td>
<td>FAIL</td>
</tr>
<tr>
<td>Complex Solution</td>
<td>2</td>
<td>8</td>
<td>286822150</td>
</tr>
<tr>
<td>Aircraft reduction</td>
<td>2</td>
<td>8</td>
<td>288156150</td>
</tr>
<tr>
<td>ROTAS No-Tunnel</td>
<td>252</td>
<td>8</td>
<td>300542700</td>
</tr>
<tr>
<td>ROTAS No-Local</td>
<td>252</td>
<td>8</td>
<td>566059200</td>
</tr>
<tr>
<td>ROTAS Final</td>
<td>252</td>
<td>8</td>
<td>253790950</td>
</tr>
</tbody>
</table>

7 Conclusions

The paper presents a constraint programming approach to solve a rotation assignment problem for airlines. The paper shows that a carefully engineered constraint model produces better-quality solutions than some existing techniques in acceptable time which is critical for the customers.
We presented in the paper the process of building the final model starting with the abstract graph model, formulating the base CP model and then improving this model by observing and correcting its deficiencies. This can serve as a guideline how CP can be used for solving real-life problems so the contribution is not only in the final solving approach but also in the process how this final model was found. Though the paper describes a particular application, we believe that the presented ideas and techniques can be applied in other areas too.

Acknowledgments. The research is supported by the Czech Science Foundation under the contract P202/10/1188. We would like to thank The Kite s.r.o. for providing example data sets.

References

An Adaptation of Path Consistency for Boolean Satisfiability: a Theoretical View of the Concept

Pavel Surynek*

Charles University in Prague
Faculty of Mathematics and Physics
Department of Theoretical Computer Science and Mathematical Logic
Malostranské náměstí 25, Praha, 118 00, Czech Republic
pavel.surynek@mff.cuni.cz

Abstract. The task of enforcing certain level of consistency in Boolean satisfiability problem (SAT problem) is addressed in this paper. The concept of path-consistency known from the constraint programming paradigm is revisited in this context. Augmentations how to make path-consistency more suitable for SAT are specifically studied. A stronger variant of path-consistency is described and its theoretical properties are investigated. It combines the standard path consistency on the literal encoding of the given SAT instance with global properties calculated from constraints imposed by the instance – namely with the maximum number of visits of a certain set by the path. Unfortunately, the problem of enforcing this variant of path-consistency turned out to be NP-hard. Hence, various types of relaxations of this stronger version of path-consistency were proposed. The relaxed version of the proposed consistency represents a trade-off between the inference strength and the complexity of its propagation algorithm. A presented theoretical analysis shows that computational costs of the proposed consistency are kept reasonably low. Preliminary experiments also show that the mentioned maximum number of visits calculated on several benchmark SAT instances provide non-trivial amount of information.

Keywords: local consistency, global consistency, path-consistency, CSP, SAT

1 Introduction and Motivation

A method how to increase the inference strength of path-consistency [13, 14] will be described. It combines the standard path-consistency on the literal encoding model [17] of the given Boolean satisfiability (SAT) instance [5] with global properties calculated from the instance. The existence of a path in a graph interpretation of the instance is being checked by the standard path-consistency. In the augmented variants, additional requirements are imposed on the path being checked to exist. Unfor-
Unfortunately, the problem of checking the existence of a path according to augmented requirements turned out to be NP-complete [16]. Hence, various relaxations that still preserve the inference strength of augmented variants above the level of the standard path-consistency were proposed and evaluated. The ultimate goal of whole design of the adaptation of path-consistency is a tool for preprocessing SAT instances. The result of preprocessing should be a simplified SAT instance that is easier to solve.

2 Notations and Definitions

Concepts of constraint satisfaction problem (CSP) [7] and Boolean satisfiability (SAT) [5] need to be established first to make reasoning about path-consistency in the context of SAT easier to understand.

Definition 1 (Constraint satisfaction problem - CSP). Let $\mathbb{D}$ be a finite set representing domain universe. A constraint satisfaction problem [7] is a triple $(X, C, D)$ where $X$ is a finite set of variables, $C$ is a finite set of constraints, and $D : X \rightarrow \mathcal{P}(\mathbb{D})$ is a function that defines domains of individual variables from $X$ (that is, $D(x) \subseteq \mathbb{D}$ is a set of values that can be assigned to the variable $x \in X$). Each constraint from $c \in C$ is of the form $\langle \langle x_1^c, x_2^c, \ldots, x_k^c \rangle, R^c \rangle$ where $k^c \in \mathbb{N}$ is called an arity of the constraint $c$, the tuple $\langle x_1^c, x_2^c, \ldots, x_k^c \rangle$ with $x_i^c \in X$ for $i = 1, 2, \ldots, k^c$ is called a scope of the constraint, and the relation $R^c \subseteq D(x_1^c) \times D(x_2^c) \times \ldots \times D(x_k^c)$ defines the set of tuples of values for which the constraint $c$ is satisfied. The task is to find a valuation of variables $v : X \rightarrow \mathbb{D}$ such that $\forall x \in X$ and $\left( v(x_1^c), v(x_2^c), \ldots, v(x_{k^c}^c) \right) \in R^c \forall c \in C$. □

A constraint $c \in C$ with the scope $\langle x_1^c, x_2^c, \ldots, x_{k^c}^c \rangle$ will be denoted as $c(\langle x_1^c, x_2^c, \ldots, x_{k^c}^c \rangle)$; this notation is useful when the ordering of variables in the scope is not known from the context; when ordering of variables in the scope matters, then a notation $c(x_1^c, x_2^c, \ldots, x_{k^c}^c)$ will be used instead.

A CSP is called binary if all the constraints have the arity of two. The expressive power of a binary CSP is not reduced in comparison with a general one since every CSP can be transformed into an equivalent binary CSP [15]. The key concept of path-consistency [14] that is addressed in this paper is defined for binary CSPs only. It is also convenient to suppose, that each pair of variables is constrained by at most one constraint.

Definition 2 (Boolean satisfiability problem - SAT). Let $B$ be a finite set of Boolean variables; that is, a set of variables that can be assigned either FALSE or TRUE. A Boolean formula $F$ over the set of variables $B$ in a so called conjunctive normal form (CNF) [12] is the construct of the form $\bigwedge_{i=1}^{N} (\bigvee_{j=1}^{K_i} l_i^j)$ where $l_i^j$ with either $l_i^j = y$ or $l_i^j = \neg y$ for some $y \in B$ for $i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, K_i$ is called a literal and $\bigvee_{j=1}^{K_i} l_i^j$ for $i = 1, 2, \ldots, N$ is called a clause. The task is to find a valuation of Boolean variables $b : B \rightarrow \{\text{FALSE, TRUE} \}$ such that $F$ evaluates to TRUE under $b$ while $\neg$ (negation), $\lor$ (disjunction), and $\land$ (conjunction) are interpreted commonly in the Boolean algebra. A formula for that such a satisfying valuation exists is called satisfiable. □
It is a well known result that the language consisting of satisfiable formulas in CNF as well as general ones is an \( NP \)-complete problem \[5, 9\]. It is not difficult to observe that the language of solvable instances of CSP is \( NP \)-complete as well since it just generalizes SAT in fact (constraints are represented by clauses) while membership of CSP into the \( NP \) class is preserved by the generalization.

## 2 Path-consistency in CSP

The standard definition of path-consistency in CSP will be recalled before the augmented versions and their relaxations are introduced. The following definition refers to general paths of variables which is not necessary in fact. However, this style of definition will be more suitable for making intended augmentations.

**Definition 3 (Path-consistency - PC).** Let \((X,C,D)\) be a binary CSP and let \(P = (x_0, x_1, \ldots, x_K)\) with \(x_i \in X\) for \(i = 0, 1, \ldots, K\) be a sequence of variables called a path. A pair of values \(d_0 \in D(x_0)\) and \(d_K \in D(x_K)\) is path-consistent with respect to \(P\) if there exists a valuation \(v: \{x_0, x_1, \ldots, x_K\} \rightarrow \mathbb{D}\) with \(v(x_0) = d_0\) and \(v(x_K) = d_K\) such that constraints \(c((x_i, x_{(i+1) \mod K}))\) are satisfied by \(v\) for every \(i = 0, 1, \ldots, K\). The path \(P\) is said to be path-consistent if all the pairs of values from \(D(x_0)\) and \(D(x_K)\) respectively are path-consistent with respect to \(P\). Finally, the CSP \((X,C,D)\) is said to be path-consistent if it is path-consistent for every path. \(\Box\)

Notice that variables forming the path in the definition do not need to be necessarily distinct. Although the notion of path-consistency seems to be computationally infeasible since there are typically too many paths, it is sufficient to check path-consistency for all the paths consisting of triples of variables only to ensure that the given CSP is path-consistent \[13, 14\]. In other words, although it seems that path-consistency captures the problem globally (a path can go through large portion of variables of the instance), it merely defines a local property.

There exist many algorithms for enforcing path-consistency in a CSP such as PC-4 \[10\] and PC-6 \[1, 3\]. They differ in the representation of auxiliary data structures and the efficiency. The common feature of path-consistency algorithms is however the process how the consistency is enforced. It is done by eliminating pairs of inconsistent values until a path-consistent state is reached (the smallest set of pairs of values such that their elimination makes the problem path-consistent is being pursued). The process of elimination of pairs of values is typically done by pruning extensional representation of constraints (lists of allowed tuples) to forbid more pairs of values.

## 3 Standard Path-consistency in SAT

The aim of this work is to modify path-consistency to make it applicable on SAT and to increase its inference strength by incorporating certain global reasoning into it. The easier task is to make path-consistency applicable on SAT - it is sufficient to model SAT as CSP. A so called literal encoding \[17\], which of the result is a binary CSP, is
particularly used. This kind of encoding is especially suitable since it allows natural expression of path-consistency in terms of graph constructs.

Let $F = \bigwedge_{i=1}^{N}(V_{i}^{R}(l_{i}^{R}))$ be a Boolean formula in CNF over a set of Boolean variables $B$. Let $D = \bigcup_{i=1}^{N}(U_{i}^{R}(l_{i}^{R}))$ be a domain universe; that is, a constant symbol with the stripe is introduced into $D$ for each literal occurrence in $F$ (notice that, each occurrence of a literal corresponds to a different constant symbol). The corresponding CSP $(X, C, D)$ using literal encoding is built as follows: $X = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$; that is, a variable is introduced for each clause of $F$; it holds for $D: X \rightarrow P(D)$ that $D(\sigma_i) = U_{j=1}^{R}(l_{j}^{R})$; that is, the domain of an $i$-th clause contains constant symbols corresponding to all its literals. A constraint $c((\sigma_{i_1}, \sigma_{i_2})) = ((\sigma_{i_1}, \sigma_{i_2}), R^c)$ is introduced over every pair of variables with $i_1, i_2 \in \{1, 2, \ldots, N\} \land i_1 \neq i_2$ where a variable $x \in B$ such that either $x \in D(\sigma_{i_1}) \land \neg x \in D(\sigma_{i_2})$ or $\neg x \in D(\sigma_{i_1}) \land x \in D(\sigma_{i_2})$ exists. Such a constraint $c((\sigma_{i_1}, \sigma_{i_2}))$ then forbids every tuple of values $(l_{i_1}, l_{i_2})$ such that there exists $x \in B$ for that either $l_{i_1} = x \land l_{i_2} = \neg x$ or $l_{i_1} = \neg x \land l_{i_2} = x$ (that is, the tuple $(l_{i_1}, l_{i_2})$ is removed from $R^c$ which has been initially set to $D(\sigma_{i_1}) \times D(\sigma_{i_2})$). A solution of the resulting CSP $(X, C, D)$ corresponds to the valuation of Boolean variables of $B$ that satisfies $F$ and vice versa [17].

![Fig. 1. An illustration of path-consistency in the CSP model of a SAT problem. The SAT problem represented by a formula $F$ shown here is a representation of the requirement of selecting an odd number of variables from every of the following sets to be true: $\{x_1, x_2\}$, $\{x_1, x_3\}$, $\{x_2, x_3\}$. Observe, that there is no satisfying valuation of $F$. However, the pair of literals $\neg x_1$ and $x_3$ from the left most variable and from the right most variable respectively are path-consistent with respect to a depicted path $P$ since they are non-conflicting and there exists a path from the left to the right consisting of edges between neighboring variables connecting allowed pairs of values (the path is marked by bold edges and by darker vertices).](image-url)
Having the CSP model of SAT it is possible to check path-consistency for the corresponding CSP model and proclaim the original SAT path-consistent or path-inconsistent accordingly. If elements of variable domains are interpreted as vertices and allowed tuples of values as directed edges connecting them, then path-consistency with respect to a given path can be interpreted as existence of paths in the resulting directed graph.

More precisely, let \( P = (\sigma_{i_0}, \sigma_{i_1}, ..., \sigma_{i_K}) \) with \( i_j \in \{1, 2, ..., N\} \) for \( j = 0, 1, ..., K \) be a sequence of variables in the literal encoding CSP model \( (X, C, D) \). A directed graph \( G_{PC}^P(P) = (V, E) \), in which path-consistency can be interpreted as the existence of paths, is defined as follows: \( V = \bigcup_{k=0}^{K} D(\sigma_{i_k}) \) and if \( (\tilde{I}_{i_j}^j, \tilde{I}_{i_{j+1}}^{j+1}) \in R^{(\sigma_{i_j}, \sigma_{i_{j+1}}) \bmod K} \) then a directed edge \( (\tilde{I}_{i_j}^j, \tilde{I}_{i_{j+1}}^{j+1}) \) is included into \( E \). A pair of values \( \tilde{I}_{i_j}^j \in D(\sigma_{i_j}) \) and \( \tilde{I}_{i_{j+1}}^{j+1} \in D(\sigma_{i_{j+1}}) \) is path-consistent with respect to the path \( P \) if there is an edge \( (\tilde{I}_{i_j}^j, \tilde{I}_{i_{j+1}}^{j+1}) \) in \( G_{PC}^P(P) \) and there exists a path from the vertex \( \tilde{I}_{i_j}^j \) to the vertex \( \tilde{I}_{i_{j+1}}^{j+1} \) in \( G_{PC}^P(P) \). The graph \( G_{PC}^P(P) \) will be called a graph interpretation of path-consistency – see Fig. 1 for illustration.

Notice that path-consistency is incomplete in the sense that a pair of values may be path-consistent even if there is no solution of the problem that contains this pair of values (see Fig. 1 again). Analogically, the problem may be path-consistent (that is, path-consistent with respect to all the paths) even if it has no solution actually. The partial reason for this weakness of path-consistency is that many constraints are ignored when a pair of values is checked. This is especially apparent if a longer path of variables is considered. Only constraints over pairs of variables neighboring in the path are considered while many constraints such as that for example over the first and the third variable in the path are ignored. This property is disadvantageous especially in SAT where stronger reasoning is typically more beneficial.

For further augmentation of path-consistency, it is also convenient to prepare a so called auxiliary constraint graph for the model with respect to the path \( P \) that reflects all the constraints over the variables of the path \( P \). It is an undirected graph \( G_{CSP}(P) = (V, E) \) and it is defined as follows: \( V = \bigcup_{j=0}^{K} D(\sigma_{i_j}) \); an edge \( \{\tilde{I}_{i_j}^j, \tilde{I}_{i_{j+1}}^{j+1}\} \) is added to \( E \) if \( (\tilde{I}_{i_j}^j, \tilde{I}_{i_{j+1}}^{j+1}) \in R^{(\sigma_{i_j}, \sigma_{i_{j+1}}) \bmod K} \); and all the edges \( \{\tilde{I}_{i_j}^j, \tilde{I}_{i_{j+1}}^{j+1}\} \) for all \( j = 1, 2, ..., K \) and \( j_1, j_2 = 1, 2, ..., K \) and \( j_1 \neq j_2 \). Observe that the auxiliary constraint graph subsumes the graph interpretation with respect to the same path. Notice also, that there is a complete subgraph over vertices corresponding to values from the domain of the same variable.

### 4 Making Path-consistency Stronger

A modification of path-consistency has been proposed to overcome mentioned limitations of the standard version. To increase inference strength of path-consistency additional requirements on the path in the graph model are imposed. These additional requirements reflect constraints over non-neighboring variables in the path of variables. As the auxiliary constraint graph represents an explicit representation of constraints, it is exploited for determining additional requirements.
4.1 An Initial Augmentation of Path-consistency

An approach adopted in this work restricts the size of the intersection of the constructed path with certain subsets of vertices in the graph interpretation of path-consistency. More precisely, let $G_{PC}(P) = (V,E)$ be a graph interpretation of path-consistency in a CSP model of SAT $(X,C,D)$. The set of vertices $V$ is partitioned into disjoint sequences $L_1, L_2, ..., L_M$ called layers (that is, $\bigcup_{i=1}^{M} L_i = V$ and $L_i \cap L_j = \emptyset \ \forall i,j \in \{1,2, ..., M\} \ \land \ i \neq j$; where denotes the union of the sequence $\mathcal{A}$, that is $\mathcal{A} = \bigcup_{i=1}^{n} \{a_i\}$ for $A = \{a_1,a_2, ..., a_n\}$). The maximum size of the intersection of the path being checked to exist with individual layers is determined using the set of constraints $C$ (notice that all the constraints over $P$ are considered – not only constraints over neighboring variables in $P$). This proposal will be called an initial augmentation of path-consistency in the rest of the text.

The concept of the initial augmentation of path-consistency comes from [16]. The process of decomposition of the set of vertices into layers is done over the corresponding auxiliary constraint graph $G_{CSP}(P)$. Vertices of $G_{CSP}(P)$ are decomposed into vertex disjoint stable sets (a stable set is a subset of vertices of a graph where no two vertices are adjacent with respect to edges). The knowledge of such decomposition can be then used to partition vertices into layers that directly correspond to found stable sets. However, determining a stable subset is a difficult task itself. Hence a greedy approach has been used to obtain an acceptable solution. More details about how to decompose vertices into layers greedily for the initial augmentation can be found in [16].

Since it is possible to assign to a variable at most one value from values corresponding to vertices of the stable set in $G_{CSP}(P)$, the maximum size of the intersection of the path with a layer is thus at most 1. Notice, that at most one value from vertices corresponding to the domain of a variable can be selected (this is due to the presence of the complete subgraph over the set of vertices corresponding to the domain of a variable in $G_{CSP}(P)$). Notice further, that if a value corresponding to a vertex in a stable set is selected than all the values corresponding to other vertices of the stable set are ruled out since they are in conflict with the selected value with respect to constraints.

A quite negative result has been obtained in [16]. It has been shown that finding a path, which conforms to the calculated maximum size of the intersection with individual layers, corresponds to finding a Hamiltonian path [4]. This is known to be an $NP$-hard problem. Hence, it is not tractable to find a path that satisfies defined requirements exactly. Moreover, initial experiments showed that it is almost impossible to make any reasonable relaxation of proposed requirements. Every relaxation of requirements on the path being constructed proposed by the author leads to weakening the modified path-consistency down to the level of the standard version of path-consistency (specifically, several adaptations of the algorithm for finding single source shortest paths [6] have been evaluated by the author).

These initial findings founded an effort to further augment requirements on the constructed path in order to allow developing stronger and more efficient relaxations. The result of this effort is a concept of a so called modified version of path-consistency.
4.2 A Modified Version of Path-consistency

Again, partitioning of vertices of $G_{PC}(P)$ into layers is supposed. In addition, the sequencing of variables in the path $P$ is exploited for defining the maximum size of the intersection of the constructed path with layers. Particularly, the path being constructed is required to conform to the calculated maximum size of the intersection with vertices of the layer preceding a given vertex of the path with respect to the sequencing of variables in $P$. The maximum size of the intersection is again imposed by the set of constraints $C$. More precisely, let $L_1, L_2, ..., L_M$ be layers of $G_{PC}(P)$; let a function $\chi: V \rightarrow \mathbb{N}$ defines requirements on the maximum size of intersections imposed by constraints as follows: $\chi(v_j^i)$ is the maximum size of the intersection of the constructed path with a set of vertices \{\text{first occurrences of literals in first four variables of the path $P$}\} with $i \in \{1,2, ..., M\}$ and $j \in \{0,1, ..., K^4\}$. Let a consistency defined by this new requirement on the constructed path be called a modified path-consistency. Observe that this new concept is a generalization of the initial augmentation described above (see Fig 2 for illustration).

![Fig 2](image)

**Fig 2.** An illustration of modified path-consistency in the CSP model of a SAT problem. The maximum size of the intersection of the constructed path with vertices preceding the given vertex (including) in its layer is calculated using constraints for each vertex - these maximum sizes are denoted as the function $\chi$. For example, having $\chi(v_1^2) = 2$ then the constructed path can intersect the subset of vertices \{first occurrences of literals in first four variables of the path $P$\} of the layer $L_1$ in at most two vertices. Observe, that these requirements on the path being constructed rules out its existence for connecting a pair of vertices from the left most variable (occurrence of literal $-x_1$) and from the right most variable (occurrence of literal $x_1$). Compare it with the standard path-consistency in Fig. 1 where the corresponding path connecting the same pair of vertices exists.

It is intractable to construct a path conforming to the maximum sizes of intersections determined by $\chi$ as in the case of the initial augmentation. Nevertheless, it is possible to make a tractable relaxation of these requirements which does not collapse down to the level of the standard path-consistency.

Let us now briefly describe such a tractable relaxation. Suppose that $\chi$ is already known (the process of calculation of $\chi$ will be described in the following section). Let
and be a pair of values for that a consistency is being checked. Two assignments will be maintained: \( \Sigma: V \rightarrow \mathbb{N}_0 \) and \( \psi: V \rightarrow \mathbb{N}_0^{M \times (K+1)} \) where \( \mathbb{N}_0^{M \times (K+1)} \) denotes matrices of the size \( M \times (K + 1) \) over \( \mathbb{N}_0 \). The assignment \( \Sigma \) will express the total number of distinct paths in \( G_{PC}^P(P) = (V, E) \) starting in \( d_0 \) and ending in a given vertex. Observe, that it is easy to calculate \( \Sigma(v) \). It is determined recursively by the expression: \( \Sigma(v) = \sum_{u \in V, (u, v) \in E} \Sigma(u) \), while \( \Sigma(d_0) = 1 \). The assignment \( \psi \) expresses statistical information about paths in \( G_{PC}^P(P) \) starting in \( d_0 \) and ending in a given vertex regarding the size of the intersection with layers. More precisely, an element of \( \psi(v) \) at \( i \)-th row and \( j \)-th column (that is, \( \psi(v)_{i,j} \)) with \( v \in V \), \( i \in \{1, 2, ..., M\} \), and \( j \in \{0, 1, ..., K\} \) represents the number of distinct paths starting in \( d_0 \) and ending in \( v \) intersecting with the layer \( L_i \) in exactly \( j \) vertices that conform to relaxed requirements (that is, the size of the intersection of these paths with \( L_i \) is \( j \)). If the mentioned conformation to relaxed requirements is omitted, the information maintained in \( \psi \) is not difficult to be calculated recursively for every vertex of \( G_{PC}^P(P) \). However, as it is algorithmically more complex calculation, it is deferred to the section devoted to algorithms.

Requirements on the size of the intersection of the constructed path with layers represented by \( \chi \) are relaxed in the following way. If it is detected that all the paths staring in \( d_0 \) and ending in \( v \) intersects the layer containing \( v \) in more vertices than it is allowed by \( \chi \), then it is possible to conclude that there is no path connecting \( d_0 \) and \( v \) that conforms to calculated maximum sizes of intersections with layers. Hence, \( v \) is unreachable from \( d_0 \) under given circumstances. The described relaxation can be expressed using defined assignments \( \Sigma \) and \( \psi \). Let \( L^v \) be a layer containing \( v \) (that is, \( v \in L^v \)). If there is some \( j > \chi(v) \) such that \( \psi(v)_{L^v,j} = \Sigma(v) \), then there is no path connecting \( d_0 \) and \( v \) conforming to the maximum sizes of intersection with layers. Observe, that although there is no \( j > \chi(v) \) such that \( \psi(v)_{L^v,j} = \Sigma(v) \), the required path still need not to exist. This is the principle which is called the relaxation in the context of this paper.

If it is detected that there is no path connecting \( d_0 \) and \( d_K \) that conforms to relaxed requirements on the maximum sizes of intersections with layers, the pair of values \( d_0 \) and \( d_K \) is said to be inconsistent with respect to the modified path-consistency.

5 Modified Path-consistency Enforcing Algorithms

It is necessary to clarify several essential steps in order to be able to enforce modified path-consistency according to the suggestion in the previous section. These essential steps are: how to construct layer decomposition of the graph interpretation of path consistency, then we need to know how to determine maximum sizes of intersections with layers, and finally how to perform modified path-consistency checking.

Constructions carried out in all these steps must regard the objective that the inference ability of the resulting modified path-consistency should as strong as possible (since every step induces a possible relaxation, this means that all these relaxations should not relax the original constraints too much).
This section is devoted to the algorithmic point of view of suggestions from previous sections. Thus aspects that have been explained informally so far will be now explained in all the details using algorithms written in pseudo-code.

5.1 Construction of the Layer Decomposition

The first step is represented by construction of the layer decomposition $L_1, L_2, ..., L_M$ of a given graph interpretation of path consistency $G_{P_C}(P) = (V, E)$ with respect to a path $P$. It has been reported how to construct layer decomposition from the knowledge of the auxiliary constraint graph $G_{CSP}(P)$ within the initial augmentation of path-consistency. The process within modified path-consistency is similar.

Algorithm 1. Construction of a layer decomposition. The input parameters are: $(X, C, D)$ which is a CSP model of the SAT instance, a path $P$, and an auxiliary constraint graph with respect to $(X, C, D)$ the path $P$ as $G_{CSP}(P)$. The output of the algorithm is a sequence of vertex disjoint subsets of $G_{P_C}(P)$ (or equivalently of $G_{CSP}(P)$) which form layer decomposition.

---

**Algorithm 1. Construction of a layer decomposition.**

The input parameters are: $(X, C, D)$ which is a CSP model of the SAT instance, a path $P$, and an auxiliary constraint graph with respect to $(X, C, D)$ the path $P$ as $G_{CSP}(P)$. The output of the algorithm is a sequence of vertex disjoint subsets of $G_{P_C}(P)$ (or equivalently of $G_{CSP}(P)$) which form layer decomposition.

**function** Construct-Layer-Decomposition$(X, C, D), P, G_{CSP}(P) = (V, E))$: sequence

1. let $P = (\sigma_0, \sigma_1, ..., \sigma_M)$
2. let $M \leftarrow 1$
3. let $\Pi \leftarrow \emptyset$
4. for $j = 0, 1, ..., K$ do
5.   let $L_M \leftarrow \emptyset$
6.   for each $u \in D(\sigma_j)$ do
7.     if $u \not\in \Pi$ then
8.       $L_M \leftarrow \emptyset$
9.       $\Pi \leftarrow \Pi \cup L_M$
10.      $M \leftarrow M + 1$
11. return $(L_1, L_2, ..., L_M)$

**function** Construct-Next-Layer$(u_0, \Pi, \sigma, C, D, P, G_{CSP}(P))$: set

1. let $P = (\sigma_0, \sigma_1, ..., \sigma_M)$
2. let $u_0 \in D(\sigma_0)$
3. let $L \leftarrow \{u_0\}$
4. for $j = k + 1, k + 2, ..., K$ do
5.   let $v_{best} \leftarrow \perp$
6.   for each $v \in D(\sigma_j)$ do
7.     if $v \not\in \Pi$ then
8.       $v_{best} \leftarrow v$
9.     if $|\{u \in L \land \{u, v \} \not\in E\}| > |\{u \in L \land \{u, v_{best} \} \not\in E\}|$ then
10.    $L \leftarrow L \cup v_{best}$
11. return $L$

It works with auxiliary constraint graphs $G_{CSP}(P)$ again while construction of each layer is done in the direction from the first variable to the last variable of the path $P$ while sequencing of variables in $P$ are respected. The process is formally expressed using pseudo-code as Algorithm 1.
For each vertex of $G_{CSB}(P)$ it is checked whether it has been already included in some of the previously constructed layers. If not, the construction of a new layer is started and the vertex is included in it. Let $u_0$ be the first vertex of the just started layer. Let it belong to the domain of a variable $\sigma_{i_k}$. Then variables following $\sigma_{i_k}$ in $P$ are traversed and vertices corresponding to their domain elements are checked whether it is advantageous to include them. Vertices not yet included in any of the previously constructed layers are considered only. At most one vertex from each variable domain is included into each layer. If there are multiple vertices remaining in the variable domain, the vertex which is disconnected from the most vertices already included in the currently constructed layer with respect to edges of $E$ is selected for inclusion. The selected vertex is concatenated to the constructed layer finally.

This way of selection of vertices to layers prefers small intersections of the path being constructed with layers. This is due to the fact that selection of vertices into layers according to the described criterion prefers that few variables can have assigned values corresponding to vertices of the layer not to violate constraints. Consequently, this selection is in accordance with the objective of obtaining strongly inferring modified path-consistency. In other words, we are trying to forbid as many paths as possible (in fact, these paths are forbidden by constraints we just need to identify as many of them as possible).

Observe that the worst case time complexity of the algorithm is $O(|P|^2|\mathcal{D}|)$. The worst case space complexity is $O(|V| + |E|)$.

### 5.2 Calculation of Maximum Sizes of Intersection with Layers

The next step is the calculation of maximum sizes of intersections of the path being constructed with layers represented by $\chi: V \rightarrow \mathbb{N}$. It is supposed that layer decomposition has been already constructed. Vertices of each layer are traversed from the beginning. This way of traversal corresponds to the traversal of vertices in the sequence of vertices of the path $P$. The process is formally expressed using pseudocode as Algorithm 2.

Let $L_l = [v^l_0, v^l_1, ..., v^l_{|L_l|}]$ with $l \in \{1, 2, ..., M\}$ be a layer which is currently considered. The value of $\chi(v^l_k)$ with $k \in \{0, 1, ..., |L_l|\}$ is calculated by considering two alternatives. The first alternative is that the vertex $v^l_k$ is considered to be included in the potential intersection with the constructed path. The second alternative is that the vertex $v^l_k$ is not included in this way. The higher value from these two alternatives is eventually assigned to $\chi(v^l_k)$.

In the first alternative, the vertices from the set $\{v^0_0, v^1_1, ..., v^l_{k-1}\}$ that are connected with $v^l_k$ are considered. These vertices can be potentially included into the intersection of the layer $L_l$ with the constructed path together with the vertex $v^l_k$. Then maximum of $\chi$ for these vertices is calculated and this value plus 1 represents the maximum size of the intersection in this alternative (in the case when maximum is not defined - the case of the empty set - the value 0 is used).

In the second alternative, the vertex $v^l_k$ is not considered for inclusion into the intersection with the path. In such a case, the value of $\chi(v^l_k)$ can be inherited from $\chi(v^l_{k-1})$ since the vertex $v^l_k$ has no influence on the size of the intersection.
Algorithm 2. The process of calculation of maximum intersection sizes. The input of the algorithm is a layer decomposition of a $G^P_C(P)$ denoted as $L$ and the auxiliary constraint graph $G_{CSP}(P)$ with respect to $(X,C,D)$ and the path $P$ as $G_{CSP}(P)$ itself. The output of the algorithm is an assignment $\chi : V \rightarrow \mathbb{N}$ which determines the maximum size of the intersection of the path being constructed with the set of vertices of the layer preceding the given vertex. The maximum size of the intersection is determined by constrains imposed by the instance.

function $\text{Calculate-Maximum-Intersection-Sizes}(L, G_{CSP}(P) = (V,E))$: assignment
1: let $L = \{L_1, L_2, \ldots, L_M\}$
2: for $l = 1, 2, \ldots, M$ do
3:     let $L_l = \{v^l_0, v^l_1, \ldots, v^l_{K^l}\}$
4:     $X_{\text{prev}} \leftarrow 0$
5:     for $k = 0, 1, \ldots, K^l$ do
6:         $X_{\text{best}} \leftarrow 0$
7:         for $j = 0, 1, \ldots, k - 1$ do
8:             if $\{v^l_j, v^l_k\} \in E$ then
9:                 if $\chi(v^l_j) > X_{\text{best}}$ then
10:                    $X_{\text{best}} \leftarrow \chi(v^l_j)$
11:                $X_{\text{prev}} \leftarrow \max(X_{\text{best}} + 1, X_{\text{prev}})$
12:         return $\chi$

The following proposition summarizes the correctness of calculation of the maximum sizes of intersection with layers and their application in forbidding paths. It is stated without the proof.

Proposition 1. Let $d_0 \in D(\sigma_{i_0})$ and $d_k \in D(\sigma_{i_k})$. If there is no path $\pi$ connecting $d_0$ and $d_k$ in $G^P_C(P)$ such that $\pi \cap \{v^l_0, v^l_1, \ldots, v^l_{K^l}\} \leq \chi(v^l_k)$ for each $l \in \{1, 2, \ldots, M\}$ and for each $k \in \{0, 1, \ldots, K^l\}$, then there is no assignment of values to variables of $P$ such that all the constraints over $P$ are satisfied. ■

The worst case time complexity of the algorithm for calculating maximum sizes of intersections is $O(|P|^2|D|)$ again. The worst case space complexity is $O(|V| + |E|)$.

5.3 Modified Path-consistency Checking

The final step is modified path-consistency checking for a pair of values $d_0$ and $d_k$ from both ends of the path $P$ being checked supposed that the layer decomposition and maximum sizes of intersections with layers have been constructed. The process is formally expressed using pseudo-code as Algorithm 3.

The algorithm traverses values from domains of the variables of the path $P$ from the beginning of the path $P$ in the graph interpretation of path-consistency $G^P_C(P) = (V,E)$. For each vertex $v \in V$ along the traversal, assignments $\Sigma$ and $\psi$ are calculated.

The major idea within the algorithm is represented by the moment when it is detected that all the paths starting in $d_0$ and ending in a given vertex exceed the maximum size of the intersection with some of the layers (lines 31-33). In such a situation, no of the paths ending in the given vertex can be prolonged into the vertex from the next variable in $P$ without violating the constraints. This claim is formalized by the
following proposition which is again stated without the proof. The proposition also formalizes the second stage of relaxation used in our proposal.

Algorithm 3. The modified path-consistency checking algorithm. The input of the algorithm is a pair of vertices \(d_0\) and \(d_K\) from both ends of the path \(P\) which also the parameter. The next parameters are the layer decomposition \(L\), CSP model of path-consistency \((X, C, D)\), and the graph interpretation of path-consistency \(G_{PC}^P(P)\). The output of the algorithm is the Boolean indicator whether the given pair of values is allowed.

\[
\text{function Check-Modified-Path-Consistency}(d_0, d_K, P, L, (X, C, D), G_{PC}^P(P) = (V, E)): \text{ boolean}
\]

1: if \((d_0, d_K) \not\in E\) then
2: return \(FALSE\)
3: let \(P = (\sigma_{i_0}, \sigma_{i_1}, ..., \sigma_{i_K})\)
4: let \(L = \{L_1, L_2, ..., L_M\}\)
5: let \(d_0 \in L_l\)
6: \(\Sigma(d_0) \leftarrow 1\)
7: \(\psi(d_0) \leftarrow 0^{M \times (K+1)}\)
8: \(\psi(d_0)_{1,1} \leftarrow 1\)
9: for each \(v \in D(\sigma_{i_0})\) do
10: if \(v \neq d_0\) then
11: \(\Sigma(v) \leftarrow 1\)
12: \(\psi(v) \leftarrow 0^{M \times (K+1)}\)
13: for \(k = 1, 2, ..., K\) do
14: for each \(v \in D(\sigma_{i_k})\) do
15: \(\Sigma(v) \leftarrow 0\)
16: \(\psi(v) \leftarrow 0^{M \times (K+1)}\)
17: let \(v \in L_{i_k}\)
18: for each \(u \in D(\sigma_{i_{k-1}})\) do
19: let \(u \in L_{i_{k-1}}\)
20: if \((u, v) \in E\) then
21: \(\Sigma(v) \leftarrow \Sigma(v) + \Sigma(u)\)
22: for \(i = 1, 2, ..., M\) do
23: for \(j = 0, 1, ..., K\) do
24: if \(i = l^*\) then
25: if \(j = 0\) then
26: \(\psi(v)_{1,0} \leftarrow 0\)
27: else
28: \(\psi(v)_{i,j} \leftarrow \psi(v)_{i,j} + \psi(u)_{i,j}\)
29: else
30: for \(j = \chi(v) + 1, \chi(v) + 2, ..., K\) do
31: if \(\psi(v)_{i,j} = \Sigma(v)\) then
32: \(\psi(v)_{i,j} \leftarrow 0^{M \times (K+1)}\)
33: else
34: if \(\psi(d_K) \neq 0^{M \times (K+1)}\) then
35: return \(TRUE\)
36: else
37: return \(FALSE\)
Proposition 2. Let \( d_0 \in D(\sigma_{i_0}) \) and \( d_k \in D(\sigma_{i_k}) \). If the modified path consistency algorithm does not find the path conforming to requirements imposed by the algorithm then there is no path \( \pi \) connecting \( d_0 \) and \( d_k \) in \( G_{pc}^o(P) \) such that \( \pi \cap \{v_0, v_1, \ldots, v_k\} \leq \chi(v_k) \) for each \( l \in \{1,2,\ldots,M\} \) and for each \( k \in \{0,1,\ldots,K\} \). \( \blacksquare \)

The worst case time complexity of the algorithm is \( O(|P|^2|D|^3) \), the worst case space complexity is \( O(|P|^2|D|^2) \).

At this moment, a question may arise whether the modified path-consistency is actually stronger than the standard path-consistency. The answer is that it really is. The particular instance, where the modified path-consistency can infer that there is no consistent pair of values while the standard path-consistency is unable to infer this, is an encoding of the pigeon hole principle [1].

6 Preliminary Experimental Evaluation

We have performed a preliminary experimental evaluation of the above suggestions. The evaluation was targeted on determining the quality of layer decomposition and maximum sizes of intersections with layers determined by proposed algorithms (Algorithm 1 and Algorithm 2).

Table 1. Maximum intersection sizes with the first layer of the layer decomposition. The intersection sizes are calculated in the graph interpretation of several satisfiability instances from SATLib. The determining of layers and maximum size of intersections was done using Algorithm 1 and Algorithm 2 respectively.

<table>
<thead>
<tr>
<th>SAT instance</th>
<th>Maximum intersection with ( L_1 = [v_0, v_1, \ldots, v_3] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ais12.cnf</td>
<td>( \chi(v_0) ) ( \chi(v_1) ) ( \chi(v_2) ) ( \chi(v_3) )</td>
</tr>
<tr>
<td>hanoi4.cnf</td>
<td>1 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>huge.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>jnh1.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>par16-1.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>par16-1-c.cnf</td>
<td>1 2 2 2 2 2 2 2</td>
</tr>
<tr>
<td>pret150_75.cnf</td>
<td>1 1 2 2 2 2 2 2</td>
</tr>
<tr>
<td>s3-3-3-8.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>ssa7552-160.cnf</td>
<td>1 1 2 2 2 2 2 2</td>
</tr>
<tr>
<td>sw100-5.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>Uraj8_5.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>uuf250-0100.cnf</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Several instances from the Satisfiability Library (SATLib) [11] and from [1] were encoded using literal encoding and graph interpretations were constructed for them with respect to the path of variables \( P \) consisting of 8 clauses starting with the first clause of the instance. The remaining clauses for the path \( P \) were selected so that they are the most constrained with already selected clauses for \( P \). This way of selection prefers choosing of highly constrained path for subsequent construction of layers and calculation of maximum sizes of intersections. In order to yet more increase constraining of the generated graph interpretation of \( G_{pc}^o(P) \) the constraint model.
is enriched by additional constraint inferred by the singleton unit propagation \cite{8} (that is, each literal is assigned the value TRUE and unit propagation is performed; assignments to literals enforced by the propagation together with assignment to the original literal form additional constraints).

To provide reproducibility of results all the results presented in this paper, the complete source code, and input data are provided at the web: http://ktiml.mff.cuni.cz/~surynek/research/csclp2010. Results regarding maximum sizes of intersections are shown in Table 1. Parts of the auxiliary constraint graph restricted on the first layer of several instances are shown in Fig 3. Although the experimental results are incomplete at the current stage of the development, they indicate that information captured by constraints over non-neighboring variables in the path of variables is relatively strongly reflected in the maximum sizes of intersections with layers.

![Fig 3. An illustration of first layers of the auxiliary constraint graph of several satisfiability instances. Several sparse and dense graphs are shown together with calculated maximum sizes of intersection of the path being constructed with the layer.](image)

7 Conclusion and Future Work

A new consistency for Boolean satisfiability has been proposed in this paper. The new type of consistency augments the standard path-consistency by exploiting global properties of the input instance. Particularly, stronger requirements are imposed on the path being checked to exist compared to the situation in the standard path-consistency – namely, the size of the intersection of the path with certain sets called layers is restricted. Preliminary experimental evaluation has shown that it is possible to relatively successfully derive the restriction on the size of the intersection with layers on several well known benchmark SAT instances.

For future work we plan to further reduce the relaxation in the calculation of the maximum sizes of intersections with layers. The simple augmentation can be done in the alternative where the new vertex is considered to be selected into the intersection. The number of vertices in the layer not ruled out by constraints when the new vertex
is included should be taken into account. Indeed, the future work is also to evaluate inference strength of the modified path-consistency checking algorithm itself (Algorithm 3). We also would like to evaluate possible applications of modified path-consistency as a preprocessing tool for SAT solving.

References

A Complete Filtering Algorithm for Cumulative Not-First/Not-Last rule in $O(n^2|H|\log n)$

Roger KAMEUGNE$^{1,2}$ and Laure Pauline FOTSO$^3$

$^1$ University of Yaounde 1, Department of Mathematics, P.O Box: 812 Yaounde-Cameroon
$^2$ University of Maroua, Higher Teachers’ Training College, Department of Mathematics, P.O Box: 46 Maroua-Cameroon
rumeugne@yahoo.fr or rumeugne@gmail.com
$^3$ University of Yaounde 1, Department of Computer Science, P.O Box: 812 Yaounde-Cameroon
lpfotso@ballstate.edu

Abstract. Many filtering rules have been designed for various classes of constraint-based disjunctive and cumulative scheduling. Among the filtering rules for cumulative resource, we have overload checking, (extended) edge-finding and not-first/not-last rule which are well known of these rules. The not-first/not-last rule which is not subsumed by the other rules is used to detect tasks that cannot run first/last regarding a set of tasks and prune their time bounds. This paper focuses on the not-first/not-last rule and presents an efficient and complete $O(n^2|H|\log n)$ algorithm for cumulative resource where $n$ is the number of tasks and $H$ the set of distinct earliest completion time of tasks with $|H| \leq n$. The algorithm uses the well known cumulative $\Theta$-tree data structure introduced by Vilím in [1]. Our algorithm thus improve the best known $O(n^3\log n)$ complete algorithm of Schutt et al [2].

1 Introduction

Overload checking [1, 12], (extended) edge-finding [6, 7, 9] and not-first/not-last [2, 3, 5, 4] are well known filtering rules for constraint-based cumulative scheduling. There are generally adapted for the disjunctive resource filtering rule. As (extended) edge-finding, the not-first/not-last rule is a constraint propagation technique for scheduling problems which determine the position of a single task $i$ relatively to a set of tasks $\Omega$ all sharing the same resource. More precisely, the rule check whether the task $i$ cannot run first/last regarding the set $\Omega$ and update the release/deadline of task $i$ accordingly. The not-first/not-last filtering rule is well understood for disjunctive scheduling problems, i.e., problems with unary resources (resource of capacity $C = 1$). Indeed, there exist efficient algorithms running in time $O(n\log n)$ where $n$ is the number of tasks on the resource [8]. For cumulative resource (resource of capacity $C > 1$), not-first/not-last is more complex because tasks may require several capacity units. Schutt et al [2] after proving that the $O(|Sc|n^3)$-implementation (where $Sc$ is the set
of distinct capacity requirements of tasks) of Nuijten [9] is incorrect and incomplete, proposed a new complete not-first/not-last algorithm which runs in time $O(n^3 \log n)$ where $n$ is the number of tasks. In [10], it is mentioned that a private communication of Nuijten presents an $O(n^3)$ not-first/not-last algorithm. To our knowledge, such an algorithm has not yet been published. In [3], we proposed a $O(n^3)$ not-first/not-last algorithm which reaches the same fix point as the complete algorithm after at most $n$ runs. Recently, Schutt and Wolf [5] and ourselves [4] independently proposed a similar sound and incomplete not-first/not-last algorithm for cumulative resource running in $O(n^2 \log n)$. These algorithms do not necessarily achieve the best filtering at the first run, but the filtering is improved at each subsequent run of the algorithm. The fix point is reached after at most $n$ iterations. This idea was already used for the disjunctive not-first/not-last rule [13, 8].

In this paper, we will present a new $O(n^2 |H| \log n)$ efficient and complete not-first/not-last algorithm for cumulative resource. The algorithm uses the well-known cumulative $\Theta$-tree data structure initially used for disjunctive filtering rule [8] and generalized for the cumulative case in [1]. It is a complete version of our sound and incomplete algorithm presented in [4].

The rest of the paper is organized as follows. The next section presents the Cumulative Scheduling Problem (CuSP) and notations used in the paper. Section 3 specifies the Not-First/Not-Last rule. In section 4, the a new complete not-first algorithm running in $O(n^2 |H| \log n)$ time and using $O(n)$ space is presented. The algorithm uses an adaptation of the left cut of the set of tasks by a task introduced Vílim in [8] and the energy envelop introduced by the same author in [1]. Section 5 concludes the paper.

2 Cumulative Scheduling Problem

A Cumulative Scheduling Problem (CuSP) is defined as a given of a resource with given capacity, a set of tasks or activities with given processing times and resource requirements, and it consists of deciding when to execute each task so that resource constraints is satisfied (tasks are processed without interruption and without exceeding the capacity of the resource at any time). Formally, this problem is defined as follow:

Definition 1 (Cumulative Scheduling Problem). A Cumulative Scheduling Problem (CuSP) is defined by the given of a set $T$ of tasks to be performed on a resource of capacity $C$. Each task $i$ must be executed (without interruption) over $p_i$ time units between an earliest start time $r_i$ (release date) and a latest end time $d_i$ (deadline). Moreover, it requires a constant amount of resource $c_i$. It is assumed that all data are integer. A solution of a CuSP is a schedule that assigns a starting date $s_i$ to each task $i$ such that:

\[
\forall i \in T : \quad r_i \leq s_i \leq s_i + p_i \leq d_i \tag{1}
\]

\[
\forall \tau : \quad \sum_{i \in T, s_i \leq \tau < s_i + p_i} c_i \leq C. \tag{2}
\]
The constraint of inequalities (1) ensure that the start time and the end time of each task is feasible while in the inequality (2) the cumulative constraint is satisfied at any time. The CuSP is a sub-problem of the Resource Constrained Project Scheduling Problem (RCPSP) where precedence constraints are relaxed and a single resource is considered at a time. The CuSP is a NP complete problem [10]. It is an extension of the decision variant of both the One-machine problem \((C = 1, c_i = 1)\) and the \(m\)-machines problem \((C = m, c_i = 1)\) and thus is NP-complete in the strong sense [11].

Let \(n = |T|\) be the number of tasks and \(e_i = c_i \cdot p_i\) be the energy of any task \(i \in T\). Throughout the paper, we assume that for any task \(i \in T\), \(r_i + p_i \leq d_i\) and \(c_i \leq C\), otherwise the problem has no solution. \(H = \{r_i + p_i, i \in T\}\) denotes the set of distinct earliest completion times of tasks and \(Sc = \{c_i, i \in T\}\) the set of distinct capacity requirements of tasks. We extend the notation from tasks to set of tasks by setting \(r_\Omega = \min_j \in \Omega r_j\), \(d_\Omega = \max_j \in \Omega d_j\), \(e_\Omega = \sum_j \in \Omega e_j\) where \(\Omega\) is a non empty set of tasks. By convention, if \(\Omega\) is the empty set, \(r_\Omega = +\infty\), \(d_\Omega = -\infty\), \(e_\Omega = 0\).

**Example 1.** Consider three identical workers (resource of capacity \(C = 3\)) who must perform four different tasks \(T = \{a, b, c, i\}\). Tasks \(a\) and \(b\) requires both 3 workers \((c_a = c_b = 3)\) for one hours each \((p_a = p_b = 1)\). Task \(c\) requires 2 workers \((c_c = 2)\) for one hours \((p_c = 1)\). Task \(i\) requires one worker \((c_i = 1)\) for four hours \((p_i = 4)\). Moreover the earliest start time of task \(a\) is four \((r_a = 4)\), two for tasks \(b\) and \(c\) \((r_b = r_c = 2)\) and one for task \(i\) \((r_i = 1)\). The latest end time of task \(i\) is 9 \((d_i = 9)\), four for tasks \(b\) and \(c\) \((d_b = d_c = 4)\) and five for task \(a\) \((d_a = 5)\). Figure 1 illustrates a solution of the CuSP of Example 1.

![Fig. 1. A solution of the CuSP of Example 1](image-url)
3 Not-First/Not-Last rule

The not-first/not-last rule is the pendant of the edge-finding rule which deduce that a task \( i \) cannot be the first (or the last) to execute in \( \Omega \cup \{i\} \) all sharing the same cumulative resource. If a task \( i \) cannot be the first (resp. the last) to execute in \( \Omega \cup \{i\} \) then the release date (resp. deadline) of task \( i \) is updated to the earliest completion time (resp. the latest start time) of the set \( \Omega \).

**Definition 3.** Let \( \Omega \subseteq T \) be a set of tasks of a CuSP of capacity \( C \). The earliest completion time of a task \( i \in T \) is defined as \( ect_i := r_i + p_i \). The earliest completion time of a set of tasks \( \Omega \) is defined as \( ECT_\Omega := \min_{j \in \Omega} ect_j \) if \( \Omega \neq \emptyset \) and \( ECT_\emptyset := +\infty \).

A similar definition is give for the latest start time for a task and a set of tasks in the following definition.

**Definition 4.** Let \( \Omega \subseteq T \) be a set of tasks of a CuSP of capacity \( C \). The latest start time of a task \( i \in T \) is defined as \( lst_i := d_i - p_i \). The latest start time of a set of tasks \( \Omega \) is defined as \( LST_\Omega := \max_{j \in \Omega} lst_j \) if \( \Omega \neq \emptyset \) and \( LST_\emptyset := -\infty \).

With these notions, Proposition 1 presents the not-first/not-last rule as it is studied in the current literature.

**Proposition 1.** [2] Let \( \Omega \) be a set of tasks and let \( i \notin \Omega \).

\[
\begin{align*}
    r_i < ECT_\Omega \land e_\Omega + c_i (\min (ect_i, d_\Omega) - r_\Omega) > C (d_\Omega - r_\Omega) \Rightarrow r_i \geq ECT_\Omega \quad \text{NF} \\
    LST_\Omega < d_i \land e_\Omega + c_i (d_\Omega - \max (lst_i, r_\Omega)) > C (d_\Omega - r_\Omega) \Rightarrow d_i \leq LST_\Omega \quad \text{NL}
\end{align*}
\]

The rule (NF) is called the Not-First rule. It consists of updating release date of every task \( i \) which cannot be the first to be executed in the set \( \Omega \cup \{i\} \). A Not-first algorithm is a procedure that performs all such deductions. Similarly, the rule (NL) is called the Not-Last rule. It consists of updating deadline of every task \( i \) which cannot be the last to be executed in the set \( \Omega \cup \{i\} \). We only consider Not-First rule in this paper (the handling of Not-Last rule is similar).

**Definition 5 (Specification of a complete not-first algorithm).** The Not-First algorithm receives as input an E-feasible CuSP. It produces as output a vector

\[
\langle LB[1], ..., LB[n] \rangle
\]

where

\[
LB[i] = \max \left( r_i, \max_{\Omega \subseteq T \setminus \{i\} \mid \gamma(\Omega,i)} ECT_\Omega \right)
\]

and

\[
\gamma(\Omega,i) := (r_i < ECT_\Omega) \land (e_\Omega + c_i (\min (ect_i, d_\Omega) - r_\Omega) > C (d_\Omega - r_\Omega))
\]

**Example 2.** Let us consider the data of the CuSP instance of Example 1. The not-first rule holds for task \( i \) and the set \( \Omega = \{a\} \) or \( \Omega = \{a, b, c\} \). But the maximum adjustment occur using the pair \( \{\{a\}, i\} \). Indeed, \( r_i = 1 < ECT_\Omega = ect_a = 5 \) and \( e_\Omega + c_i (\min (ect_i, d_\Omega) - r_\Omega) = 3 + 1 \cdot (5 - 4) = 4 > 3 = C (d_\Omega - r_\Omega) \).

Hence, \( LB[i] = 5 \).
4 A Complete Not-First Algorithm in $\mathcal{O}(n^2|H| \log n)$

A complete algorithm regarding the not-first rule finds the most upper bound $LB[i]$ for any task $i$. An incomplete algorithm as it is presented in [3, 5, 4] finds a lower bound $LB'[i]$ such that $r_i < LB'[i] \leq LB[i]$ holds if $r_i < LB[i]$ and $LB'[i] = r_i$ otherwise. In this section, we introduce a new not-first algorithm with $\mathcal{O}(n^2|H| \log n)$ time and $\mathcal{O}(n)$ space complexity. The algorithm is a complete version of our previous sound and incomplete algorithm presents in [4]. It is based on an adaptation of the concept of the left cut of $T$ by a task introduced by Vilím in [8] and the energy envelop introduced by the same author in [1].

We firstly describe how the left cut and energy envelop can be used to check the not-first condition. Secondly, the usage of the cumulative $\Theta$-tree for an efficient computation of the energy envelop of a tasks set is described and the new not-first algorithm is presented. At last, the completeness and the correctness of the new algorithm is proven as well as it complexity.

**Definition 6.** Let $j$ and $i$ be two tasks with $i \neq j$. Let $Ect \in H$ be an earliest completion time of a task. The left cut of $T$ by task $j$ relatively to a task $i$ and the earliest completion time $Ect$ is the set of tasks noted $LCut(T, j, Ect, i)$ and define as:

$$LCut(T, j, Ect, i) := \{k, k \in T \land k \neq i \land r_i < Ect \land Ect \leq ect_k \land d_k \leq d_j\}.$$  

**Definition 7.** Let $\Theta$ be a set of tasks and $i$ be a task. The energy envelope of $\Theta$ relatively to task $i$ noted $Env(\Theta, i)$ is the real number define by:

$$Env(\Theta, i) := \max_{\theta \in \Theta} \{(C - c_i)r_\theta + e_\theta\}.$$  

A task $i \in T$ and a task set $\Omega \setminus \{i\}$ satisfy the not-first condition if $r_i < ECT_\Omega$ and $e_\Omega + c_i(\min(ect_i, d_\Omega) - r_\Omega) > C(d_\Omega - r_\Omega)$ hold. In this paper, we will prove that the not-first condition can be checked using the following inequality for all pair $(i, j)$ of tasks with $i \neq j$ and a given earliest completion time $Ect$:

$$Env(LCut(T, j, Ect, i), i) > Cd_j - c_i \min(ect_i, d_j). \tag{4}$$

**Theorem 1.** Let $i \in T$ be any task of a CuSP. There exists a set of tasks $\Omega \subseteq T \setminus \{i\}$ satisfying the condition of the not-first rule if and only if there is a task $j \in T \setminus \{i\}$ and an earliest completion time $Ect \in H$ such that inequality (4) holds for tasks $i$, $j$ and the earliest completion time $Ect$.

**Proof.** Let $i \in T$ be any task of a CuSP. We show both direction (forward and backward) of the equivalence.

$\Rightarrow$: Let us assume that there is a subset $\Omega \subseteq T \setminus \{i\}$ such that the condition of the not-first rule holds for $\Omega$ and $i$, i.e. $r_i < ECT_\Omega$ and $e_\Omega + c_i(\min(ect_i, d_\Omega) - r_\Omega) > C(d_\Omega - r_\Omega)$. We have to show the existence of a task $j \in T \setminus \{i\}$ and an earliest completion time $Ect \in H$ that satisfies $Env(LCut(T, j, Ect, i), i) > Cd_j - c_i \min(ect_i, d_j)$. 


Let $j \in \Omega$ be a task with $d_j = d_{\Omega}$. Because $r_i < ECT_i$ and $i \notin \Omega$, it holds $\Omega \subseteq L\text{Cut}(T, j, ECT_i, i)$. The inequality $e_i + c_i (\min(ect_i, d_{\Omega}) - r_i) > C(d_{\Omega} - r_i)$ holds since the condition $\gamma(\Omega, i)$ holds and it is equivalent to

$$C - c_i) r_{\Omega} + e_{\Omega} > C d_j - c_i \min(ect_i, d_j).$$

(5)

According to Definition 7 and the inclusion $\Omega \subseteq L\text{Cut}(T, j, ECT_i, i)$ it follow that

$$Env(L\text{Cut}(T, j, ECT_i, i), i) \geq (C - c_i) r_{\Omega} + e_{\Omega} > C d_j - c_i \min(ect_i, d_j).$$

(6)

Hence the inequality holds for task $j$ and the earliest completion time $ECT_i$.

$\Leftarrow$: Let $j \in T \setminus \{i\}$ be a task and $ECT \in H$ be an earliest completion time that satisfies the inequality $Env(L\text{Cut}(T, j, ECT, i), i) > C d_j - c_i \min(ect_i, d_j)$. We have to show the existence of a subset $\Omega \subseteq T \setminus \{i\}$ such that the condition of the not-first rule is satisfied.

Let $\Omega' \subseteq L\text{Cut}(T, j, ECT, i)$ be the tasks set that determines $Env(L\text{Cut}(T, j, ECT, i), i)$, i.e. $Env(L\text{Cut}(T, j, ECT, i), i) = (C - c_i) r_{\Omega'} + e_{\Omega'}$. From $d_{\Omega'} \leq d_j$ it follows that

$$C d_j - c_i \min(ect_i, d_j) \geq C d_{\Omega'} - c_i \min(ect_i, d_{\Omega'}).$$

(7)

Indeed,

- if $ect_i \leq d_{\Omega'}$ then

$$C d_j - c_i \min(ect_i, d_j) = C d_j - c_i \cdot ect_i$$

$$\geq C d_{\Omega'} - c_i \cdot ect_i$$

$$= C d_{\Omega'} - c_i \min(ect_i, d_{\Omega'})$$

- if $ect_i > d_{\Omega'}$ then

$$C d_j - c_i \min(ect_i, d_j) \geq (C - c_i) d_j$$

$$\geq (C - c_i) d_{\Omega'}$$

$$= C d_{\Omega'} - c_i \min(ect_i, d_{\Omega'})$$

From inequality (7), it follows that

$$Env(L\text{Cut}(T, j, ECT, i), i) = (C - c_i) r_{\Omega'} + e_{\Omega'}$$

$$\geq C d_j - c_i \min(ect_i, d_j)$$

$$\geq C d_{\Omega'} - c_i \min(ect_i, d_{\Omega'})$$

and the not-first condition holds for $\Omega'$ and $i$.

Let $i \in T$ be a task and let $ECT \in H$ be an earliest completion time. Let $T \setminus \{i\} = \{j_1, j_2, \ldots, j_{n-1}\}$ be the set of tasks sorted in non-decrease order of deadline i.e. $d_{j_1} \leq d_{j_2} \leq \cdots \leq d_{j_{n-1}}$. Then $L\text{Cut}(T, j_1, ECT, i) \subseteq L\text{Cut}(T, j_2, ECT, i) \subseteq \cdots \subseteq L\text{Cut}(T, j_{n-1}, ECT, i)$. With the ordering of the tasks, the set $L\text{Cut}(T, j, ECT, i)$ can be quickly recomputed from the previous set. Our
previous not-first algorithm \cite{4} works essentially as follow: For each tasks \(i, j \in T\) with \(i \neq j\) we define the left cut of \(T\) by a task \(j\) relatively to the task \(i\) as the set

\[
\text{LCut}(T, j, i) = \{k, \ k \in T \land k \neq i \land r_i < \text{ect}_k \land d_k \leq d_j\}
\]

and checks after sorted the set of tasks by non-decrease order of deadline, the smallest index \(l\) such that \(\text{Env} \left( \text{LCut}(T, j, l), i \right) > C d_{j_l} - c_i \min_{k \in \text{ect}(i, d_{j_l})}\) holds. If so, it updates the release date of task \(i\) to \(r_i = \text{ECT}_{\text{LCut}(T, j, l)}\). This is sound but unfortunately still incomplete as the following example shows.

**Example 3.** Let us consider the data of the CuSP of Example 1. With \(T \setminus \{i\} = \{b, c, a\}\) the set of tasks sorted in non-decrease order of deadline, the not-first condition is detected firstly using the set \(\text{LCut}(T, a, i) = \{b, c, a\}\). Then the release date of task \(i\) is updated to \(\text{ECT}_{\text{LCut}(T, a, i)} = 3\) which is not the most lower bound as it is shown in Example 1.

The concept of our algorithm is to iterate over all left cut of \(T\) \(\text{LCut}(T, j, i)\) for each task \(i \in T\) and to find the maximal earliest completion time \(\text{Ect} \in H\) for which the not-first rule holds. In order to do that, we iterate over all \(\text{LCut}(T, j, \text{Ect}, i)\) after sorted the set \(H\) in increase order of earliest completion time. The first adjustment correspond to the maximum adjustment since the other detection will performed weak adjustment (with the ordering of \(H\)). To detect the not-first condition as it is provide in formula (4), we need to compute the energy envelop of each \(\text{LCut}(T, j, \text{Ect}, i)\). We use the cumulative \(\Theta\)-tree data structure introduce by Vilím \cite{1} to quickly compute this values. The algorithm organize set \(\text{LCut}(T, j, \text{Ect}, i)\) in a balanced binary tree in order to reduce the complexity. Therefore, all classic operation (adding, remove an element in the tree) are done in \(O(\log n)\) and we can integrated the computation of the energy envelop without changed this complexity. The concept will be explained in the following.

In Algorithm 1 the necessary left cut of \(T\) are determined by the two outer loops. The cumulative \(\Theta\)-tree is created innermost loop and it is used to organize set \(\text{LCut}(T, j, \text{Ect}, i)\) in a balanced binary tree. The innermost loop iterate over the set of tasks sorted be non-decreasing deadline. Thus, the different tasks set \(\text{LCut}(T, j, \text{Ect}, i)\) are built step-by-step beginning with the maximal earliest completion time. The ”break” of lines 13 and line 19 allow us to quickly restarts the detection and the update of a new task. Indeed, for a given to tasks \(j\) and \(j'\) with \(d_j \leq d_{j'}\) it follows by the inclusion of \(\text{LCut}(T, j, \text{Ect}, i) \subseteq \text{LCut}(T, j', \text{Ect}, i)\) if the not first condition holds for both set of task with task \(i\) then the same adjustment will be performed (line 13). When a task \(i \in T\) is updated to \(\text{Ect} \in H\) then for the next value of earliest completion time \(\text{Ect}' \in H\) will performed a weak adjustment since the set \(H\) is iterated in decrease order of values (line 19).

To reduce the complexity of the algorithm and to quickly compute the energy envelop of \(\Theta = \text{LCut}(T, j, \text{Ect}, i)\), we organize set \(\text{LCut}(T, j, \text{Ect}, i)\) in a balanced binary tree called \(\Theta\)-tree \cite{1}. Tasks are represented by leaf nodes and sorted by increasing order of \(r_i\) from left to right. Each node \(v\) of the tree holds the following
Algorithm 1 An $O(n^2|H| \log n)$ Not-first algorithm for cumulative resource allocation

**Input:** $H$ is an array of distinct earliest completion time sorted in decrease order

**Private:** $\Theta$: cumulative $\Theta$-tree of tasks set $LCut(T, j, Ect, i)$ balanced by $r_j$

**Result:** $LB'$ array of the new lower bound of release date of tasks.

1: for all $i \in T$ do
2: $LB'[i] := r_i$;
3: end for
4: for all $i \in T$ do
5: for all $Ect \in H$ // $H$ sorted in decreasing order of $Ect$ do
6: if $r_i < Ect$ then
7: $\Theta := \emptyset$;
8: for all $j \in T$ // $T$ sorted in non-decreasing order of $d_j$ do
9: if $Ect \leq r_j \land j \neq i$ then
10: $\Theta := \Theta \cup \{j\}$;
11: if $Env(\Theta, i) > Cd_j - c_i \min(ect_i, d_i)$ then
12: $LB'[i] := \max(LB'[i], Ect)$;
13: break;
14: end if
15: end if
16: end for
17: end if
18: if $LB'[i] == Ect$ then
19: break;
20: end if
21: end for
22: end for
23: for all $i \in T$ do
24: $r_i := LB'[i]$;
25: end for
values:

\[ e_v := e_{\text{Leaves}}(v) \]  
\[ \text{Env}_v(i) := \text{Env}(\text{Leaves}(v), i) \]

where \( \text{Leaves}(v) \) is a set of all tasks represented by leaves of the subtree rooted in \( v \). For a leaf node representing a task \( k \in T \), the values in the tree are set to:

\[ e_v := e_k \]

\[ \text{Env}_v(i) := \text{Env}(\{k\}, i) = (C - c_i)r_k + e_k \]

For internal node \( v \), these values can be computed recursively from their children nodes \( \text{left}(v) \) and \( \text{right}(v) \) as it is specified in Proposition 2.

**Proposition 2.** For a given task \( i \), for an internal node \( v \), values \( e_v \) and \( \text{Env}_v(i) \) can be computed by recursively as follows:

\[ e_v := e_{\text{left}(v)} + e_{\text{right}(v)} \]  
\[ \text{Env}_v(i) := \max\{\text{Env}_{\text{left}(v)}(i) + e_{\text{right}(v)}, \text{Env}_{\text{right}(v)}(i)\} \]

**Proof.** Similar to the proof of Proposition 2 in [1] by replacing \( C \) in the proof of the second item by \( C - c_i \).

**Example 4.** Five tasks \( a \) \((r_a = 0; d_a = 10; p_a = 5; c_a = 1)\), \( b \) \((r_b = 1; d_b = 5; p_b = 1; c_b = 2)\), \( c \) \((r_c = 1; d_c = 5; p_c = 1; c_c = 1)\), \( d \) \((r_d = 2; d_d = 5; p_d = 1; c_d = 1)\) and \( e \) \((r_e = 3; d_e = 5; p_e = 2; c_e = 3)\) share a resource of capacity 3. Let \( i = a \). Figure 2 illustrates the cumulative \( \Theta \)-tree for \( \Theta = \text{LCut}(T, b, 2, a) = \{b, c, d, e\} \). For example, the values at the root of the tree are \( e_{\text{LCut}(T, b, 2, a)} = 10 \) and \( \text{Env}(\text{LCut}(T, b, 2, a), a) = 12 \).

![Fig. 2. \( \Theta \)-tree representing the left cut of \( T \) by task \( b \) relatively to task \( a \)](image)

For a given task \( i \in T \) and a given earliest completion time \( Ect \in H \), using formula (10) and (11), the computation of the values \( e_v \) and \( \text{Env}_v(i) \) can be integrated within usual operations (adding and removing a node) with balanced binary trees without changing their complexity known to be in \( \log n \) [1].
Before show that Algorithm 1 is correct, complete, and has a complexity of $O(n^2|H| \log n)$, let us prove a property of its innermost loop. Let $i \in T$ be a task and let $\text{Ect} \in H$ be an earliest completion time ($H$ sorted in decrease order of it values). Furthermore, let $\Theta_j := \text{LCut}(T, j, \text{Ect}, i)$ with $j \in T = \{1, 2, ..., n\}$ ($T$ sorted in non-decrease order of deadlines) and $\Theta_0 := \emptyset$.

**Proposition 3.** Let $i \in T$ be a task and let $\text{Ect} \in H$ be an earliest completion time. In Algorithm 1, it holds before the $j$th iteration of the inner loop (8-16):

\[
\Theta_{j-1} = \emptyset \quad \text{and} \quad \text{Env}(\Theta_{j-1}, i) = \text{Env}(\Theta, i)
\]

**Proof.** By induction over the array $T = \{1, 2, ..., n\}$ of tasks sorted in non-decrease order of deadline.

Basis $j = 1$: For a given task $i \in T$ and a given earliest completion $\text{Ect} \in H$, the innermost loop was not iterated at all. From this it follows that $\Theta = \emptyset$ and then no energy envelop can be computed. Because $\Theta_{j-1} = \emptyset = \emptyset$ the basis holds.

Inductive step $j \rightarrow j + 1$: The inductive hypothesis is that for all $j' < j$ before the $j'$th loop iteration the condition (12) and (13) hold. Because tasks in the array $T$ are sorted by non-decrease deadline, it holds that $\Theta_{j-1} = \Theta_j \setminus \{j\}$. In order to show the induction step two cases have to be considered, namely $j \in \Theta_j$ and $j \notin \Theta_j$.

First we consider the case $j \in \Theta_j$. In this case the conditions of line 9 is true for the $j$th loop iteration. According to the induction hypothesis $\Theta = \Theta_{j-1}$ was updated with $\Theta \cup \{j\}$ (line 10). Thus, the conditions (12) and (13) hold.

The second case is $j \notin \Theta_j$. In this case the conditions of line 9 is not true. Because $j$ is not added to the tree it follows that $\Theta_j = \Theta_{j-1}$ and by the induction hypothesis the proposition holds.

**Theorem 2.** Algorithm 1 is correct, complete and runs in $O(n^2|H| \log n)$. After its computation it holds: $LB'[i] = LB[i] = \max \left( min_{D \subseteq T \setminus \{i\}} \gamma(D) \right)$ with $\gamma(O, i) \overset{def}{=} (r_i < ECT_O) \land (e_O + c_i (\min(ect_i, d_O) - r_O)) > C(d_O - r_O))$.

**Proof.** First the correctness and completeness of Algorithm 1 is shown, followed by the proof for its runtime.

The algorithm is correct and complete if the following holds for every task $i \in T$:

\[
LB'[i] = \max \left( r_i, \max_{D \subseteq T \setminus \{i\}} \gamma(D) \right)
\]

with $\gamma(O, i) \overset{def}{=} (r_i < ECT_O) \land (e_O + c_i (\min(ect_i, d_O) - r_O)) > C(d_O - r_O))$.

Let $i \in T$ be any task. We wan to shown that $LB'[i] = LB[i]$.

If $LB[i] = r_i$, then no detection was founded by the not-first rule. According to Theorem 1, the condition of line 11 is not satisfied and no adjustment is
performed by Algorithm 1. At the end, the last loop of Algorithm 1 allow us to have $LB'[i] = r_i = LB[i]$.

Without loss of generality, let $r_i < LB[i]$. Then a task set exists $\Omega \subseteq T \setminus \{i\}$ for which the following holds:

$$LB[i] = ECT_\Omega \land e_\Omega + c_i (\min(ect_i, d_\Omega) - r_\Omega) > C (d_\Omega - r_\Omega).$$  \hspace{1cm} (14)$$

Let $j \in \Omega$ be any tasks with $r_j = r_\Omega$ and let $Ect := ECT_\Omega \in H$ be the earliest completion time of task set $\Omega$. Hence, $\Omega \subseteq \Theta_j$, thus the condition (14) also holds for the set $\Theta_j$ since $Env(\Theta_j, i) \geq (C - c_i)r_\Omega + e_\Omega > C d_j - c_i \min(ect_i, d_j)$.

According to Theorem 1 and Proposition 3, Algorithm 1 detect the not-first condition (line 11) and update the release date of task $r_i$ to $LB'[i] = Ect = ECT_\Omega = LB[i]$. In other words, Algorithm 1 is correct and complete.

Let us now show that the runtime is $O(n^2 |H| \log n)$. The first loop: line 1 to line 3 is $O(n)$. The second loop: line 4 to line 22 comprises an inner loop from line 5 to line 21. In the innermost loop (8-16) the time complexity of line 10 is $O(\log n)$. Consequently, the time complexity of the inner loop is $O(n |H| \log n)$ and is $O(n^2 |H| \log n)$ for the outer loop. The last loop: line 23 to line 25 is $O(n)$. Thus, the total time complexity of the algorithm is $O(n) + O(n^2 |H| \log n) + O(n) = O(n^2 |H| \log n)$.

5 Conclusion

This paper focuses on the not-first/not-last filtering rule for cumulative scheduling. It is a pendant of the edge-finding in constraint-based disjunctive and cumulative scheduling. The not-first/not-last rule which is not subsumed by the other rules is used to detect tasks that cannot run first/last regarding a set of tasks and prune their time bounds. The contribution of this paper is an improvement of the time complexity of the well known complete not-first/not-last algorithm of Schutt et al [2] from $O(n^3 \log n)$ to $O(n^2 |H| \log n)$ where $n$ is the number of tasks and $H$ the set of distinct earliest completion times of tasks with $|H| \leq n$.

The algorithm is based on an adaptation of the left cut of the set of tasks notion introduce by Vilím in [8] and the energy envelop introduce by the same author in [1]. It is a complete version of our sound and incomplete algorithm presents in [4].

The future work focuses on an efficient implementation of the presented algorithm in a constraint programming solver and compared it with the version of not-first/not-last algorithm presented in [4,5] since both algorithm have the same complexity at the fixed point.

References

Almost Square Packing

Helmut Simonis and Barry O’Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.simonis|b.osullivan}@4c.ucc.ie

Abstract. The almost square rectangle packing problem involves packing all rectangles with sizes $1 \times 2$ to $n \times (n + 1)$ (almost squares) into an enclosing rectangle with minimal area. This extends the previously studied square packing problem by adding an additional degree of freedom for each rectangle, deciding in which orientation the item should be packed. We show how to extend the model and search strategy that worked well for square packing to solve the new problem. Some adapted versions of known redundant constraints improve overall search times. Based on a visualization of the search tree, we derive a decomposition method which initially only looks at the subproblem given by one of the cumulative constraints. This decomposition leads to further modest improvements of execution times. We find a solution for problem size 26 for the first time and dramatically improve best known times for finding solutions for smaller problem sizes by up to three orders of magnitude.

1 Introduction

The almost square rectangle packing problem [9, 12, 13] involves packing all rectangles with sizes $1 \times 2$ to $n \times (n + 1)$ into an enclosing rectangle of minimum area. The orientation of the rectangles can be freely chosen, adding an additional degree of freedom compared to the previously studied square packing problem [8, 10, 11, 15, 18]. General rectangle packing is an important problem in a variety of real-world settings. For example, in electronic design automation, the packing of blocks into a circuit layout is essentially a rectangle packing problem [14, 16]. Rectangle packing problems are also motivated by applications in scheduling [10, 11, 15]. Rectangle packing is an important application domain for constraint programming, with significant research into improved constraint propagation methods reported in the literature [1–7, 19].

2 Constraint Programming Model

We initially use the established constraint model [2, 6, 18] for the rectangle packing problem. Each item to be placed is defined by domain variables $X$ and $Y$ for the origin in the $x$ and $y$ dimension respectively, and two domain variables $W$ and $H$ for the width and the height of the rectangle, respectively. In the particular case of packing almost squares, $W$ and $H$ can take only two possible
values \((n \text{ and } n + 1)\), and must be different from each other. The constraints are expressed by a non-overlapping constraint in two dimensions and two (redundant) \textsc{Cumulative} constraints that work on the projection of the packing problem in \(x\) or \(y\) direction. This is illustrated by Figure 1. We use SICStus Prolog 4.0.4 (on a 3GHz Intel Xeon 5450 with 3.25GB of memory), which provides both \textsc{Cumulative} \cite{1} and \textsc{Disjoint2} \cite{3} constraints.

![Fig. 1. The basic constraint programming model.](image)

### 2.1 Generating Candidate Enclosing Rectangles

To find the enclosing rectangle with smallest area, we need a decomposition strategy that generates sub-problems with fixed enclosing rectangle sizes. We enumerate on demand all pairs \(\text{Width, Height}\) in order of increasing area \(\text{Width} \times \text{Height}\) that satisfy

\[
[\text{Width, Height}] :: n..\infty, \text{Width} \geq \text{Height}
\]

\[
\sum_{i=1}^{n} i \times (i + 1) \leq \text{Width} \times \text{Height}
\]

\[
k = \left\lfloor \frac{\text{Height} + 1}{2} \right\rfloor, \text{Width} \geq \sum_{j=k}^{n} j
\]

Equation 1 provides a simple bound on the required area, considering all large items that cannot be stacked on top of each other, which, thus, must fit horizontally. For candidates with the same area, we try them by increasing
Height, i.e. for two subproblems with the same surface we try the “less square-like” solution first. We then solve the rectangle packing problem for each such candidate enclosing rectangle in turn, until we find the first feasible solution. By construction, this is an optimal solution. The number of candidates seems to grow linearly with the amount of slack allowed. In difference to the square packing problem, we find that the optimal solution in many cases does not use any slack at all, and the number of candidates to be tested remains quite small.

### 2.2 Symmetry Removal

The model so far contains a number of symmetries, which we need to remove as we may have to explore the complete search space. We restrict the domain of the largest square of size $n \times (n + 1)$ to be placed in an enclosing rectangle of size $Width \times Height$ to

$$X \:: 1.1 + \left\lfloor \frac{Width - n}{2} \right\rfloor \ , \ Y \:: 1.1 + \left\lfloor \frac{Height - n}{2} \right\rfloor .$$

Other symmetries are discussed below, but are not yet handled as part of the constraint model.

### 3 Search

We studied a number of different search strategies for square packing in [18]. The best method found used an interval labeling approach, first assigning the $X$ variables to intervals, small enough to create obligatory parts, then fixing the $X$ variables to values, and then repeating the process for the $Y$ variables. For the problem sizes studied (up to 27) this provided the best solutions, when fixing the interval size to a fraction between 0.2 and 0.3 of the square width.

In the almost square packing problem, we have to assign $W$ and $H$ variables in addition to the $X$ and $Y$ variables. As the $W$ and $H$ variables of one rectangle are linked by a disequality, and can only take two possible values, it is enough to assign $W$, this will force the assignment of the $H$ variable.

When should we assign the $W$ variables in the search process? We have studied three cases:

- **eager** Assign all $W$ variables before assigning any $X$ variables, leading to multiple problems with oriented rectangles
- **lazy** Assign the $W$ variables once all $X$ variables have been assigned to intervals, but before assigning fixed values for $X$
- **mixed** For each rectangle, ordered by decreasing size, first assign the $W$ variable, then fix the $X$ variable to an interval. Repeat this process for all rectangles, before assigning the $X$ variables to values.

Not surprisingly, the mixed method clearly outperforms the two other methods. In Figures 2, 3, and 4 we show the node distribution of the search for the first
solution, considering problem size 17. The display shows the number of TRY and FAIL nodes at each level of the search tree. A TRY node is generated, when we try to assign an interval or value to a variable and the resulting propagation succeeds. A FAIL node is generated when the assignment leads to a failure and backtracking. The displays are generated with CP-Viz [17], a generic visualization tool for finite domain constraint solvers.

**Fig. 2. Eager Orientation (N=17)**

For the eager method (Figure 2) we see that failures only start once we begin to assign the $X$ variables to intervals. The initial fixing of the rectangle orientation leads to an exponential growth of the search tree (straight line on the left side of the graph due to the log-scale), peaking at over a million nodes at level 23. Note that after the assignment of the $X$ variables only 20 possible solutions remain. Starting with the assignment of the $Y$ variables, the search tree expands again, but only to a few hundred nodes.

For the lazy method (Figure 3), the overall structure of the graph is similar, although the maximal width of the search tree (again, over a million nodes) is reached earlier, at the end of the $X$ variable interval assignment. Forcing the orientation of the rectangles then leads to a rapid elimination of candidate solutions. Although failures occur earlier in the search, the propagation is not powerful enough to eliminate unfeasible candidates without knowing the orientation of the rectangles.

In the mixed method (Figure 4), the propagation can eliminate more partial assignments early in the search, so that the maximal width of the tree is around 20000 nodes. Figure 5 compares the three methods considering only the TRY
Fig. 3. Lazy Orientation (N=17)

Fig. 4. Interleaved Orientation (N=17)
nodes. We see that the search for the last $X$ variable assignments and for finding the $Y$ variables is quite similar, but that the mixed method clearly outperforms the two other methods early in the search.

**Fig. 5. Assignment Strategies Compared (N=17)**

The overall structure of the search tree is remarkably similar for most problem sizes, Figure 6 shows the node distribution for problem size 20. An exception is problem size 21, shown in Figure 7. This shows the node distribution for the $46 \times 77$ candidate rectangle with no slack. Even after the orientation and $X$ interval assignment of all rectangles a large number of partial assignments remains, which are only reduced by the assignment of the $X$ variables to particular values. But there is no solution to this problem, therefore all possible assignments must be enumerated.

The optimal solution for size 26 is shown in Figure 8. This result has not been previously published. Previous work [12] only obtained solutions for problem sizes up to 25.

The results for the basic model are shown in Table 1. It shows the problem size $N$, the total *Surface* of the rectangles to be placed, the number of candidate enclosing rectangles studied ($K$), the *Width* and *Height* of the optimal enclosing rectangle, its *Area* and the amount of lost space (*Lost*). It then counts the number of backtracking steps and the time required to find the first solution, the total number of solutions for the given enclosing rectangle, and the number of backtracking steps and time required to enumerate all such solutions. Note that the total number of all optimal solutions can be higher, as there can be
Fig. 6. Node Distribution (N=20)

Fig. 7. Infeasible Problem Instance (N=21)
candidate rectangles with the same optimal area which are not explored by our algorithm, which stopes at the first feasible candidate.

The total number of solutions varies widely with the problem size. For the problem sizes (6, 9, 10, 12, 21) where the optimal solution is not perfect (i.e. requiring some slack), the number of solutions increases as the 1×2 rectangle can be placed in many of the empty spaces.

In general, if a solution contains two (consecutive) rectangles which share a common edge, then we can exchange these rectangles creating a new solution. In Figure 8 for example, the rectangles 22×21 and 20×21 (on the left) can be exchanged. Indeed, in Figure 8 there are 5 such pairs of rectangles, which can be flipped independently, leading to 32 symmetrical solutions.

3.1 Redundant Constraints

We described in [18] two methods which were quite effective in reducing problem complexity.

- The first was to ignore the 1×1 square when setting up the constraints, while still reserving space for it in the enclosing rectangle. This both reduced the amount of unnecessary work inside the constraints dealing with this small square, and avoided symmetries in the search when the 1×1 square was placed in all possible empty places.
- The second idea was to eliminate certain X and Y values, when squares were placed close to the border of the enclosing space. If a large object is placed near a border, then it might be impossible to fill the gap between the border and the object with the few available, smaller items and the slack allowed (empty space). These gap limits can be precomputed and domain values can be removed a priori, reducing the search space.

For the almost square packing problem, the smallest item is the 1×2 rectangle. If we remove it from the problem, we might find an infeasible solution, if an assignment exists where all empty space is allocated to non-connected 1×1 pieces. Fortunately, that situation rarely occurs; Figure 9 shows a case for size 16. We correct this by enforcing an additional non-overlapping constraint at the
Table 1. Basic Model Results

<table>
<thead>
<tr>
<th>N</th>
<th>Surface</th>
<th>K</th>
<th>Width</th>
<th>Height</th>
<th>Area</th>
<th>Loss</th>
<th>First Solution</th>
<th>All Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Back</td>
<td>Time</td>
</tr>
<tr>
<td>4</td>
<td>40 1</td>
<td>4</td>
<td>10</td>
<td>40 0.00</td>
<td>2</td>
<td>00:00</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>70 1</td>
<td>5</td>
<td>14</td>
<td>70 0.00</td>
<td>4</td>
<td>00:00</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>112 3</td>
<td>6</td>
<td>19</td>
<td>114 1.79</td>
<td>16</td>
<td>00:00</td>
<td>216</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>168 3</td>
<td>12</td>
<td>14</td>
<td>168 0.00</td>
<td>19</td>
<td>00:00</td>
<td>65</td>
<td>76</td>
</tr>
<tr>
<td>8</td>
<td>240 4</td>
<td>15</td>
<td>16</td>
<td>240 0.00</td>
<td>6</td>
<td>00:00</td>
<td>12</td>
<td>83</td>
</tr>
<tr>
<td>9</td>
<td>330 6</td>
<td>14</td>
<td>24</td>
<td>336 1.82</td>
<td>54</td>
<td>00:00</td>
<td>9170</td>
<td>3137</td>
</tr>
<tr>
<td>10</td>
<td>440 6</td>
<td>17</td>
<td>26</td>
<td>442 0.45</td>
<td>323</td>
<td>00:00</td>
<td>1854</td>
<td>1379</td>
</tr>
<tr>
<td>11</td>
<td>572 3</td>
<td>22</td>
<td>26</td>
<td>572 0.00</td>
<td>99</td>
<td>00:00</td>
<td>4</td>
<td>268</td>
</tr>
<tr>
<td>12</td>
<td>728 8</td>
<td>21</td>
<td>35</td>
<td>735 0.96</td>
<td>546</td>
<td>00:00</td>
<td>25180</td>
<td>13795</td>
</tr>
<tr>
<td>13</td>
<td>910 3</td>
<td>26</td>
<td>35</td>
<td>910 0.00</td>
<td>1900</td>
<td>00:00</td>
<td>42</td>
<td>6197</td>
</tr>
<tr>
<td>14</td>
<td>1120 4</td>
<td>28</td>
<td>40</td>
<td>1120 0.00</td>
<td>2937</td>
<td>00:00</td>
<td>4</td>
<td>9604</td>
</tr>
<tr>
<td>15</td>
<td>1360 4</td>
<td>34</td>
<td>40</td>
<td>1360 0.00</td>
<td>14440</td>
<td>00:00</td>
<td>4</td>
<td>50592</td>
</tr>
<tr>
<td>16</td>
<td>1632 4</td>
<td>32</td>
<td>51</td>
<td>1632 0.00</td>
<td>15967</td>
<td>00:01</td>
<td>544</td>
<td>48711</td>
</tr>
<tr>
<td>17</td>
<td>1938 3</td>
<td>34</td>
<td>57</td>
<td>1938 0.00</td>
<td>210878</td>
<td>00:14</td>
<td>16</td>
<td>398759</td>
</tr>
<tr>
<td>18</td>
<td>2280 4</td>
<td>30</td>
<td>76</td>
<td>2280 0.00</td>
<td>9734</td>
<td>00:00</td>
<td>110288</td>
<td>152032</td>
</tr>
<tr>
<td>19</td>
<td>2660 4</td>
<td>35</td>
<td>76</td>
<td>2660 0.00</td>
<td>102235</td>
<td>00:08</td>
<td>526</td>
<td>3240741</td>
</tr>
<tr>
<td>20</td>
<td>3080 4</td>
<td>35</td>
<td>88</td>
<td>3080 0.00</td>
<td>351659</td>
<td>00:34</td>
<td>1988</td>
<td>3612859</td>
</tr>
<tr>
<td>21</td>
<td>3542 5</td>
<td>39</td>
<td>91</td>
<td>3549 0.20</td>
<td>14036353</td>
<td>21:38</td>
<td>3250117</td>
<td>720146935</td>
</tr>
<tr>
<td>22</td>
<td>4048 3</td>
<td>44</td>
<td>92</td>
<td>4048 0.00</td>
<td>58206362</td>
<td>01:37:30</td>
<td>688</td>
<td>122563947</td>
</tr>
<tr>
<td>23</td>
<td>4600 3</td>
<td>40</td>
<td>115</td>
<td>4600 0.00</td>
<td>14490682</td>
<td>30:12</td>
<td>6784</td>
<td>136039535</td>
</tr>
<tr>
<td>24</td>
<td>5200 3</td>
<td>40</td>
<td>130</td>
<td>5200 0.00</td>
<td>27475258</td>
<td>55:05</td>
<td>96</td>
<td>99731414</td>
</tr>
<tr>
<td>25</td>
<td>5850 5</td>
<td>45</td>
<td>130</td>
<td>5850 0.00</td>
<td>35282646</td>
<td>01:23:12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>6552 5</td>
<td>42</td>
<td>156</td>
<td>6552 0.00</td>
<td>92228265</td>
<td>03:28:26</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
end of the search, where we add the $1 \times 2$ piece back to the problem. If there is no room to place that item, the constraint will fail and we backtrack to find another candidate for the relaxed problem, until a valid solution is generated.

**Fig. 9.** Pseudo Solution $N=16$ Width=32 Height=51 with $1 \times 2$ item removed; This can not be extended to a complete solution

<table>
<thead>
<tr>
<th>10x9</th>
<th>8x9</th>
<th>14x13</th>
</tr>
</thead>
<tbody>
<tr>
<td>12x11</td>
<td>6x5</td>
<td>13x12</td>
</tr>
<tr>
<td>15x14</td>
<td>7x8</td>
<td>10x11</td>
</tr>
<tr>
<td>16x17</td>
<td>4x3</td>
<td>16x15</td>
</tr>
</tbody>
</table>

The precomputation of infeasible gap values can also be done for the almost square case, although the domain restrictions are somewhat weaker.

The effect of the redundant constraints are shown in Table 2. Ignoring the $1 \times 2$ rectangle leads to a small, but consistent improvement (Not One Column) compared to the Basic Model. Removing values close to the border of the placement area (Gap Column) has a more significant effect, while combining both leads to the best results.

### 3.2 Impact of Interval Size

In [18], we also studied the impact of the chosen interval size on the performance of the algorithm. We repeated these tests for the almost square packing problem, which lead to a similar conclusion. Setting the interval to 0.3 times the size of the item leads to the best performance, both in number of search nodes and execution time. As Figure 10 shows, the effect is rather restricted, with an obvious effect only visible for problem size 21, which is the only large instance which requires some slack.
Table 2. Redundant Constraint Model Results

<table>
<thead>
<tr>
<th>N</th>
<th>Basic Model</th>
<th>Not One</th>
<th>Gap</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Back Time</td>
<td>Back Time</td>
<td>Back Time</td>
<td>Back Time</td>
</tr>
<tr>
<td>4</td>
<td>2 00:00</td>
<td>2 00:00</td>
<td>2 00:00</td>
<td>2 00:00</td>
</tr>
<tr>
<td>5</td>
<td>4 00:00</td>
<td>3 00:00</td>
<td>2 00:00</td>
<td>1 00:00</td>
</tr>
<tr>
<td>6</td>
<td>16 00:00</td>
<td>16 00:00</td>
<td>6 00:00</td>
<td>6 00:00</td>
</tr>
<tr>
<td>7</td>
<td>19 00:00</td>
<td>18 00:00</td>
<td>10 00:00</td>
<td>9 00:00</td>
</tr>
<tr>
<td>8</td>
<td>6 00:00</td>
<td>5 00:00</td>
<td>17 00:00</td>
<td>10 00:00</td>
</tr>
<tr>
<td>9</td>
<td>54 00:00</td>
<td>54 00:00</td>
<td>27 00:00</td>
<td>27 00:00</td>
</tr>
<tr>
<td>10</td>
<td>323 00:00</td>
<td>323 00:00</td>
<td>159 00:00</td>
<td>159 00:00</td>
</tr>
<tr>
<td>11</td>
<td>99 00:00</td>
<td>99 00:00</td>
<td>54 00:00</td>
<td>54 00:00</td>
</tr>
<tr>
<td>12</td>
<td>546 00:00</td>
<td>546 00:00</td>
<td>274 00:00</td>
<td>274 00:00</td>
</tr>
<tr>
<td>13</td>
<td>1900 00:00</td>
<td>1900 00:00</td>
<td>1040 00:00</td>
<td>1040 00:00</td>
</tr>
<tr>
<td>14</td>
<td>2937 00:00</td>
<td>2936 00:00</td>
<td>1505 00:00</td>
<td>1501 00:00</td>
</tr>
<tr>
<td>15</td>
<td>14440 00:00</td>
<td>14425 00:00</td>
<td>7632 00:00</td>
<td>7617 00:00</td>
</tr>
<tr>
<td>16</td>
<td>15967 00:01</td>
<td>9338 00:00</td>
<td>7264 00:00</td>
<td>3989 00:00</td>
</tr>
<tr>
<td>17</td>
<td>210878 00:14</td>
<td>210850 00:13</td>
<td>107639 00:07</td>
<td>107611 00:07</td>
</tr>
<tr>
<td>18</td>
<td>9734 00:00</td>
<td>9734 00:00</td>
<td>5550 00:00</td>
<td>5550 00:00</td>
</tr>
<tr>
<td>19</td>
<td>102235 00:08</td>
<td>102235 00:08</td>
<td>13694 00:01</td>
<td>13690 00:01</td>
</tr>
<tr>
<td>20</td>
<td>351659 00:34</td>
<td>355964 00:33</td>
<td>157312 00:14</td>
<td>161410 00:14</td>
</tr>
<tr>
<td>21</td>
<td>14036353 21:38</td>
<td>10859861 16:01</td>
<td>9499957 14:14</td>
<td>6524396 09:13</td>
</tr>
<tr>
<td>22</td>
<td>78206362 01:37:30</td>
<td>58214183 01:33:03</td>
<td>17312971 24:37</td>
<td>17319946 23:54</td>
</tr>
<tr>
<td>23</td>
<td>14490682 00:12</td>
<td>14490682 29:16</td>
<td>6400629 11:01</td>
<td>6400629 10:33</td>
</tr>
<tr>
<td>24</td>
<td>27475258 55:05</td>
<td>27475258 53:11</td>
<td>9801577 16:39</td>
<td>9801577 16:10</td>
</tr>
</tbody>
</table>

Fig. 10. Impact of Interval Size
4 Decomposition

We saw in Figure 4 that only rather few complete assignments of the $X$ variables have to be tested to find an optimal solution for the problem. This suggests a further decomposition where we solve the first part of the problem, the orientation of the rectangles and the assignment of the $X$ variables, without considering the $Y$ variables at all. For this we only need the cumulative constraint for the $X$ variables, the second cumulative and the non-overlapping constraints are stated only once the first subproblem has been solved, before we start the assignment of the $Y$ variables. This will avoid waking these constraints repeatedly as the $X$ variables are assigned. Given the number of nodes in the search tree, this can add to significant savings. At the same time, we may lose important propagation due to these constraints, and therefore increase the size of the search tree of the subproblem. Experiments shows that this is not the case. Table 3 compares backtracking steps and execution times for the basic model without and with the redundant constraints and the decomposed model, also without and with the redundant constraints. The number of backtracks is the same for all problem instances except 10 and 12. This is a clear indication that the non-overlapping constraint and the second cumulative are not contributing anything to the search in the initial phase. The difference in execution times are solely due to avoiding unnecessary calls to these constraints in the first phase of the search. The savings are limited, but still worthwhile. In the last two columns (Decomposed Reified) we show results for a model where we replace the disjoint constraint of SICStus with reified sets of inequalities for each pair of rectangles. This is a much weaker form of the non-overlapping constraint, but the results for the decomposed model are quite similar. Clearly, the non-overlapping constraint affects the performance only in a minor way.

Do we need the non-overlapping constraint at all? In [1] perfect placement problems were solved by creating all solutions for the cumulative projections in $x$ and $y$ direction, and then combining them with a checker for the non-overlapping constraint. This will not be competitive for the almost square packing problem. We have seen above (Table 1) that some problem instances have millions of solutions. There will be a similar number of solutions for solving the $x$ cumulative alone. Testing each of those solutions against all solutions of the $y$ cumulative will be too expensive.

We can try to push the non-overlapping constraint to the overall end of the search, and use it only as a checker. This will mean that in the second part of the search we only use a single cumulative constraint in the $y$ direction. Experiments indicate that this is not competitive.

5 Comparison

In Table 4, we compare our results to those reported in [12]. Note that we only count backtracking steps, not the total number of nodes as in [12]. We can see that even our basic model dramatically outperforms Korf et al for large
In this paper we have extended our previous results [18] for packing squares into account. But the differences are not uniform with the problem size, the differences for instances 21 and 22 are much smaller.

Table 4. Comparison with [12]

<table>
<thead>
<tr>
<th>Size</th>
<th>Korf, Moffitt and Pollack Nodes</th>
<th>Pure Back</th>
<th>Times</th>
<th>Redundant Back</th>
<th>Decomposition Back</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>107161</td>
<td>107611</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>15</td>
<td>14407</td>
<td>14407</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>16</td>
<td>19007</td>
<td>19007</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>17</td>
<td>23707</td>
<td>23707</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>18</td>
<td>28407</td>
<td>28407</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>19</td>
<td>33107</td>
<td>33107</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>20</td>
<td>37807</td>
<td>37807</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>21</td>
<td>42507</td>
<td>42507</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>22</td>
<td>47207</td>
<td>47207</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>23</td>
<td>51907</td>
<td>51907</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>24</td>
<td>56607</td>
<td>56607</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
<tr>
<td>25</td>
<td>61307</td>
<td>61307</td>
<td>06:11</td>
<td>161410</td>
<td>161410</td>
<td>06:11</td>
</tr>
</tbody>
</table>

6 Conclusion

In this paper we have extended our previous results [18] for packing squares into the smallest enclosing rectangle to packing “almost squares”, rectangles of
sizes \( n \times (n + 1) \). For problem size \( N \), this adds \( 2^N \) additional choices. Using the existing constraint model and carefully interleaving the assignment of \( X \) intervals and the orientation of the rectangles, we can solve the problem to optimality up to size 26, extending the previously best results [12] by one instance and obtaining a large reduction in execution time. For this problem type, a further decomposition of the problem into two phases is suggested by a visualization of the search tree. We first solve the problem in \( x \) direction with a single cumulative constraint, interleaving the orientation of the rectangles with the assignment of intervals to the \( X \) variables, before fixing the \( X \) values. Only then do we state the second cumulative constraint and the non-overlapping constraint. Together with some redundant constraints, this leads to a further reduction of the search space required.

### Acknowledgment

This work was supported by Science Foundation Ireland (Grant Number 05/IN/1886). The authors wish to thank Mats Carlsson, who provided the SICStus Prolog 4.0.4 used for the experiments.

### References


Creating Tests for a Family of Cost Aware Resource Constraints

Tarik Hadzic and Helmut Simonis

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{t.hadzic,h.simonis}@4c.ucc.ie

Abstract. In this paper we describe a benchmark generator for a family of cost aware resource constraints. These constraints have been introduced in the context of electricity cost minimization, but can also be applied for other use cases like manpower scheduling. None of the existing resource constraint benchmarks include a cost element, we therefore created our own test generator to evaluate the quality of different constraint implementations. The generator is written in Java, produces an XML instance format, and is available on the website of the authors.

1 Motivation

Time variable electricity tariffs have been introduced in many countries to properly price the cost of electricity generation over changing demand at different times of the day. In Ireland, the All Island Electricity Market (http://allislandmarket.com) generates a whole-sale market price in half hour time slots, with prices sometimes varying by a factor of ten between peak and non-peak periods. The high cost at peak utilization is linked to the use of inefficient, expensive stand-by generation plant with an increased carbon footprint. Shifting large-scale customer demand away from peak utilization increases stability of the national grid, and can possibly delay or avoid heavy investment in new generator capacity.

Figure 1 shows demand and price data for two one week periods in January and June 2010. We can see that demand varies significantly during the day, but also from day to day in the week and between seasons. Prices follow the demand, with especially high prices at periods close to the national dispatch limit.

In Ireland, wind power plays an increasing role as a renewable energy source. Unfortunately, its integration in the national grid is problematic, due to the relative isolation of the Irish electricity grid, and wind energy’s time variable, and only partially known, supply level. Matching electricity use to changes in wind energy supply is quite difficult to imagine for domestic usage, but can be possible for industrial users, given the right pricing incentive. Widespread

* This work was supported by Science Foundation Ireland (Grant Numbers 05/IN/1886 and 05/IN.1/I886 TIDA 09).
adaption of electric cars is seen as another way of fully utilizing the wind energy potential, but this is still years in the future.

The current adaption of time variable tariffs for large scale electricity users is still quite limited. This is partially due to the cost of the required smart metering tools and the necessary detailed understanding of electrical usage over short time periods, but also due to a lack of decision support tools which allow the exploitation of time variable tariffs to reduce costs. We are trying to fill this gap using constraint programming.

Our current work is therefore focused on the problem of solving industrial scheduling problems considering time variable energy cost, and using available renewable energy in the most efficient way. This requires agile scheduling tools, which can react quickly to changes in prices, while still considering other optimization criteria like product quality, factory throughput and just-in-time production.

Constraint Programming has been a very successful tool in solving large scale industrial scheduling problems [3,13,14], creating flexible scheduling tools for industries in many domains. Its main advantage over competing technologies is the ease of adapting a system to a changing environment, and the potential to incorporate user-defined strategies and heuristics besides powerful mathematical reasoning techniques.

So far our work [11,12,10] has considered the core algorithms to include resource costs into scheduling constraints. We have initially considered an extension of the cumulative constraint, and then extended that work for other resource types, disjunctive machines, multiple parallel disjunctive machines and machine-choice resource types. We have proposed different algorithms to compute lower bounds on the resource cost of a schedule, and compared their theoretical and practical power. The best results use an adaptation of the Linear Programming model for the cumulative constraint from [5]. Experimental results indicate that very good estimates (better than 98% of the optimal value) can be achieved. We
have used that lower bound on the cost to perform reduced cost based filtering for the different resource types described. Initial results showed that a significant number of domain values can be removed by this filtering. To validate and improve these findings we require test cases on which we can systematically test the performance of our methods. As none of the existing scheduling benchmarks consider resource cost, we decided to create our own instance generator and make it publically available. This paper defines the problem domain, introduces the instance generator and describes its capabilities, and compares it to the existing benchmark tests for constraint-based scheduling.

Our current methods are focused on industrial scheduling problems for large scale electricity consumers in manufacturing and other industries. But the underlying technology is also applicable in other areas of significant electricity usage like HVAC (heating/ventilation/air-conditioning) for buildings, or cooling systems in the food industry or for data centres. These are domains where there also is significant overlap with other projects at our institute (4C), and existing collaboration within UCC and with industrial partners. Note that these problem types are quite different from the project scheduling problems that are considered in many of the existing scheduling benchmarks.

2 Constraints Family

We consider four variants of the energy cost aware scheduling constraints:

**CumulativeCost** The total energy consumption is limited by a hard limit, and each task consumes a fixed amount of energy during its execution. The cost is the total cost of energy consumed over time, priced at different levels.

**DisjunctiveCost** This constraint models a disjunctive machine, where tasks consume different amounts of energy. The order of the tasks on the machine will therefore affect total energy cost.

**ParallelMachineCost** This considers multiple disjunctive machines, which all contribute to an overall energy limit. Tasks are fixed on their machine, only the order of the tasks can be affected.

**MachineChoiceCost** We consider multiple disjunctive machines with an overall energy use limit. Tasks can move between machines, with possibly different duration and resource use on every machine.

We now discuss each of the possible constraints in more detail.

2.1 CumulativeCost

We start with the CumulativeCost constraint, which looks at the overall energy consumption of a set of tasks over time. It is an extension of the classical cumulative constraint \[\text{Cumulative}([s_1, s_2, \ldots, s_n], [d_1, d_2, \ldots, d_n], [r_1, r_2, \ldots, r_n], l, p),\]
describing $n$ tasks with start $s_i$, fixed duration $d_i$ and resource use $r_i$, with an overall resource limit $l$ and a scheduling period end $p$. Our new constraint is denoted as

$$\text{CumulativeCost}(\text{Areas}, \text{Tasks}, l, p, \text{cost}).$$

The $\text{Areas}$ argument is a collection of $q$ areas $\{A_1, \ldots, A_q\}$, which do not overlap and partition the entire available resource area $[0, p] \times [0, l]$. Each area $A_j$ has a fixed position $x_j, y_j$, fixed width $w_j$ and height $h_j$, and fixed per-unit cost $c_j$. Consider the example in Fig. 2 (left). It has 5 areas, each of width 1 and height 3. Our definition allows that an area $A_j$ could start above the bottom level ($y_j > 0$). This reflects the fact that the unit-cost does not only depend on the time of resource consumption but also on its volume. In our electricity example, in some environments, a limited amount of renewable energy may be available at low marginal cost, generated by wind-power or reclaimed process heat. Depending on the tariff, the electricity price may also be linked to the current consumption, enforcing higher values if an agreed limit is exceeded.

We choose the numbering of the areas so that they are ordered by non-decreasing cost ($i \leq j \Rightarrow c_i \leq c_j$); in our example costs are 0, 1, 2, 3, 4. There could be more than one area defined over the same time slot $t$ (possibly spanning over other time-slots as well). If that is the case, we require that the area "above" has a higher cost. The electricity consumed over a certain volume threshold might cost more.

The $\text{Tasks}$ argument is a collection of $n$ tasks. Each task $T_i$ is described by its start $s_i$ (between its earliest start $s_i$ and latest start $\bar{s}_i$), and fixed duration $d_i$ and resource use $r_i$. In our example, we have three tasks with durations 1, 2, 1 and resource use 2, 2, 3. The initial start times are $s_1 \in [2, 5]$, $s_2 \in [1, 5]$, $s_3 \in [0, 5]$. For a given task allocation, variable $a_j$ states how many resource units of area $A_j$ are used. For the optimal solution in our example we have $a_1 = 2, a_2 = 3, a_3 = 2, a_4 = 0, a_5 = 2$. Finally, we can define our constraint:

**Definition 1.** Constraint $\text{CumulativeCost}$ expresses the following relationships:

\[
\begin{align*}
\forall 0 \leq t < p & : \quad pr_t := \sum_{\{i|s_i \leq t < s_i + d_i\}} r_i \leq l & (1) \\
\forall 1 \leq i \leq n & : \quad 0 \leq s_i \leq s_i + d_i \leq \bar{s}_i + d_i \leq p & (2) \\
\text{ov}(t, pr_t, A_j) & := \begin{cases} 
\max(0, \min(y_j + h_j, pr_t) - y_j) & x_j \leq t < x_j + w_j \\
0 & \text{otherwise}
\end{cases} & (3) \\
\forall 1 \leq j \leq q & : \quad a_j = \sum_{0 \leq t < p} \text{ov}(t, pr_t, A_j) & (4) \\
\text{cost} & = \sum_{j=1}^{q} a_j c_j & (5)
\end{align*}
\]

For each time point $t$ we first define the resource profile $pr_t$ (the amount of resource consumed at time $t$). That profile must be below the overall resource limit
l, as in the standard cumulative. The term $\text{ov}(t, pr_t, A_j)$ denotes the intersection of the profile at time $t$ with area $A_j$, and the sum of all such intersections is the total resource usage $a_j$. The cost is computed by weighting each intersection $a_j$ with the per-unit cost $c_j$ of the area.

Note that our constraint is a strict generalization of the standard cumulative. Since enforcing generalized arc consistency (GAC) for Cumulative is NP-hard [5], the complexity of enforcing GAC over CumulativeCost is NP-hard as well.

**Fig. 2.** An example with 3 tasks and 5 areas. Areas are drawn as rounded rectangles at the top, the tasks to be placed below, each with a line indicating its earliest start and latest end. The optimal placement of tasks has cost 15 (left). The LP model produces a lower bound 12 (right).

In [12] we considered different algorithms to compute a lower bound on the resource cost of the CumulativeCost constraint. The best model was based on [5], which describes an LP relaxation of the classical cumulative constraint. We extend this model to handle the cost component directly (equations (6)-(15)).

We introduce binary variables $y_{it}$ which state whether task $i$ starts at time $t$. For each task, exactly one of these variables will be one (constraint (12)). Equations (11) connect the $s_t$ and $y_{it}$ variables. Continuous variables $pr_t$ describe the resource profile at each time point $t$, all values must be below the resource limit $l$. The profile is used in two ways: In (13), the profile is built by cumulating all active tasks at each time-point. In (14), the profile overlaps all areas active at a time-point, where the contribution of area $j$ at time-point $t$ is called $z_{jt}$ (a
continuous variable ranging between zero and \( h_j \). Adding all contributions of an area leads to the resource use \( a_j \) for area \( j \). This model combines the start-time based model of the cumulative with a standard LP formulation of the convex, piece-wise linear cost of the resource profile at each time point. Note that this model relies on the objective function to fill up cheaper areas to capacity before using more expensive ones. Enforcing the integrality in (8) leads to a mixed integer programming model \( DMIP \), relaxing the integrality constraint leads to the LP model \( DLP \). The MIP model solves the cumulative-cost constraint to optimality, thus providing an exact bound for the constraint. We can ignore the actual solution if we want to use the constraint in a larger constraint problem.

\[
\text{lb} = \min \sum_{j=1}^{q} a_j c_j
\]

∀ \( 0 \leq t < p \) : \( pr_t \in [0, l] \)  \hspace{1cm} (6)

∀ \( 1 \leq i \leq n, 0 \leq t < p \) : \( y_{it} \in \{0, 1\} \)  \hspace{1cm} (7)

∀ \( 1 \leq j \leq q, \forall x_j \leq t < x_j + w_j \) : \( z_{jt} \in [0, h_j] \)  \hspace{1cm} (8)

∀ \( 1 \leq j \leq q \) : \( 0 \leq a_j \leq a_j \leq \overline{a_j} \leq w_j h_j \)  \hspace{1cm} (9)

∀ \( 1 \leq i \leq n \) : \( s_i = \sum_{t=0}^{p-1} ty_{it} \)  \hspace{1cm} (10)

∀ \( 1 \leq i \leq n \) : \( \sum_{t=0}^{p-1} y_{it} = 1 \)  \hspace{1cm} (11)

∀ \( 0 \leq t < p \) : \( pr_t = \sum_{1 \leq i \leq n} \sum_{t' \leq t < t' + d_i} y_{it'} r_i \)  \hspace{1cm} (12)

∀ \( 0 \leq t < p \) : \( pr_t = \sum_{j=1}^{q} z_{jt} \)  \hspace{1cm} (13)

∀ \( 1 \leq j \leq q \) : \( a_j = \sum_{t=x_j}^{x_j+w_j-1} z_{jt} \)  \hspace{1cm} (14)

2.2 DisjunctiveCost

The DisjunctiveCost constraint is the analog generalization of the disjunctive constraint, which allows one task run be run on a machine at any time. Given the areas and tasks as defined before, we can describe the constraint by adding

\[
\forall i, j | i \neq j : \quad s_i + d_i \leq s_j \vee s_j + d_j \leq s_i
\]

(15) to constraints (1) - (5).

In the LP/MIP model, we extend constraints (6) - (15) with the condition

\[
\forall 0 \leq t < p : \quad \sum_{1 \leq i \leq n} \sum_{t' \leq t < t' + d_i} y_{it'} \leq 1
\]

(16)
which states that at each time point only one task can be active. Note that the overall resource limit \( l \) becomes meaningless, as one only task consumes resources at any time point. The overall resource limit can be checked a priori by comparing it against the resource requirements \( r_i \) of the tasks. There is still a minimization problem arranging the tasks in a sequence such that total energy cost is minimal.

This constraint is useful to model a factory which contains a single, disjunctive resource as the main energy consumer. Modelling that machine as a CumulativeCost resource (together with a finite domain disjunctive constraint, say) will lead to a massive underestimation of the energy cost required.

### 2.3 ParallelMachineCost

In the previous section we considered a situation where a single disjunctive machine dominates the energy consumption. We now consider a case where \( b \) disjunctive machines run in parallel, and each task is fixed to one of those disjunctive machines. Let \( 1 \leq m_i \leq b \) be the machine on which task \( i \) is assigned. The ParallelMachineCost constraint then consists of constraints (1) - (5) plus the constraints

\[
\forall 1 \leq k \leq b, \forall i, j | i \neq j : \quad m_i \neq m_j \lor s_i + d_i \leq s_j \lor s_j + d_j \leq s_i \quad (18)
\]

We can express this condition in the LP/MIP model by adding constraints of the form

\[
\forall 1 \leq k \leq d, \forall 0 \leq t < p : \sum_{i | m_i = k} \sum_{t' \leq t < t' + d_i} y_{it'} \leq 1 \quad (19)
\]

to constraints (6) - (13).

This ParallelMachineCost describes how all tasks compete for energy, and tasks on a single machine must be scheduled with overlap. This model produces stronger bounds than a CumulativeCost constraint on the tasks alone, as it considers the disjunctive behaviour as well.

### 2.4 MachineChoiceCost

The final variant of the constraint family we consider looks at a situation where we have \( b \) machines, and tasks can be assigned on alternative machines, possibly with a different duration and resource consumption on each machine. Let \( d_{ik} \) be the duration of task \( i \) on machine \( k \), and \( r_{ik} \) the energy demand for task \( i \) on machine \( k \). The variable \( m_i \in M_i \) denotes the machine on which task \( i \) has been assigned. It must take a value in the set of possible machines \( M_i \) for task \( i \).
Definition 2. Constraint MachineChoiceCost expresses the following relationships:

\[ \forall 0 \leq t < p : \quad pr_t := \sum_{\{i|s_i \leq s_t < s_t + d_{im_i}\}} r_{im_i} \leq l \]  
\( (20) \)

\[ \forall 1 \leq i \leq n : \quad 0 \leq s_i \leq s_t < s_t + d_{im_i} \leq s_t + d_{im_i} \leq p \]  
\( (21) \)

\[ ov(t, pr_t, A_j) := \begin{cases} \max(0, \min(y_j + h_j, pr_t) - y_j) & x_j \leq t < x_j + w_j \\ 0 & \text{otherwise} \end{cases} \]  
\( (22) \)

\[ \forall 1 \leq j \leq q : \quad a_j = \sum_{0 \leq t < p} ov(t, pr_t, A_j) \]  
\( (23) \)

\[ \text{cost} = \sum_{j=1}^{q} a_j c_j \]  
\( (24) \)

We use 0/1 variables \( y_{itk} \) to state that task \( i \) starts at time \( t \) on machine \( k \). If a machine can not run on some machine \( k \), then all entries \( y_{itk} \) for that machine will be zero.

\[ 1b = \min \sum_{j=1}^{q} a_j c_j \]  
\( (25) \)

\[ \forall 0 \leq t < p : \quad pr_t \in [0, l] \]  
\( (26) \)

\[ \forall 1 \leq k \leq b, \forall 1 \leq i \leq n, 0 \leq t < p : \quad y_{itk} \in \{0, 1\} \]  
\( (27) \)

\[ \forall 1 \leq j \leq q, \forall x_j \leq t < x_j + w_j : \quad z_{jt} \in [0, h_j] \]  
\( (28) \)

\[ \forall 1 \leq j \leq q : \quad 0 \leq a_j \leq a_j \leq \overline{a_j} \leq w_j h_j \]  
\( (29) \)

\[ \forall 1 \leq i \leq n : \quad s_i = \sum_{1 \leq k \leq d} \sum_{t=0}^{p-1} y_{itk} \]  
\( (30) \)

\[ \forall 1 \leq i \leq n : \quad \sum_{1 \leq k \leq b} \sum_{t=0}^{p-1} y_{itk} = 1 \]  
\( (31) \)

\[ \forall 0 \leq t < p : \quad pr_t = \sum_{1 \leq k \leq b} \sum_{t' \leq t < t' + d_{ik}} y_{it'k} r_{ik} \]  
\( (32) \)

\[ \forall 0 \leq t < p : \quad pr_t = \sum_{j=1}^{q} z_{jt} \]  
\( (33) \)

\[ \forall 1 \leq j \leq q : \quad a_j = \sum_{t=x_j}^{x_j + w_j - 1} z_{jt} \]  
\( (34) \)

\[ \forall 1 \leq k \leq b, \forall 0 \leq t < p : \quad \sum_{1 \leq i \leq n} \sum_{t' \leq t < t' + d_{ik}} y_{it'k} \leq 1 \]  
\( (35) \)
3 An Instance Generator

This section explains how to use the instance generator for the cost-scheduling instances used in the ongoing research on energy-efficient scheduling [11,12,10]. The instance generator is available in the form of a Java jar file CostInstance.jar which can be downloaded from [http://4c.ucc.ie/~thadzic/CostInstance.jar](http://4c.ucc.ie/~thadzic/CostInstance.jar).

3.1 Parameters

Once the file CostInstance.jar is downloaded, it can be used to generate a scheduling instance by executing:

```
java -cp CostInstance.jar Instance <parameters>
```

where `<parameters>` are:

- `-instanceType` | 0 - CumulativeCost, 1 - DisjunctiveCost, 2 - ParallelMachineCost
- `-n` number of required tasks
- `-m` number of required areas
- `-d_max` maximum duration of a task
- `-r_max` maximum resource consumption of a task
- `-s_diff_portion` portion of the horizon restricting the start time domain
- `-util` utilization of the total available area
- `-cost_distr` cost distribution, 0 - explicitly given, 1 - random
- `-w` width of each area
- `-machineNo` number of machines for parallel machine instances
- `-randomSeed` initial random seed
- `-maxCost` maximal random cost of an area
- `-costFileName` a valid file name containing a vector of costs (or input "no-file")

We will now discuss each parameter in details.

**instanceType** An integer from the set \{0, 1, 2\} designating the type of the instance. Value 0 represents an instance of CumulativeCost, 1 - DisjunctiveCost and 2 - ParallelMachineCost. At the moment, generation of MachineChoiceCost instances is not supported.

**n** An integer denoting the number of required tasks. The resulting instance is guaranteed to have the `n` tasks.

**m** An integer denoting the number of required areas. It is *not guaranteed* that the resulting instance would have `m` areas if that would conflict with the required utilization level (see parameter `util`). The number of areas is tightly related to the horizon `p` through parameter `w` (the width of each area): `p = m \cdot w`. 
**d_max** An integer denoting the maximal duration of a task. For each task \(i\), duration \(d_i\) is randomly selected from \([1, d_{\text{max}}]\).

**r_max** An integer denoting the maximal resource use of a task. For each task \(i\), resource use \(r_i\) is randomly selected from \([1, r_{\text{max}}]\).

**s_diff** A fractional value denoting the portion of the horizon that restricts the maximal length of the start-time interval \([s_i, s_{\text{max}}]\) for each task \(i\). Let \(s_{\text{max}}\) denote \(p \cdot s_{\text{diff}}\). Then it holds: \(0 \leq s_i \leq s_{\text{max}} \leq s_i + s_{\text{max}}\) if \(s_{\text{diff}}\) is set to a negative value than each task \(i\) has a maximal feasible start-time interval \([0, p - d_i]\).

**util** An integer value between 0 and 100 designating the percentage of the required utilization i.e. the tightness of the instance. The higher the utilization level the harder it is to find a feasible schedule and the more expensive is the cheapest schedule. In case of CumulativeCost and ParallelMachineCost instances, utilization denotes the required ratio of the total task volume and the total available area

\[
\text{util} = \frac{\sum_{i=1}^{n} d_i \cdot r_i}{l \cdot p}.
\]  

(36)

For DisjunctiveCost, the utilization denotes the required ratio between the total task length and the horizon:

\[
\text{util} = \frac{\sum_{i=1}^{n} d_i}{p}.
\]  

(37)

The required level of utilization is achieved by adjusting the overall resource limit \(l\) in case of CumulativeCost and ParallelMachineCost:

\[
l = \frac{\sum_{i=1}^{n} d_i \cdot r_i}{\text{util} \cdot p}
\]  

(38)

and adjusting the horizon \(p\) (i.e. the number of areas \(m = \frac{p}{w}\)) in case of DisjunctiveCost:

\[
p = \frac{\sum_{i=1}^{n} d_i}{\text{util}}.
\]  

(39)

As a result, even though the required number of areas can be specified as the input, the actual number of areas in the instance depends on the required utilization level. Please note that since resource limit \(l\) and number of areas \(m\) have to be integers, due to rounding in equation (38) and (39) the actual utilization can somewhat defer from the desired utilization level.

**cost_distr** An integer from the set \([0, 1]\) indicating explicitly given cost distribution for value 0, and random cost distribution for value 1. For random cost distribution, we select a cost for each area from interval \([0, \text{maxCost} - 1]\).

For generating instances with explicitly given costs, we use externally specified vector of costs that we cyclically repeat (see parameter costFileName below). For a given number of areas \(m\) and width \(w\) we cyclically repeat the vector of costs until the \(m\) areas are generated. For example, for a vector of costs 10, 20, 30 and \(m = 5\) areas, we generate costs 10, 20, 30, 10, 20. In case
the external file is not provided (file name input "no-file") we use default, hardcoded vector of electricity costs which consists of 48 values, covering 24 hour period for a particular day on the Irish electricity market. For both distributions we subtract from the cost of each area the cost of the cheapest area. This ensures that the costs are normalized, i.e. that the cheapest area has a cost 0.

w An integer denoting the width of each area. It links the horizon \( p \) to the number of areas \( m \) through equation \( p = m \cdot w \).

\textbf{machineNo} An integer denoting the number of machines for \texttt{ParallelMachineCost} instances.

\textbf{randomSeed} An integer denoting the random seed used for generating tasks and areas (if negative then the current system time is used).

\textbf{maxCost} A positive integer indicating a maximal cost of an area in a random cost distribution. The cost of each area is a random number from interval \([0, \text{maxCost} - 1]\). If a non-positive value is provided, \( \text{maxCost} \leq 0 \), then default maximal cost is used, \( \text{maxCost} = 150 \).

\textbf{costFileName} A file name in which the cost vector is specified. The cost information is used if the parameter \texttt{cost_distr} is set to 0. The file should contain a single line with space seperated integers. The integers are then cyclically repeated as costs of \( m \) areas. For example, if the file contains three entries: 10 20 30 and we want to generate 5 areas, the respective costs will be: 10, 20, 30, 10, 20. In case the external file is not provided, the input "no-file" is expected. In that case, a default, hardcoded vector is used, consisting of 48 values of electricity costs, covering 24 hour period for a particular day on the Irish electricity market.

\textbf{Discussion} Please note that the current version generates only the "non-stacking" areas, i.e. over each time point there is only one area defined. Furthermore, each area has the height equal to the maximal resource availability \( l \). Furthermore, for \texttt{DisjunctiveCost} instances, the resource limit \( l \) does not influence the utilization, and is set to be higher then the highest resource consumption among the tasks, \( l = \max \{ r_i + 1 \mid i = 1, \ldots, n \} \). In case of \texttt{CumulativeCost} and \texttt{ParallelMachineCost} instances, if the size of task volume is small in comparison of the length of horizon, in order to achieve the required utilization, the computed overall resource limit \( l \) might be smaller that the resource use of individual tasks, leading to an infeasible instance.

3.2 XML Output Format

The output of the generator is an instance in the XML format. The root of the file is an element \texttt{<instance>} with attributes \texttt{horizon} (denoting the task period \( p \)) and \texttt{resource-limit} (denoting the overall resource limit \( l \)). The \texttt{<instance>} consists of \texttt{<tasks>}, \texttt{<areas>} and \texttt{<machines>}. Element \texttt{<tasks>} consist of multiple \texttt{<task>} elements, each of which defines a task, through attributes \texttt{start_min}
(earliest start time \(s_i\)), \texttt{start max} (latest start time \(\pi_i\)), \texttt{duration} and \texttt{resource}. Element \texttt{<areas>} consist of multiple \texttt{<area>} elements, each of which defines an area through attributes \(x\) and \(y\) (defining the coordinates of the lower-left corner), \texttt{width}, \texttt{height} and \texttt{cost}. Finally, element \texttt{<machines>} consists of multiple \texttt{<machine>} elements, each of which lists the set of tasks that are scheduled to be executed on that machine through attribute \texttt{tasks}.

Furthermore, \texttt{<tasks>}, \texttt{<areas>} and \texttt{<machines>} have auxiliary attribute \texttt{number} that indicates how many \texttt{<task>}, \texttt{<area>} and \texttt{<machine>} elements are to follow. And finally, each \texttt{<task>}, \texttt{<area>} and \texttt{<machine>} element is uniquely identified through an attribute \texttt{id}. In fact, the list of tasks in each machine element is given by listing the corresponding task ids.

The formal specification of the XML output format is given in the XML schema file, whose structure is given in Fig. 3. The file can be downloaded from \url{http://4c.ucc.ie/~thadzic/resourcecost.xsd}.

As an illustration, consider a result of executing:

\begin{verbatim}
java -cp CostInstance.jar Instance 2 5 7 6 8 0.5 50 0 3 2 123 150 no-file
\end{verbatim}

the generator produces the XML file in Figure 4. The result is an instance of a parallel machine scheduling problem, with overall resource limit \(l = 6\) and horizon \(p = 21\). The instance has \(n = 5\) tasks, with \(m = 7\) cost areas, each of width \(w = 3\). The tasks are distributed over \texttt{machineNo = 2} machines. For each task we specify its start time interval, duration and resource consumption. For each area we specify its left-most coordinate \((x, y)\), its width, its height and its cost per resource unit. For each machine we list the indexes of tasks that are scheduled for execution on such machine.

4 Existing Benchmarks for Resource Constraints

To the best of our knowledge, this is the first generator for cost-aware scheduling instances. However, there are several sources of classical scheduling instances, where typically the objective is the \textit{makespan minimization}.

In [15] the authors describe the set of widely cited flow-shop, job-shop and open-shop scheduling instances, for which the generating code is made available at the online collection of OR data sets, the \textit{OR-Library}\(^1\). Patterson [9] introduced a set of 110 multi-resource scheduling instances involving between 7 and 50 activities, which presented the accumulation of readily-available instances in the literature at that time. Alvarez et al. [2] introduced 144 instances containing 27, 51 and 103 activities respectively. Kolisch et al. [8] introduced a number of general resource-constrained project scheduling instances using the \textit{ProGen} instance generator. They subsequently extended the instance set into a PSPLIB library that is available online [7]. Subsequently, Baptiste and Le Pape [4] introduced the set of more cumulative instances (the \textit{BL} benchmark) since in most of the existing benchmarks the instances are highly disjunctive (i.e. many pairs of activities cannot execute in parallel).

\(^1\)\url{http://people.brunel.ac.uk/~mastjjb/jeb/info.html}
Fig. 3. The XML schema, describing the output format of the output XML file.
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<instance limit="6" horizon="21"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="resourcecost.xsd"/>
<tasks number="5" />
<task id="0" start_min="4" start_max="4" duration="3" resource="2" />
<task id="1" start_min="1" start_max="5" duration="4" resource="5" />
<task id="2" start_min="8" start_max="8" duration="6" resource="5" />
<task id="3" start_min="0" start_max="10" duration="3" resource="2" />
<task id="4" start_min="9" start_max="9" duration="3" resource="3" />
<areas number="7" />
<area id="0" x="0" y="0" width="3" height="6" cost="14" />
<area id="1" x="3" y="0" width="3" height="6" cost="11" />
<area id="2" x="6" y="0" width="3" height="6" cost="7" />
<area id="3" x="9" y="0" width="3" height="6" cost="7" />
<area id="4" x="12" y="0" width="3" height="6" cost="0" />
<area id="5" x="15" y="0" width="3" height="6" cost="7" />
<area id="6" x="18" y="0" width="3" height="6" cost="7" />
<machines number="2" />
<machine id="0" tasks="2 4 3" />
<machine id="1" tasks="0 1" />
</machines>

Fig. 4. An xml file output of executing java -cp CostInstance.jar Instance 2 5 7 6 8 0.5 50 0 3 2 123 150 no-file
The most comprehensive instance generator is presented in the body of work by Kolisch et al [8]. The authors present an instance generator ProGen for the general class of project scheduling instances. The generator is able to produce multi-modal or single-mode instances over multiple resources, where in each mode, each task can consume a different amount or each resource. The instance start times are not controlled by limiting the length of the start time interval. Instead, the tightness of precedence relationships is controlled by generating the activity-on-node network. The scarcity level of resource is controlled through a resource strength (RS) parameter, where $RS \in [0,1]$. Their method restricted to single-resource instances corresponds roughly to computing the minimal feasible resource consumption $l_{\text{min}} = \min_{i=1}^{n}\{r_i\}$, and maximal resource consumption $l_{\text{max}}$ that corresponds to the peak resource usage for the earliest start schedule. Then, the actual resource limit is generated as a convex combination: $l = l_{\text{min}} + RS \cdot (l_{\text{max}} - l_{\text{min}})$. A good overview of the benchmark library and related datasets is presented in [6].

5 Summary and Future Work

In this paper we presented a Java-based benchmark generator for a family of cost aware resource constraints. To the best of our knowledge, this is the first instance generator for the cost-aware scheduling instances. The generator is written in Java, produces an XML instance format, and is available on the website of the authors.

In future, we plan to extend the generator to support the generation of MachineChoiceCost instances as well as other classes of resource-constrained scheduling problems that would be used in the ongoing research.

References


(Co)inductive Semantics for Constraint Handling Rules*

Rény Haemmerlé

School of Computer Science, Universidad Politécnica de Madrid, Spain

Abstract. Constraint Handling Rules (CHR) shares with its spiritual ancestor, Constraint Logic Programming (CLP), nice declarative semantics consisting in a direct translation into logic. However, whereas fixpoint semantics is an important foundation of CLP, there is no equivalent notion for CHR that captures behaviors of both simplification and propagation rules. In this paper we address the problem of giving a fixpoint semantics for the whole language of CHR. firstly, we show that logical reading of a state with respect a set of simplifications can be characterized by a least fixpoint over the transition system generated by the abstract operational semantics of CHR. Similarly we demonstrate that logical reading with respect a set of propagations can be characterized by a greatest fixpoint. Then, in order to take advantage of both types of rules without losing fixpoint characterization, we presented an operational semantics closed to the one recently proposed by Betz et al. We show this semantics can be characterized by two nested fixpoints. Finally we show the resulting language is an elegant framework to program using coinductive reasoning.

1 Introduction

Owing to its origins in the tradition of Constraint Logic Programming (CLP) \cite{14}, Constraint Handling Rules (CHR) \cite{7} feature declarative semantics through direct translation to classical logic \cite{2}. However no attempts to provide fixpoint semantics to the whole CHR, sound and complete w.r.t. this declarative semantics has succeeded so far. That is particularly surprising taking into account that the fixpoint semantics is an important foundation of CLP. It is perhaps because CHR is the combination of two inherently distinct kinds of rules that this formulation is not so simple. On the one hand, the so-called constraint simplification rules (CSR) replace constraints by simpler ones while preserving their meaning. On the other hand, the so-called constraint propagation rules (CPR) add redundant constraints in a monotonic way. Even if in the declarative semantics the two notions merge into a single formalism, one of the main interests of the language comes from the explicit distinctions between the two. Indeed it is well known

* This work was funded in part by the Madrid Regional Government under the PROMETIDOS project, and the Spanish Ministry of Science under the TIN-2008-05624 DOVES project.
that propagations are useful in practice but have to be managed in a different way than simplifications to avoid trivial non-termination (see for instance [7,5]).

Soundness (i.e. each derivation corresponds a deduction) and completeness (i.e. each deduction corresponds a derivation) of the operational semantics of CSR with respect to its classical logic semantics have been proved by Abdennadher et al. [2]. However it is worth noticing that the completeness result is limited to terminating programs. On the other hand, the accuracy of CPR with respect to its classical logic semantics is only given through its translation into CSR. Since any set of propagation rules trivially loops when seen as simplifications, the completeness result does not apply to CPR.

It is well known that termination captures least fixpoint (l.f.p.) of states of a transition system. Quite naturally non-termination captures the greatest fixpoint (g.f.p.). Starting from this observation, we show in the present paper that if they are considered independently, CSR and CPR can be characterized by a sound and complete l.f.p. (or inductive) and g.f.p. (or coinductive) semantics respectively. In second place, in order to take advantage of both types of rules without losing fixpoint characterization, we present an operational semantics of close to the one recently proposed by Betz et al. [5]. Subsequently we demonstrate that this new semantics can be characterized by two nested fixpoints. Also we show it is simply implementable in accurate way and that it is an elegant framework for programming with coinductive reasoning on infinite (or non-well-founded) objects [3].

The remainder of this paper is structured as follows: Sect. 2 states syntax of CHR and summarizes two existing operational semantics. In Sect. 3, after summarizing some classical notations and results about fixpoints, we present two fixpoint semantics for CHR. These semantics built over the transition system induced by the abstract semantics of CHR, will offer a sound and complete characterization of logical reading of queries w.r.t. CSR and CPR, respectively. In Sect. 4 we defined an hybrid semantics closed to the one introduced by Betz et al. [5]. We prove this new operational semantics can be characterized by a l.f.p. nested within a g.f.p., and give an implementation via a source-to-source transformation. In Sect. 5, we illustrate the power of the language through some examples. Finally, we discuss related work in Sect. 6 before conclude in Sect. 7. Due to space limitation, some details and all proofs are omitted in the present version but can be found in a technical report [12].

2 Preliminaries on CHR

In this section we introduce syntax of CHR and present an abstract and a concrete semantics. In the next sections both semantics will be used, the former as a semantics foundations for our different fixpoint semantics, and the latter as a target to implementation purpose.
2.1 Syntax and declarative semantics of CHR

The formalisation of CHR assumes a language of built-in constraints containing the equality \( = \) over some ground complete constraint theory \( \mathcal{C} \) and defines user-defined constraints using a different set of predicate symbols. In the following we will denote variables by lower case letters, \( x, y, z, \ldots \), sets of variables by capital letters \( X, Y, Z, \ldots \), and (user-defined or built-in) constraints by lowercase letters \( c, d, e, \ldots \). By a slight abuse of notation we will confuse conjunction and multisets of constraints, forget braces around multiset and use the comma, for multiset union. We note \( \text{fv}(F) \) the set of free variables of any formula \( F \).

A CHR program is a finite set of rules of the form \( (r \ @ \ \mathcal{K} \ \mathcal{H} \ \mapsto \ \mathcal{G} \ | \ \mathcal{C}, \mathcal{B}) \) where \( \mathcal{K}, \mathcal{H} \) are multisets of user-defined constraints, called kept head and removed head respectively, \( \mathcal{G} \) is a conjunction of built-in constraints called guard, \( \mathcal{C} \) is a multiset of user-defined constraints, \( \mathcal{B} \) is a conjunction of built-in constraints, and \( r \) is an arbitrary identifier assumed unique in the program called rule name. Rules, where both heads are empty, are prohibited. The empty guard \( \top \) can be omitted together with the symbol \( | \). Similarly empty kept-head can be omitted together with the symbol \( \{ \}. The local variables of rule are the variable occurring in the guards and in the body but not in the head that is \( \text{lv}(r) = \text{fv}(\mathcal{G}) \cup \text{fv}(\mathcal{C}) \setminus \text{fv}(\mathcal{K}) \). CHR rules are divided into two classes: simplification rules if the removed head is non empty and propagation rules otherwise. Propagation rules can be written using the alternative syntax \( (r \ @ \ \mathcal{K} \mapsto \mathcal{G} | \mathcal{C}, \mathcal{B}) \).

A syntactical state is a tuple \( \langle \mathcal{C}; \mathcal{E}; X \rangle \). \( \mathcal{C} \) is a multiset of CHR constraints called CHR store, \( \mathcal{E} \) is a conjunction of built-in constraints called built-in store and \( X \) is a set of variable, called global variables. The set of strictly local variables of a given state \( \sigma = \langle \mathcal{C}; \mathcal{E}; X \rangle \) is \( \text{slv}(\sigma) = (\mathcal{V} \setminus (\text{fv}(\mathcal{C}) \cup X)) \). In the following \( \Sigma \) and \( \Sigma_b \) will denote the set of syntactical states and the states of purely built-in states (i.e. states of the form \( \langle \emptyset; \mathcal{C}; X \rangle \)).

We recall now the declarative semantics of CHR \[1\]. It consists in a conjunction of universal quantified formulas. The logical reading of a CHR rule \( (r \ @ \ \mathcal{K} \ \mathcal{H} \ \mapsto \ \mathcal{G} | \ \mathcal{C}, \mathcal{B}) \) is a logical equivalence if the guard and the kept-head are satisfied, i.e. \( \forall ((\mathcal{K} \land \mathcal{G}) \rightarrow (\mathcal{H} \leftrightarrow \exists Y (\mathcal{G} \land \mathcal{C} \land \mathcal{B}))) \) where \( Y \) are the local variables of the rules. The glue equality axiom for a CHR constraint symbol \( \gamma \), is the implicationation \( \forall ((\gamma(x) \land x = y) \rightarrow \gamma(y)) \), where \( x \) and \( y \) are disjoint sets of pairwise distinct variables. The logical reading of a program \( \mathcal{P} \) within a constraints theory \( \mathcal{C} \) is the conjunction of the logical readings of the rules of \( \mathcal{P} \), the glue equility axioms of all CHR symbols used in \( \mathcal{P} \), together with the constraints theory \( \mathcal{C} \). It will be denoted by \( \mathcal{P}^c \). The logical reading of a state \( \sigma = \langle \mathcal{C}; \mathcal{E}; X \rangle \) is the formula \( \exists Z.(\mathcal{C} \land \mathcal{E}) \). A C-model of a program \( \mathcal{P} \) is a set \( M \) of valued CHR constraints such that, for any \( I \), model of \( \mathcal{C} \), \( M \cup I \) is a model for the theory \( \mathcal{P}^c \).

2.2 Equivalent-base operational semantics

Here, we recall the equivalence-based operational semantics \( \omega_e \) of Raiser et al. \[17\]. It is close to the very abstract semantics \( \omega_a \) of Fruwirth \[7\], the most general operational semantics of CHR. We prefer the former because it includes an
explicit notion of equivalence, that will simplify many formulations. Because it
the most abstract operational semantics we consider in this paper, we will refer
to it as the abstract semantics. For the sake of generality, we present it in a
parametric form, according some logically sound equivalence relation.

An equivalence relation $\equiv_i$ is (logically) sound if two states equivalent w.r.t.
$\equiv_i$ have logically equivalent readings. The equivalent class of a some state $\sigma$ by
$\equiv_i$ will be note $[\sigma]_i$. For a given program $P$ and a sound equivalence $\equiv_i$, the
syntactical transition $\rho \# \in P$ and the $\equiv_i$-transition, $\rho \rightarrow_i$, are the least relation on
states verifying the following rules:

$$
\frac{r \in P \mid \mathcal{H} \Rightarrow \mathcal{G} \mid C, B \in \rho \sigma \setminus X \cap X = \emptyset}{\langle \mathcal{K}, \mathcal{D}, G \land E; X \rangle \equiv \langle \mathcal{C}, \mathcal{D}, B \land G \land E; X \rangle} \quad \sigma_1 \equiv_i \sigma_1', \sigma_2 \equiv_i \sigma_2' \quad \rho \rightarrow_i \sigma_1, \sigma_2
$$

where $\rho$ is a renaming. If such transition is possible, we will say that the tuple
$r\#([K\setminus\mathcal{H}\setminus\mathcal{G}]$ is a redex for the state $\sigma$. We denoted the transitive closure of the
relation $(\rightarrow_i \cup \equiv_i)$ by $\rightarrow_i^*$.

The abstract equivalence is the least relation $\equiv_a$ defined over $\Sigma$ and verifying:
1. $\langle C; y = t \land D; X \rangle \equiv_a \langle G[Y \setminus t]; y = t \land D; X \rangle$
2. $\langle C; X \rangle \equiv_a \langle D; X \rangle$
3. $\langle C; D; X \rangle \equiv_a \langle C; D; X \rangle$ if $C \models \exists Y \exists D \iff \exists Y' \exists E$ where $Y = \text{slv}((C; D; X))$ and

\[ Y' = \text{slv}((C; E; X)). \]
4. $\langle C; E; X \rangle \equiv_a \langle C; E; \{x\} \cup X \rangle$ where $x \notin \text{fv}(C, E)$.

Note this equivalence is logically sound [17]. The abstract transition is defined as the $\equiv_a$-transition.

### 2.3 Concrete operational semantics of CHR

This subsection presents the operational semantics $\omega_p$ of De Koninck et al. [6].
In this framework rules are annotated with explicit priorities that reduce the
non-determinism in the choice of the rule to apply. As initially proposed by Ab-
dennadher [11], this semantics include a partial control that prevents the trivial
looping of propagation rules by restricting their firing only once on same inst-
ances. By opposition to the abstract semantics we have just defined, we will
call this semantics concrete.

An identified constraint is a pair noted $c\#i$, associating a CHR constraint $c$
with an integer $i$. For any identified constraints, we define the function $\text{chr}(c\#i) = c$
and $\text{id}(c\#i) = i$, and extend them to sequences and set of identified constraints.
A token is a tuple $(r, i)$, where $r$ is a rule name and $i$ is a sequence of integers. A
concrete CHR state is a tuple of the form $\langle C; D; E; T \rangle^X_n$ where $C$ is a multiset
of CHR and built-in constraints, $D$ is a multiset of identified constraints, $E$ is a
conjunction of built-in constraints, $T$ is a set of tokens and $n$ is an integer. We
assume moreover that the identifier of each identified constraints in the CHR
store is unique and smaller than $n$. For any program $P$ the transition relation
$\rho \rightarrow_c$ is defined as follows:
3 Transition system semantics for pure CSR and CPR

In this section we propose fixpoint semantics for both CSR and CPR programs. Before, we recall briefly useful notations and results about fixpoints.\(^1\)

3.1 Fixpoints

Let assume some arbitrary complete lattice \((\mathcal{L}, \supseteq, \sqcap, \sqcup, \top, \bot)\). A function \(f : \mathcal{L} \rightarrow \mathcal{L}\) is monotonic if \(f(\mathcal{X}) \supseteq f(\mathcal{Y})\) whenever \(\mathcal{X} \supseteq \mathcal{Y}\). An element \(\mathcal{X} \in \mathcal{L}\) is a fixpoint for \(f : \mathcal{L} \rightarrow \mathcal{L}\) if \(f(\mathcal{X}) = \mathcal{X}\). \(\mathcal{X}\) is a least fixpoint for \(f\), noted \(\mu \mathcal{X}.f(\mathcal{X})\), if it is a fixpoint and \(\mathcal{Y} \supseteq \mathcal{X}\) whenever \(\mathcal{Y}\) is a fixpoint for \(f\). The greatest fixpoint of a function \(f (\nu \mathcal{X}.f(\mathcal{X}))\) is defined by duality.

**Theorem 1 (Knaster–Tarski).** Let \(f\) be a monotonic function on sets. Then \(F\) has a l.f.p. \(\mu \mathcal{X}.f(\mathcal{X})\) and a greatest fixpoint \(\nu \mathcal{X}.f(\mathcal{X})\).

3.2 Inductive semantics for CSR

In this section we give a fixpoint semantics for CSR. We call it inductive transition system semantics because it is defined as a least fixpoint built over the transition system induced by the abstract semantics.

**Definition 1.** For a given program \(\mathcal{P}\) and an given sound equivalence relation \(\equiv_i\), the existential immediate cause operator \(\langle \mathcal{P} \rangle_i\) is defined as:

\[
\langle \mathcal{P} \rangle_i (\mathcal{X}) = \{ \sigma \in \Sigma \mid \text{there exists } \sigma' \in \Sigma \text{ such that } \sigma \xrightarrow{c} \sigma' \text{ implies } \sigma' \in \mathcal{X} \}
\]

The inductive (transition system) semantics of a CSR program \(\mathcal{Q}\) is defined as:

\[
\mathcal{M}_i^{\mathcal{C}} (\mathcal{Q}) = \mu \mathcal{X}.(\langle \mathcal{Q} \rangle_a (\mathcal{X}) \cup (\Sigma_{\mathcal{b}} \setminus \{\theta; \bot; \bot\})).
\]

The following theorem, which can be viewed as a reformulation of the main results of \[2\], states the accuracy of the inductive semantics with respect to the logical reading of CSR program. In other words, a state is in the inductive semantics of a CSR program if and only if its logical reading is valid in the theory \(\mathcal{P}_c\), i.e. any interpretation of \(\mathcal{P}_c\) is a model for the logical reading of the state. Before stating the theorem we introduce the notion of confluence.

\(^1\) For more details one can refer, for example, to Lloyd’s Book [15].
Definition 2. A transition system $\rightarrow$ is confluent if whenever $\sigma \rightarrow^* \sigma_1$ and $\sigma \rightarrow^* \sigma_2$ hold, there exist a state $\sigma'$ such that $\sigma_1 \rightarrow^* \sigma'$ and $\sigma_2 \rightarrow^* \sigma'$.

Theorem 2. Let $P$ be program such that $\rightarrow_a$ is confluent, and let $(D; E; X)$ be a state having a derivation terminating in a purely built-in state. We have:

$$(D; E; X) \in M^C_i(P) \text{ if and only if } P^C \models \exists (D \land E).$$

3.3 Coinductive semantics for CPR

We continue by giving a similar characterization for CPR. This semantics is defined by the g.f.p. of the universal cause operator. Hence, we call it coinductive transition system semantics. As it is the case for the linear logic semantics of CHR [4], this characterization requires $C$ to be an intuitionistic constraints system. Indeed, we will assume from now and until end of the paper that $C$ is defined by a constraint theory based on intuitionistic logic and with non-logical axioms of the form $\forall (C \rightarrow \exists Z.D)$, where $C$ and $D$ stand for conjunctions of built-in constraints.

Definition 3. For a given program $P$ and an given sound equivalence relation $\equiv_i$, the universal immediate cause operator $[P]_i$ are is defined as:

$$[P]_i(X) = \{ \sigma \in \Sigma \mid \text{for all } \sigma' \in \Sigma, \sigma \rightarrow_i \sigma' \text{ implies } \sigma' \in X \}$$

The coinductive (transition system) semantics of a CPR program $Q$ is defined as:

$$M^C_{co}(Q) = \nu X. ([Q]_a(X) \cap (\Sigma \setminus \{0; 1; 0\})$$

The next theorem states the accuracy of the coinductive semantics with respect to the logical reading of CPR. Since for most of CPR programs, the theory $P^C$ is trivially incomplete, one has get content with satisfiability instead of validity. That is to say, a state is in the coinductive semantics of a CPR program if and only if $P^C$ has at least one interpretation that is a model for the logical reading of the state. Notice that for the completeness direction, we have to ensure that the constraints are provide in a sufficient quantity to engage the derivation. To state the theorem we assume the following notation:

$$P^n = \{(r @ K \land H \iff G \mid B, L, n \cdot P) \mid (r @ K \land H \iff G \mid B, L, P) \in P\}$$

Theorem 3. Let $P$ be a CPR program, $n$ be some integer bigger than the maximal number of constraints occurring in the head of any rule of $P$. We have:

$$(n \cdot D; E; X) \in M^C_{co}(P^n) \text{ if and only if } P^C \not\models \exists (D \land E)$$

where $n \cdot D$ is the scalar product of the multiset $D$ by $n$.

Next result shows it is sufficiently to consider only one fair derivation to know if a state is in the coinductive semantics of a program.

Definition 4. An finite or infinite derivation is fair, if any redex appearing in one of the state of the derivation is reduced within the derivation.

Proposition 1. $\sigma \in M^C_{co}(P)$ iff there is a fair and consistent derivation starting from $\sigma$. 
4 Hybrid semantics

In this section, we aim at obtaining a fixpoint semantics for the whole language. Nonetheless, one has to notice that the completeness result for CSR needs, among other, the termination of $P^{-\rightarrow a}$, while the equivalent result for CPR is based on the monotonic evolution of the constraints store along derivations. Hence combining naively CSR and CPR will break both properties, leading consequently to an incomplete model. In order to provide an accurate semantics with result to the logical semantics enjoying the advantages of both kinds of rules, we introduce a hybrid operational semantics, we note $\omega_h$, following preliminaries ideas of Betz et al. [5]. In practice we will assume the CHR constraints is divided into two kinds of constraints, the so-called persistent constraints, acting as classical logic statements and linear constraints, acting as linear logic statements.

4.1 Hybrid operational semantics

From now, let’s assume the set of user-defined constraints is divided into two sets: the linear symbols and the persistent symbols. CHR constraints built from the linear (resp. persistent) symbols are called linear (resp. persistent) constraints. Hybrid rules are CHR rules where the kept head is purly persistent and the removed head is purly linear. We will denote by $\Sigma_p$, the set of purely persistent states (i.e. states of the form $\langle P; C; X \rangle$ where $P$ is a set of persistent constraints).

In the following we denote by $P^i$ and $P^p$ respectively the set of hybrid simplifications (rule with a non empty removed head) and hybrid propagations (rule with an empty removed head) of a hybrid program $P$.

Definition 5. The hybrid equivalence is the smallest relation $\equiv_{h}$ over syntactical states containing $\equiv_a$ and satisfying the following condition:

$\langle c, c; C; D; X \rangle \equiv_h \langle c, C; D; X \rangle$ if $c$ is a persistent constraints

The hybrid transition, $P^{-\rightarrow_h}$, is defined as the $\equiv_h$-transition. We say that a finite or infinite derivation $\sigma_0 \overset{P^i}{\rightarrow_h} \sigma_1 \overset{P^p}{\rightarrow_h} \ldots$ is consistent if so are all $\sigma_i$’s.

4.2 Logical model for confluent programs

In this section, we tackle the problem of proving consistency of the logical reading of an hybrid program. Indeed, contrary to the results of Sect. 3, the proof of Thm. 5 needs among others the considered programs to be consistent. To this aim, we construct a interpretation for confluent programs in the pure CLP style [14] and demonstrate it provides a model for confluent programs.

Definition 6. For any program $P$, the immediate (backward) consequence operator $T_C^P : 2^{B_C} \rightarrow 2^{B_C}$ is defined as:

$T_C^P(X) = \{ c \in H_\rho | \exists \text{ a rule } (K \setminus H) \leftrightarrow (G | B_c, B_h) \in P \}$

such that $C \models (G \land B_h) \rho$ and $(K, B_c)_\rho \subset X$
It is worth noting that $T_C^P$ is nothing more than the immediate consequence operator of the CLP program derived form $P$ when simplification rules are read as backward implications (i.e. implications from the right to the left hand side). Indeed, one can consider the implication $c_1 \land \cdots \land c_n \leftarrow \exists Z(G \land C \land B \land K)$ as the conjunction of the $n$ implications $(c_1 \leftarrow \exists Z(G \land C \land B \land K)), \ldots, (c_n \leftarrow \exists Z(G \land C \land B \land K))$, $T_C^P$ being obviously monotonic, Thm. 1 ensures it has a l.f.p.

### Theorem 4.
Let $P$ be a hybrid program such that $\frac{\leftarrow}{h}$ is confluent. $\mu X. (T_C^P(X))$ is the least $C$-model of $P$.

One can note that since CSR programs can be trivially viewed as an hybrid programs (by considering all CHR constraints as linear), the theorem closes a conjecture of Abdennadher et al. [2] about consistency of general confluent CSR programs. Indeed, it is worth noting that original proof of consistency of confluent program is limited to range-restricted programs (i.e. programs without local variables).

#### 4.3 Hybrid transition system semantics

We present the transition system semantics for hybrid programs. This semantics are expressed by two fixpoints of the immediate cause operators

### Definition 7. The hybrid coinductive (transition system) semantics of a hybrid program $P$ is

$$M_C^P(P) = \nu X. ((P|_h(X) \cap \mu Y. ((P|_h(Y) \cup (\Sigma \setminus \{\emptyset; \bot; \emptyset\}|_h)))$$

The following theorem states soundness and completeness of the hybrid coinductive semantics for data sufficient states with respect confluent programs. In this paper, we do not address the problem of proving confluence of hybrid programs, but believe it can be tackled by extending quite straightforwardly the work of Abdennadher et al. [2] or by adequately instantiating the notion of abstract critical pair we proposed in a previous work [13].

### Definition 8. A hybrid state $\sigma$ is data-sufficient with respect a hybrid program $P$, if for any state $\sigma'$ accessible form $\sigma$ there exists a purely persistent state $\sigma''$ accessible form $\sigma'$.

### Theorem 5. Let $P$ be a hybrid program such that $\frac{\leftarrow}{h}$ is confluent, and let $\langle L, P; E; X \rangle$ be a data-sufficient state with respect to $P$. We have:

$$\langle L, P; E; X \rangle \in M_C^P(P) \text{ if and only if } P^C \not\vdash \exists (L \land P \land E).$$

The next proposition states it is enough to consider only one fair derivation.

### Definition 9. An infinite hybrid derivation $\sigma_0 \xrightarrow{\sigma_1} \sigma_1 \xrightarrow{\sigma_2} \ldots$ is fair if for any state $\sigma_i$ and any redex $(r \# \langle P|\emptyset; D; X \rangle)$ of $\sigma_i$, where $r$ is a propagation rule there is a state $\sigma_j$ s.t. the reduced redex of $\sigma_j$ is $(r \# \langle P|\emptyset; E; X \rangle)$ with $C \models (\exists Y(E \rightarrow \exists Y.D))$ and $Y = (fv(E, D) \setminus X)$. 

### Proposition 2. Let $P$ be a confluent and data-sufficient hybrid program and $\sigma$ be a data-sufficient hybrid state. Any fair derivation starting form $\sigma$ is consistent if and only if $\sigma \in M_C^P(P)$.
Let $P^\circ$ the program $P$ where simplifications and propagations are given with the priorities 3 and 4 respectively:

step 1 Apply the following rules until convergence:

If $(p :: c, d, K \iff G | B, L, P)$ is in $P^\circ$, with $C |\exists (c = d)$
then add the rule $(p :: c, K \iff c = d \land G | B, L, P)$ to $P^\circ$

step 2 Substitute any rule $(p :: c_1, \ldots, c_m \iff G | B, L, d_1, \ldots, d_n)$ by
$(p :: K, a(X_1, c_1), \ldots, a(X_m, c_m) \iff G | B, L, f(d_1), \ldots, f(d_n))$
where $x_1, \ldots, x_m$ are pairwise distinct variables

step 3 Add to $P^\circ$ the rules:
1 :: stamp @ $f(X), c_f(Y) \iff f(Y, X), c_f(Y + 1)$
2 :: set @ $a(Y, X) \setminus a(Z, X) \iff Y > Z | \top$
5 :: unfreeze @ $f(Y, X), c_a(Y) \iff a(Y, X), c_a(Y + 1)$

Fig. 1. Source-to-source translation for hybrid programs

4.4 Implementation

We continue by addressing the question of implementing of the hybrid semantics in a sound and complete way. For this purpose, we assume without loss of generality that the constraint symbols $f/1$, $f/2$, $a/2$, $c_f/1$, and $c_a/1$ are fresh with respect the program $P$ we considered. The implementation of a hybrid program $P$ consists into a source-to-source translation $P^\circ$ intended to be executed in the concrete semantics $\omega_p$. This transformation is given in detail by Fig. 1. In order to be executed an hybrid state $\sigma$ has to be translated into a concrete state $\sigma^\circ$ as following: if $L$ and $(c_1, \ldots, c_n)$ are multisets of linear and persistent constraints respectively, then $\langle L, d_1, \ldots, d_n; \emptyset; V \rangle^\circ = \langle \langle L, f(d_1), \ldots, f(d_n), c_f(0), c_a(0); \emptyset; \emptyset \rangle^V \rangle_0$

Before going further, let us give some intuitions about behaviour of the translation. If a rule needs two occurrences of the same persistent constraint, the step 1 will insert an equivalent rule which needs only one occurrence of the constraint. In the translation each persistent constraint can be applied in three different successive states: fresh indicated by $f/1$, frozen indicated by $f/2$, and alive indicated by $a/2$. The step 2 ensures, on the one hand, that only alive constraints can be used to launch a rule, and, on the other hand, that the persistent constraints of the right-hand side are inserted as fresh. Each frozen and alive constraint is associated to a time stamp indicating the order in which it has been asserted. The fresh constraint are time stamped and marked as frozen as soon as possible by stamp, the rule of highest priority (the constraint $c_f/1$ indicating the next available time stamp). Only if no other rule can be applied, the unfreeze rule changes the oldest frozen constraint into an alive constraint meanwhile preserving its time stamp (the constraint $c_a/1$ indicating the next constraint to unfreeze). Rule set prevents trivial loops, by removing the youngest occurrence of two identical constraints.
The two following theorems state, our implementation is sound and complete with respectively to failure. Thm. 6 shows furthermore that the implementation we propose here is sound with respect to finite successes. It is worth noting that it is hopeless to look for a complete implementation with respect to success, since the problem to know if a state is in the coinductive semantics is undecidable. The intuition being this last claim is that otherwise it would possible to solve the halting problem.

Theorem 6 (Soundness w.r.t. successes and failures). Let $P$ be a confluent hybrid program, $\langle \langle D; \emptyset; E; \emptyset \rangle \rangle P X_0 \rightarrow^* P \langle \langle \emptyset; D'; E'; T \rangle \rangle_i X_i \not\rightarrow^*_P$ be a terminating derivation, and $Y = \text{fv}(E) \setminus X$. $\langle C; D; E \rangle \in M_C^X(P)$ holds if and only if so does $C \models \exists Y. E'$. 

Theorem 7 (Completeness w.r.t. failures). If $\langle C; D; E \rangle \not\models M_C^X(P)$, then any derivation form $\langle \langle C^0; \emptyset; D; \emptyset \rangle \rangle \rightarrow^*_P$ finitely fail.

5 Applications

In this section, we illustrate the power of CHR for the coinductive reasoning when it is provided with its fixpoint semantics. In particular we show it is an elegant framework to realize coinductive equality proofs for language and regular expressions as presented by Rutten [18]. It is worth noticing that, despite we do not address the problem here, the coinductive semantics of CHR provide also an ideal theoretical framework to tackle infinite data structure such as rational tree [16].

5.1 Coinductive language equality proof

Firstly, let us introduce the classical notion of binary automaton in a slightly different way than usual. A binary automaton is a pair $(L, f)$ where $L$ is a possibly infinite set of states and $f : L \rightarrow \{a, b\} \times L \times L$ is a function called destructor. Let us assume some automaton $(L, f)$. For any state $L \in L$ such that $f(L) = (T, L_a, L_b)$, we write $L \xrightarrow{a} L_a$ and $L \xrightarrow{b} L_b$, and $t(L) = T$. $L(L) = \{a_1 \ldots a_n | L \xrightarrow{a_1} L_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} L_n \land t(L_n) = 1\}$ is the language accepted by a state $L$. A bisimulation between states is a relation $R \subset L \times L$ verifying:

If $K \not\sim L$ then

\[
\begin{cases}
  t(K) = t(L), \\
  K \xrightarrow{a} K_a, L \xrightarrow{a} L_a, K_a \not\sim L_a, \text{ and} \\
  K \xrightarrow{b} K_b, L \xrightarrow{b} L_b, K_b \not\sim L_b,
\end{cases}
\]

On the contrary to the standard definition, in the presented setting, automata do not have an initial state and may have an infinite number of states. As represented here, automata are particular coalgebra [3]. Due to the space limitations,
we will not enter in details in the topic of coalgebra\footnote{We invite unfamiliar readers to refer to the gentle introduction of Rutten \cite{rutten2000coalgebras}} but only state the following \textit{Coinductive Proof Principle} \cite{hsiung2003coinductive}, which gives rise the representation of automata as coalgebra:

In order to prove the equality of the languages recognized by two states $K$ and $L$, it is sufficient to establish the existence of a bisimulation relation in $\mathcal{L}$ that includes the pair $(K, L)$.

A nice application of CPR consists in the possibly to directly represent coalgebra and prove bisimulation between states. For instance, one can easily represent a finite automata, using variables for states and binary user-defined constraints $f$ for the destructor function. Fig. 2 gives an example of automaton and its representations as a multiset $\mathbb{D}$ of CHR constraints. Once the representation of automata is fixed, one can transpose the definition of bisimulation into a single propagation rule:

$$f(L, (L_t, L_a, L_b)), f(K, (K_t, K_a, K_b)), L \sim K \implies L_t = K_t, L_a \sim K_a, L_b \sim K_b$$

Using the coinductive proof principal and the Thm. \cite{hsiung2003coinductive} it is simple to prove that two states of the coalgebra represented by $\mathbb{D}$ accept or not the same language. For example, to conclude that $L_1$ and $K_1$ recognize the same language while $L_1$ and $K_2$ do not, one can prove the execution of $\langle \mathbb{D}, L_1 \sim K_1; \top; \emptyset \rangle$ never reaches inconsistent states, while there are inconsistent derivations starting from $\langle \mathbb{D}, L_1 \sim K_2; \top; \emptyset \rangle$.

\section{Coinductive solver for regular expressions}

We have just shown that CPR are an nice framework for coinductive reasoning about coalgebra. Nonetheless the explicit representation of the automata by user defined constraints (as in Fig. 2) will limit ourselves to finite state automata. One simple idea to overpass this limitation is to represent infinite automata in an implicit way. For instance, one can represent states using regular expressions and implement a computation of the destructor function using derivatives \cite{rutten2000coalgebras}.

Let us assume the following syntax for regular expressions:

$$E ::= a | b | E^* | E, E | L \quad L ::= \emptyset | [E|L]$$

\footnote{We invite unfamiliar readers to refer to the gentle introduction of Rutten \cite{rutten2000coalgebras}.}
where $a$ and $b$ are characters, ($^*$) and (,) stand for the Kleene star and the concatenation respectively, and the list corresponds to the alternation of its elements. Here follows a possible implementation of the destructor function:

\[
\begin{align*}
  f([], R) & \iff R = (0, [], []). \\
  f([K], R) & \iff R = (T, A, B), f(K, (Kt, Ka, Kb)), f(L, (Lt, La, Lb)), \text{ or}(Kt, Lt, T), \text{union}(Ka, La, A), \text{union}(Kb, Lb, B). \\
  f(a, R) & \iff R = (0, [[*]], []). \\
  f(b, R) & \iff R = (0, [[]], [[*]]). \\
  f(K^*, R) & \iff R = (1, [(Ka, [K^*])], [(Kb, [K^*])]), f(K, (\omega, Ka, Kb)). \\
  f((K, L), R) & \iff f(K, (Kt, Ka, Kb)), f(L, (La, R), (Kb, Kb, L, R)). \\
  f(0, Ka, Kb, L, R) & \iff R = (0, [(Ka, L)], [(Kb, L)]). \\
  f(1, Ka, Kb, L, R) & \iff R = (T, A, B), f(L, (T, La, Lb)), \text{union}([(Ka, L)], La, A), \text{union}([(Kb, L)], Lb, B).
\end{align*}
\]

where \text{or}/3 unifies its third argument with the Boolean disjunction of its two first elements, and \text{union}/3 unifies its last arguments with the union of the lists given as first arguments. Now one can adapt the encoding of bisimulation given in the previous section as follows:

\[\text{L} \sim \text{K} \implies \text{nonvar}(\text{L}), \text{nonvar}(\text{K}), f(\text{L}, (\text{T}, \text{La}, \text{Lb})), f(\text{K}, (\text{Ka}, \text{Kb})), \text{La} \sim \text{Ka}, \text{Lb} \sim \text{Kb}.\]

We are now able to prove equality of regular expression using the implementation of CHR hybrid semantics provided in Sect. 4.4 (The complete program can be found in the long version of this paper \cite{12}). For example the following state leads to an irreducible consistent state, proving thanks to the Coinductive Proof Principal, Thm. 5 and Thm. 6 that the two regular expressions recognize the same language:

\[
(((b^*, a)^*, (a, b^*))^* \sim [[^*], (a, [a, b]^*], ([a, b]^*, (a, (a, [a, b]^*))))]; T; \emptyset)^\omega
\]

It should be underlined that the use of simplifications instead of propagations, for encoding the destructor function is essential here in order to avoid the saturation of the memory by tokens. Just notice that the confluence of the simplification part needed by Thm. 5 and Thm. 6 can be easily inferred, since the program is deterministic.

Of course, no one should be surprised that equivalence of regular expressions is decidable. The interesting point here is that the notion of coalgebra and bisimulation can be transposed naturally to CHR. Moreover it is worth noticing the program given has the properties required by a constraints solver. Firstly the program is effective, i.e. it can actually prove or disprove if that two expressions are equal. The first part of the claim can be proved using Kleene theorem \cite{18} and the idempotency and commutativity of the alternation, enforced here by the \text{union}/3 predicate. The second part is direct by the completeness w.r.t. failures. Secondly, the program is incremental: it can indeed deal with partially instanced expression by freezing some computations provided without enough information. Last but not the least, one can easily add to the system new expressions (as for instance $e$, $E?$, or $E^+$). For this purpose it is just necessary to provide a new simplification rule for computing the result of the corresponding destructor function. For example we can add to the program the following rule:

\[
\begin{align*}
  f([K], R) & \iff R = (T, A, B), f(K, (Kt, Ka, Kb)), f(L, (Lt, La, Lb)), \text{ or}(Kt, Lt, T), \text{union}(Ka, La, A), \text{union}(Kb, Lb, B). \\
  f(\text{La}, R) & \iff R = (0, [[^*]], []). \\
  f(\text{Lb}, R) & \iff R = (0, [[]], \text{union}/3). \\
  f(\text{Ka}, \text{Ka}, \text{Lb}, \text{R}) & \iff R = (\omega, Ka, Kb, L, R). \\
  f(\text{Kb}, \text{Kb}, \text{La}, \text{R}) & \iff R = (0, [(Ka, L)], [(Kb, L)]).
\end{align*}
\]
and prove as previously that \( a^+ \) and \((a, a^*)\) recognize the same language while \( a^+ \) and \( a^* \) do not:

\[
f(K^+, R) \iff R = (T, [K_a, (K_a, K^+)], [K_b, (K_b, K^+)]), f(K, (K_a, K_b)).
\]

6 Related work

Although \( \omega_h \) was inspired by the semantics \( \omega_l \) Betz et al. [5], it differs on some important points. In \( \omega_l \), the kind of a constraint (linear or persistent), is not statically fixed, but dynamically determined according the type of rules from which it was produced. Note the static distinction between the two kinds of constraint is necessary to express the side conditions of \( \text{Thm. 5} \). Another significant difference is that the completeness of \( \omega_l \) is not limited to range restricted programs. We were able to drop this restriction because we were interested by the logical meaning of propagations and not in their interpretation as simplifications. Likewise the implementation proposed for \( \omega_l \) does not ensure fairness, leading to loops over programs having trivially no correct answer.

The transition system semantics of CHR have important similitude with the fixpoint semantics of CLP [14]. Both are defined by fixpoint point of a somehow similar operator and fully abstract the logical meaning of programs. Nonetheless the coinductive semantics of CHR is not compositional. That is not a particular drawback of our semantics, since the logical semantics we aim to characterize is not compositional i.e. if the logical reading of two states are independently consistent, then one cannot ensure that so is their conjunction. It should be noticed that this non-compositionally prevents the immediate cause operator to be defined over a \( \mathcal{C} \)-base as it is done in CLP, and requires a definition over set of states.

In [9], Gabrielli and Meo defined a fixpoint semantics for CSR built on traces and focusing on input/output behaviour. To a certain extent this semantics can be related to our inductive semantics. It have been then extended to handle propagations by adding into the states an explicit token store in order to remember the propagation history [10]. Such an extension leads to a complicated model which is furthermore incomplete with respect to logical semantics. In [8], Gabrielli and Levi defined a fixpoint semantics for non-deterministic concurrent constraint programming in the typical logic programming style. Nevertheless this semantics is based on infinite unfoldings which seems difficult to transpose to standard CHR, which allows multiple head and forbid disjunctions. It seems that any other formulations of fixpoint semantics for concurrent constraint programming are more removed from typical logic programming style and are, in any case, limited to single head rules.

Concerning other programming frameworks designed to deal with coinductive reasoning, one can cite the Co-Logic Programming of Simon et al. [19], which extends LP by using a g.f.p. in place of l.f.p. in the traditional Herbrand
semantics. But because they limit themselves to rational terms, the implementa-
tion proposed is incomplete with respect to the logical reading of programs. In fact, the incompleteness with respect to failures is directly inherited from classical LP, while the incompleteness with respect to successes is due to the non-continuity of the immediate consequence operators when considering any reasonable subset of the infinite terms [15]. That contrasts with CHR where any intuitionistic constraint theory can be chosen without losing completeness with respect to failures. Another interesting framework is the circular coinductive rewriting (ccrw) of Goguen et al. [11]. This elegant proof method seems more powerful than CHR regarding only pure coinductive proofs but is perhaps a less general purpose language. As one of the particular advantage of CHR over ccrw one can referred to the incremental nature of this former.

7 Conclusion

We have defined l.f.p. semantics for CHR simplification rules and a g.f.p. semantics for CHR propagations rules, and proved both to be accurate with respect the logical reading of the rules. By using a hybrid operational semantics with persistent constraints similar to the one of Betz et al., we were able to characterize CHR programs combining both simplification and propagations by a fixpoint semantics without losing completeness with respect to logical semantics. In doing so, we have improved noticeably results about logical semantics of CHR. For instance, we close a conjecture of Abdennadher et al. about consistency of confluent programs. Subsequently we proposed an implementation of this hybrid semantics and showed it is an elegant framework for programming coinductive reasoning.

The observation, that non-termination of all derivations starting from a given state ensures this latter to be in the coinductive semantics of an hybrid programme, suggests that the analysis of universal non-termination of a CHR program might be worth investigating. The comparison of CHR with ccrw may suggest there should exists possibilities to improve the completeness with respect to success of the implementation we have proposed.

References