

Explaining Time-Table-Edge-Finding Propagation for the Cumulative Resource Constraint

Andreas Schutt, Thibaut Feydy, and Peter J. Stuckey

Optimisation Research Group, National ICT Australia, and Department of Computing and Information Systems, The University of Melbourne, Victoria 3010, Australia

{andreas.schutt,thibaut.feydy,peter.stuckey}@nicta.com.au

Abstract. Cumulative resource constraints can model scarce resources in scheduling problems or a dimension in packing and cutting problems. In order to efficiently solve such problems with a constraint programming solver, it is important to have strong and fast propagators for cumulative resource constraints. Time-table-edge-finding propagators are a recent development in cumulative propagators, that combine the current resource profile (time-table) during the edge-finding propagation. The current state of the art for solving scheduling and cutting problems involving cumulative constraints are lazy clause generation solvers, *i.e.*, constraint programming solvers incorporating nogood learning, have proved to be excellent at solving scheduling and cutting problems. For such solvers, concise and accurate explanations of the reasons for propagation are essential for strong nogood learning. In this paper, we develop a time-table-edge-finding propagator for `cumulative` that explains its propagations. We give results using this propagator in a lazy clause generation system on resource-constrained project scheduling problems from various standard benchmark suites. On the standard benchmark suite PSPLib, we are able to improve the lower bound of about 60% of the remaining open instances, and close 6 open instances.

1 Introduction

A cumulative resource constraint models the relationship between a scarce resource and activities requiring some part of the resource capacity for their execution. Resources can be workers, processors, water, electricity, or, even, a dimension in a packing and cutting problem. Due to its relevance in many industrial scheduling and placement problems, it is important to have strong and fast propagation techniques in constraint programming (CP) solvers that detect inconsistencies early and remove many invalid values from the domains of the variables involved. Moreover, when using CP solvers that incorporate “fine-grained” nogood learning it is also important that each inconsistency and each value removal from a domain is explained in such a way that the full strength of nogood learning is exploited.

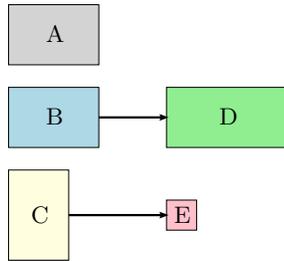


Fig. 1: Five activities with precedence relations.

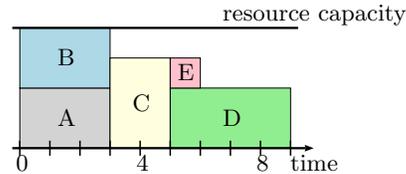


Fig. 2: A possible schedule of the activities.

In this paper, we consider *renewable* resources, *i.e.*, resources with a constant resource capacity over time, and *non-preemptive* activities, *i.e.*, whose execution cannot be interrupted, with fixed processing times and resource usages. In this work, we develop explanations for the time-table-edge-finding (TTEF) propagator [34] for use in lazy clause generation (LCG) solvers [22,10].

Example 1. Consider a simple cumulative resource scheduling problem. There are 5 activities A, B, C, D, and E to be executed before time period 10. The activities have processing times 3, 3, 2, 4, and 1, respectively, with each activity requiring 2, 2, 3, 2, and 1 units of resource, respectively. There is a resource capacity of 4. Assume further that there are precedence constraints: activity B must finish before activity D begins, written $B \ll D$, and similarly $C \ll E$. Figure 1 shows the five activities and precedence relations, while Fig. 2 shows a possible schedule, where the start times are: 0, 0, 3, 5, and 5 respectively.

In CP solvers, a cumulative resource constraint can be modelled by a decomposition or, more successfully, by the global constraint `cumulative` [2]. Since the introduction of this global constraint, a great deal of research has investigated stronger and faster propagation techniques. These include time-table [2], (extended) edge-finding [21,33], not-first/not-last [21,29], and energetic-reasoning propagation [4,6]. Time-table propagation is usually superior for *highly disjunctive* problems, *i.e.*, in which only some activities can run concurrently, while (extended) edge-finding, not-first/not-last, and energetic reasoning are more appropriate for *highly cumulative* problems, *i.e.*, in which many activities can run concurrently.[4] The reader is referred to [6] for a detailed comparison of these techniques.

Vilim [34] recently developed TTEF propagation which combines the time-table and (extended) edge-finding propagation in order to perform stronger propagation while having a low runtime overhead. Vilim [34] shows that on a range of highly disjunctive open resource-constrained project scheduling problems from the well-established benchmark library PSPLib,¹ TTEF propagation can generate lower bounds on the project deadline (*makespan*) that are superior to those

¹ See <http://129.187.106.231/psplib/>.

found by previous methods. He uses a CP solver without nogood learning. This result, and the success of LCG on such problems, motivated us to study whether an explaining version of this propagation yields an improvement in performance for LCG solvers.

In general, nogood learning is a resolution step that infers redundant constraints, called *nogoods*, given an inconsistent solution state. These nogoods are permanently or temporarily added to the initial constraint system in order to reduce the search space and/or to guide the search. Moreover, they can be used to short circuit propagation. How this resolution step is performed is dependent on the underlying system.

LCG solvers employ a “fine-grained” nogood learning system that mimics the learning of modern Boolean satisfiability (SAT) solvers (see e.g. [20]). In order to create a strong nogood, it is necessary that each inconsistency and value removal is explained concisely and in the most general way possible. For LCG solvers, we have previously developed explanations for time-table and (extended) edge-finding propagation [26]. Moreover, for time-table propagation we have also considered the case when processing times, resource usages, and resource capacity are variable [24]. Explanations for the time-table propagator were successfully applied on resource-constraint project scheduling problems [26,27] and carpet cutting [28] where in both cases the state-of-the-art of exact solution methods were substantially improved.

Explanations for the propagation of the **cumulative** constraint have also been proposed for the PaLM [14,13] and SCIP [1,7,12] frameworks. In the PaLM framework, explanations are only considered for time-table propagation, while the SCIP framework additionally provides explanations for energetic reasoning propagation and a restricted version of edge-finding propagation. Neither framework consider bounds widening in order to generalise these explanations as we do in this paper. Other related works include [32], which presents explanations for different propagation techniques for problems only involving disjunctive resources, *i.e.*, cumulative resources with unary resource capacity, and generalised nogoods [15]. A detailed comparison of explanations for the propagation of **cumulative** resource constraints in LCG solvers can be found in [24].

The contributions of this paper are:

- We define a new simpler TTEF propagator for **cumulative**.
- We define how to explain the propagation of this propagator.
- We compare the performance of the TTEF propagator with explanation, against time-table propagation with explanation.
- We improve the lower bounds of a large proportion of the open instances in the well studied PSPLib, and close 6 instances from PSPLib.
- We improve the lower bounds and close many more instances on less studied highly cumulative benchmarks.

2 Cumulative Resource Scheduling

In cumulative resource scheduling, a set of (non-preemptive) activities \mathcal{V} and one cumulative resource with a (constant) resource capacity R is given where

an activity i is specified by its *start time* S_i , its *processing time* p_i , its *resource usage* r_i , and its *energy* $e_i := p_i \cdot r_i$. In this paper we assume each S_i is an integer variable and all others are assumed to be integer constants. Further, we define est_i (ect_i) and lst_i (lct_i) as the *earliest* and *latest* start (completion) time of i .

In this setting, the cumulative resource scheduling problem is defined as a constraint satisfaction problem that is characterised by the set of activities \mathcal{V} and a cumulative resource with resource capacity R . The goal is to find a solution that assigns values from the domain to the start time variables S_i ($i \in \mathcal{V}$), so that the following conditions are satisfied.

$$\begin{aligned} est_i &\leq S_i \leq lst_i, & \forall i \in \mathcal{V} \\ \sum_{i \in \mathcal{V}: \tau \in [S_i, S_i + p_i)} r_i &\leq R & \forall \tau \end{aligned}$$

where τ ranges over the time periods considered. Note that this problem is NP-hard [5].

We shall tackle problems including cumulative resource scheduling using CP with nogood learning. In a CP solver, each variable $S_i, i \in \mathcal{V}$ has an initial domain of possible values $D^0(S_i)$ which is initially $[est_i, lst_i]$. The solver maintains a current domain D for all variables. CP search interleaves propagation with search. The constraints are represented by propagators that, given the current domain D , creates a new smaller domain D' by eliminating infeasible values. The current *lower* and *upper bound* of the domain $D(S_i)$ are denoted by $lb(S_i)$ and $ub(S_i)$, respectively. For more details on CP see e.g. [23].

For a learning solver we also represent the domain of each variable S_i using Boolean variables $\llbracket S_i \leq v \rrbracket, est_i \leq v < lst_i$. These are used to track the reasons for propagation and generate nogoods. For more details see [22]. We use the notation $\llbracket v \leq S_i \rrbracket, est_i < v \leq lst_i$ as shorthand for $\neg \llbracket S_i \leq v - 1 \rrbracket$, and treat $\llbracket v \leq S_i \rrbracket, v \leq est_i$ and $\llbracket S_i \leq v \rrbracket, v \geq lst_i$ as synonyms for *true*. Propagators in a learning solver must explain each reduction in the domain by building a clausal explanation using these Boolean variables.

3 TTEF Propagation

TTEF propagation was developed by Vilim [34]. The idea of TTEF propagation is to split the treatment of activities into a fixed and free part. The former results from the activities' compulsory part whereas the latter is the remainder. The fixed part of an activity i is characterised by the length of its *compulsory part* $p_i^{TT} := \max(0, ect_i - lst_i)$ and its fixed energy $e_i^{TT} := r_i \cdot p_i^{TT}$. The free part has a processing time $p_i^{EF} := p_i - p_i^{TT}$, a latest start time $lst_i^{EF} := lst_i + p_i^{TT}$, and a free energy of $e_i^{EF} := e_i - e_i^{TT}$. An illustration of this is shown in Figure 3.

TTEF propagation reasons about the energy available from the resource and energy required for the execution of activities in specific time windows. Let \mathcal{V}^{EF} be the set of activities with a non-empty free part $\{i \in \mathcal{V} \mid p_i^{EF} > 0\}$. The start and end times of these windows are determined by the earliest start and

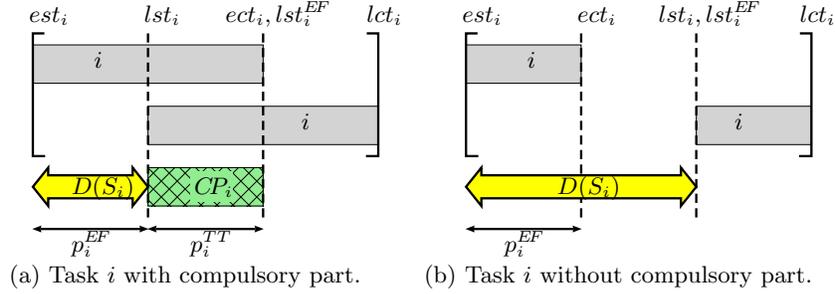


Fig. 3: A diagram illustrating an activity i when started at est_i or lst_i , and its possible range of start times, as well as the compulsory part CP_i (the hatched box), and the fixed and free parts of the processing time.

the latest completion times of two activities $\{a, b\} \subseteq \mathcal{V}^{EF}$. These time windows $[begin, end)$ are characterised by the so-called *task intervals* $\mathcal{V}^{EF}(a, b) := \{i \in \mathcal{V}^{EF} \mid est_a \leq est_i \wedge lct_i \leq lct_b\}$ where $a, b \in \mathcal{V}^{EF}$, $begin := est_a$, and $end := lct_b$.

It is not only the free energy of activities in the task interval $\mathcal{V}^{EF}(a, b)$ that is considered, but also the energy resulting from the compulsory parts in the time window $[est_a, lct_b)$. This energy is defined by $ttEn(a, b) := ttAfter[est_a] - ttAfter[lct_b]$ where $ttAfter[\tau] := \sum_{t \geq \tau} \sum_{i \in \mathcal{V}: lst_i \leq t < ect_i} r_i$ is the total energy of all compulsory parts occurring at time τ and after.

Furthermore, we also consider activities $i \in \mathcal{V}^{EF} \setminus \mathcal{V}^{EF}(a, b)$ in which a portion of their free part must be run within the time window as described in [34]. Suppose activity i starts after est_a , i.e., $est_a \leq est_i$. Then activity i 's free part consumes at least $r_i \cdot (lct_b - lst_i^{EF})$ energy units in $[est_a, lct_b)$ assuming $lst_i^{EF} < lct_b$. We define the energy contributed by such activities by $rsEn(a, b) := \sum_{i \in \mathcal{V}^{EF} \setminus \mathcal{V}^{EF}(a, b): est_a \leq est_i} r_i \cdot \max(0, lct_b - lst_i^{EF})$. Note that this is a special case of energetic reasoning that is cheaper to compute.

In summary, TTEF propagation considers three ways in which an activity i can contribute to energy consumption within a time window determined by a task interval $\mathcal{V}^{EF}(a, b)$. First, the free parts that must fully be executed in the time window; second, some free parts that must partially be run in the time window, and third, the compulsory parts that must lie within the time window; Thus, the considered length of an activity i is

$$p_i(a, b) := \begin{cases} p_i & i \in \mathcal{V}^{EF}(a, b) \\ \max(0, lct_b - lst_i) & i \notin \mathcal{V}^{EF}(a, b) \wedge est_a \leq est_i \\ \max(0, \min(lct_b, ect_i) - \max(est_a, lst_i)) & \text{others} \end{cases}$$

The considered energy consumption is $e_i(a, b) := r_i \cdot p_i(a, b)$ in the time window. An illustration of the three cases is shown in Fig. 4(a).

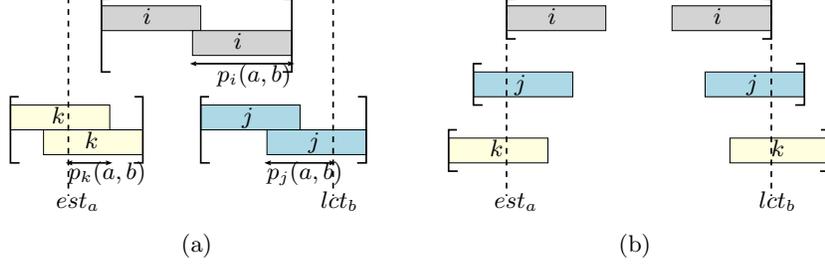


Fig. 4: (a) Diagram explaining the cases of energy contribution: task i is completely included in the task interval and its entire length is considered; task j starts after est_a , and the length from its latest start time to lct_b is considered; and task k has the intersection of its compulsory part with $[est_a, lct_b)$ considered. (b) Diagram illustrating how the bounds can be weakened in explanation, to still ensure that at least $p_l(a, b)$ for $l \in \{i, j, k\}$ is used with $[est_a, lct_b)$.

3.1 Explanation for the TTEF Consistency Check

The consistency check is one part of TTEF propagation that checks whether there is a resource overload in any task interval.

Proposition 1 (Consistency Check). *The cumulative resource scheduling problem is inconsistent if*

$$R \cdot (lct_b - est_a) - energy(a, b) < 0$$

where $energy(a, b) := \sum_{i \in \mathcal{V}^{EF}(a, b)} e_i^{EF} + ttEn(a, b) + rsEn(a, b)$.

This check can be done in $\mathcal{O}(l^2 + n)$ runtime, where $l = |\mathcal{V}^{EF}|$, if the resource profile is given.

The algorithm we use for the consistency check is shown in Alg. 1. It is different from that proposed by Vilim [34]. The main difference is that by iterating over the end times in decreasing order we can calculate the minimal available free energy $minAvail$ from the previous iteration. If the reduction in this free energy for the next iteration cannot make it negative we know that none of the task intervals in this iteration can lead to resource overload, and we can skip the entire set of task intervals. This optimization is highly useful high up in the search tree when there is little chance of resource overload. In preliminary experiments on instances with 30 activities, the number of checked task intervals could be reduced about 60% on average.

The algorithm iterates on each end time in decreasing order. For each end time the algorithm first checks if no propagation is possible with this end time (lines 5-6), and if so skips to the next. Otherwise it examines each possible start time, updating the free energy used E for the new task interval (lines 13-14), and calculating the energy available $avail$ in the task interval (line 15). If

Algorithm 1: TTEF consistency check.

Input: X an array of activities sorted in non-decreasing order of the earliest start time.
Input: Y an array of activities sorted in non-decreasing order of the latest completion time.

```
1  $end := \infty$ ;  $minAvail := \infty$ ;  
2 for  $y := n$  down to 1 do  
3    $b := Y[y]$ ;  
4   if  $lct_b = end$  then continue;  
5   if  $end \neq \infty$  and  $minAvail \neq \infty$  and  
    $minAvail \geq R \cdot (end - lct_b) - ttAfter[lct_b] + ttAfter[end]$  then  
6     continue;  
7    $end := lct_b$ ;  
8    $E := 0$ ;  $minAvail := \infty$ ;  
9   for  $x := n$  down to 1 do  
10     $a := X[x]$ ;  
11    if  $end \leq est_a$  then continue;  
12     $begin := est_a$ ;  
13    if  $lct_a \leq end$  then  $E := E + e_a^{EF}$ ;  
14    else if  $lst_a^{EF} < end$  then  $E := E + r_a \cdot (end - lst_a^{EF})$ ;  
15     $avail := R \cdot (end - begin) - E - ttEn(a, b)$ ;  
16    if  $avail < 0$  then  
17      explainOverload( $begin, end$ );  
18      return false;  
19    if  $avail < minAvail$  then  $minAvail := avail$ ;  
20 return true;
```

this is negative it explains the overload in the interval and returns *false*. If not it updates the minimum available energy and examines the next task interval (line 19).

A naïve explanation for a resource overload in the time window $[est_a, lct_b)$ only considers the current bounds on activities' start times S_i .

$$\bigwedge_{i \in \mathcal{V}: p_i(a,b) > 0} \llbracket est_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lct_i \rrbracket \rightarrow \perp$$

However, we can easily generalise this explanation by only ensuring that at least $p_i(a, b)$ time units are executed in the time window. This results in the following explanation.

$$\bigwedge_{i \in \mathcal{V}: p_i(a,b) > 0} \llbracket est_a + p_i(a, b) - p_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lct_b - p_i(a, b) \rrbracket \rightarrow \perp$$

Figure 4(b) shows how the explanations are weakened for the tasks shown in Figure 4(a). Note that this explanation expresses a resource overload with respect to energetic reasoning propagation which is more general than TTEF.

Let $\Delta := \text{energy}(a, b) - R \cdot (lct_b - est_a) - 1$. If $\Delta > 0$ then the resource overload has extra energy. We can use this extra energy to further generalise the explanation, by reducing the energy required to appear in the time window by up to Δ . For example, if $r_i \leq \Delta$ then the lower and upper bound on S_i can simultaneously be decreased and increased by a total amount in $\{1, 2, \dots, \min(\lfloor \Delta / r_i \rfloor, p_i(a, b))\}$ units without resolving the overload. If $r_i \cdot p_i(a, b) \leq \Delta$ then we can remove activity i completely from the explanation. In a greedy manner, we try to maximally widen the bounds of activities i where $p_i(a, b) > 0$, first considering activities with non-empty free parts. If Δ_i denotes the time units of the widening then it holds that $p_i(a, b) \geq \Delta_i \geq 0$ and $\sum_{i \in \mathcal{V}: p_i(a, b) > 0} \Delta_i \cdot r_i \leq \Delta$ and we create the following explanation.

$$\bigwedge_{i \in \mathcal{V}: p_i(a, b) - \Delta_i > 0} \llbracket est_a + p_i(a, b) - p_i - \Delta_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lct_b - p_i(a, b) + \Delta_i \rrbracket \rightarrow \perp$$

The last generalisation mechanism can be performed in different ways, e.g. we could widen the bounds of activities that were involved in many recent conflicts. By default we generalise the tasks in order. We experimented with different policies, but found any reasonable generalization policy was equally effective.

3.2 Explanation for the TTEF Start Times Propagation

Propagation on the lower and upper bounds of the start time variables S_i are symmetric; consequently we only present the case for the lower bounds' propagation. To prune the lower bound of an activity u , TTEF bounds propagation tentatively starts the activity u at its earliest start time est_u and then checks whether that causes a resource overload in any time window $[est_a, lct_b)$ ($\{a, b\} \subseteq \mathcal{V}^{EF}$). Thus, bounds propagation and its explanation are very similar to that of the consistency check.

The work of Vilim [34] considers four positions of u relative to the time window: *right* ($est_a \leq est_u < lct_b < ect_u$), *inside* ($est_a \leq est_u < ect_u \leq lct_b$), *through* ($est_u < est_a \wedge lct_b < ect_u$), and *left* ($est_u < est_a < ect_u \leq lct_b$). The first two of these positions correspond to edge-finding propagation and the last two to extended edge-finding propagation. In this work we fully consider the *right* and *inside* positions, i.e., $est_a \leq est_u$ (note that a could be u), and only opportunistically consider the *through* and *left* positions.

The calculation of a *right* or *inside* bounds update of u with respect to the time interval $[est_a, lct_b)$ are identical. Then, the bounds update rule is

$$R \cdot (lct_b - est_a) - \text{energy}(a, b, u) < 0 \rightarrow est_a + \left\lceil \frac{\text{rest}(a, b, u)}{r_u} \right\rceil \leq S_u \quad (1)$$

where $\text{energy}(a, b, u) := \text{energy}(a, b) - e_u(a, b) + r_u \cdot (\min(lct_b, ect_u) - est_u)$ and

$$\begin{aligned} \text{rest}(a, b, u) := & \text{energy}(a, b, u) - (R - r_u) \cdot (lct_b - est_a) \\ & - r_u \cdot (\min(lct_b, ect_u) - est_u) . \end{aligned}$$

The first two terms in the sum of $energy(a, b, u)$ give the energy consumption in the time window $[est_a, lct_b)$ of all considered activities except u , whereas the last term is the required energy of u in $[est_a, lct_b)$ if it is scheduled at est_u . The propagation, including explanation generation, can be performed in $\mathcal{O}(l^2 + k \cdot n)$ runtime, where $l = |\mathcal{V}^{EF}|$ and k is the number of bounds' updates, if the resource profile is given. Moreover, TTEF propagation does not necessarily consider each $u \in \mathcal{V}^{EF}$, but those only that maximise $\min(e_u^{EF}, r_u \cdot (lct_b - est_a)) - r_u \cdot \max(0, lct_b - lst_u^{EF})$ and satisfy $est_a \leq est_u$.

The pseudo-code for lower bounds propagation is shown in Algorithm 2. Similarly to the consistency check the task intervals are explored in an order using the latest completion time and all decreasing start times, before considering the next completion time.

If the global variable *opportunistic* is set to *true* then the algorithm first (lines 11-17) opportunistically searches for and records an upper bound change of the first task a if possible by using the calculated minimum available energy $minAvail$ in the task interval $[minBegin, lct_b)$ where $compIn(minBegin, end, a)$ (line 12) is the energy of the compulsory part of a in that task interval. This upper bound change is an extended edge finding propagation. It then updates the free energy used E for the new task interval, and updates the task u which requires maximum energy $enReqU$ in the new task interval (lines 18-23). It then calculates the energy available $avail$ in the task interval $[est_a, lct_b)$ (line 24), updates the interval with minimum available energy (lines 25-26) needed for the extended edge finding propagation, and records a lower bound change of the task u requiring most energy if this is possible (lines 27-33). Only after all task intervals are visited are the bounds actually changed by *updateBound*.

The procedure *explainUpdate(begin, end, v, oldbnd, newbnd)* explains the bound change of v to $newbnd$ where *oldbnd* is the old bound (the difference allows calculating which bound is being updated). A naïve explanation for a lower bound update from est_u to $newLB := \lceil rest(a, b, u) / r_u \rceil$ with respect to the time window $[est_a, lct_b)$ additionally includes the previous and new lower bound on the left and right hand side of the implication, respectively, in comparison to the naïve explanation for a resource overload.

$$\llbracket est_u \leq S_u \rrbracket \wedge \bigwedge_{i \in \mathcal{V} \setminus \{u\} : p_i(a,b) > 0} \llbracket est_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lst_i \rrbracket \rightarrow \llbracket newLB \leq S_u \rrbracket$$

As we discussed in the case of resource overload, we perform a similar generalisation for the activities in $\mathcal{V} \setminus \{u\}$, and for u we decrease the lower bound on the left hand side as much as possible so that the same propagation holds when u is executed at that decreased lower bound.

$$\begin{aligned} & \llbracket est_a + lct_b - newLB + 1 - p_u \leq S_u \rrbracket \wedge \\ & \bigwedge_{i \in \mathcal{V} \setminus \{u\} : p_i(a,b) > 0} \llbracket est_a + p_i(a, b) - p_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lct_b - p_i(a, b) \rrbracket \\ & \rightarrow \llbracket newLB \leq S_u \rrbracket \quad (2) \end{aligned}$$

Algorithm 2: TTEF lower bounds propagator on the start times.

Input: X an array of activities sorted in non-decreasing order of the earliest start time.

Input: Y an array of activities sorted in non-decreasing order of the latest completion time.

```
1 for  $i \in \mathcal{V}^{EF}$  do  $est'_i := est_i; lst'_i := lst_i;$ 
2  $end := \infty; k := 0;$ 
3 for  $y := n$  down to 1 do
4    $b := Y[y];$ 
5   if  $lct_b = end$  then continue;
6    $end := lct_b; E := 0; minAvail := \infty; minBegin := \infty; u := -\infty;$ 
    $enReqU := 0;$ 
7   for  $x := n$  down to 1 do
8      $a := X[x];$ 
9     if  $end \leq est_a$  then continue;
10     $begin := est_a;$ 
11    if opportunistic and  $minAvail \neq \infty$  and
       $minAvail < r_a \cdot (\min(end, lct_a) - \max(minBegin, lst_a^{EF}))$  then
12       $rest := minAvail + compln(minBegin, end, a);$ 
13       $ubA := minBegin + \lceil rest/r_a \rceil - p_a;$ 
14      if  $lst'_a > ubA$  then
15         $expl := explainUpdate(minBegin, end, a, lst'_a, ubA);$ 
16         $Update[++k] := (a, ub, ubA, expl);$ 
17         $lst'_a := ubA;$ 
18      if  $lct_a \leq end$  then  $E := E + e_a^{EF};$ 
19      else
20         $enIn := r_a \cdot \max(0, end - lst_a^{EF});$ 
21         $E := E + enIn;$ 
22         $enReqA := \min(e_a^{EF}, r_a \cdot (end - est_a)) - enIn;$ 
23        if  $enReqA > enReqU$  then  $u := a; enReqU := enReqA;$ 
24         $avail := R \cdot (end - begin) - E - ttEn(a, b);$ 
25        if opportunistic and  $avail < minAvail$  then
26           $minAvail := avail; minBegin := begin;$ 
27        if  $enReqU > 0$  and  $avail - enReqU < 0$  then
28           $rest := E - avail - r_a \cdot \max(0, end - lst_a);$ 
29           $lbU := begin + \lceil rest/r_u \rceil;$ 
30          if  $est'_u < lbU$  then
31             $expl := explainUpdate(begin, end, u, est'_u, lbU);$ 
32             $Update[++k] := (u, lb, lbU, expl);$ 
33             $est'_u := lbU;$ 
34 for  $z := 1$  to  $k$  do  $updateBound(Update[z]);$ 
```

Again this more general explanation expresses the energetic reasoning propagation and the bounds of activities in $\{i \in \mathcal{V} \setminus \{u\} \mid p_i(a, b) > 0\}$ can further be generalised in the same way as for a resource overload. But here the available energy units Δ for widening the bounds is $rest(a, b, u) - r_u \cdot (newLB - 1) + 1$.

Table 1: Specifications of the benchmark suites.

suite	sub-suites	#inst	#act	p_i	#res	notes
AT [3]	ST27/ST51/ST103	48 each	25/49/101	1–12	6 each	
PSPLib [16]	J30 [17]/J60/J90	480 each	30/60/90	1–10	4 each	
	J120	600	30	1–10	4	
BL [4]	BL20/BL25	20 each	20/25	1–6	3 each	
PACK [8]		55	15–33	1–19	2–5	
KSD15_D [18]		480	15	1–250	4	based on J30
PACK_D [18]		55	15–33	1–1138	2–5	based on PACK

Hence, $0 \leq \Delta < r_u$ indicate that the explanation only can further be generalised a little bit. We perform this generalisation as for the overload case.

4 Experiments on Resource-constrained Project Scheduling Problems

We carried out extensive experiments on RCPSP instances comparing our solution approach using both time-table and/or TTEF propagation. We compare the obtained results on the lower bounds of the makespan with the best known so far. Detailed results are available at <http://www.cs.mu.oz.au/~pjs/rcpsp>.

We used six benchmark suites for which an overview is given in Table 1 where #inst, #act, p_i , and #res are the number of instances, number of activities, range of processing times, and number of resources, respectively. The first two suites are highly disjunctive, while the remainder are highly cumulative.

The experiments were run on a X86-64 architecture running GNU/Linux and a Intel(R) Core(TM) i7 CPU processor at 2.8GHz. The code was written in Mercury [30] using the G12 Constraint Programming Platform [31].

We model an instance as in [26] using global cumulative constraints `cumulative` and difference logic constraints ($S_i + p_i \leq S_j$), resp. In addition, between two activities i, j in disjunction, *i.e.*, two activities which cannot concurrently run without overloading some resource, the two half-reified constraints [9] $b \rightarrow S_i + p_i \leq S_j$ and $\neg b \rightarrow S_j + p_j \leq S_i$ are posted where b is a Boolean variable.

We run cumulative constraint propagation using different phases:

- (a) time-table consistency check in $\mathcal{O}(n + p \log p)$ runtime,
- (b) TTEF consistency check in $\mathcal{O}(l^2 + n)$ runtime as defined in Section 3.1,
- (c) time-table bounds' propagation in $\mathcal{O}(l \cdot p + k \cdot \min(R, n))$ runtime, and
- (d) TTEF bounds' propagation in $\mathcal{O}(l^2 + k \cdot n)$ runtime as defined in Section 3.2

where k, l, n, p are the numbers of bounds' updates, unfixed activities, all activities, and height changes in the resource profile, respectively. Note that in our setup phase (d) TTEF bounds' propagation does not take into account the bounds' changes of the phase (c) time-table bounds' propagation. For the experiments, we consider four settings of the `cumulative` propagator: `tt` executes

phases (a) and (c), **ttef(c)** (a–c), **ttef** (a–d) with *opportunistic* set to *false*, and **ttef+** (a–d) with *opportunistic* set to *true*. Each phase is run once for each execution of the propagator. The propagator is itself run multiple times in the usual propagation fixpoint calculation. Note that phases (c) and (d) are not run if either phase (a) or (b) detects inconsistency.

4.1 Upper Bound Computation

For solving RCPSP we use the same branch-and-bound algorithm as we used in [26], but here we limit ourselves to the search heuristic **HOTRESTART** which was the most robust one in our previous studies [25,26]. It executes an adapted search of [4] using serial scheduling generation for the first 500 choice points and, then, continues with an activity based search (a variant of **VSIDS** [20]) on the Boolean variables representing a lower part $x \leq v$ and upper part $v < x$ of the variable x 's domain where x is either a start time or the makespan variable and v a value of x 's initial domain. Moreover, it is interleaved with a geometric restart policy [35] on the number of node failures for which the restart base and factor are 250 failures and 2.0, respectively. The search was halted after 10 minutes.

The results are given in Tables 2 and 3. For each benchmark suite, the number of solved instances (**#svd**) is given. The column **cmpr(a)** shows the results on the instances solved by all methods, where a is the number of such instances. The left entry in that column is the average runtime on these instances in seconds, and the right entry is the average number of failures during search. The entries in column **all(a)** have the same meaning, but here all instances are considered where a is the total number of instances. For unsolved instances, the number of failures after 10 minutes is used.

Table 2 shows the results on the highly disjunctive RCPSPs. As expected, the stronger propagation (**ttef(c)**, **ttef**) reduces the search space overall in comparison to **tt**, but the average runtime is higher by a factor of about 5%–70% for **ttef(c)** and 50%–100% for **ttef**. Interestingly, **ttef(c)** and **ttef** solved respectively 1 and 2 more instances on **J60** and closed the instance **j120_1.1** on **J120** which has an optimal makespan 105. This makespan corresponds to the best known upper bound. However, the stronger propagation does not generally pay off for a CP solver with nogood learning on highly disjunctive RCPSPs. The opportunistic extended edge finding **ttef+** does not pay off on the highly disjunctive problems.

Table 3 presents the results on highly cumulative RCPSPs which clearly shows the benefit of **TTEF** propagation, especially on **BL** for which **ttef(c)** and **ttef** reduce the search space and the average runtime by a factor of 8, and **PACK** for which they solved 23 instances more than **tt**. On **PACK_D**, **ttef(c)** is about 50% faster on average than **tt** while **ttef** is slightly slower on average than **tt**. The opportunistic extended edge finding is beneficial on the the hardest highly cumulative problems **PACK** and **PACK_D**. No conclusion can be drawn on **KSD15_D** because the instances are too easy for LCG solvers.

Table 2: UB results on highly disjunctive RCPSPs.

	J30					J60				
	#svd	cmpr(480)		all(480)		#svd	cmpr(429)		all(480)	
tt	480	0.12	1074	0.12	1074	430	1.82	5798	64.25	93164
ttef(c)	480	0.20	1103	0.20	1103	431	2.00	4860	64.39	80845
ttef	480	0.23	991	0.23	991	432	3.04	5191	64.87	62534
ttef+	480	0.28	1045	0.28	1045	430	3.58	5172	66.63	58577
	J90					J120				
	#svd	cmpr(398)		all(480)		#svd	cmpr(278)		all(600)	
tt	400	4.01	7540	104.09	132234	283	8.92	13636	322.35	398941
ttef(c)	400	4.90	7263	105.69	104297	282	11.13	14387	324.73	297562
ttef	400	6.57	7277	106.66	72402	283	13.30	11881	324.66	186597
ttef+	398	6.05	6165	107.52	70436	282	12.53	11016	325.41	168803
	AT									
	#svd	cmpr(129)		all(144)						
tt	132	8.90	19997	66.22	87226					
ttef(c)	130	9.36	16466	69.41	72056					
ttef	129	13.55	17239	74.60	63554					
ttef+	129	15.82	18060	76.68	61665					

Table 3: UB results on highly cumulative RCPSPs.

	BL				PACK					
	#svd	cmpr(40)		all(40)		#svd	cmpr(16)		all(55)	
tt	40	0.16	2568	0.16	2568	16	77.65	245441	447.69	699615
ttef(c)	40	0.02	370	0.02	370	39	37.22	122038	186.79	292101
ttef	40	0.02	269	0.02	269	39	44.44	105751	188.23	257747
ttef+	40	0.06	484	0.06	484	39	36.42	95704	185.69	262506
	KSD15_D					PACK_D				
	#svd	cmpr(480)		all(480)		#svd	cmpr(37)		all(55)	
tt	480	0.01	26	0.01	26	37	32.72	42503	218.26	184293
ttef(c)	480	0.01	26	0.01	26	37	23.96	32916	212.37	170301
ttef	480	0.01	26	0.01	26	37	36.93	37004	221.11	157015
ttef+	480	0.13	26	0.13	26	37	23.13	28489	212.14	152950

4.2 Lower Bound Computation

The lower bound computation tries to solve RCPSPs in a destructive way by converging to the optimal *makespan* from below, *i.e.*, it repeatedly proves that there exists no solution for current makespan considered and continues with an incremented *makespan* by 1. If a solution is found then it is the optimal one. For these experiments we use the search heuristic HOTSTART as we did in [25,26]. This heuristic is HOTRESTART (as described earlier) but no restart. We used the same parameters as for HOTRESTART. For the starting *makespan*, we choose the best known lower bounds on J60, J90, and J120 recorded in the PSPLib at <http://129.187.106.231/psplib/> and [34] at <http://vilim.eu/>

Table 4: LB results on AT, PACK, and PACK_D

	AT (12)	PACK (16)	PACK_D (18)
ttef(c)	5/4/3 +52	0/4/12 +100	0/7/11 +632
ttf	7/2/3 +44	1/4/11 +101	2/6/10 +618
ttf+	7/2/3 +45	0/2/14 +105	3/5/10 +638

Table 5: LB results on J60, J90, and J120

		J60			J90					J120									
		+1	+2	+3	+1	+2	+3	+4	+5	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
1 min	ttef(c)	4	1	-	12	1	-	-	-	27	8	4	-	-	-	2	-	-	-
	ttf	7	5	-	25	14	3	1	-	90	20	10	5	2	-	-	2	-	-
	ttf+	10	5	-	29	12	3	1	-	83	20	7	8	2	-	-	1	1	-
10 mins	ttef(c)	21	2	-	25	7	-	-	-	68	16	4	4	2	-	-	1	1	-
	ttf	13	6	3	35	17	6	3	1	116	39	9	9	4	1	-	-	1	1
	ttf+	19	7	3	33	17	6	2	1	111	35	9	9	5	1	-	-	1	1

`petr/cpaior2011-results.txt`. On the other suites, the search starts from *makespan* 1. Due to the tighter *makespan*, it is expected that the TTEF propagation will perform better than for upper bound computation on the highly disjunctive instances. The search was cut off at 10 minutes as in [25,26].

Table 4 shows the results on AT, PACK, and PACK_D restricted to the instances that none of the methods could solve using the upper bound computation, The number of instances for each class is shown in parentheses in the header. An entry $a/b/c + d$ for method x means that x achieved respectively a -times, b -times and c -times a worse, the same and a better lower bound than tt , while the $+d$ is the sum of lower bounds' differences of method x to tt . On PACK and PACK_D, ttef(c) and ttf clearly perform better than tt . On the highly disjunctive instances in AT, ttef(c) and tt are almost balanced whereas tt could generate better lower bounds on more instances than ttef. The lower bounds' differences on AT are dominated by the instance `st103_4` for which ttef(c) and ttf retrieved a lower bound improvement of 54 and 53 time periods with respect to tt . Opportunistic extended edge finding ttf+ is beneficial on the highly disjunctive benchmarks, but can only better tt on PACK.

The more interesting results are presented in Tab. 5 because the best lower bounds are known for all the remaining open instances (48, 77, 307 in J60, J90, J120).² An entry in a column $+d$ shows the number of instances for that the corresponding method could improve the lower bound by d time periods. On these instances, we run at first the experiments with a runtime limit of one minute as it was done in the experiments for TTEF propagation in [34] but he used a CP solver without nogood learning. tt could not improve any lower

² Note that the PSPLib still lists the instances `j60_25_5`, `j90_26_5`, `j120_8_3`, `j120_48_5`, and `j120_35_5` as open, but we closed the first four ones in [26] and [19] closed the last one.

bound because its corresponding results are already recorded in the PSPLib. `ttef(c)`, `ttef`, and `ttef+` improved the lower bounds of 59, 183 and 173 instances, respectively, which is about 13.7%, 42.3% and 40.0% of the open instances. Although, the experiments in [34] were run on a slower machine³ the results confirm the importance of nogood learning. For the experiments with 10 minutes runtime, `ttef(c)`, `ttef` and `ttef+` could improve the lower bounds of more instances, namely 151, 264 and 258 instances, respectively, which is about 35.0%, 61.1% and 59.7%. Again for the highly disjunctive instances the opportunistic extended edge finding does not pay off, although interestingly it gives the best results on J60. Moreover, 3, 1, and 1 of the remaining open instances on J60, J90, and J120, respectively, could be solved optimally. See App. A for the listing of the closed instances and the new lower bounds.

5 Conclusion and Outlook

We present explanations for the recently developed TTEF propagation of the global `cumulative` constraint for lazy clause generation solvers. These explanations express an energetic reasoning propagation which is a stronger propagation than the TTEF one.

Our implementation of this propagator was compared to time-table propagation in lazy clause generation solvers on six benchmark suites. The preliminary results confirm the importance of energy-based reasoning on highly disjunctive RCPSPs for CP solvers with nogood learning.

Moreover, our approach with TTEF propagation was able to close six open instances. It also improves the best known lower bounds for 264 of the remaining 432 remaining open instances on RCPSPs from the PSPLib.

In the future, we want to integrate the extended edge-finding propagation into TTEF propagation as it was originally proposed in [34], to perform experiments on cutting and packing problems, and to study different variations of explanations for TTEF propagation. Furthermore, we want to look at a more efficient implementation of the TTEF propagation as well as an implementation of energetic reasoning.

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. This work was partially supported by Asian Office of Aerospace Research and Development grant 10-4123.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. *Mathematical Programming Computation* 1, 1–41 (2009)

³ Intel(R) Core(TM)2 Duo CPU T9400 on 2.53GHz

2. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling* 17(7), 57–73 (1993)
3. Alvarez-Valdés, R., Tamarit, J.M.: *Advances in Project Scheduling*, chap. Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis, pp. 113–134. Elsevier (1989)
4. Baptiste, P., Le Pape, C.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints* 5(1-2), 119–139 (2000)
5. Baptiste, P., Le Pape, C., Nuijten, W.: Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research* 92, 305–333 (1999)
6. Baptiste, P., Le Pape, C., Nuijten, W.: *Constraint-Based Scheduling*. Kluwer Academic Publishers, Norwell, MA, USA (2001)
7. Berthold, T., Heinz, S., Lübbecke, M., Möhring, R., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In: Lodi, A., Milano, M., Toth, P. (eds.) *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Lecture Notes in Computer Science, vol. 6140, pp. 313–317. Springer Berlin / Heidelberg (2010)
8. Carlier, J., Néron, E.: On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research* 149(2), 314–324 (2003)
9. Feydy, T., Somogyi, Z., Stuckey, P.J.: Half reification and flattening. In: Lee, J.H.M. (ed.) *Proceedings of Principles and Practice of Constraint Programming – CP 2011*. Lecture Notes in Computer Science, vol. 6876, pp. 286–301. Springer (2011)
10. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent [11], pp. 352–366
11. Gent, I.P. (ed.): *Proceedings of Principles and Practice of Constraint Programming – CP 2009*, Lecture Notes in Computer Science, vol. 5732. Springer Berlin / Heidelberg (2009)
12. Heinz, S., Schulz, J.: Explanations for the cumulative constraint: An experimental study. In: Pardalos, P.M., Rebennack, S. (eds.) *Proceedings of Experimental Algorithms – SEA 2011*. Lecture Notes in Computer Science, vol. 6630, pp. 400–409. Springer Berlin / Heidelberg (2011)
13. Jussien, N.: The versatility of using explanations within constraint programming. Research Report 03-04-INFO, École des Mines de Nantes, Nantes, France (2003)
14. Jussien, N., Barichard, V.: The PaLM system: explanation-based constraint programming. In: *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*. pp. 118–133. Singapore (2000)
15. Katsirelos, G., Bacchus, F.: Generalized nogoods in CSPs. In: Veloso, M.M., Kambhampati, S. (eds.) *Proceedings on Artificial Intelligence – AAAI 2005*. pp. 390–396. AAAI Press / The MIT Press (2005)
16. Kolisch, R., Sprecher, A.: PSPLIB – A project scheduling problem library. *European Journal of Operational Research* 96(1), 205–216 (1997)
17. Kolisch, R., Sprecher, A., Drexel, A.: Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science* 41(10), 1693–1703 (1995)
18. Koné, O., Artigues, C., Lopez, P., Mongeau, M.: Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research* 38(1), 3–13 (2011)

19. Liess, O., Michelon, P.: A constraint programming approach for the resource-constrained project scheduling problem. *Annals of Operations Research* 157(1), 25–36 (Jan 2008)
20. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: *Proceedings of Design Automation Conference – DAC 2001*. pp. 530–535. ACM, New York, NY, USA (2001)
21. Nuijten, W.P.M.: *Time and Resource Constrained Scheduling*. Ph.D. thesis, Eindhoven University of Technology (1994)
22. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. *Constraints* 14(3), 357–391 (2009)
23. Schulte, C., Stuckey, P.J.: Efficient constraint propagation engines. *ACM Transactions on Programming Languages and Systems* 31(1), Article No. 2 (2008)
24. Schutt, A.: *Improving Scheduling by Learning*. Ph.D. thesis, The University of Melbourne (2011), <http://repository.unimelb.edu.au/10187/11060>
25. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Why cumulative decomposition is not as bad as it sounds. In: *Gent [11]*, pp. 746–761
26. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. *Constraints* 16(3), 250–282 (2011)
27. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. *Journal of Scheduling* pp. 1–17 (2012), online first
28. Schutt, A., Stuckey, P., Verden, A.: Optimal carpet cutting. In: Lee, J. (ed.) *Principles and Practice of Constraint Programming – CP 2011*. *Lecture Notes in Computer Science*, vol. 6876, pp. 69–84. Springer Berlin / Heidelberg (2011)
29. Schutt, A., Wolf, A.: A new $\mathcal{O}(n^2 \log n)$ not-first/not-last pruning algorithm for cumulative resource constraints. In: Cohen, D. (ed.) *Proceedings of Principles and Practice of Constraint Programming – CP 2010*. *Lecture Notes in Computer Science*, vol. 6308, pp. 445–459. Springer Berlin / Heidelberg (2010), 10.1007/978-3-642-15396-9_36
30. Somogyi, Z., Henderson, F., Conway, T.: The execution algorithm of Mercury, an efficient purely declarative logic programming language. *The Journal of Logic Programming* 29(1–3), 17–64 (1996)
31. Stuckey, P.J., García de la Banda, M.J., Maher, M.J., Marriott, K., Slaney, J.K., Somogyi, Z., Wallace, M.G., Walsh, T.: The G12 project: Mapping solver independent models to efficient solutions. In: Gabbrielli, M., Gupta, G. (eds.) *Proceedings of Logic Programming – ICLP 2005*. *Lecture Notes in Computer Science*, vol. 3668, pp. 9–13. Springer Berlin / Heidelberg (Oct 2005)
32. Vilím, P.: Computing explanations for the unary resource constraint. In: Barták, R., Milano, M. (eds.) *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2005*. *Lecture Notes in Computer Science*, vol. 3524, pp. 396–409. Springer Berlin / Heidelberg (2005)
33. Vilím, P.: Edge finding filtering algorithm for discrete cumulative resources in $\mathcal{O}(kn \log n)$. In: *Gent [11]*, pp. 802–816
34. Vilím, P.: Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg, T., Beck, J. (eds.) *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2011*. *Lecture Notes in Computer Science*, vol. 6697, pp. 230–245. Springer Berlin / Heidelberg (2011)
35. Walsh, T.: Search in a small world. In: *Proceedings of Artificial intelligence – IJCAI 1999*. pp. 1172–1177. Morgan Kaufmann (1999)

A Closed Instances and New Lower Bounds on PSPLib

From the open instances, we closed the instances 9.3 (100), 9.9 (99), 25.10 (108) on J60, 5.6 (86) on J90, and 1.1 (105), 8.6 (85) on J120 where the number in brackets shows the optimal makespan. We computed new lower bounds on the remaining open instances from the PSPLib. Tables 6–8 list these new lower bounds where the column “inst” shows the name of the instance and the column “LB” the corresponding new lower bound.

Table 6: New lower bounds on J60.

inst	LB	inst	LB	inst	LB	inst	LB	inst	LB	inst	LB
9.1	85	9.5	81	9.6	106	9.7	103	9.8	95	9.10	89
13.2	103	13.3	84	13.4	98	13.7	82	13.8	115	13.9	96
25.2	96	25.4	106	25.6	106	29.1	97	29.3	115	29.6	144
29.8	97	29.10	112	41.3	90	41.10	106	45.1	90	45.5	100
								45.10	104		

Table 7: New lower bounds on J90.

inst	LB	inst	LB	inst	LB	inst	LB	inst	LB	inst	LB
5.3	84	5.5	109	5.7	106	5.8	97	5.9	114	5.10	95
9.3	98	9.4	120	9.5	127	9.6	113	9.7	103	9.8	111
9.10	105	13.2	119	13.3	105	13.5	109	13.7	116	13.8	113
13.10	114	21.7	106	21.8	108	25.1	117	25.2	122	25.3	113
25.5	110	25.6	113	25.7	123	25.8	133	25.9	98	25.10	119
29.2	122	29.4	139	29.6	117	29.7	160	29.8	146	29.9	120
37.2	114	41.1	129	41.2	154	41.3	149	41.4	142	41.5	116
41.7	146	41.8	148	41.9	110	41.10	144	45.1	143	45.2	138
45.4	126	45.5	164	45.6	163	45.7	129	45.8	150	45.9	145
46.9	86									45.10	157

B Best Lower and Upper Bounds Retrieved

For comparison with future papers on RCPSP, Tables 9–11 show the best lower and upper bounds for AT, PACK, and PACK.D retrieved by one of the methods tt, ttef(c), and ttef. The column “inst” shows the instance name and the column “LB/UB” the corresponding lower and upper bound. If these bounds are equal then only one number is given.

Table 8: New lower bounds on J120.

inst	LB	inst	LB	inst	LB	inst	LB	inst	LB	inst	LB		
6.1	136	6.2	127	6.5	118	6.6	143	6.8	142	6.9	151	6.10	158
7.1	99	7.3	98	7.4	107	7.6	117	7.7	114	7.8	93	7.9	87
7.10	112	8.2	102	8.5	100	8.9	91	8.10	92	9.4	85	11.1	159
11.2	147	11.3	189	11.4	182	11.5	195	11.6	192	11.7	152	11.8	154
11.10	165	12.1	127	12.2	112	12.4	122	12.5	155	12.6	116	13.1	124
13.3	116	13.4	109	13.6	96	13.9	84	14.2	91	14.5	94	14.7	90
16.1	181	16.3	221	16.4	191	16.6	195	16.8	183	17.5	124	17.6	134
18.8	102	18.9	89	18.10	97	26.1	155	26.2	160	26.3	160	26.4	161
26.5	140	26.6	179	26.7	148	26.8	168	26.9	162	26.10	179	27.1	107
27.2	111	27.3	142	27.4	105	27.5	106	27.6	136	27.7	122	27.8	136
27.9	122	27.10	111	28.1	106	31.1	182	31.2	179	31.3	160	31.4	195
31.5	187	31.6	183	31.7	195	31.8	177	31.9	176	31.10	202	32.1	144
32.2	123	32.5	133	32.6	122	32.8	132	33.1	105	33.2	107	33.3	102
33.4	107	33.5	134	33.8	107	33.9	109	34.1	76	34.2	103	34.3	100
34.5	102	36.1	201	36.3	219	36.5	214	36.7	196	36.9	203	37.2	141
37.5	195	37.8	169	37.9	138	38.1	105	38.2	119	38.4	138	38.6	119
38.7	103	38.10	137	39.2	105	40.1	80	42.1	107	46.1	173	46.2	187
46.3	163	46.5	140	46.7	162	46.9	157	46.10	184	47.1	130	47.3	122
47.4	122	47.5	126	47.6	133	47.7	114	47.8	126	47.10	130	48.4	123
51.1	186	51.2	200	51.3	197	51.4	200	51.6	193	51.7	186	51.8	187
51.9	192	51.10	201	52.1	161	52.2	172	52.3	126	52.4	158	52.5	158
52.6	186	52.7	143	52.8	149	52.9	143	52.10	134	53.1	139	53.2	110
53.4	138	53.5	110	53.6	101	53.8	135	53.10	126	54.1	102	54.5	107
54.6	105	54.8	100	54.9	105	57.1	174	57.2	151	57.3	176	57.5	170
57.6	177	57.7	156	57.9	158	58.2	122	58.3	117	58.4	139	58.5	116
58.6	135	58.7	143	58.8	126	58.9	127	59.5	104	59.6	112	59.8	107
59.9	117	59.10	128	60.3	88	60.7	91						

Table 11: Lower and upper bounds for PACK_D.

inst	LB/UB								
001	612	002	745/747	003	624/625	004	1381	005	983
006	1119	007	1082	008	1274	009	1594/1951	010	1216
011	940	012	1234/1241	013	829	014	1565	015	1198
016	1783/1813	017	1641/1651	018	1462/1480	019	1527/1542	020	1661
021	1606	022	1787	023	1092	024	1625	025	2061/2147
026	926	027	1789/1793	028	1899/1962	029	1233	030	597
031	1949	032	2943	033	3390	034	2371	035	2305
036	2175/2191	037	3328/3614	038	2180	039	2730/2734	040	3024
041	679	042	838	043	2439	044	3050	045	2712
046	3243/3277	047	2740/2745	048	2446	049	675	050	2687/2716
051	838	052	2253	053	2521	054	2750	055	2628