Maximising the Net Present Value for Resource-constrained Project Scheduling

Andreas Schutt[1], Geoffrey Chu[1], Peter J. Stuckey[1], and Mark G. Wallace[2]

[1] National ICT Australia, Department of Computing and Information Systems, The University of Melbourne, Victoria 3010, Australia
{andreas.schutt,geoffrey.chu,peter.stuckey}@nicta.com.au
[2] Faculty of Information Technology, Monash University, Victoria 3100, Australia
mark.wallace@monash.edu

Abstract. The Resource-constrained Project Scheduling Problem (RCPSP), in which a schedule must obey the resource constraints and the precedence constraints between pairs of activities, is one of the most studied scheduling problems. An important variation of the problem (RCPSPDC) is to find a schedule which maximises the net present value (discounted cash flow), when every activity has a given cash flow associated with it. Given the success of lazy clause generation (LCG) approaches to solve RCPSP with and without generalised precedence relations it seems worthwhile investigating LCG's use on RCPSPDC. To do so, we must construct propagators for the net-present-value constraint that explain their propagation to the LCG solver. In this paper we construct three different propagators for net-present-value constraints, and show how they can be used to rapidly solve RCPSPDC.

1 Introduction

The Resource-constrained Project Scheduling Problem (RCPSP) is one of the most studied scheduling problems. It consists of scarce resources, activities and precedence relations between pairs of activities where a precedence relation expresses that an activity can be run after the execution of its preceding activity is finished. Each activity requires some units of resources during their execution. The aim is to build a schedule that satisfies the resource and precedence constraints. Here, we assume renewable resources (*i.e.*, their supply is constant during the planning period) and non-preemptive activities (*i.e.*, once started their execution cannot be interrupted). Usually the objective in solving RCPSP problems is to minimise the makespan, *i.e.*, to complete the entire project in the minimum total time. But another important objective is to maximise the net present value (*npv*), because it better captures the financial aspects of the project. In this formulation each activity has an associated cash flow which may be a payment (negative cash flow) or a receipt (positive cash flow). These cash flows are discounted with respect to a discount rate, which makes it, in general, beneficial for the *npv* to execute activities with a positive (negative) cash flow as early (late) as possible. The problem herein is to maximise the *npv* for a given

RCPSP problem. We denote the problem RCPSPDC, *i.e.*, RCPSP with discounted cash flows. It is classified as $m, 1|cpm, \delta_n, c_j|npv$ [10] or $PS|prec|\sum C_j^F \beta^{C_j}$ [3].

Optimisation of the net present value for project scheduling problems was first introduced in [17] which also gave some characteristics for a non-linear program to maximise the net present value of resource-unconstrained project scheduling problems. This problem equates to RCPSPDC without resource constraints and is—in contrast to RCPSPDC—polynomial time solvable. In the remainder, we denote this resource-unconstrained problem by PSPDC.

Normally, methods for RCPSPDC relax the problem to PSPDC for obtaining an upper bound on the *npv*. Different algorithms for PSPDC with and without generalised precedence relations have been proposed where such relations allow arbitrary distances between start times of activities. A good overview can be found in [23]. These algorithms are usually based on linear programs or graph algorithms. Two important works are by Grinold [7] which shows that PSPDC can be transformed to a linear program, and by Demeulemeester *et al.* [6] which presents an efficient graph algorithm for PSPDC.

Different complete and incomplete methods for RCPSPDC with or without generalised precedence relations have been proposed, the reader is referred to [8] for a more extensive literature overview of solution approaches for RCPSPDC and other variants or extensions of RCPSP.

Most complete methods for RCPSPDC use a branch-and-bound algorithm to maximise the *npv*. The algorithm is interleaved with dominance rules and an upper bound computation on the *npv* by relaxing the current RCPSPDC to PSPDC. The approaches in [11,27,15] are based on the branch-and-bound algorithm in [4,5] for RCPSP. This algorithm is based on a scheduling generation scheme which resolves resource conflicts by adding new precedence constraints between activities in conflict. The method in [27] improves upon the one in [11] whereas the work [15] considers RCPSPDC with generalised precedence relations.

The only method, we are aware of, that uses constraint programming techniques to solve project scheduling problem with discounted cash flows, is presented in [12]. This method tackles a variant of RCPSPDC, called construction RCPSPDC, in which the resource capacities are decision variables and are associated with cost. However, no details are given whether the *npv* objective is used for bounds propagation on start time variables as we do.

A state of the art method for solving RCPSP and its variations is constraint programming with learning, or lazy clause generation (LCG) [16]. LCG combines the power of complex `cumulative` constraint propagation with the learning of where the previous search failed, and can solve many RCPSPs faster than all other known methods [22,21]. It seems worthwhile to explore whether we can use the same technology to solve RCPSPDC.

To do so we must create a propagator for the net-present-value constraint, and, *critically* for its use in LCG, extend the propagator to *explain* its propagations. This is a requirement for using the propagator in an LCG system and gaining the most out of learning.

The contributions of this paper are: ***(i)*** We define three propagators for the net-present-value constraint: a simple propagator, a direct propagator that takes into account precedence relations, and a linear programming based approach (using the formulation of [7]) that also considers precedence relations. ***(ii)*** We give experimental results that show that the LCG approach to solve RCPSPDC is highly competitive, and can find optimal solutions for many instances.

2 Resource-constrained Project Scheduling Problem with Discounted Cash Flows

The RCPSPDC is defined as follows: A set of activities $\mathcal{V} = \{1, \ldots, n\}$ is subject to precedence relations in $\mathcal{P} \subset \mathcal{V}^2 \times \mathbb{Z}$ between two activities, and scarce resources in $\mathcal{R}$. The goal is to find a schedule $s = (s_i)_{i \in \mathcal{V}}$ that respects the precedence and resource constraints, and maximises the *npv* $\sum_{i=1}^{n} e^{-\alpha s_i} c_i$ where α is the discount rate, s_i is the *start time* of the activity i, and c_i is the discounted cash flow for activity i to start at time 0.[1]

Each activity i has a finite *duration* p_i and requires (non-negative) r_{ik} units of resource k, $k \in \mathcal{R}$ for its execution where r_{ik} is the *resource requirement* or *usage* of activity i for resource k. A resource $k \in \mathcal{R}$ has a constant *resource capacity* R_k over the planning period which cannot be exceeded at any point in time. The *planning period* is given by $[0, t_{max})$ where t_{max} is the maximal project duration. Each resource k is modeled by a single cumulative constraint [1]: $\texttt{cumulative}(s, p, [r_{ik} | i \in \mathcal{V}], R_k)$.

In this paper we consider precedence relations $(i, j, d_{ij}) \in \mathcal{P}$ between the activities i and j are subject to the constraint $s_i + d_{ij} \leq s_j$. We restrict attention to precedence constraints where $d_{ij} \geq 0$ and the graph $\mathcal{P}$ is acyclic. This is the standard form of RCPSPDC and extending to generalised precedence relations does not fundamentally change the algorithms required.

The RCPSPDC problem can be stated as follows:

$$\text{maximise} \quad \sum_{i=1}^{n} e^{-\alpha s_i} c_i \tag{1}$$

$$\text{subject to} \quad s_i + d_{ij} \leq s_j \qquad \forall (i, j, d_{ij}) \in \mathcal{P}, \tag{2}$$

$$\texttt{cumulative}(s, p, [r_{ik} | i \in \mathcal{V}], R_k) \qquad \forall k \in \mathcal{R}, \tag{3}$$

$$0 \leq s_i \leq t_{max} - p_i \qquad \forall i \in \mathcal{V}. \tag{4}$$

The objective is to maximise the net present value (Eq. 1) which is subject to the precedence constraints (Eq. 2), and the resource constraints (Eq. 3). All start times must be non-negative and all activities must be scheduled in the planning period (Eq. 4).

We shall tackle this problem using constraint programming (CP) with learning. In a CP solver each variable $s_i, 1 \leq i \leq n$ has an initial domain of possible

[1] Note that much of the work on PSPDC and RCPSPDC considers discounted cash flows at the end of activities, and find solutions to end time variables, here we use start times. The discounted cash flow at the end of activities is $c_i e^{\alpha p_i}$.

values $D^0(s_i)$ which is initially $[0, t_{max} - p_i]$. The solver maintains a current domain D for all variables. The 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. For more details on CP see e.g. [18].

For a learning solver we also represent the domain of each variable s_i using Boolean variables $[\![s_i \leq v]\!], 0 \leq v < t_{max} - p_i$. These are used to track the reasons for propagation and generate nogoods. For more details see [16]. We use the notation $[\![s_i \geq v]\!], 1 \leq v \leq t_{max} - p_i$ as shorthand for $\neg[\![s_i \leq v-1]\!]$, and treat $[\![s_i \geq 0]\!]$ and $[\![s_i \leq t_{max} - p_i]\!]$ as synonyms for *true*. Propagators in a learning solver must explain each reduction in domain by building a clausal explanation using these Boolean variables.

Example 1. Consider the initial domain D^0 where $D^0(s_1) = D^0(s_2) = D^0(s_3) =$ [0,10]. Suppose later we have that $D(s_3) = [3, 8]$, or equivalently in the Boolean formulation $[\![s_3 \leq 8]\!] \wedge \neg[\![s_3 \leq 2]\!]$. Propagation of the constraint $s_3 + 0 \leq s_1$ gives a new domain where $D'(s_1) = [3, 10]$. We can explain this propagation using the clause $[\![s_3 \geq 3]\!] \rightarrow [\![s_1 \geq 3]\!]$ or equivalently $\neg[\![s_3 \leq 2]\!] \rightarrow \neg[\![s_1 \leq 2]\!]$. Similarly propagation of the constraint $s_2 + 2 \leq s_3$ gives a new domain where $D'(s_2) = [0, 6]$. We can explain this using $[\![s_3 \leq 8]\!] \rightarrow [\![s_2 \leq 6]\!]$. □

Efficient propagators for `cumulative` constraints [22] and precedence constraints (as a special case of linear constraints [9]) which explain themselves are well understood.

Optimisation problems are typically solved in CP via branch and bound. Given an objective *obj* which is to be maximised, when a solution is found with objective value o, a new constraint $obj > o$ is posted to enforce that we only look for better solutions in the subsequent search.

3 The Net-Present-Value Constraint Propagator

In order to solve the RCPSPDC using CP we need to build a propagator for the objective constraint. In this section we present three different propagators.

3.1 Propagator without Precedence Relations

The max-NPV problem without precedence relations is

$$\begin{aligned} \text{maximise} \quad & \sum_{i=1}^{n} e^{-\alpha s_i} c_i \\ \text{subject to} \quad & \min D(s_i) \leq s_i \leq \max D(s_i) \qquad 1 \leq i \leq n. \end{aligned}$$

This is easy to solve by simply setting $s_i = \min D(s_i)$ when $c_i > 0$ and $s_i = \max D(s_i)$ when $c_i < 0$. Define $T^+ = \{i \mid i \in \{1, \ldots, n\}, c_i > 0\}$ and $T^- = \{i \mid i \in \{1, \ldots, n\}, c_i < 0\}$.

In a branch and bound solution process, we will be trying to improve on a previous solution, and the constraint on the objective can be represented as:

$\sum_{i=1}^{n} e^{-\alpha s_i} c_i > o$ where o is the current best solution found. We can build a propagator directly for this as follows:

Consistency: Calculate $DC = \sum_{i \in T^+} e^{-\alpha \min D(s_i)} c_i + \sum_{i \in T^-} e^{-\alpha \max D(s_i)} c_i$. If $DC \leq o$ then fail.

Example 2. Consider 5 activities with cash flows -100, 125, -150, 200, 20 and $\alpha = 0.01$. Suppose the current domain is $D(s_1) = [0, 7]$, $D(s_2) = [3, 8]$, $D(s_3) = [0, 4]$, $D(s_4) = [1, 5]$ and $D(s_5) = [4, 8]$. Then $DC = -100e^{-0.01(7)} + 125e^{-0.01(3)} - 150e^{-0.01(4)} + 200e^{-0.01(1)} + 20e^{-0.01(4)} = 101.17$. Suppose the current best solution is $o = 105$ then the propagator would trigger backtracking. □

Bounds propagation: If $DC > o$ then propagate bounds as follows. For each $i \in T^+$, let $DC_i = DC - e^{-\alpha \min D(s_i)} c_i$. Let $u_i = \lfloor -\alpha^{-1}(\ln(o - DC_i) - \ln(c_i)) \rfloor$ and set $D(s_i) = D(s_i) \cap [0, u_i]$. Similarly for each $i \in T^-$, let $DC_i = DC - e^{-\alpha \max D(s_i)} c_i$ and $l_i = \lceil -\alpha^{-1}(\ln(DC_i - o) - \ln(-c_i)) \rceil$ and set $D(s_i) = D(s_i) \cap [l_i, t_{max}]$.

Example 3. Consider the problem of Ex. 2 but assume the current best solution is $o = 100$. Then we can propagate bounds. $DC_2 = -20.13$ and hence a new upper bound on s_2 is $u_2 = \lfloor -100(\ln(100 - -20.13) - \ln(125)) \rfloor = \lfloor 3.97 \rfloor = 3$. Similarly $DC_4 = -96.84$ and a new upper bound for s_4 is $u_4 = \lfloor -100(\ln(100 - -96.84) - \ln(200)) \rfloor = \lfloor 1.59 \rfloor = 1$. Now $DC_5 = 81.95$ and a potential new upper bound for s_5 is $u_5 = \lfloor -100(\ln(100 - 81.95) - \ln(20)) \rfloor = \lfloor 10.26 \rfloor = 10$, but this is not lower than its current upper bound so there is no propagation. □

Explanation: The explanation of failure simply uses the appropriate bounds of the variables. If $DC \leq o$ then we generate explanation $\bigwedge_{j \in T^+} [\![s_j \geq \min D(s_j)]\!] \wedge \bigwedge_{j \in T^-} [\![s_j \leq \max D(s_j)]\!] \rightarrow \mathit{false}$. Similarly for bounds propagation if $DC > o$ we generate an explanation $\bigwedge_{j \in T^+ - \{i\}} [\![s_j \geq \min D(s_j)]\!] \wedge \bigwedge_{j \in T^-} [\![s_j \leq \max D(s_j)]\!] \rightarrow [\![s_i \leq u_i]\!]$ for changing the upper bound of s_i where $c_i > 0$, and $\bigwedge_{j \in T^+} [\![s_j \geq \min D(s_j)]\!] \wedge \bigwedge_{j \in T^- - \{i\}} [\![s_j \leq \max D(s_j)]\!] \rightarrow [\![s_i \geq l_i]\!]$ for changing the lower bound of s_i where $c_i < 0$.

Example 4. An explanation for the failure of Ex. 2 is $[\![s_1 \leq 7]\!] \wedge [\![s_2 \geq 3]\!] \wedge [\![s_3 \leq 4]\!] \wedge [\![s_4 \geq 1]\!] \wedge [\![s_5 \geq 4]\!] \rightarrow \mathit{false}$. An explanation for the new upper bound of s_4 in Ex. 3 is $[\![s_1 \leq 7]\!] \wedge [\![s_2 \geq 3]\!] \wedge [\![s_3 \leq 4]\!] \wedge [\![s_5 \geq 4]\!] \rightarrow [\![s_4 \leq 1]\!]$. □

Strengthening Explanations: The explanations described above are not necessarily *the strongest possible, as* there may be weaker left hand sides of the explanation which still explain the right hand side consequence.

Assume $c_i > 0$ and $s_i \leq u_i$ is propagated. Then $DC_i + e^{-\alpha(u_i+1)} c_i \leq o$. Let $r = o - (DC_i + e^{-\alpha(u_i+1)} c_i)$ be the remainder before the bounds propagation will be weakened. We can relax the bounds on the other variables s_i, changing DC and DC_i as long as r remains non-negative. This generates a stronger (more reusable) explanation.

Example 5. Consider the explanation for the propagation of $s_4 \leq 1$ in Ex. 3. The remainder before the propagation is $r = 100 - (-96.84 + e^{-0.01(2)} 200) = 0.80$.

If we relax the bound on s_5 to 0 the new DC is 101.96. The difference is 101.96 - 101.17 = 0.79, hence the same upper bound for s_4 holds. Since $s_5 \geq 0$ is universally true we can remove it from the explanation obtaining a stronger explanation $[\![s_1 \leq 7]\!] \wedge [\![s_2 \geq 3]\!] \wedge [\![s_3 \leq 4]\!] \rightarrow [\![s_4 \leq 1]\!]$. Weakening any bound further drops the remainder to negative, so this explanation is as strong as possible. □

There are lots of strategies for strengthening explanations. Given that propagator is used for solving RcpspDc problems we may have a better weakening strategy by attempting to weaken the bounds for activities in T^+ (or T^-) simultaneously by the same amount, since if they are connected by precedence relations weakening one may have no effect. Given the remainder r we can look for a set of activities $I \subseteq T^+$ where reducing their lower bounds by 1 gives DC' where $DC' - DC \leq r$. If there is still a remainder we can continue. We can similarly increase the upper bounds by 1 for a set of activities in T^-.

Note that we can also handle the *npv* constraint by decomposing to $\sum_{i=1}^{n} c_i q_i \wedge \bigwedge_{i=1}^{n} q_i = e^{-\alpha s_i}$, and implementing the real variables q_i using views [19], but strengthening explanations is not directly possible in this approach.

3.2 Propagator with Precedence Relations

The precedence constrained max-NPV problem, PspDc, is

$$\text{maximise} \quad \sum_{i=1}^{n} e^{-\alpha s_i} c_i \tag{5}$$

$$\text{subject to} \quad \min D(s_i) \leq s_i \leq \max D(s_i) \qquad 1 \leq i \leq n, \tag{6}$$

$$s_i \leq s_j - d_{ij} \qquad \forall (i, j, d_{ij}) \in \mathcal{P}. \tag{7}$$

We can rephrase this only in terms of precedence constraints by adding a *source* activity 0 and a *sink* activity $n+1$ which are fixed to time 0 and t_{max} respectively, and adding *bounds precedence relations* $(0, i, \min D(s_i)), 1 \leq i \leq n$, and $(i, n+1, t_{max} - \max D(s_i)), 1 \leq i \leq n$, to $\mathcal{P}$, obtaining the reformulated problem with the extended set $\mathcal{P}$ in which the constraints (6) are replaced by the bounds precedence relations and the constraints $s_0 = 0$ and $s_{n+1} = t_{max}$ are added.

An algorithm for determining the optimal solution of PspDc is given by Demeulemeester *et al.* [6] (see also [27]). The algorithm is a hill climbing algorithm where, starting from an initial solution, groups of activities are shifted together until an optimal schedule is found. It maintains a graph with activities as nodes and precedence relations as edges. From this graph two *active trees* are built that are rooted at the source and the sink. For edges in these trees, it holds that their corresponding precedence relations are tight (at equality) in the current solution. These precedence relations/edges are called *active*. The initial solution is constructed by scheduling all activities at their earliest start time with respect to the precedence constraints, except the sink which is scheduled at t_{max}. By keeping track of which edges are active, a simple graph traversal is sufficient to find subtrees of the two active trees where the accumulated discount cash flow is

negative. The set of activities belonging to such subtrees can be shifted together towards the end of the schedule to improve the *npv*. By performing these shifts in a systematic manner, the final solution is guaranteed to be optimal.

The original algorithm has inefficiencies resulting from recalculating discounted cash flows repeatedly after each change in the active trees. Also for our use in a propagator we want to differentiate between the inter-activity precedence relations, *i.e.*, the original precedence relations, and the bounds precedence relations. We modify the Demeulemeester *et al.* [6] algorithm as follows: ***(i)*** We avoid recomputation by tracking which activities need to be reconsidered after a change in the active precedence relations. ***(ii)*** We prioritise inter-activity precedence relations over bounds precedence relations.

The modified algorithm maintains for each activity $i, 1 \leq i \leq n$, a unique parent pr_i in the tree (either $(i, pr_i, d_{ipr_i}) \in P_a$ or $(pr_i, i, d_{pr_i i}) \in P_a$). A unique parent ensures that the precedence relations form a tree. It also maintains for each activity i: the activity start time s_i, the discounted cash flow dc_i for the entire subtree under activity i, and the *root* r_i of the subtree containing i, that is the activity that is either connected to the source or the sink by an active precedence relation.

The algorithm precNPV (Algorithm 1) constructs an initial tree using initial defined the same as in the original Demeulemeester algorithm. Essentially it moves every activity to its earliest start time, then moves activities with negative cash flow and no successor as late as possible.

After the initial tree is constructed, the sets S and E contain the root nodes attached to the source and the sink, respectively. We now proceed to the main part of the algorithm, where through a sequence of shifts to the right, we construct an optimal solution for PspDc.

The algorithm repeatedly chooses a root activity to process, from S if possible and then from E. The subtree rooted at an activity is processed and the discounted cash flow dc for the subtree is calculated. If $dc \geq 0$, then it is recorded as the tree cost tc_i for the root activity i. If $dc < 0$, then we can improve the current solution by taking the whole subtree and shifting it to the right. We do this by calling moveTree, after which tc_i is set to 0. Calling moveTree changes the set of active precedence relations. This may cause certain subtrees to change or merge. We use the new active edges $(P'_a - P_a)$ to find the changed subtrees and schedule them for re-evaluation. When all root activities have been processed and no further shifts can improve the objective, then the solution is optimal and we can return the discounted cash flow for the entire schedule.

The recursive evaluation procedure recNPV(i,P_a,par) (Algorithm 2) returns the correct discounted cash flow for the subtree of activities rooted by i using active precedence relations P_a, where i has parent par. It stores par in pr_i in order to avoid traversing the parent precedence relation. It visits all the children of i in the tree, and detaches child subtrees which have negative discounted cash flow and moves them right. It sums up the cash flows of the (attached) children to determine the total discounted cash flow dc for the subtree, storing it in dc_i.

Algorithm 1: precNPV()

```
1  initial(); S := {i | pr_i = 0}; E := {i | pr_i = n + 1};
2  while ∃i ∈ S ∪ E do
3      P'_a := P_a;
4      if ∃i ∈ S then
5          S := S − {i}; (P_a, pt, dc) := recNPV(i, P_a, 0);
6          if dc < 0 then
7              P_a := P_a − {(0, i, min D(s_i))}; P_a := moveTree(pt, P_a); tc_i := 0;
8          else tc_i := dc;
9      else E := E − {i}; (P_a, pt, dc) := recNPV(i, P_a, n + 1); tc_i := dc;
10     for (k, j, d_kj) ∈ P_a − P'_a do
11         if (0, r_j, min D(s_{r_j})) ∈ P_a then S := S ∪ {r_j};
12         else E:= E ∪ {r_j};
13 return Σ_{i=1}^{n} tc_i;
```

Algorithm 2: recNPV(i,P_a,par)

```
1  pr_i := par; pt := {i}; dc := e^{−αs_i} c_i;
2  for (i, j, d_ij) ∈ P_a where j ≠ par do
3      (P_a, pt', dc') := recNPV(j, P_a, i);
4      if dc' ≥ 0 then pt := pt ∪ pt'; dc := dc + dc';
5      else P_a := P_a − {(i, j, d_ij)}; P_a := moveTree(pt',P_a);
6  for (j, i, d_ji) ∈ P_a where j ≠ par do
7      (P_a, pt', dc') := recNPV(j, P_a, i); pt := pt ∪ pt'; dc := dc + dc';
8  dc_i := dc;
9  return (P_a, pt, dc);
```

It returns dc, the set of activities in the subtree pt and the possibly modified active precedence relations P_a.

The movement of trees is extracted as a separate procedure, moveTree (Algorithm 3), which moves the subtree pt to the right as far as possible. It prefers inter-activity over bounds precedence relations. It updates the root information for the moved subtree.

Note that at the end of the algorithm each active subtree rooted at i (whether its parent is the sink or source or any other activity) is such that if $dc_i > 0$ then the active precedence relation connecting to the parent pr_i is of the form $(pr_i, i, d_{pr_i i})$, and if $dc_i < 0$ then the active precedence relation is of the form (i, pr_i, d_{ipr_i}). This condition ensures optimality of the solution.

Example 6. Consider a PspDc for a set of 10 activities with cash flow -100, 125, -150, 200, 20, -30, 20, 100, -100, 30 respectively each with domains $D(s_i) = [0, 7]$ and precedence relations $(1, 2, 1)$, $(3, 4, 1)$, $(4, 2, 1)$, $(4, 5, 3)$, $(5, 6, 1)$, $(6, 7, 1)$, $(8, 9, 1)$, $(9, 10, 1)$, $(10, 5, 1)$. The active tree built by initial is shown in Figure 1(a). Active precedence relations are shown outlined, while inactive precedence relations are shown dashed. Most bounds precedence relations are omitted for clarity. Initially $S = \{3, 8\}$ and $E = \{\}$. Processing $i = 3$ we find that the calculation of

Algorithm 3: moveTree(pt,P_a)

```
1 m := +∞;
2 for (l, k, d_lk) ∈ P − P_a, l ∈ pt, k ∉ pt do
3     if s_k − s_l − d_lk < m ∨ (s_k − s_l − d_lk = m ∧ e = (_, n + 1, _)) then
4         m := s_k − s_l − d_lk; e := (l, k, d_lk); r := r_k;
5         if k = n + 1 then r := l;
6 P_a := P_a ∪ {e}; for k ∈ pt do s_k := s_k + m; r_k := r;
7 return P_a;
```

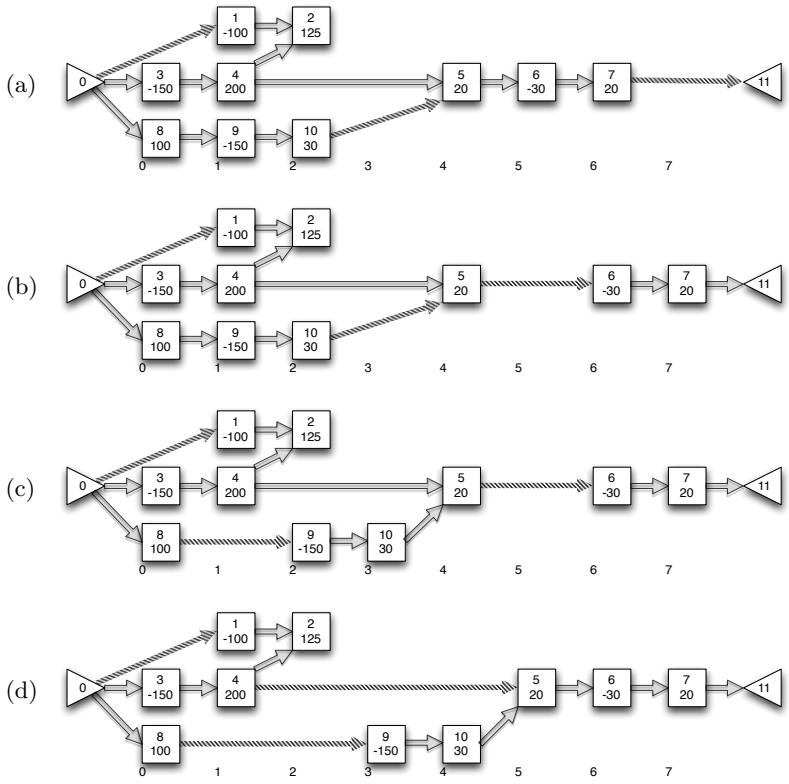

Fig. 1. Figures illustrating progress of the PspDc algorithm.

the discounted cash flow for the subtree rooted at 6 is $-30e^{-0.01(5)}+20e^{-0.01(6)} = -9.70$ and hence moveTree is called on $pt = \{6, 7\}$. This moves the subtree to the sink and makes the edge $(7, 11, 0)$ active, and sets the root of these nodes as 7. The processing of $i = 3$ finishes with final discounted cash flow $tc_3 = -100e^{-0.01(1)} + 125e^{-0.01(2)} - 150e^{0.01(0)} + 200e^{-0.01(1)} + 20e^{-0.01(4)} = 90.75$. The resulting active tree is shown in Figure 1(b). The newly added edge $(7, 11, 0)$ causes 7 to be added to E.

Processing $i = 8$ we find that the discounted cash flow for the subtree rooted at 9 is $-150e^{-0.01(1)} + 30e^{-0.01(2)} = -119.10$ and hence moveTree is called on $pt = \{9, 10\}$. This moves the subtree to the right, adding the active edge $(10, 5, 1)$ and sets the root of these nodes to 3. The processing of $i = 8$ finishes with final discounted cash flow $tc_8 = 100$. The resulting active tree is shown in Figure 1(c). The newly added edge causes 3 to be added to S.

Reprocessing $i = 3$ we find the discounted cash flow for the subtree rooted at 5 is $dc = 20e^{-0.01(4)} - 150e^{-0.01(2)} + 30e^{-0.01(3)} = -98.70$ and this moves the subtree $\{5, 9, 10\}$ to the right, adding the active edge $(5, 6, 1)$ and setting the root of these activities to 7. The call returns setting $tc_3 = -100e^{-0.01(1)} + 125e^{-0.01(2)} - 150e^{0.01(0)} + 200e^{-0.01(1)} = 71.53$. The new edge (re-)adds 7 to E.

Processing $i = 7$ we determine the discounted cash flow for the entire subtree $\{5, 6, 7, 9, 10\}$ as $dc_7 = 20e^{-0.01(5)} - 30e^{-0.01(6)} + 20e^{-0.01(7)} - 150e^{-0.01(3)} + 30e^{-0.01(4)} = -107.32$. The total discounted cash flow is $tc_3 + tc_7 + tc_8 = 64.21$. The final active tree is unchanged. □

When we solve the PspDc problem in a branch and bound solver we are given a current best solution o and need to satisfy

$$\sum_{i=1}^{n} e^{-\alpha s_i} c_i > o \wedge \bigwedge_{i=1}^{n} \min D(s_i) \leq s_i \leq \max D(s_i) \wedge \bigwedge_{(i,j,d_{ij}) \in P} s_i \leq s_j - d_{ij} \ .$$

We construct a propagator for this conjunction of constraints as follows:

Consistency: To check consistency of the objective constraint we run precNPV using the current domain D (after all precedence constraints have reached a fixpoint). If the optimal solution DC returned is $\leq o$ then we have detected that we cannot improve the current best solution and the propagator triggers backtracking.

Bounds propagation: If $DC > o$ we record the start time values returned by precNPV as tentative start times s'. We may be able to reduce the domains of some variables. We can do so in a manner analogous to the unconstrained problem as follows. We construct an unconstrained max-NPV problem as follows. Let $S = \{i \mid pr_i = 0\}$ and $E = \{i \mid pr_i = n + 1\}$ be the roots of the active subtrees in the optimal solution. The unconstrained problem has activities $S \cup E$ (corresponding to the active subtrees) where the cash flow c'_i for each activity is given by the $tc_i = e^{-\alpha s'_i} c'_i$ where tc_i is calculated by precNPV. We now do bounds propagation completely analogously to the unconstrained case described in Section 3.1. A slight extension is we can now have activities with $c'_i = 0$. We never generate new bounds for such activities.

The bounds propagation is correct since the unconstrained problem *overestimates* the effect of changing activity start times. Consider an active subtree rooted at i with $c'_i > 0$. Let $DC_i = DC - e^{-\alpha \min D(s_i)} c'_i$. The new upper bound u_i is calculated as $u_i = \lfloor -\alpha^{-1}(\ln(o - DC_i) - \ln(c'_i)) \rfloor$. It updates the domain of s_i if it is stricter than the current bound. $D(s_i) = D(s_i) \cap [0, u_i]$. Suppose we set $s_i = u_i + 1$. Then the unconstrained problem with $s_i = u_i + 1$ set would have an optimal value $o' \leq o$ and hence violates the objective constraint. This solution is

the optimal solution to a relaxation of the original maximisation problem with $s_i = u_i + 1$ where precedence relations not in P_a are relaxed. But any solution to the original constrained problem with $s_i = u_i + 1$ set is also a solution to the relaxed problem, hence any solution o'' of the original problem with $s_i = u_i + 1$ is such that $o'' \leq o' \leq o$ and would also violate the objective constraint. An analogous argument holds for new lower bounds. Hence the bounds propagation is correct.

Example 7. Consider the problem of Ex. 6, and suppose the current best solution is $o = 63$. We create three pseudo-activities: one for the tree rooted at 3 which has $71.53 = e^{-0.01(0)}c'_3$ or $c'_3 = 71.53$; one for the tree rooted at 7 which has $-107.32 = e^{-0.01(7)}c'_7$ or $c'_7 = -115.10$; and one for the tree rooted at 8 which has $c'_8 = 100$. The propagation for s_3 gives $u_3 = \lfloor -100(\ln(63 - -7.32) - \ln(71.53))\rfloor = \lfloor 1.71\rfloor = 1$. The propagation for s_8 gives $u_8 = \lfloor -100(\ln(63 - -35.79) - \ln(100))\rfloor = \lfloor 1.22\rfloor = 1$. The propagation for s_7 gives new potential lower bound $l_7 = \lceil -100(\ln(171.53 - 63) - \ln(107.32))\rceil = \lceil -1.12\rceil = -1$ and hence there is no propagation. □

Mapping the PspDc to an unconstrained max-NPV problem only allows us to improve the bounds on the activities which are the roots of active subtrees. We need to extend the approach to find bounds on other activities.

Let j be an activity in the active subtree rooted at i (that is $r_j = i$). We split the subtree rooted at i into two parts, one containing j and the other containing i and create two pseudo-activities for them. We then solve an unconstrained max-NPV problem.

We subtract dc_j from the discounted cash flows computed for all the subtrees on the path from j to the root i and see for which subtrees the discounted cash flow changes sign. Let

$$k = \text{indexmin}\{|dc_{k'}| \mid k' \text{ on the path from } i \text{ to } j, dc_{k'} \times (dc_{k'} - dc_j) \leq 0\}.$$

be the activity where the discounted cash flow changes sign, but the cash flow is minimally distant from 0. Note that if no activity changes sign we let $k = j$. In an optimal solution of the problem with only precedence relations P_a, if we set s_j to be some $u_j + 1$ then activities in the tree rooted at k move to stay attached to s_j, while the remaining activities stay attached to i.

We replace the pseudo-activity for i with two pseudo-activities: one rooted at i has $pt_i = dc_i - dc_k$, and c'_i such that $pt_i = e^{-\alpha s'_i}c'_i$; and another rooted at j which has $pt_j = dc_k$ and c'_j such that $pt_j = e^{-\alpha s'_j}c'_j$. We propagate the new unconstrained max-NPV problem as before.

Example 8. Consider the problem of Ex. 6, and suppose the current best solution is $o = 63.5$.

Suppose we are interested in bounding s_4 in the tree with root 3. Now $dc_4 = -100e^{-0.01(1)} + 125e^{-0.01(2)} + 200e^{-0.01(1)} = 221.53$, and $dc_3 = 71.53$ Since $dc_3 - dc_4 < 0$ we have that $k = 3$. We determine that $71.53 = c'_4e^{-0.01(1)}$ and hence $c'_4 = 72.25$. Calculating DC for the 3 pseudo-activities $\{4, 7, 8\}$ gives

$DC = 72.25e^{-0.01(1)} - 115.10e^{-0.01(7)} + 100e^{-0.01(0)} = 64.21$. Now $DC_4 = 64.21 - 72.25e^{-0.01(1)} = -7.32$ and then $u_4 = \lfloor -100(\ln(63.5 - -7.32) - \ln(72.25)) \rfloor = \lfloor 1.99 \rfloor = 1$.

Suppose we are interested in bounding s_2 in the tree with root 3. $dc_2 = -100e^{-0.01(1)} + 125e^{-0.01(2)} = 23.52$, and $dc_4 - dc_2 > 0$ and $dc_3 - dc_2 > 0$ hence there are no changes in sign and $k = 2$. We determine $c'_3 = -150e^{-0.01(0)} + 200e^{-0.01(1)} = 48.01$. Calculating DC for the 4 pseudo-activities $\{2, 3, 7, 8\}$ gives $DC = 24.00e^{-0.01(2)} + 48.01e^{-0.01(0)} + -115.10e^{-0.01(7)} + 100e^{-0.01(0)} = 64.21$. Now $DC_2 = 64.21 - 24.00e^{-0.01(2)} = 40.69$ and then $u_2 = \lfloor -100(\ln(63.5 - 40.69) - \ln(24.00)) \rfloor = \lfloor 5.09 \rfloor = 5$. □

Explanation: The reason for preferring inter-activity precedence relations over bounds precedence relations is driven by explanation. The justification for inconsistency is the set of active precedence relations P_a, but since inter-activity precedence relations are globally true they are not required in an explanation. Hence the explanation of inconsistency is simply $\bigwedge_{j \in S} [\![s_j \geq \min D(s_j)]\!] \wedge \bigwedge_{j \in E} [\![s_j \leq \max D(s_j)]\!] \rightarrow \mathit{false}$ where S and E are defined as above. Explaining the bounds propagation is analogous $\bigwedge_{j \in S - \{i\}} [\![s_j \geq \min D(s_j)]\!] \wedge \bigwedge_{j \in E} [\![s_j \leq \max D(s_j)]\!] \rightarrow [\![s_i \leq u_i]\!]$ for changing the upper bound of s_i where $c'_i > 0$, and $\bigwedge_{j \in S} [\![s_j \geq \min D(s_j)]\!] \wedge \bigwedge_{j \in E - \{i\}} [\![s_j \leq \max D(s_j)]\!] \rightarrow [\![s_i \geq l_i]\!]$ for changing the lower bound of s_i where $c'_i < 0$. The explanations are independent of whether the bounds propagation was for the root of an active subtree, or an internal activity (in which case e.g. $S - \{i\} = S$).

Strengthening Explanations: We can strengthen explanations of failure and bounds propagation in the precedence constrained case in a manner analogous to the unconstrained case. One extra consideration is not to weaken a bound in a way that would violate a non-active precedence relation. We consider weakening a bound only to the point that it would make a non-active precedence hold at equality. We then consider weakening a different bound. We may reconsider the original bound if the other variable in non-active precedence relation which limited its weakening is itself weakened.

3.3 Using a generic LP propagator for PspDc

Although the PspDc appears non-linear at first sight, a simple substitution turns it into a linear problem. This was first noted by Grinold [7]. Let $q_i = e^{-\alpha s_i}$ for each i. Then the PspDc stated in (5–7) on page 6 becomes

$$\begin{array}{rll} \text{maximise} & \sum_{i=1}^{n} q_i c_i & \\ \text{subjected to} & e^{-\alpha \max D(s_i)} \leq q_i \leq e^{-\alpha \min D(s_i)} & 1 \leq i \leq n, \\ & q_i e^{-\alpha d_{ij}} \geq q_j & \forall (i, j, d_{ij}) \in \mathcal{P}. \end{array}$$

The objective and the constraints in PspDc are linear in q_i. Thus it is possible to implement the max-NPV constraint using a generic linear programming propagator [2]. We treat the variables q_i as views [19] of the original start times.

A generic LP implementation of the max-NPV constraint differs from the specialised implementation of the previous subsection in several ways. The LP propagator is not able to take advantage of the special form of the linear program, thus it takes $O(n^2)$ to perform a pivot (equivalent to shifting a subtree), whereas the specialised algorithm only takes $O(n)$ to perform a shift. Secondly, the LP propagator cannot differentiate between inter-activity and bounds precedence relations, thus its explanations for failure are inferior to the specialised version. On the plus side, the LP propagator uses dual simplex, thus rather than recreating an optimal solution from scratch when the previous one is violated by bounds changes, it simply modifies the previous optimal solution to make it feasible again. This is more incremental than the specialised propagator and can save some work.

4 Experiments

We use a branch-and-bound algorithm in connection with different binary search heuristics which are used in different stages of the solution process. The following heuristics are used

SMALLEST**:** Select the start time variable s_i with the smallest overall value v in its domain (breaking ties using input order): branch first $s_i \leq v$ and then $s_i \geq v + 1$.

LARGEST**:** Select the start time variable s_i with the largest overall value v in its domain (breaking ties with input order): branch first $s_i \geq v$ and then $s_i \leq v - 1$.

BIDIR**:** An adaption of the bi-directional heuristic in [24].

1. Select and branch using SMALLEST but only from variables s_i where all predecessors of i are already scheduled, breaking ties first by most positive cash flow.
2. Otherwise select and branch using LARGEST but only from variables s_i where all successors of i are already scheduled, breaking ties first on least negative cash flow.
3. Otherwise select using SMALLEST from any activities with positive cash flow, or finally using LARGEST for any activities with negative cash flow.

SLB**:** Start with SMALLEST, if at least half of the activities have a positive cash flow, otherwise use LARGEST. After finding the first solution switch to BIDIR.

VSIDS**:** Use activity based search [14] to choose the variable bound that is most associated with recent failure.

HOTSTART**:** Run SLB until a solution is found. If it takes less than $50 \times |V|$ conflicts then continue SLB until this number is reached. Then switch to VSIDS and restart the search.

In preliminary experiments, VSIDS and BIDIR had problems in quickly finding a first solution, especially if the deadline was tight to the optimal makespan. Later in the search SMALLEST, LARGEST and BIDIR had problems improving the current solution, whereas VSIDS did not. These results and the fact that a similar

Table 1. Comparison of the different methods with VDH01.

Search	LP	CP(unc)	CP(prec+unc)	CP(prec)	VDH01
SLB	388/1056	757/1036	376/1067	371/1067	
	0/937/18503	0/1487/17953	0/903/18537	0/887/18553	
	29215/6.03s	68142/9.43s	24111/5.86s	25404/5.80s	
VSIDS	31/1161	63/1159	17/1166	17/1165	0/0
	0/157/19283	0/269/19171	0/152/19288	0/146/19294	598/832/18010
	7565/1.51s	12325/2.36s	6092/1.51s	6304/1.46s	1344849/9.06s
HOTSTART	62/1140	117/1130	53/1143	51/1142	
	0/216/19224	0/336/19104	0/203/19237	0/200/19240	
	8027/1.81s	12019/2.71s	6147/1.79s	6423/1.72s	

kind of hybrid was successful on RCPSP [22] motivated the construction of SLB and HOTSTART.

The heuristics VSIDS and HOTSTART are used with the Luby restart policy [13] with a restart base of 500 conflicts. For HOTSTART, we only allow the VSIDS phase to restart.

We carried out extensive experiments on the benchmark set available at `www.projectmanagement.ugent.be/npv.html`. From there we also retrieved a Windows executable of the complete method presented in [27]. This method (denoted VDH01) is compared with our approaches. For that purpose, we ran VDH01 on a X86-64 architecture running Windows Server 2008 and a Intel(R) Core(TM)2 Quad CPU Q9400 processor with 2.66 GHz whereas our approach was run on a X86-64 architecture running GNU/Linux and a Intel(R) Xeon(R) CPU E54052 processor with 2 GHz. We implemented our approach using the LCG solver Chuffed. We used a time limit of 100 seconds.

The benchmark set consists of 1080 RCPSP instances involving four resources with 360 instances with each of 10, 20, and 30 activities. A more detailed specification of these instances can be found in [27]. These instances were extended with cash flows of activities, a discount rate of 0.01, and a deadline: the cash flows were generated by VDH01 with the percentage of negative cash flows set at 0, 20, 40, 60, 80, or 100 percent and the deadline is given as the optimal makespan + 0, 5, 10. In total, the set includes 19440 RCPSPDC instances.

Table 1 compares VDH01 with our LCG approaches: **LP** using the LP propagator of Section 3.3, **CP(unc)** using the propagator of Section 3.1, **CP(prec)** using the propagator of Section 3.2. and **CP(prec+unc)** using both propagators simultaneously. The top field of an entry for a propagator shows numbers of instances for which our method found worse/better solutions than VDH01. The middle field lists the numbers of instances respectively that the method could not find a solution; found a solution, but did not prove optimality; and found the optimal solution and proved its optimality. The last entry contains the average number of explored nodes and the average runtime per instance.

Our method could find a solution for each instance, whereas VDH01 had trouble doing so, especially when the deadline was tight. Moreover, in the majority of cases our methods find a better solution. The number of explored nodes

Table 2. Detailed results for our approach with the best search heuristic VSIDS.

dl / cf	0%	20%	40%	60%	80%	100%
	2/66	1/79	0/79	0/75	0/85	0/82
0	0/11/1069	0/8/1072	0/6/1074	0/4/1076	0/9/1071	0/11/1069
	7399/1.56s	5038/1.12s	4844/1.03s	4289/0.97s	6037/1.25s	8780/1.69s
	6/38	2/46	0/44	0/66	0/86	0/99
5	0/14/1066	0/8/1072	0/2/1078	0/5/1075	0/10/1070	0/15/1065
	9440/2.26s	5759/1.40s	2661/0.60s	3731/0.92s	6768/1.76s	11206/2.42s
	5/29	1/17	0/27	0/50	0/90	0/107
10	0/16/1064	0/2/1078	0/1/1079	0/3/1077	0/3/1077	0/18/1062
	10514/2.64s	4743/1.19s	2291/0.52s	3169/0.87s	5085/1.33s	11713/2.67s

clearly shows that LCG drastically reduces the search, and the LP and CP(prec) propagators provide stronger pruning than CP(unc). Interestingly, the LP approach is almost as fast as CP(prec) which may relate to its more incremental approach. Overall the CP(prec) approach is best, combining it with CP(unc) reduces search but overall does not pay off.

Surprisingly, HOTSTART is not as effective as it was for RCPSP. It appears that SLB does not concentrate the following VSIDS search on critical conflicts.

Table 2 shows the results of our best method CP(prec) with the best performing search heuristic VSIDS. These results are split concerning the different deadlines (`dl`) and percentages of negative cash flow activities (`cf`). Interestingly, our method slightly decays as the deadline increases for instances with only either positive or negative cash flow activities whereas it improves for instances with mixed cash flow activities. The hardest instances for our method are either all positive or all negative cash flows which confirms the results in [27].

5 Conclusion and Future Work

We have shown how we can use LCG [16] to quickly find optimal solutions to RCPSPDC problems. We have developed 3 propagators that explain their propagation in order to tackle the problem. They each lead to a comprehensively better solution than the previous state-of-the-art complete method [27]. While any form of complete search methods can struggle with very large problems, they can be combined with incomplete methods in order to solve larger problems. Future work concentrates on the integration of the presented method in a large neighbourhood search [25].

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. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling 17(7), 57–73 (Dec 1993)
2. Beringer, H., Backer, B.D.: Satisfiability of boolean formulas over linear constraints. In: IJCAI. pp. 296–304 (1993)
3. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(1), 3–41 (1999), `http://www.sciencedirect.com/science/article/B6VCT-3YSXJP1-P/2/7006a62859d7b4d83e72b0ea50c8b88b`
4. Demeulemeester, E.L., Herroelen, W.S.: A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 38(12), 1803–1818 (1992)
5. Demeulemeester, E.L., Herroelen, W.S.: New benchmark results for the resource-constrained project scheduling problem. Management Science 43(11), 1485–1492 (1997)
6. Demeulemeester, E.L., Herroelen, W.S., Van Dommelen, P.: An optimal recursive search procedure for the deterministic unconstrained max-npv project scheduling problem. Tech. rep., Katholieke Universiteit Leuven (1996), Research Report 9603
7. Grinold, R.C.: The payment scheduling problem. Naval Research Logistics Quarterly 19(1), 123–136 (1972)
8. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207(1), 1–14 (2010)
9. Harvey, W., Stuckey, P.: Improving linear constraint propagation by changing constraint representation. Constraints 8(2), 173–207 (2003)
10. Herroelen, W.S., Demeulemeester, E.L., De Reyck, B.: A classification scheme for project scheduling. In: Weglarz, J. (ed.) Project Scheduling, International Series in Operations Research and Management Science, vol. 14, pp. 1–26. Kluwer Academic Publishers (1999)
11. Icmeli, O., Erengüç, S.S.: A branch and bound procedure for the resource constrained project scheduling problem with discounted cash flows. Management Science 42(10), 1395–1408 (1996)
12. Liu, S.S., Wang, C.J.: Resource-constrained construction project scheduling model for profit maximization considering cash flow. Automation in Construction 17(8), 966–974 (2008)
13. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Inf. Proc. Let. 47(4), 173 – 180 (1993)
14. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Procs. of DAC2001. pp. 530–535 (2001)
15. Neumann, K., Zimmermann, J.: Exact and truncated branch-and-bound procedures for resource-constrained project scheduling with discounted cash flows and general temporal constraints. Central European Journal of Operations Research 10(4), 357–380 (2002)
16. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
17. Russell, A.H.: Cash flows in networks. Management Science 16(5), 357–373 (1970)
18. Schulte, C., Stuckey, P.: Efficient constraint propagation engines. ACM Transactions on Programming Languages and Systems 31(1), Article No. 2 (2008)

19. Schulte, C., Tack, G.: Views iterators for generic constraint implementations. In: Procs. of CP2005. pp. 817–821 (2005)
20. Schutt, A., Feydy, T., Stuckey, P., Wallace, M.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (2011)
21. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving the resource constrained project scheduling problem with generalized precedences by lazy clause generation (Sep 2010), http://arxiv.org/abs/1009.0347
22. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (2011)
23. Schwindt, C., Zimmermann, J.: A steepest ascent approach to maximizing the net present value of projects. Mathematical Methods of Operations Research 53, 435–450 (2001)
24. Selle, T., Zimmermann, J.: A bidirectional heuristic for maximizing the net present value of large-scale projects subject to limited resources. Naval Research Logistics (NRL) 50(2), 130–148 (2003)
25. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.F. (eds.) Principles and Practice of Constraint Programming – CP98, Lecture Notes in Computer Science, vol. 1520, pp. 417–431. Springer Berlin / Heidelberg (1998), http://dx.doi.org/10.1007/3-540-49481-2_30
26. Vanhoucke, M.: A scatter search heuristic for maximising the net present value of a resourceconstrained project with fixed activity cash flows. International Journal of Production Research 48(7), 1983–2001 (2010)
27. Vanhoucke, M., Demeulemeester, E.L., Herroelen, W.S.: On maximizing the net present value of a project under renewable resource constraints. Management Science 47, 1113–1121 (Aug 2001)