Maximising the Net Present Value of Large Resource-Constrained Projects

Hanyu Gu[1], Peter J. Stuckey[2], and Mark G. Wallace[3]

[1] National ICT Australia, Victoria Research Laboratory,
The University of Melbourne, Victoria 3010, Australia
Hanyu.Gu@nicta.com.au
[2] National ICT Australia, Department of Computing and Information Systems,
The University of Melbourne, Victoria 3010, Australia
pstuckey@unimelb.edu.au
[3] Monash University, Victoria 3100, Australia
mark.wallace@monash.edu

Abstract. The Resource-constrained Project Scheduling Problem (RCPSP), in which a schedule must obey resource constraints and precedence constraints between pairs of activities, is one of the most studied scheduling problems. An important variation of this problem (RCPSPDC) is to find a schedule which maximises the net present value (discounted cash flow). Large industrial applications can require thousands of activities to be scheduled over a long time span. The largest case we have (from a mining company) includes about 11,000 activities spanning over 20 years. We apply a Lagrangian relaxation method for large RCPSPDC to obtain both upper and lower bounds. To overcome the scalability problem of this approach we first decompose the problem by relaxing as fewer as possible precedence constraints, while obtaining activity clusters small enough to be solved efficiently. We propose a hierarchical scheme to accelerate the convergence of the subgradient algorithm of the Lagrangian relaxation. We also parallelise the implementation to make better use of modern computing infrastructure. Together these three improvements allow us to provide the best known solutions for very large examples from underground mining problems.

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 constraints between pairs of activities where a precedence constraint 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 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 $m, 1|cpm, \delta_n, c_j|npv$ [13] 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 [21]. Different complete and incomplete methods for RCPSPDC with or without generalised precedence constraints have been proposed, the reader is referred to [12] 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 approaches in [14,24,19] are based on the branch-and-bound algorithm in [6,7] for RCPSP. These algorithms use a scheduling generation scheme which resolves resource conflicts by adding new precedence constraints between activities in conflict. The method in [24] improves upon the one in [14] whereas the work [19] considers RCPSPDC with generalised precedence constraints. Recently we developed lazy clause generation approaches to RCPSPDC [22] which provide the state of the art complete methods for RCPSPDC.

But complete methods can only solve problems up to a hundred activities. To cope with large industrial applications with thousands of activities, various rules based heuristics [2,20] are used in practice. However these approaches are not robust especially for problems with tight constraints. Decomposition methods are widely used for large-scale combinatorial optimisation problems. Lagrangian relaxation was successfully applied on RCPSP for up to 1000 jobs [18]. It has also been applied to RCPSPDC with good results [17] and is more robust than rule based heuristics. However our experience shows that this method scales badly for example underground mining problems with over a few thousand activities. The main obstacle is that the Lagrangian relaxation problem, which is a max-flow problem, cannot be solved efficiently. Furthermore the Lagrangian dual problem requires many more iterations to converge. We address these two problems in this paper to improve the Lagrangian relaxation method to generate tight upper and lower bounds for RCPSPDC problems with up to 10,000 activities.

The contributions of this paper are: ***(i)*** We propose a general approach to relax as fewer as possible of the precedence constraints but still obtain activity clusters small enough to be solved efficiently. ***(ii)*** We define a hierarchical scheme to accelerate the convergence of the Lagrangian multipliers for the precedence constraints when using subgradient algorithm. ***(iii)*** We parallelise the algorithm to make more effective use of modern multi-core desktop computers. ***(iv)*** We produce highly competitive results on very large underground mining problems within a reasonable computing time.

2 Lagrangian Relaxation of Resource Constraints

Let T be the deadline of the project, α the discount rate, R_k be the capacity of resource $k \in R$, r_{jk} be the resource requirement of activity $j \in J$ on resource k, p_j be the processing time of activity j, c_j is the discounted cash flow for activity j to start at time 0, and precedence constraint $(i,j) \in L$ if activity j cannot start before activity i completes. The RcpspDc problem can be stated as follows:

$$NPV = \text{maximise } \textstyle\sum_j e^{-\alpha s_j} c_j \tag{1}$$

$$\text{subject to} \quad s_i + p_i \leq s_j \qquad (i,j) \in L \tag{2}$$

$$\texttt{cumulative}(s, p, [r_{jk} | j \in J], R_k) \qquad k \in R, \tag{3}$$

$$0 \leq s_j \leq T - p_j \qquad j \in J. \tag{4}$$

where $s_j, j \in J$ is the start time of activity j. This model uses the global constraint `cumulative`, and hence requires a CP technology to solve directly.

The time-indexed formulation for the RcpspDc problem, breaks the start time variables into binary form, and encodes the resource constraints explicitly for each time point, giving a binary program. Let w_{jt} be the discounted cash flow of activity j when starting at time t, i.e. $w_{jt} = e^{-\alpha t} c_j$. The time-indexed formulation is [8]:

$$NPV = \text{maximise } \textstyle\sum_j \sum_t w_{jt} x_{jt} \tag{5}$$

$$\text{subject to} \quad \textstyle\sum_t x_{jt} = 1 \qquad j \in J \tag{6}$$

$$\textstyle\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_j-1} x_{js} \leq 1 \qquad \forall (i,j) \in L, t = 0, \cdots, T \tag{7}$$

$$\textstyle\sum_j r_{jk} (\sum_{s=t-p_j+1}^{t} x_{js}) \leq R_k \qquad k \in R, t = 0, \cdots, T \tag{8}$$

$$\text{all variables binary} \tag{9}$$

where the binary variable $x_{jt} = 1$ if activity j starts at t ($s_j = t$), and $x_{jt} = 0$ otherwise. The assignment constraints (6) ensure that each activity has exactly one start time. The precedence constraints (7) imply that activity j cannot start before $t + p_j$ if activity i starts at or after time t for each $(i,j) \in L$. The resource constraints (8) enforce that the resource consumption of all activities processed simultaneously must be within the resource capacity.

The Lagrangian Relaxation Problem (LRP) obtained by relaxing the resource constraints (8) with Lagrangian multipliers λ_{kt}, $k \in R$, $t = 0, \cdots, T$ is

$$Z_{LR}(\lambda) = \text{maximise } LRP(x) \tag{10}$$

$$\text{subject to} \quad (6), (7), (9) \tag{11}$$

where

$$LRP(x) = \sum_j \sum_t w_{jt} x_{jt} + \sum_{k \in R} \sum_t \lambda_{kt} (R_k - \sum_j r_{jk} (\sum_{s=t-p_j+1}^{t} x_{js})) \tag{12}$$

By rearranging the terms in (12) we have

$$LRP(x) = \sum_j \sum_t z_{jt} x_{jt} + \sum_{k \in R} \sum_t \lambda_{kt} R_k \tag{13}$$

where

$$z_{jt} = w_{jt} - \sum_{s=t}^{t+p_j-1} \sum_{k \in R} \lambda_{ks} r_{jk} \tag{14}$$

It is well-know [10] that $Z_{LR}(\lambda)$ is a valid upper bound of RcpspDc for $\lambda \geq 0$.

The polytope described by (6), (7), and (9) is integral [4]. However, it is inefficient to solve LRP using a general LP solver. Instead, it can be transformed into a minimum cut problem [18] and solved efficiently by a general max-flow algorithm. We briefly explain the network flow model of [18], denoted by $G = (V, A)$ in order to be self-contained. The node set $V = \{a, b\} \cup \bigcup_{i \in J} V_i$ is defined as

- a is the source node and b is the sink node.
- $V_i = \{v_{it} | t = e(i), \cdots, l(i)+1\}$ where $e(i)$ and $l(i)$ are the earliest and latest start time of activity i respectively.

The directed edge set $A = A^a \cup A^b \cup A_i^A \cup A_{i,j}^P$ is defined as

- $A^a = \{(a, v_{j,e(j)}) | \forall j \in J\}$ are the auxiliary edges connecting the source and the first node of each activity. All edges in A^a have infinite capacity.
- $A^b = \{(v_{j,l(j)+1}, b) | \forall j \in J\}$ are the auxiliary edges connecting the last node of each activity and the sink. All edges in A^b have infinite capacity.
- $A_i^A = \{(v_{it}, v_{i,t+1}) | t = e(i), \cdots, l(i)\}$ are the assignment edges that forms a chain for each activity. $(v_{it}, v_{i,t+1})$ has capacity z_{it}.
- $A_{i,j}^P = \{(v_{it}, v_{j,t+p_i}) | e(i) + 1 \leq t \leq l(i), e(j) + 1 \leq t + p_i \leq l(j)\}$ are the precedence edges for $(i, j) \in L$. All the precedence edges have infinite capacity.

Since the network flow model has $O(|J|T)$ nodes and $O((|J| + |L|)T)$ edges, a state of the art max-flow solver [5] can solve it in $O(|J||L|T^2 \log(T))$. We use the push-relabel implementation in c++ BOOST BGL [23] which is based on the source code of the authors of [5]. This implements the highest-label version of the push-relabel method with global relabeling and gap relabeling heuristics. With just 1400 activities and $T = 4000$ the network has about 5 million nodes and it takes on average 4 minutes to solve the maximal flow problem. For some larger cases we could not even set up the network model on a desktop computer with 16GB memory.

3 Lagrangian Relaxation of Precedence Constraints

To overcome the scalability problem of the max-flow algorithm we further relax some precedence constraints so that activities can form clusters that are independent from each other. We partition the set of activities J into $J = J_1 \cup J_2 \cup \cdots \cup J_U$

where U is the given number of clusters. The multi-cut of this partition is defined as the set of precedence constraints $E = \{(i,j) \in L | i \in J_p, j \in J_q, p \neq q\}$. Denote the set of precedence relations that hold on cluster J_i by $L_i = \{(k,j) \in L | k \in J_i, j \in J_i\}$. Obviously we have $L = E \cup L_1 \cup \cdots \cup L_U$. To reduce the number of Lagrangian multipliers introduced for the relaxed precedence relations, we use the weak form of the precedence constraints [18]

$$\sum_t t(x_{jt} - x_{it}) \geq p_i, \quad \forall (i,j) \in E \tag{15}$$

We give the details here how (15) can be derived by (6) and (7) even when x_{jt} is allowed to be fractional. By rewriting (6) as

$$\sum_{0}^{s=t+p_j-1} x_{js} + \sum_{s=t+p_j} x_{js} = 1$$

and replacing the common terms in (7) we get

$$\begin{aligned} & \textstyle\sum_{s=t}^{T} x_{is} + 1 - \sum_{s=t+p_j} x_{js} \leq 1 \\ \Rightarrow\ & x_{it} + \textstyle\sum_{s=t+1}^{T} x_{is} - (\sum_{s=t+1}^{T} x_{js} - \sum_{s=t+1}^{t+p_i-1} x_{js}) \leq 0 \\ \Rightarrow\ & \textstyle\sum_{s=t+1}^{T} (x_{js} - x_{is}) \geq \sum_{s=t+1}^{t+p_i-1} x_{js} + x_{it} \end{aligned} \tag{16}$$

By summing up (16) over all t we have

$$\begin{aligned} \sum_t t(x_{jt} - x_{it}) &\geq \sum_t \sum_{s=t+1}^{t+p_i-1} x_{js} + \sum_t x_{it} \\ &= \sum_{k=1}^{p_i-1} \sum_{s=k} x_{jk} + 1 \\ &= \sum_{k=1}^{p_i-1} \sum_{s=0} x_{jk} + 1 = p_i \end{aligned}$$

which exploits that $x_{jt} = 0$, $t = 0, \cdots, p_i - 1$.

By relaxing the precedence constraints (15) with Lagrangian multipliers μ we can obtain a Decomposable Lagrangian Relaxing Problem (DLRP)

$$Z_{LR}(\lambda, \mu) = \text{maximise } LRP(x) + \textstyle\sum_{v=(i,j) \in E} \mu_v (\sum_t t(x_{jt} - x_{it}) - p_i) \tag{17}$$

$$\text{subject to} \qquad (6),\ (9) \tag{18}$$

$$\textstyle\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_j-1} x_{js} \leq 1 \quad \forall (i,j) \in L \backslash E, t = 0, \cdots, T \tag{19}$$

Let the set of precedence constraints in the multi-cut that have activity i as predecessor be $P_i = \{(i,j) \in E\}$, the set of precedence constraints in the multi-cut that have activity i as successor be $S_i = \{(j,i) \in E\}$. By rearranging the items in the objective function of DLRP in (17), we get

$$LRP(x) + \sum_j \sum_t (\sum_{e \in S_j} t\mu_e - \sum_{e \in P_j} t\mu_e) x_{jt} - \sum_{e=(i,j) \in E} \mu_e p_i \tag{20}$$

By ignoring the constant terms the DLRP can be decomposed into U independent subproblems on each of the clusters J_k, $k = 1, \cdots, U$

$$\text{maximise} \quad \sum_{j \in J_k} \sum_t (z_{jt} + \sum_{e \in S_j} t\mu_e - \sum_{e \in P_j} t\mu_e) x_{jt} \tag{21}$$

$$\text{subject to} \quad \sum_t x_{jt} = 1 \quad j \in J_k \tag{22}$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_j-1} x_{js} \leq 1 \quad \forall (i,j) \in L_k, t = 0, \cdots, T \tag{23}$$

$$\text{all variables binary} \tag{24}$$

Since each subproblem has smaller size, the max-flow solver can solve DLRP much faster than LRP. Also these subproblems can be solved in parallel utilising the multi-core computers that are now very popular. If main memory of the computer is a bottleneck we can construct the network flow model of each cluster on the fly. In this way we have solved the DLRP with over a hundred million variables within 500M memory.

The upper bound will become worse, i.e., $Z_{LR}(\lambda) \leq Z_{LR}(\lambda, \mu)$ since the weak form of the precedence constraints is used. Our goal therefore is to relax as fewer as possible the precedence constraints but still obtain activity clusters small enough to solve efficiently as a maximal flow problem. This can be formulated as the Min-Cut Clustering problem (MCC) as in [15]

$$\text{minimise} \quad \sum_{g=1}^{U} \sum_{e \in L} z_{eg} \tag{25}$$

$$\text{subject to} \quad \sum_{g=1}^{U} x_{ig} = 1 \qquad i \in J \tag{26}$$

$$x_{ig} - x_{jg} \leq z_{eg} \qquad \forall e = (i,j) \in L, g = 1, \cdots, U \tag{27}$$

$$l \leq \sum_{i \in J} x_{ig} \leq u \qquad g = 1, \cdots, U \tag{28}$$

$$\text{all variables binary} \tag{29}$$

where U is the upper bound of the number of clusters, x_{ig} is 1 if activity i is included in the cluster g, and otherwise 0. The set partitioning constraints (26) make sure that each activity is contained in only one cluster; constraints (27) and the minimisation of (25) imply that the binary variable z_{eg} is 1 if the predecessor activity of e is included in the cluster g but the successor activity is in a different cluster, and otherwise 0; constraints (28) ensure that the cluster size is within the specified range $[l, u]$.

MCC is also NP-hard, and only small problems can be solved to optimality. For our purpose the cluster size constraints (28) are just soft constraints. We can use heuristics to generate good partitions very quickly. Our experimentation with METIS [16] shows that the project with 11,000 activities can be partitioned into 100 balanced parts within 0.1 second and only 384 precedence constraints need to be relaxed.

4 Hierarchical Subgradient Algorithm (HSA)

The upper bound obtained by solving DLRP can be tightened by optimising the Lagrangian Dual Problem(LDP) as

$$\min_{\lambda \geq 0, \mu \geq 0} \quad Z_{LR}(\lambda, \mu) \tag{30}$$

We use the *standard subgradient algorithm* (SSA) [10] which updates the Lagrangian multipliers at the ith iteration (λ^i, μ^i) according to

$$(\lambda^{i+1}, \mu^{i+1}) = \left[(\lambda^i, \mu^i) - \delta^i \frac{Z_{LR}(\lambda^i, \mu^i) - LB^*}{||g^i_\lambda||^2 + ||g^i_\mu||^2}(g^i_\lambda, g^i_\mu)\right]^+ \tag{31}$$

where $[\cdot]^+$ denotes the non-negative part of a vector, δ^i is a scalar step size, LB^* is the best known lower bound, and (g^i_λ, g^i_μ) is a subgradient calculated as

$$g^i_\lambda(k,t) = R_k - \sum_j r_{jk}(\sum_{s=t-p_j+1}^{t} x^i_{js}) \tag{32}$$

and

$$g^i_\mu(j,k) = \sum_t t(x^i_{kt} - x^i_{jt}) - p_j \quad \forall (j,k) \in E \tag{33}$$

where x^i is the optimal solution of DLRP at the ith iteration.

In practice δ^i is reduced by a factor ρ if Z_{LR} is not improved by at least Δ^i after p iterations. The algorithm can terminate when δ^i is small enough to avoid excessive iterations.

The subgradient algorithm tends to converge slowly for problems of high dimensions due to the zig-zag phenomenon [25]. For large RcpspDc problems we observed that the convergence of the precedence multipliers μ was extremely slow using the updating rule (31). The reasons could be

- The contribution of the precedence multipliers in the objective function value Z_{LR} is trivial. It can be even smaller than Δ^i which is used to test if the upper bound is improved. Too small Δ^i can only lead to excessive iterations before δ^i can be reduced.
- In (31) the resource component of the subgradient $||g^i_\lambda||^2$ is much larger than the precedence component $||g^i_\mu||^2$, which may lead to steps too small for the convergence of μ.

Good μ can lead to near-feasible solution with respect to the precedence constraints, which is important for Lagrangian relation based heuristics to produce good lower bound. We will demonstrate this in Section 6.

To accelerate the convergence of precedence multipliers we introduce a *hierarchical subgradient algorithm* (HSA) which has two levels. At the first level we update the multipliers according to (31) for a certain number of iterations i_1 and then move to the second level by just updating the precedence multipliers as

$$\mu^{i+1} = \left[\mu^i - \delta^i \delta^i_\mu \frac{Z_{LR}(\lambda^i, \mu^i)}{||g^i_\mu||^2} g^i_\mu\right]^+ \tag{34}$$

Only δ^i_μ is reduced at the second level if Z_{LR} is not improved after p iterations. After a certain number of iterations i_2 the algorithm will switch back to the first level. This process is repeated until some stopping criterion are met.

5 Lagrangian Relaxation based Heuristic

The Lagrangian relaxation DLRP produces upper bounds for the original NPV problem. But in practice we are interested in finding feasible solutions of high value. We can use the Lagrangian relaxation solution to create a heuristic which created strong solutions. Combining Lagrangian relaxation with list scheduling has been previously successfully applied to different variants of RCPSP [18] [17] problems.

The basic idea is motivated by the intuition that violation of relaxed constraints tend to be reduced in the course of the subgradient optimization. Therefore, a near-feasible solution of the Lagrangian relaxation contains valuable information on how conflicts on constraints can be resolved. In [18] the activities are sorted in the increasing order of the so called $\alpha-$point. Assume the Lagrangian relaxation solution assigned start time $l_j, j \in J$ to each activity. The $\alpha-$point is calculated as $l_j + \alpha * p_j$ which is the earliest time that at least α of the activity has completed if the activity starts at l_j. α is normally evenly chosen between 0 and 1. A parallel list scheduling scheme [11] is then employed to produce feasible solutions. For RCPSPDC left and right shifting techniques are used to further improve the solution quality [17] of the parallel list scheduling using just the start time in the Lagrangian relaxation solution.

We use the parallel list scheduling scheme to generate feasible solution at each iteration of the subgradient algorithm. The list scheduling scheme starts at time $t = 0$, and determines the subset of candidate activities $j \in A$ for scheduling as those whose predecessors have all completed $s_i + p_i \leq t, (i, j) \in L$, and whose start time $l_j \leq t$. The candidate activities $j \in A$ are then scheduled, so s_j is set, in increasing order of l_j, where the resource requirements are satisfied. Candidate activities $j \in A$ with positive cash flow ($c_j > 0$) are moved as early as possible (so it may be the case that $s_j < l_j$). After an activity i is scheduled, its successor j where $(i, j) \in L$ activities may become eligble for scheduling and are added to the candidates A. The process continues until no activity is schedulable at or before time t. We then set $t := t + 1$ and repeat.

A schedule created in this way will almost always have a makespan ms larger than the deadline T. To overcome this we simply modify the Lagrangian relaxation solution $l_j, j \in J$ used to drive the heuristic. We set $l_j := l_i - T + m$ for all $j \in J$. This means there is many more candidate activities at the start of the search, and also allows activities with negative cash flows to be scheduled earler.

6 Experiments

We implemented the algorithms in C++. BOOST version 1.49.0 [1] is used for the max-flow solver and multi-threading. METIS version 5.0 [16] is used to solve the MCC problem. All tests were run on a Dell PowerEdge R415 computer with two 2.8GHz AMD 6-Core Opteron 4184 cpu which has 3M L2 and 6M L3 Cache, and 64 GB memory. We use the same set of parameters for all the tests. For the subgradient algorithm we use $\delta^0 = 1$ in (31), $\delta^0_\mu = 0.01$ in (34), the threshold for

Table 1. Test cases for large RcpspDc

case name	$\|J\|$	$\|L\|/\|J\|$	$\|R\|$	T	NC	(U,$\|E\|$)		
caNZ_def	1410	1.18	7	3040	14	(10,38)	(50,126)	(100,233)
caW	2817	1.26	2	3472	1	(10,120)	(50,314)	(100,578)
caGL	3838	1.16	5	2280	17	(10,59)	(50,174)	(100,269)
caZ	5032	1.36	5	8171	1	(50,274)	(100,427)	
caCH	8284	1.24	4	7251	1	(100,623)	(200,866)	
caZF	11769	1.16	5	6484	1	(100,384)	(200,595)	

significant objective value improvement is $\Delta^i = 0.0001 Z_{LR}(\lambda^i, \mu^i)$, the number of iterations for reducing δ^i is $p = 3$ and the factor $\rho = 0.5$, the maximal number of iterations for each level of HSA is $i_1 = i_2 = 10$, and the maximal number of iterations for the list scheduling heuristic is $i_3 = 20$.

6.1 Test Cases

Table 1 shows six of the eight test cases we obtained from a mining consulting company. The other two have no resource constraints and can be solved to optimality within 2 minutes. The first column in the table is the name of each case. The next two columns are the number of activities $|J|$ and the average number of successors of each activity in the precedence constraints L. The fourth column is the number of resources $|R|$. The fifth column is the number of Natural Clusters (NC) which is the number of clusters without relaxing any precedence constraints. The remaining columns give the pairs of the number of obtained clusters (U) after relaxing the number of precedence constraints (E) by solving MCC. These test cases ranges from about 1400 activities to about 11,000 activities. The average number of successors per activity is small for all of these test cases. However only the smaller canNZ_def and caGL have natural clusters. Even for these two cases the natural clusters are not balanced in size. For example 38 precedence constraints have to be relaxed for canNZ_def to have 10 balanced clusters which is smaller than the number of natural clusters. It can be seen that more precedence constraints have to be relaxed when the number of clusters required increases. We omit here the running times of METIS because all MCC instances for our six test cases in Table 1 can be solved within 0.1 seconds.

6.2 Relaxing Resource Constraints only

Without relaxing precedence constraints we can only solve the three smaller test cases and the results are shown in Table 2. The makespan of the feasible solution with the best *npv*, the upper bound(ub) and lowerbound(lb) are reported in columns 2 to 4. The fifth column is the optimality gap calculated as $(ub - lb)/ub$. The next two columns are the total number of iterations for the subgradient algorithm, and the total cpu time. The last two columns are the number of nodes $|V|$ and number of edges $|A|$ in the network model for solving the Lagrangian relaxation problem. All times are in seconds. Entries in bold are the best over

Table 2. Test results for relaxing resource constraints only

case name	makespan	ub	lb	gap	ite	time	$\|V\|$	$\|A\|$
canNZ_def	3040	**1.199E9**	**1.140E9**	0.0496	81	10370	2874917	5854335
caW	3471	**6.014E8**	4.681E8	0.2217	72	12336	6178671	13449822
caGL	2269	**1.055E9**	**1.021E9**	0.0318	86	60885	6371416	13224808

Table 3. Test results of SSA for comparison with HSA

case name	U	makespan	ub	lb	gap	ite	time
canNZ_def	10	3035	1.202E9	1.047E9	0.128	96	2261
caW	10	—	6.036E8	—	—	62	2103
caGL	10	2237	1.058E9	1.010E9	0.046	89	7887
caZ	100	7969	3.003E8	1.259E8	0.581	100	19357
caCH	200	**7251**	3.031E9	2.326E9	0.232	100	18885
caZF	200	**6337**	3.979E8	2.091E8	0.474	100	11371

entries in Tables 2–5. It can be seen that canNZ_def and caGL have very good optimality gaps which are under 5%, while caW has quite a large gap. Although caW and caGL have similar sizes of network flow model, caGL is much slower to solve. The reason could be that caGL has larger edge capacities which can also affect the performance of max-flow algorithm.

6.3 Relaxing both Resource and Precedence Constraints

We first study how convergence can be affected after relaxing precedence constraints by comparing the standard subgradient algorithm (SSA) and the heirarchical subgradient algorithm (HSA). The maximal number of iterations is set to be 100 for all tests. We use 10 cores to speed up the tests.

The results for SSA and HSA are reported in Table 3 and Table 4 respectively. For the three smaller test cases it can be seen clearly that the upper bounds are trivially worsened by relaxing precedence constraints. However the lower bounds become significantly worse if SSA is used. The test case caW could not even find a feasible solution. In sharp contrast HSA finds a better lower bound for caW than without precedence constraint relaxation. This shows that HSA makes the Lagrangian multipliers associated with precedence constraints converge much faster. For the three larger cases HSA produces much better lower bounds than SSA especially for caZ. However SSA got a better upper bound for caCH. The reason is that HSA only uses half the number of iterations on updating the Lagrangian multipliers associated with the resource constraints. All the following tests will use HSA because of the overwhelming advantages on lower bounds over SSA.

We study how the quality of bounds are affected by the number of clusters. The results are reported in Tables 4 and 5. For the three smaller test cases, both upper bounds and lower bounds consistently become worse with the number of clusters increased. The lower bounds are more adversely affected than the upper

Table 4. Test results of HSA for comparison with SSA

case name	U	makespan	ub	lb	gap	ite	time
canNZ_def	10	3033	**1.200E9**	1.136E9	0.054	100	6330
caW	10	**3456**	6.017E8	**4.803E8**	0.202	100	6657
caGL	10	2254	1.056E9	1.019E9	0.035	100	9523
caZ	100	7931	**2.919E8**	**1.735E8**	0.406	100	23076
caCH	200	**7251**	3.060E9	**2.449E9**	0.200	100	44353
caZF	200	6368	3.952E8	2.394E8	0.394	100	17318

Table 5. Test results for effects of the number of clusters

case name	U	makespan	ub	lb	gap	ite	time
canNZ_def	10	3033	**1.200E9**	1.136E9	0.054	100	6330
canNZ_def	50	3030	1.202E9	1.125E9	0.064	100	1349
canNZ_def	100	**3025**	1.203E9	1.100E9	0.086	100	967
caW	10	**3456**	6.017E8	**4.803E8**	0.202	100	6657
caW	50	3457	6.025E8	4.615E8	0.234	100	4390
caW	100	3460	6.036E8	4.380E8	0.274	100	3699
caGL	10	2254	1.056E9	1.019E9	0.035	100	9523
caGL	50	2228	1.060E9	1.017E9	0.040	100	3574
caGL	100	**2218**	1.060E9	1.010E9	0.047	100	2337
caZ	50	**7909**	2.856E8	1.714E8	0.400	100	31180
caZ	100	7931	**2.919E8**	**1.735E8**	0.406	100	23076
caCH	100	**7251**	**3.023E9**	2.365E9	0.218	100	64758
caCH	200	**7251**	3.060E9	**2.449E9**	0.200	100	44353
caZF	100	6427	**3.860E8**	**2.398E8**	0.379	100	27663
caZF	200	6368	3.952E8	2.394E8	0.394	100	17318

bounds. For the three larger test cases the worsening of upper bounds is more than 1% after the number of clusters is doubled. However the lower bounds become significantly better on caZ and caCH. The reason could be that list scheduling is not robust. But it can also be that the HSA does not converge well on the Lagrangian multipliers related to the precedence constraints. By setting $i_1 = i_2 = 50$ and keeping the maximal number of iterations the same we get lower bound 1.86E8 for caZ with $U = 50$. This suggests more work need to be done to adaptively tune parameters of HSA.

We next study the impact on solution time from using the techniques in the paper. We calculate the speedup factor f_d due to solving DLRP instead of LRP as

$$f_d = ATL/ATD$$

where ATD is the average solution time of DLRP while ATL is the average solution time of LRP. We also calculate the speedup factor f_p due to solving DLRP using multi-cores as

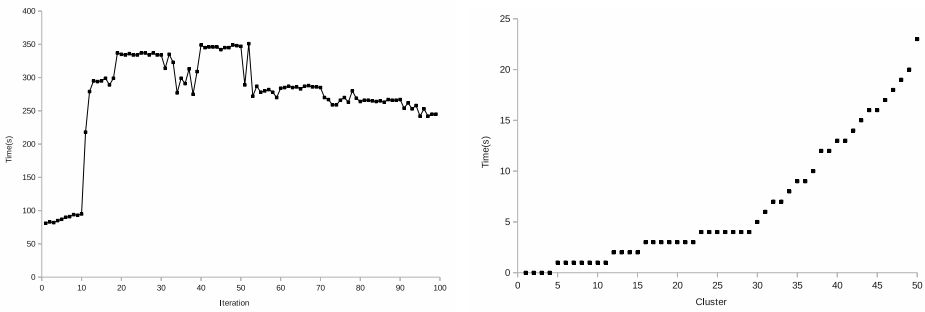
$$f_p = ATD/ATM$$

Table 6. Test results for speedups due to decomposition and parallelization as an effect of the number of clusters

case name	$U = 10$			$U = 50$			$U = 100$		
	f_d	f_p	$f_d * f_p$	f_d	f_p	$f_d * f_p$	f_d	f_p	$f_d * f_p$
canNZ_def	0.94	2.16	2.02	1.68	5.63	9.49	2.46	5.38	13.24
caW	0.92	2.81	2.57	0.63	6.18	3.90	0.76	6.11	4.63
caGL	1.50	4.97	7.43	3.32	5.97	19.81	5.11	5.93	30.29

where ATM is the average solution time of DLRP with multi-cores. We implemented a thread pool using the BOOST thread library. If the number of clusters is larger the number of cores available the longest solution time first rule [9] is used to schedule the threads. Solution time is estimated based on the previous iterations. If the estimation of the solution time produces the same list of threads as using the real solution time, this rule has performance guarantee 4/3 .

The results for smaller cases are reported in Table 6 using 10 threads. It can be seen that the benefit from increasing the number of clusters is small. The reason is that the complexity of the max-flow algorithm depends not just on the size of the cluster but also on the number of precedence constraints on it. The capacities of the edges can also be an important factor. It is interesting to see that f_d is even less than 1. We show the solution time of caW for each iteration with $U = 50$ in Figure 6.3(a). The solution times at the first few solutions are quite small. After the capacities of edges are updated according to the HSA, the solution times increase quickly. We also show the solution time of caW for each cluster with $U = 50$ in Figure 6.3(b). It can be seen that some clusters have much larger solution time than others.



(a) Time for each iteration (b) Time for each cluster

Fig. 1. Solutions times for caW with $U = 50$

We cannot calculate f_d for the larger cases. f_p is similar to those of the smaller cases.

Table 7. Comparison with J and D's methods in 5 hours limit

	LR				J		D	
case name	Makespan	ub	*npv*	$\|U\|$	makespan	*npv*	Makespan	*npv*
canNZ_def	3040	1.199E9	**1.140E9**	1	**3014**	1.040E9	3559	9.810E8
caW	3456	6.017E8	**4.803E8**	10	**3417**	4.670E8	3472	4.740E8
caGL	2254	1.056E9	**1.019E9**	10	**2196**	9.920E8	2283	1.010E9
caZ	**7931**	2.997E8	**1.735E8**	100	8171	1.670E8	8078	1.670E8
caCH	**7251**	3.215E9	**2.449E9**	200	**7251**	1.750E9	8097	1.700E9
caZF	**6368**	3.952E8	**2.394E8**	200	6384	2.290E8	6904	2.290E8

The mining consulting company also provides us with the results from two other research teams J and D. J sets a time limit of 5 hours but does not disclose details of their method. D sets a time limit of 1 hour and is based on Ant Colony Optimisation. We report our results with a time limit of 5h in Table 7 along with J and D's results. Although D's method is the fastest, it has difficulty in finding good solutions when the deadline is tight. It has the largest makespans for all the cases except for caZ. We don't know how it performs if running time is extended to 5 hours. For the three smaller cases we cannot produce solutions with the same makespan as J's. However our *npv* are significantly larger with small increase of the makespan. For the three larger cases our method produces solutions with tightest makespan and best *npv*. For caCH our *npv* is almost 40% higher than J's.

7 Conclusion

We have applied a Lagrangian relaxation method for large RcpspDc problems by relaxing both resource and precedence constraints. By minimising the number of relaxed precedence constraints we can still achieve relatively tight upper and lower bounds. Together with a parallel implementation and a hierarchical subgradient algorithm this approach produced highly competitive results on very large underground mining problems within reasonable computing time. Further work is needed to combine with the CP technologies to improve solution qualities.

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.

References

1. Boost.org, `http://www.boost.org/`
2. Baroum, S., Patterson, J.: The development of cash flow weight procedures for maximizing the net present value of a project. Journal of Operations Management 14, 209–227 (1996)

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)
4. Chaudhuri, S., Walker, R., Mitchell, J.: Analyzing and exploiting the structure of the constraints in the ilp approach to the scheduling problem. IEEE Trans. Very Large Scale Integration (VLSI) Systems 2, 456–471 (1994)
5. Cherkassky, B., Goldberg, A.: On implementing push-relabel method for the maximum flow problem. pp. 157–171. IPCO '95
6. 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)
7. Demeulemeester, E.L., Herroelen, W.S.: New benchmark results for the resource-constrained project scheduling problem. Management Science 43(11), 1485–1492 (1997)
8. Doersch, R., Patterson, J.: Scheduling a project to maximize its present value: A zero-one programming approach. Management Science 23, 882–889 (1977)
9. Dósa, G.: Graham's example is the only tight one for $P||C_{max}$. Annales Universitatis Scientarium Budapestiensis de Rolando Eötös nominatae, Sectio Mathematica 47, 207–210 (2004)
10. Fisher, M.: The lagrangian relaxation method for solving integer programming problems. Management Science 27, 1–18 (1981)
11. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Tech. J. (45), 1563–1581 (1966)
12. 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)
13. 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)
14. 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)
15. Johnson, E., Mehrotra, A., Nemhauser, G.: Min-cut clustering. Mathematical Programming 62, 133–151 (1993)
16. Karypis, G.: METIS, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 5.0 (2011), `http://glaros.dtc.umn.edu/gkhome/views/metis`
17. Kimms, A.: Maximizing the net present value of a project under resource constraints using a lagrangian relaxation based heuristic with tight upper bounds. Annals of Operations Research 102, 221–236 (2001)
18. Möhring, R.H., Schulz, A.S., Stork, F., Uetz, M.: Solving project scheduling problems by minimum cut computations. Management Science 49(3), 330–350 (2003)
19. 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)
20. Padman, R., Smith-Daniels, D.E., Smith-Daniels, V.L.: Heuristic scheduling of resource-constrained projects with cash flows. Naval Research Logistics 44, 365–381 (1997)

21. Russell, A.H.: Cash flows in networks. Management Science 16(5), 357–373 (1970)
22. Schutt, A., Chu, G., Stuckey, P., Wallace, M.: Maximizing the net-present-value for resource constrained project scheduling. In: Beldiceanu, N., Jussien, N., Pinson, E. (eds.) International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR). p. to appear. LNCS, Springer (2012)
23. Siek, J.G., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library: User Guide and Reference Manual. Addison-Wesley (2001)
24. 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)
25. Zhao, X., Luh, P.B.: New bundle methods for solving lagrangian relaxation dual problems. J. Optim. Theory Appl. 113, 373–397 (May 2002), http://dx.doi.org/10.1023/A:1014839227049