A new scheduling technique for the resource–constrained project scheduling problem with discounted cash flows

Pieter Leyman[1] and Mario Vanhoucke[1,2,3]

[1]Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Gent (Belgium), pieter.leyman@ugent.be, mario.vanhoucke@ugent.be
[2]Operations and Technology Management Centre, Vlerick Business School, Reep 1, 9000 Gent (Belgium)
[3]Department of Management Science and Innovation, University College London, Gower Street, London, WC1E 6BT (United Kingdom)

Abstract

In this paper we discuss the resource–constrained project scheduling problem with discounted cash flows (RCPSPDC). We introduce a new schedule construction technique which moves sets of activities to improve the project net present value (NPV) and consists of two steps. In particular the inclusion of individual activities into sets, which are then moved together, is crucial in both steps. The first step groups activities based on the predecessors and successors in the project network, and adds these activities to a set based on their finish time and cash flow. The second step on the contrary does so based on the neighbouring activities in the schedule, which may but need not include precedence related activities. The proposed scheduling method is implemented in a genetic algorithm (GA) metaheuristic and we employ a penalty function to improve the algorithm's feasibility with respect to a tight deadline. All steps of the proposed solution methodology are tested in detail and an extensive computational experiment shows that our results are competitive with existing work.

Keywords: Net present value; resource–constrained project scheduling; metaheuristics

1 Introduction

The resource–constrained project scheduling problem (RCPSP) has been extensively discussed in literature in the past few decades. Subsequently, the problem has been covered in a multitude of extensions and variations (Hartmann and Briskorn, 2010). Whereas the basic RCPSP and most its extensions focus on total project duration minimization, other variations aim to minimize resource idle time, minimize project costs or maximize the project net present value (NPV).

In this paper, we focus on the RCPSP with discounted cash flows (RCPSPDC) and aim to maximize a project's total NPV. The RCPSPDC can be seen as a variant to the RCPSP in which each activity has a cash flow, which can be either positive or negative. Furthermore, since the focus is on maximizing the NPV, a deadline is imposed on the project to avoid that activities with negative cash flows may be delayed indefinitely.

The remainder of this paper is organized as follows. Section 2 start with a literature overview of the existing papers on the RCPSPDC. In section 3 we discuss the mathematical problem formulation, whereas in section 4 we go into detail about our schedule generation procedure and illustrate all of its steps on a problem example. Section 5 gives an overview of our proposed metaheuristic and results of a computational experiment are shown in section 6. We finish with a conclusion in section 7.

2 Literature overview

Overviews of the existing literature on the RCPSPDC and its extensions have been given by Herroelen et al. (1997) and Mika et al. (2005). In this manuscript we briefly discuss all papers to–date which handle the RCPSPDC as formulated in section 3. We first discuss the existing exact methods, then the multi–pass heuristics, and finally the metaheuristic procedures.

Exact procedures: Yang et al. (1995) employ a branch–and–bound procedure to solve the RCPSPDC and make use of a depth–first search. The authors apply node fathoming rules to reduce the size of the tree and show that these rules significantly reduce computation times. Icmeli and Erengüç (1996) also propose a branch–and–bound procedure which introduces additional precedence relations to avoid resource conflicts. Branching is done according to the minimal delaying alternatives concept proposed by Demeulemeester and Herroelen (1992) and their results outperform other existing procedures. Another branch–and–bound procedure is used by Baroum and Patterson (1996) and tested on instances from Patterson's dataset, with networks consisting of up to 51 activities. Vanhoucke et al. (2001) propose a branch–and–bound procedure for the RCPSPDC based on an exact recursive method for the resource–unconstrained case. The procedure can solve relatively small problems to optimality within a limited computation time. Schutt et al. (2012) use lazy clause generation for the RCPSPDC and come up with three appropriate propagators for maximizing the NPV. These propagators are tested using a branch–and–bound algorithm and several binary search heuristics. The authors compare their results with those of Vanhoucke et al. (2001) and conclude that their proposed method finds both better and a higher number of feasible solutions than those of the benchmark.

Multi–pass heuristics: Russell (1986) proposed the first heuristic for the RCPSPDC and uses six rules to solve the problem. The work has shown that heuristics which perform well for duration minimization do not necessarily provide good results for NPV maximization. A backward scheduling method is used by Smith-Daniels and Aquilano (1987), with a lump–sum payment at activity completion and cash outflows occurring at the start of each activity.

Metaheuristics: Zhu and Padman (1999) apply a tabu search procedure to the RCPSPDC and show considerably better results than any single–pass heuristic available. Kimms (2001) employs Lagragian relaxation and derives tight upper bounds. Based on these upper bounds feasible solutions are constructed, which are shown to be very close to the optimal solutions. Selle and Zimmermann (2003) schedule large–scale projects subject to resource constraints and temporal constraints. A new bi–directional scheduling approach is proposed which simultaneously schedules activities forward and backward. The new approach manages to outperform existing scheduling approaches. Vanhoucke (2010) employs a scatter search metaheuristic and makes use of a bi–directional schedule generation scheme and a recursive search method to improve the project NPV. The results are compared with both the exact procedure of Vanhoucke et al. (2001) and several different metaheuristics coded by the author, including the genetic algorithm of Vanhoucke (2009). It is concluded that the proposed method outperforms all others. Next, Gu et al. (2012) discuss the RCPSPDC for large project instances. The authors apply Lagrangian relaxation to projects with up to 11,000 activities and employ three improvements to ensure scalability of their algorithm.

These steps involve the relaxation of precedence constraints, parallel implementation and a hierarchical subgradient algorithm. The results produced are highly competitive given a reasonable computation time. Gu et al. (2013) improve on the results of Schutt et al. (2012) by making use of a Lagrangian relaxation based forward–backward improvement heuristic, to ensure tight deadlines are met. The authors' forward–backward method used is that of Li and Willis (1992), who use the activity start (finish) times of the previously generated forward (backward) schedule to construct the current schedule. The authors compare their procedure with the algorithm of Vanhoucke (2010) based on a 5 minutes time limit and find that their method performs best. It is however important to note that the results of Vanhoucke (2010) are based on a 5,000 schedule limit instead of a run time restriction and have an average computation time of 2.2 seconds.

Based on this overview, we conclude that the papers of Vanhoucke (2010) and of Gu et al. (2013) are the most recent ones which discuss a metaheuritic solution methodology for the RCPSPDC. Thus, we will compare the results of our algorithm with the procedures of these papers in section 6.

3 Problem formulation

A project can be represented as an activity–on–the–node (AoN) network $G(N, A)$ with N representing the nodes or project activities, and A the network arcs or precedence relations between the activities. For the precedence relations a time–lag of zero is assumed. Each activity, numbered from start dummy 0 to end dummy $n+1$, has a duration d_i, renewable resource demand r_{ik} and cash flow c_i. The latter is composed by discounting the pre-specified cash flow cf_{it} of an activity at time t to its finish time and can be mathematically formulated as follows: $c_i = \sum_{t=1}^{d_i} cf_{it} e^{\alpha(d_i - t)}$. Each renewable resource k has a limited constant availability of a_k. The decision variables f_i contain the finish time for each activity i. Finally, the project has a deadline δ_{n+1}.

The RCPSPDC was proven to be NP–hard by Blazewicz et al. (1983) and can be represented as $m, 1|cpm, \delta_n, c_i|npv$ according to the classification scheme of Herroelen et al. (1999), and as $PS|prec|\sum C_i^F \beta^{C_i}$ according to Brucker et al. (1999).

Mathematically, the problem is formulated as follows:

$$\text{Maximize} \sum_{i=1}^{n} c_i \cdot e^{-\alpha f_i} \tag{1}$$

Subject to:

$$f_i \leq f_j - d_j, \quad \forall (i,j) \in A, \tag{2}$$

$$\sum_{i \in S(t)} r_{ik} \leq a_k, \quad k = 1, \ldots, R; \; t = 1, \ldots, \delta_{n+1}, \tag{3}$$

$$f_{n+1} \leq \delta_{n+1}, \tag{4}$$

$$f_i \text{ integer}, \quad \forall i \in N \tag{5}$$

The objective function (1) aims to optimize the project NPV, whereas constraints (2) ensure precedence feasibility. Constraints (3) impose the renewable resource limits with $S(t)$ the set of activities in progress at time t. Constraint (4) enforces deadline feasibility and finally constraints (5) ensure the decision variables are integer.

4 Schedule generation

In this section, we discuss the schedule generation employed by the metaheuristic (see section 5) and all of the included steps. A distinction is made between two variants of the scheduling approach based on the percentage of activities with a negative cash flow. If this percentage is lower than or equal to 50%, we start with a forward schedule generation scheme (SGS) and subsequently delay activities. If more than 50% of the activities have a negative cash flow, a backward SGS is first applied followed by the advancing of activities. The overall flow of the schedule generation can be seen in figure 1. The subsequent subsections go into detail about each individual step of the scheduling process.

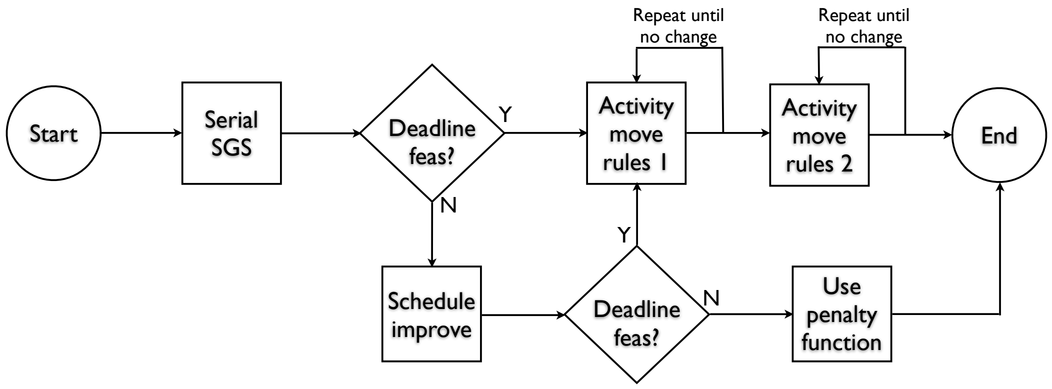


Figure 1: Schedule generation flow

4.1 Initial schedule and deadline feasibility

The first step in the scheduling process is the construction of an initial deadline–feasible schedule. To construct this schedule we use the well–known forward serial schedule generation scheme (SSGS) (Kolisch, 1996) if no more than half of the activities have a negative cash flow. If the result is feasible with respect to the project deadline, we continue with the first set of activity move rules as described in section 4.2. If however, the schedule is infeasible (*D–Infeas*) we apply the forward–backward improvement method of Li and Willis (1992) to reduce the project duration, until no further improvement is possible. If the resulting schedule after applying Li and Willis (1992) is feasible, we proceed with the activity move rules. If even after the schedule improvement method the schedule is still infeasible, we add the following penalty function to the project NPV:

$$\begin{cases} NPV = -Y_1 + NPV_{infeas} \cdot Y_2^{f_{n+1}-\delta_{n+1}} & \text{if } NPV_{infeas} \geq 0 \\ NPV = -Y_1 + \frac{NPV_{infeas}}{Y_2^{f_{n+1}-\delta_{n+1}}} & \text{otherwise} \end{cases} \tag{6}$$

with Y_1 and Y_2 variables to be tested in section 6.1, and NPV_{Infeas} the NPV of the *D–Infeas* schedule.

- Y_1 represents a large fixed value which is subtracted from the project NPV to ensure that the infeasible solution's NPV is considerably worse than that of any feasible solution.

- Y_2 aims to penalize the NPV based on the difference between the solution's project duration and the project deadline. As such this parameter has a fractional value in order to exponentially reduce the project's NPV. The value of the parameter furthermore depends on the absolute value of the NPV, with a higher absolute NPV requiring a parameter value closer to 1 than a lower absolute NPV.

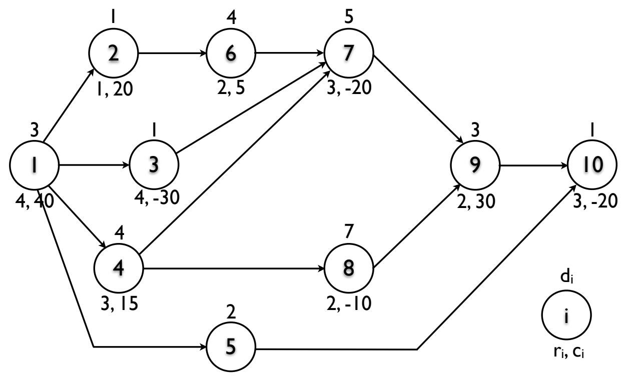
As an example, assume a project with a total duration of 19, a project NPV of 800 and a deadline of 17. If $Y_2 = 0.9$ we first adjust the NPV by multiplying 800 with 0.9^{19-17} and the remaining NPV is 648. We then subtract Y_1 from this value, assume $Y_1 = 1,000$, and receive -352. This way the NPV of the *D–Infeas* project has been reduced from 800 to -352. If by comparison the project duration is 18 instead of 19, the adjusted NPV would be -280. As such, a schedule with a lower deadline violation is penalized less than one with a higher deadline violation. Both are however reduced to a value well below that of NPV_{Infeas}.

If the project NPV is on the contrary relatively small, assume 80, and the deadline is again 17, the corrected NPV would be -935.20 for a project duration of 19 and -928 for a duration of 18. In this case the absolute difference between both corrected values is much smaller than it was for a larger project NPV (7.2 versus 72). If we however apply a lower value for Y_2, e.g. 0.50, the corrected NPV becomes -980 and -960 respectively. As a result, the difference between both corrected NPV has increased (20 versus 7.2), more clearly distinguishing both schedules from one another.

In case more than half of the activities have a negative cash flow, the backward instead of the forward scheduling approach is selected. If the resulting schedule proves to be infeasible with respect to the project deadline, we again apply the method of Li and Willis (1992) until no improvement can be found. The schedule is then once more evaluated. If it is still *D–Infeas* we use the penalty function, otherwise we move on to the activity move rules.

4.2 Activity move rules

This subsection gives an overview of the rules applied to delay (advance) activities, starting from the initial forward (backward) schedule constructed by the SSGS. In order to illustrate these rules, we use an example which is shown at the top of figure 2. We assume a single renewable resource with an availability of 5 and a project deadline of 22. The initial schedule, based on the forward SSGS ($\leq$50% negative cash flows), is shown at the bottom of figure 2 with the renewable resource (RR) on the vertical axis and the time on the horizontal axis. The activities are scheduled according to the priority list (PL) (1, 2, 3, 4, 6, 7, 5, 8, 9, 10). The initial NPV based on a discount rate of 1% is 25.72 (= 38.82 + 19.22 - 28.82 + 13.85 - 4.30 + 4.62 - 17.56 - 8.61 + 25.06 - 16.54). Note that the schedule shown is deadline–feasible, and no schedule improvement is needed. Finally, since we start from a forward schedule, we aim to delay sets of activities.



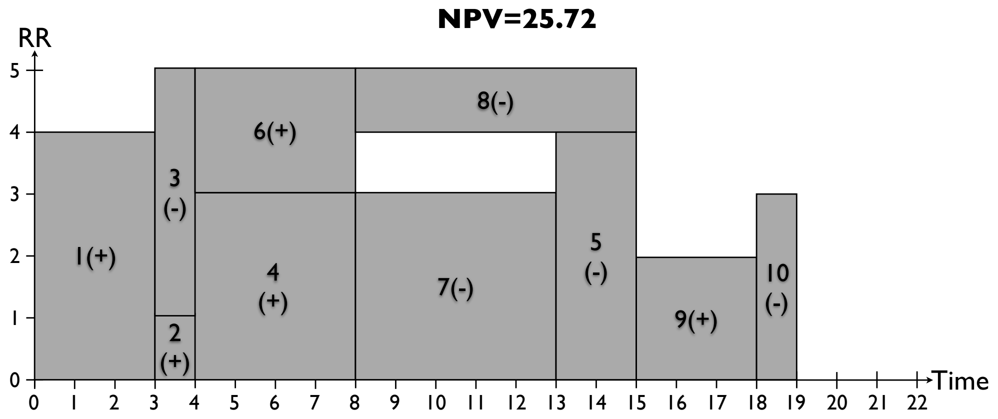


Figure 2: Example 1: Network & initial schedule example

4.2.1 Network–based moves

The first set of rules delays or advances activities based on the precedence relations in the project network. Depending on whether the initial schedule is an earliest finish (forward scheduling) or latest finish (backward scheduling) schedule, the procedure aims to delay or advance activities. First, we discuss the delay algorithm in detail and then we highlight the differences when activities should be advanced.

With an initial forward schedule, we delay both individual activities with a negative cash flows and sets of activities with a negative cumulative NPV. Starting from the last activity in the PL each activity is evaluated with respect to its cash flow. If the activity cash flow is negative, the activity finish time is not equal to its latest finish time and a delay is possible based on the successors, the algorithm delays the activity as much is possible. If however a delay is impossible due to at least one successor with a start time equal to the activity's finish time, algorithm 1 is applied. This recursive method includes all activities which should be delayed together in a set. Starting from the current activity it adds all successors which start immediately after this activity and then recursively calls the procedure again for each of these successors. For each of those, all predecessor activities with both a negative cash flow and a finish time not smaller than the initial activity's finish time are also added to the activity set. Note that algorithm 1 does not take the predecessors of the start activity into account, because these will be considered in a next iteration. Once this set is complete and algorithm 1 returns to the start activity, the cumulative NPV of the set is calculated and evaluated. If it is negative, the minimum delay is calculated based on the earliest successor not in the set, and this for all activities in the set. The global minimum of all these minima then constitutes the maximum allowable delay and can be formulated as follows: $\delta = min_{i \in setAct, k \in S_i, k \notin setAct} f_k - d_k - f_i$. Then each activity in the set has its finish time increased by this δ. If the schedule is feasible with respect to the renewable resources the current finish times are retained. If the schedule is infeasible we decrease δ by 1 and continue until we find a feasible solution or until δ reaches 0. When the search for a delay is complete, the procedure moves on the previous activity in the PL. Finally, the algorithm is repeated until no more changes occur and the procedure reaches the first activity in the PL without delaying any activity.

Algorithm 1 Get all successors

```
GetAllSuc (current activity $i$, start activity $k$, $set$[], $cumNPV$)
  $\forall j \in S_i$
      If $j \notin set \land f_j - d_j = f_i$
          Add $j$ to $set$
          Add $NPV_j$ to $cumNPV$
          GetAllSuc ($j$, $k$, $set$, $cumNPV$)
  $\forall j \in P_i$
      If $j \notin set \land f_j + d_j = f_i \land j \neq k \land f_j \geq f_k \land f_j < LF_j \land CF_j < 0$
          Add $j$ to $set$
          Add $NPV_j$ to $cumNPV$
          GetAllSuc ($j$, $k$, $set$, $cumNPV$)
  Return set
```

If the initial schedule is a backward one, the goal is to advance both individual activities with a positive cash flow and sets of activities with a positive cumulative NPV. This states the first major difference with the delay procedure, where we aimed to delay activities. As a consequence, we now have to consider the finish time of predecessors instead of the start time of successors when determining the potential change in activity finish time. Finally, algorithm 1 needs to be inverted to a **GetAllPred** variant which finds all predecessors of an activity and any successors with both a positive cash flow and a finish time no more than that of the start activity.

It is important to note that at most half of the activities are considered for a move. Recall that we start from a forward (backward) schedule if at most (less than) half of the activities has a negative cash flow meaning that at most half of the activities has to be considered for a delay (advance).

To illustrate these rules we go back to the example. Recall that the PL of the example was (1, 2, 3, 4, 6, 7, 5, 8, 9, 10). Starting from the final activity, we first consider activity 10. Since it has a negative cash flow and no successors it is delayed until time 22. Next is activity 9 which however has a positive cash flow and is skipped. We move on to activity 8 which has a negative cash flow but cannot be delayed due to its successor 9. Algorithm 1 is used and returns the set {8, 9} since 9 is the only successor of 8. No other activities are included since 9's only other predecessor is 7, but 7 finishes 2 time units before the start of 9. The cumulative NPV of both 8 and 9 is positive so they are not considered for delay. To further illustrate why only 8 and 9 are in the set, consider the top left of figure 3 which shows the project network but with only those precedence relations which constitute equalities in the mathematical model of section 3. In the figure activity 9 is only connected to 8, which still has 4 as a predecessor, but this activity is not taken into account. This way the set is limited to the activities 8 and 9. Next in the PL is activity 5 which has a negative cash flow and whose only successor is 10. As such, activity 5 can be delayed beyond activity 9 to time 21. Note that due to this delay, the ordering of the activities in the schedule is changed. After 5 also activity 7 can be delayed because its only successor is 9 which is scheduled later. Activities 6 and 4 are not delayed because they have a positive cash flow, whereas 3 cannot be delayed due to the renewable resource constraint. Finally, activities 2 and 1 are also not eligible for delay because of their positive cash flow.

Since at least one delay has occurred, the procedure starts again. Activities 10 and 9 can be skipped because the former is scheduled at its latest finish time and the latter has a positive cash flow. Next is activity 8 which cannot be delayed because of its successor 9. If algorithm 1 is applied this time, it returns the set {7, 8, 9} because 7 as a predecessor of 9 now has a finish time equal to its successor's start time. The cumulative NPV of the three activities is negative and their maximum allowable delay is 1. Since this delay is feasible with respect to the renewable resource, the three activities are all delayed by 1 time unit. The top right graph of figure 3 illustrates that activity 7 is now also added to the set because its finish time equals activity 9's start time and hence both are connected. No further delays are possible both in this iteration and the procedure's next, so it terminates. The resulting schedule can be found at the bottom of figure 3.

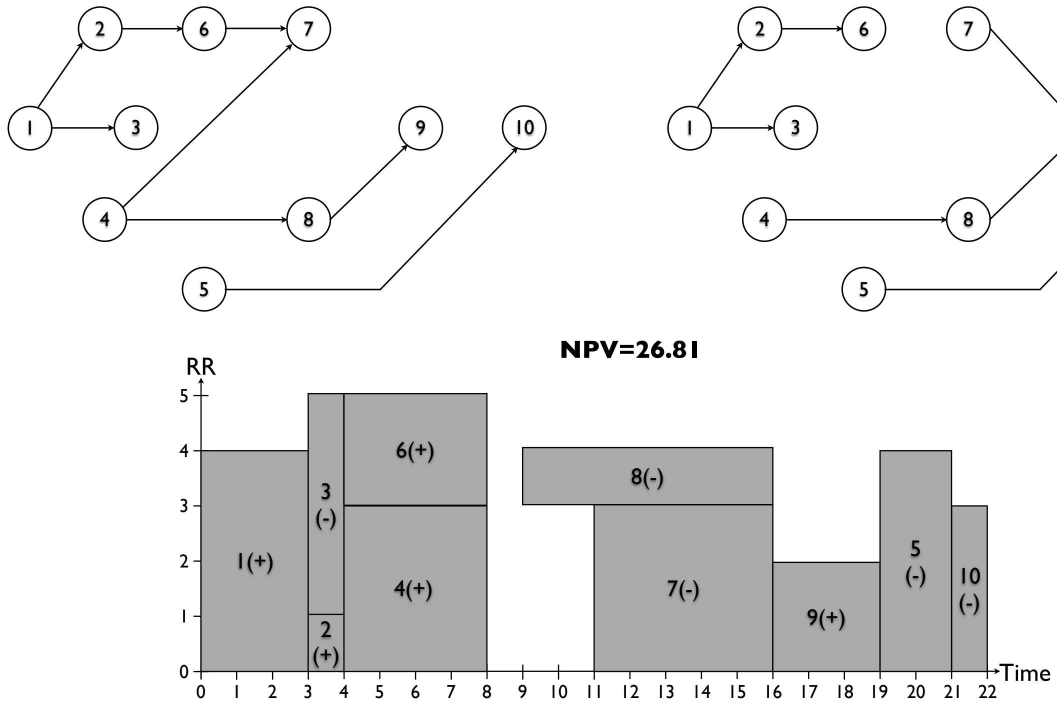


Figure 3: Example 1: Network–based delays in the example

4.2.2 Schedule–based delays

The second set of rules delays or advances activities based on neighbouring activities in the project schedule, which can but need not be precedence related. Whereas in section 4.2.1 activities were grouped together based on their precedence relations, we here group activities together based on their neighbours, which means that one activity's finish time equals another's start time. These neighbouring activities may be precedence related, but can also be scheduled one after another because of the renewable resources. This way, the first set of rules is extended, because these rules may not delay activities due to a resource conflict with other activities. In such cases, it may however be possible that if all of these activities would be considered together, in spite of the absence of any precedence relations between some of them, a delay would be beneficial. Algorithm 2 shows how to find all such neighbouring activities. The manner in which delays occur and with what δ are the same as for the network–based delays in section 4.2.1.

Algorithm 2 Get all later neighbours

```
GetLaterNeighb (current activity $i$, start activity $k$, $set[]$, $cumNPV$)
  $\forall j \in N \wedge f_i = f_j - d_j$
      If $j \notin set$
          Add $j$ to $set$
          Add $NPV_j$ to $cumNPV$
          GetLaterNeighb ($j$, $k$, $set$, $cumNPV$)
  $\forall j \in N \wedge f_i - d_i = f_j$
      If $j \notin set \wedge j \neq k \wedge f_j \geq f_k \wedge f_j < LF_j \wedge CF_j < 0$
          Add $j$ to $set$
          Add $NPV_j$ to $cumNPV$
          GetLaterNeighb ($j$, $k$, $set$, $cumNPV$)
  Return set
```

Just like for the first set of rules, we distinguish between a delay and an advance case. The differences between a forward and backward approach are however the same as those discussed in section 4.2.1 and are hence not repeated here. Obviously, in this case we also need a reverse version of algorithm 2 to find an activity's earlier neighbours.

Also, similar to the first set of rules, only at most half of the activities are considered for a move.

Let us apply this second set of rules to the example. Starting from the back of the PL and the schedule of figure 3 it should be clear that activities 10, 9, 8, 5 and 7 should not be considered anymore since they cannot be delayed any further. The only remaining activity with a negative cash flow is 3, but as stated in section 4.2.1 there are insufficient renewable resources available to allow for a delay. This means that the first set of rules will never delay activity 3 because none of its successors have a start time equal to activity 3's finish time. If we however apply algorithm 2 we notice that both activities 4 and 6 have a start time equal to 3's finish time, hence both are added to the activity set. This way, the resulting set is {3, 4, 6} and has a negative cumulative NPV. The maximum allowable delay is 1, because of activity 8 as a successor of 4, and is feasible with respect to the renewable resource. To further illustrate why exactly these activities are included in the set, consider both graphs at the top of figure 4 which show the neighbours of each activity both before and after the delay.

Although the procedure for this second set of rules is repeated because a change has occurred, no further improvements are possible. The bottom of figure 4 shows the example schedule after the second set of rules has been applied, which is also the optimal one.

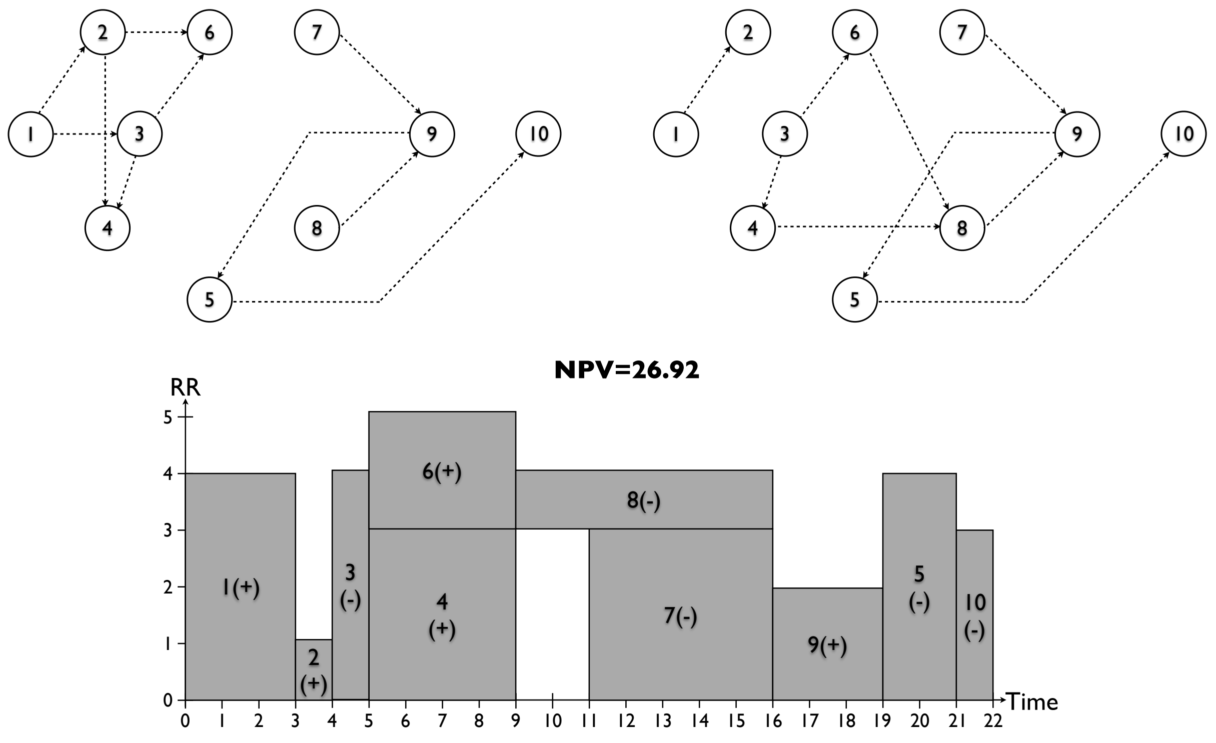


Figure 4: Example 1: Schedule–based delays in the example

Note that for both sets of activity move rules, the activities are considered starting from the last activity in the PL. This is done to allow the enveloping metaheuristic (see section 5) to fully exploit its potential. Let us illustrate the potential problems of not using the discussed approach, but for instance start with the activities which have the *smallest cumulative NPV*, based on the example network in figure 5. The project deadline is 15, the renewable resource availability is 5 and the discount rate is again 1%. If we start from an initial schedule with finish times {0, 1, 5, 9, 10, 10} then it should be clear that improvement is possible, as both activities 4 and 5 have a negative cash flow and are only succeeded by the dummy end activity. The optimal solution in this case is to delay activity 5 to time 15, and delay 4 to 14, as can be seen from the bottom left schedule in figure 5. If we would however employ a heuristic such as the *smallest cumulative NPV first* rule, activity 4 will always be delayed first, as can be seen in the bottom right schedule in figure 5. What is worse is that regardless of the activity ordering, meaning that regardless of which one of both activities has the highest priority and occurs after the other in the PL, activity 4 will be moved first. This means that not only will our solution always be suboptimal, it will also always be the same independent of the PL provided by the metaheuristic. Thus, we have chosen to consider activities for delay in the backward order of the PL, rather than a static heuristic like the one discussed here. Initial tests with heuristics such as *smallest cumulative NPV first* and *largest finish time first* confirmed this reasoning.

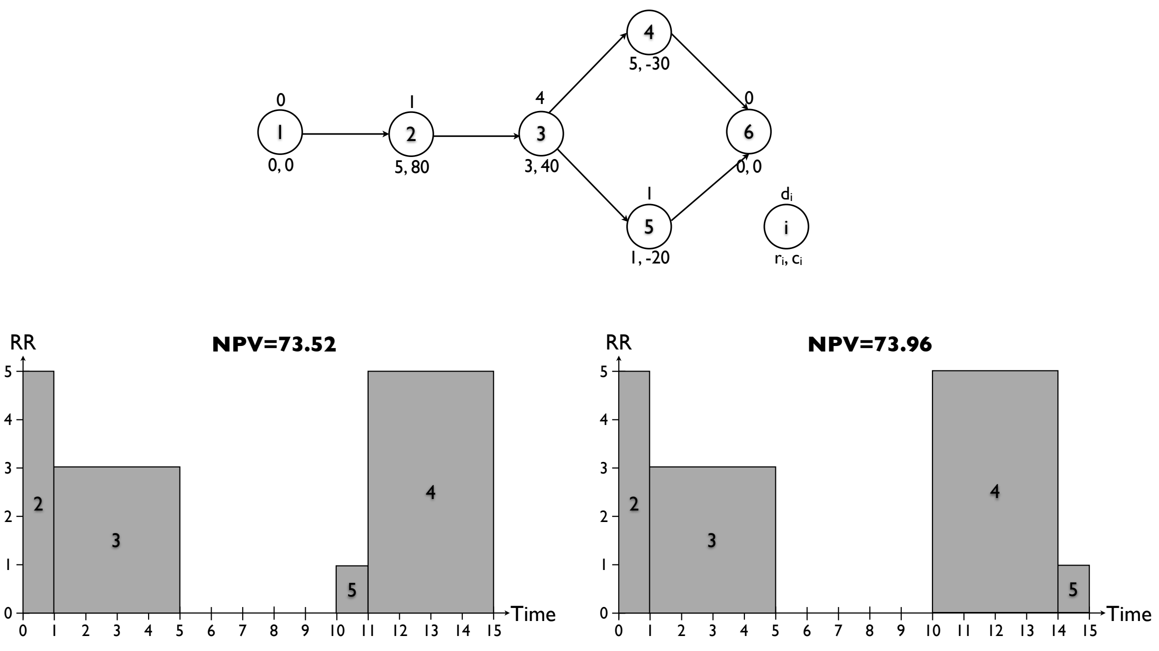


Figure 5: Example 2: Network, optimal and suboptimal schedule

5 Genetic algorithm

Whereas in the previous section the focus was on the schedule generation of a solution, we here discuss the proposed metaheuristic, namely a genetic algorithm (GA).

The GA was first proposed by Holland (1975) and is inspired by evolutionary biology. The technique makes use of such operators as selection, crossover and mutation to combine existing solution into new ones, and has already extensively been used in existing project scheduling literature. A GA typically has a selection operator which selects parent elements to be combined in new offspring using a crossover operator and which constitutes the intensification step of the algorithm. A small percentage of these children are then mutated to diversify the population. Finally, a population update is applied to reduce the population size by retaining the best parents and replacing the rest by the best children.

We continue this section by going into detail about the selected solution representation, and then discuss each part of our GA in more detail. The differences between two options of our GA are discussed in detail in the following subsections.

5.1 Representation

In section 4.2.2 we emphasized the importance of delaying activities in the order of the solution's PL. We can further improve upon this by using the topological order (TO) instead of the PL representation (Valls et al., 2003, 2004). The TO representation as used by Debels et al. (2006) not only ensures the ordering of activities is precedence feasible, but also takes the actual activity start or finish time into account. Once all activity move rules have been applied to a schedule, its corresponding PL is transformed into a finish time ordered activity list. Finally, if two activities have the same finish time, we break ties by randomly selecting one of them to go first in the TO representation.

To illustrate the TO we take another look at the first example. Recall that there, we started with the PL (1, 2, 3, 4, 6, 7, 5, 8, 9, 10). Based on the final schedule of figure 4 the TO representation becomes (1, 2, 3, 6, 4, 7, 8, 9, 5, 10), with 6 randomly chosen from {4, 6} as tie–breaker to go earlier in the list since both have the same finish time, and 7 randomly chosen from {7, 8}.

5.2 Initial population

The initial population is generated by creating random precedence feasible PLs for each element in the population. Whereas typically the initial population is larger than the actual population, our tests indicated that better results are achieved by simply copying the initial population as the start population of the metaheuristic. This initial population generation is the same for both GA1 and GA2.

5.3 Selection

For the selection operator of our GA we have chosen to implement two alternatives. Selection 1 for GA1 constitutes a four–tournament selection for both parents. This implies that for both the father and mother four elements are randomly selected from the population, of which in both cases the best one is retained.

In the case of GA2 we employ an elite selection, which means that the father is always selected out the pool of best X_2 elements. This does however not mean that the set of best X_2 elements is always the same. Once the population as a whole has been updated, the set with size X_2 is updated as well since the new elements may contain better solutions. Each element in the population is evaluated with respect to its NPV and the best X_2 elements are stored as an elite set. This way this subset always contains the best X_2 elements once a population update has been done. The mother on the contrary is still chosen using a four–tournament selection.

Other selection operators for both parents in GA1 and the mother in GA2, such as a roulette wheel and rank selection were tested, but the selection discussed earlier performed best.

5.4 Crossover

Our crossover operator is a one–point crossover as typically used in a GA, for both GA1 and GA2. We also tested the two–point crossover, the MCUOX (Ulusoy et al., 2001), and the partially–mapped crossover, but they were all outperformed by the one–point crossover.

5.5 Mutation

For both GA1 and GA2 we use the same mutation operator, namely a two–activity swap. The difference between both algorithms however is the employed mutation rates R_1 and R_2. For GA1 we expect that a relatively low mutation rate is sufficient since its methods correspond with those typically used in literature. GA2 on the other hand requires a higher rate to serve as counterweight to the elite selection of section 5.3 to ensure the population is diverse enough. Our results of section 6.1 confirm these assumptions.

Alternatively, the mutation operators scramble, inversion and insertion were tested, but they performed worse than the proposed swap operator.

5.6 Evaluation and population update

For updating the population we retain the best X_1 parents for GA1 and best X_2 for GA2. The rest of the parents are replaced by the best newly generated children. Note that the X_2 parameter also determines the size of the pool out of which the father is always selected in GA2.

6 Computational results

In this section we show and discuss our computational results by first configuring both GAs, and then by comparing with the best known results from literature.

In terms of test data, we have chosen to use the same projects as employed by Vanhoucke (2010), downloaded from *www.projectmanagement.ugent.be*. On this site the author's solutions, computation times and employed upper bounds are also available for each datafile. The dataset itself consists of 720 networks with 25, 50, 75 or 100 activities. Each datafile can be executed with 6 different cash flow files, with the percentage of negative cash flows (*%Neg*) ranging from 0% to 100%, in steps of 20%. Furthermore, a deadline is imposed based on the optimal RCPSP project duration of the procedure by Demeulemeester and Herroelen (1992) for the projects with 25 activities and on the best known heuristic solutions of Debels and Vanhoucke (2007) for a higher number of activities. This minimum project duration is then increased by a specific percentage (*D–Incr*), ranging from 5% to 20% in steps of 5%, and constitutes the project deadline. As such, the problem set contains 720 * 6 * 4 = 17,280 problem instances. The order strength (OS) and resource constraindness (RC) of the networks are both either 0.25, 0.50 or 0.75. The resource usage (RU) is 2 or 4. Finally, we employ a discount rate (*DiscRate*) of 1%. A summary of the parameters of the dataset is given in table 1.

Parameter	**Values**
#Act	25, 50, 75, 100
%Neg	0, 20, 40, 60, 80, 100
D–Incr	5, 10, 15, 20
OS	0.25, 0.50, 0.75
RC	0.25, 0.50, 0.75
RU	2, 4
DiscRate	1%

Table 1: Parameters dataset

6.1 Configuration of the algorithm

In this subsection we configure both our GAs. We first give an overview of the values of the parameters of both algorithms and of the penalty function. Then we show that the proposed order in which the two sets of activity move rules are applied is the best and illustrate the relevance of both steps. Finally, we compare both GAs and also show the results if 5,000 random schedules had been generated instead of making use of the proposed metaheuristic. This way the added value of our GA approaches is also emphasized. Each time we make use of the 5,000 schedules termination criterion.

The parameters that were tested are the population sizes P_1 and P_2, the number of retained elements X_1 and X_2 and the mutation rates R_1 and R_2. Computational experiments showed an optimal P_1 and P_2 equal to 50, X_1 equal to 10 and X_2 equal to 5, and a mutation rate R_1 of 20% and R_2 of 95%. Note that the assumption of GA2 needing a higher mutation rate R_2 than R_1 of GA1 as stated in section 5.5 is confirmed. The tested population sizes range from 25 to 100 in steps of 5, whereas the number of retained elements R_1 and R_2 tested range from 1 to 15 in steps of 1. Finally, the mutation rates were tested in steps of 5%, from 5% to 95%.

The Y_1 and Y_2 parameters of the penalty function have also been tested.

- The tested values for Y_1 range from $15,000$ to $25,000$ in steps of $1,000$ and its optimal value equals $20,000$, much larger than that of any feasible solution's NPV.
- Y_2 was initially tested with values between 0.50 and 0.95 in steps of 0.05. Additional finetuning of the latter was done to further improve the performance of the penalty function. The parameter's optimal value was found to depend on the absolute value of the average NPV ($AvNPV$) of all test instances with a fixed percentage of negative cash flows. The values for Y_2 are found in table 3. It should be clear based on the table that the suggested relation does indeed exist. As the absolute value of $AvNPV$ decreases, so does the factor Y_2. Observe in table 3 that starting from 0% negative cash flows Y_2 decreases along with the absolute average NPV until 60%, whereas from 80% on both increase together. This way, our results confirmed what we already suspected in section 4.1, namely that a lower Y_2 value is required for a lower absolute project NPV, to more clearly distinguish solutions with a different deadline violation from one another.

%Neg	0%	20%	40%	60%	80%	100%
AvNPV	7,930.42	4,763.23	1,436.95	263.86	-1,369.12	-5,736.01
Y_2	0.995	0.95	0.85	0.70	0.92	0.97

Table 2: Parameters of the penalty function

Next, we compare different combinations of the activity move rules of section 4 and use GA2 to make this comparison. The reason that GA2 and not GA1 is used for the comparison, is because GA2 not only performs better, but also because it more clearly illustrates the value of each step. To validate both distinct steps, each step is first excluded from the method. These results are displayed in the table under *NoStep1* (the network–based moves are excluded) and *NoStep2* (the schedule–based moves are excluded). Finally, we also tested a switch in the order in which the network– and schedule–based moves are applied (*Switch1&2*). Recall that otherwise we first apply step 1 and then step 2, hence for the *Switch1&2* case we first apply step 2 and then step 1. As a benchmark, we use the case in which only single activity moves are applied (*OnlySingle*).

The results of this analysis are shown in table **??** and are based on their deviations from the best known solutions for the resource–unconstrained max–NPV problem, as discussed by Vanhoucke (2006). The table displays the relative average deviation (*%AvDev*) which stands for the percentage average deviation of the solutions of this manuscript in comparison with the results for the resource–unconstrained case. A lower value corresponds with a better solution. The results from the comparison with the max–NPV in the table only take those data files into account for which all alternatives are able to find deadline feasible solutions, this to ensure a correct comparison. The table also compares all options based on the percentage average difference (*%AvDiff*), which shows the performance of the proposed method in comparison with the scatter search of Vanhoucke (2010). A positive value corresponds with a better performance of the proposed option, whereas a negative value means that the scatter search has better results. Finally, the percentage of better solutions (*%Better*) found by each combination in comparison with the scatter search is shown.

Based on the results in the table it can be seen that *BothSteps* does better than all other options on all three performance criteria, which means that the proposed scheduling steps and their ordering are justified.

%Neg	0%	20%	40%	60%	80%	100%
AvNPV	7,930.42	4,763.23	1,436.95	263.86	-1,369.12	-5,736.01
Y_2	0.995	0.95	0.85	0.70	0.92	0.97

Table 3: Parameters of the penalty function

Finally, table 4 compares GA1 and GA2 with 5,000 random schedules, only problem instances for which all three methods could find a deadline feasible solution are taken into account. The percentage feasible (*%Feas*) shows the number of deadline feasible solutions found.

Based on the table the following can be concluded:

- GA2 performs best both overall and for all values of the six parameters of the dataset. Hence, in section 6.2 where we compare with existing methods, we only show the results of GA2.
- GA2 furthermore reports the largest percentage of feasible solutions found. Additionally, the percentage of feasible solutions found between on the one hand both GAs and the 5,000 random schedules on the other hand illustrates the added value of the proposed penalty function as part of our metaheuristic.
- The difference in performance between GA2 and GA1 in terms of *%Neg* is larger when *%Neg*$>$ 50% compared to the cases with *%Neg*$\leq$ 50%.
- The larger the *OS* the better GA2 performs in comparison with GA1.
- The smaller the *RC* the better GA2 performs in comparison with GA1.

6.2 Comparison with literature

To the best of our knowledge, the scatter search of Vanhoucke (2010) and the Lagrangian–based heuristic of Gu et al. (2013) are the best known algorithms for the RCPSPDC. Both however report their results based on a different termination criterion. Whereas Vanhoucke (2010) uses the 5,000 schedules criterion, Gu et al. (2013) terminate after 5 minutes. As such we compare with both algorithms separately.

First, in table 5 we compare our GA with Vanhoucke (2010). The percentage of instances for which the best known solution is found (*%Best*) constitutes the number of instances for which either method found the best known solutions. Note that the sum of both percentages is larger than 100% because of the cases in which both algorithms have the same result are included twice.

		GA1		GA2		5,000 randoms	
		%AvDev	%Feas	%AvDev	%Feas	%AvDev	%Feas
%Neg	0%	30.77	98.33	30.53	98.85	32.87	91.98
	20%	28.49	98.75	28.02	99.34	32.60	92.22
	40%	73.57	98.99	72.53	98.85	92.47	93.36
	60%	572.16	97.36	551.35	97.40	743.61	78.72
	80%	402.13	97.60	388.02	98.33	518.50	79.10
	100%	277.84	97.12	270.08	98.44	329.13	78.82
D–Incr	5%	251.73	92.78	248.68	94.21	306.27	61.13
	10%	124.45	99.81	114.68	99.93	154.02	84.19
	15%	344.50	99.51	335.10	100.00	441.55	96.97
	20%	147.74	100.00	143.43	100.00	189.31	99.84
#Act	25	198.93	99.44	194.41	99.51	235.53	98.56
	50	319.23	98.84	305.53	99.03	412.73	89.77
	75	151.73	97.18	148.71	98.06	200.35	79.70
	100	183.76	96.64	177.89	97.55	230.93	74.10
OS	0.25	147.94	96.82	143.68	98.00	201.63	83.07
	0.50	213.97	97.66	208.39	97.95	265.62	83.02
	0.75	280.69	99.60	270.24	99.65	344.55	90.50
RC	0.25	368.35	97.38	357.92	98.13	468.72	84.55
	0.50	137.72	98.28	132.20	98.75	172.02	85.07
	0.75	145.22	98.42	140.48	98.73	181.19	86.98
RU	2	271.19	97.36	263.06	98.38	344.73	81.16
	4	166.71	98.69	160.97	98.69	208.10	89.91
Overall		216.21	98.03	209.34	98.54	272.84	85.53

Table 4: Comparison between GA1, GA2 and 5,000 randoms

The following conclusions can be drawn from the table:

- Overall our results outperform those of Vanhoucke (2010).
- Our proposed method performs better comparatively with a decreasing value of the *OS*, and an increasing value of the *RC* and *RU*.
- A similar trend can be observed as *#Act* increases, but it is less pronounced.
- For the instances with *%Neg* equal to 40% or 60% the *%AvDev* of GA2 is worse than that of Vanhoucke (2010), but this is offset by a larger *%Best*.

Table 6 shows a more detailed overview of the comparative performance of both algorithms. In this table both algorithms are compared based on the percentage average difference (*%AvDiff*) between them, which shows the performance of the proposed method in comparison with the scatter search of Vanhoucke (2010). A positive value corresponds with a better performance of our GA2, whereas a negative value means that the scatter search has better results. The percentages of better, equal and worse (*%Better*, *%Equal* and *%Worse*) solutions by GA2 in comparison with the scatter search are also shown. Again, only problem instances for which both methods could find a deadline feasible solution are taken into account in both tables. Based on both table 5 and 6, it can be seen that GA2 outperforms the algorithm of Vanhoucke (2010). The only case for which a slightly worse performance can be observed based on all comparative factors is when *#Act* is set to 25.

		This paper			Vanhoucke (2010)		
		AvDev	%AvDev	%Feas	AvDev	%AvDev	%Feas
%Neg	0%	4,480.55	30.84	98.85	4,535.76	31.16	99.79
	20%	2,569.39	28.59	99.34	2,626.96	29.17	99.79
	40%	1,022.16	71.21	98.85	1,064,49	69.78	99.79
	60%	703.17	487.85	97.40	758.32	482.66	99.79
	80%	933.29	415.22	98.33	1,023.84	442.50	99.79
	100%	2,011.40	240.45	98.44	2,048.43	252.93	99.79
D–Incr	5%	2,119.16	228.94	94.21	2,192.98	234.86	99.17
	10%	1,970.61	133.91	99.93	2,052.91	152.41	100.00
	15%	1,919.04	337.93	100.00	1,945.71	332.34	100.00
	20%	1,830.39	145.83	100.00	1,873.86	149.71	100.00
#Act	25	371.41	192.59	99.51	370.89	192.46	100.00
	50	1,295.24	294.05	99.03	1,327.73	291.75	99.86
	75	2,458.97	153.97	98.06	2,540.08	160.30	99.72
	100	3,744.92	204.38	97.55	3,858.48	223.42	99.58
OS	0.25	2,254.87	166.00	98.00	2,356.44	179.67	99.69
	0.50	2,046.98	203.56	97.95	2,098.45	210.55	99.69
	0.75	1,577.06	263.71	99.65	1,593.62	260.23	100.00
RC	0.25	1,887.70	342.07	98.13	1,942.33	342.11	99.58
	0.50	1,983.81	141.25	98.75	2,039.25	149.67	99.90
	0.75	2,000.15	151.81	98.73	2,058.96	160.33	99.90
RU	2	1,696.00	251.76	98.38	1,746.26	252.45	99.79
	4	2,217.88	171.18	98.69	2,280.20	181.82	99.79
Overall		1,957.38	211.40	98.54	2,013.68	217.08	99.79

Table 5: Comparative computational results part 1 (5,000 schedules)

As already stated the algorithm of Gu et al. (2013) compares with the results of Vanhoucke (2010) based on a 5 minutes stopping criterion instead of the 5,000 schedules criterion. This means that in order to properly compare our own method with the former we should also terminate after 5 minutes. We have however chosen the employ a stopping criterion of 12,500 schedules with a maximum time limit of 5 minutes per instance. We furthermore use a 2.5GHz Dual Core processor, whereas Gu et al. (2013) use a computing cluster where each node consists of two 2.8GHz 6–Core processors.

The results of the comparison with the algorithm of Gu et al. (2013) can be found in table 7. The results used of the latter are those of the CP–LR since the authors show this method clearly performs best compared to others. This CP–LR is one of several alternatives tested by them and constitutes a hybrid approach of constraint programming and Lagrangian relaxation.

		%AvDiff	%Better	%Equal	%Worse
%Neg	0%	0.80	58.59	13.28	28.13
	20%	1.18	66.41	9.47	24.12
	40%	2.01	56.87	6.92	36.21
	60%	37.07	60.11	5.85	34.04
	80%	22.22	61.33	3.35	35.31
	100%	1.15	59.54	12.80	27.65
D–Incr	5%	7.66	61.13	9.14	29.73
	10%	21.19	64.70	7.67	27.63
	15%	5.73	55.02	9.28	35.69
	20%	7.92	61.11	8.43	30.46
#Act	25	-1.35	30.03	32.43	37.54
	50	14.52	66.99	1.50	31.51
	75	16.97	73.68	0.24	26.09
	100	12.69	71.67	0.00	28.33
OS	0.25	12.90	70.01	3.29	26.70
	0.50	9.49	60.85	7.44	31.71
	0.75	9.63	50.75	15.02	34.23
RC	0.25	12.96	59.52	8.09	32.40
	0.50	10.29	61.41	8.23	30.36
	0.75	8.76	60.51	9.55	29.95
RU	2	10.27	55.68	10.26	34.06
	4	11.06	65.26	6.99	27.75
Overall		10.67	60.48	8.62	30.90

Table 6: Comparative computational results part 2 (5,000 schedules)

Table 7 is constructed similarly to table 5, of which the latter compares our GA2 with the scatter search of Vanhoucke (2010). This way, a similar analysis can be made and the following conclusions can be drawn:

- The overall results show a better performance of our GA2 than the CP–LR of Gu et al. (2013).
- In particular our *%Best* is larger in nearly all cases.
- Comparatively our method performs better as *#Act* increases meaning that it is more robust as the project size increases. This can be seen by an increasing *%Best* of our method as the number of activities increases, whereas the results of Gu et al. (2013) show alternating decreases and increases. The gap between both methods in terms of *%AvDev* furthermore becomes larger in favour of GA2 with an increasing number of activities.

In table 8 we show the average computation times of our own algorithm (*AvTime*) and the percentage of instances for which the 5 minutes limit was actually reached (*%Limit*). Only for a very small number of instances is the 5 minutes limit actually reached.

		This paper			Gu et al. (2013)		
		%AvDev	%Best	%Feas	%AvDev	%Best	%Feas
%Neg	0%	30.55	84.36	99.48	30.64	74.31	99.90
	20%	28.13	87.26	99.51	28.42	73.20	99.90
	40%	63.69	80.68	99.41	67.03	64.60	99.90
	60%	476.10	75.93	98.68	460.68	66.95	99.93
	80%	422.20	70.95	99.20	422.21	65.39	99.93
	100%	234.78	84.38	99.17	247.07	63.52	99.93
D–Incr	5%	228.47	82.49	97.04	222.64	74.16	99.65
	10%	138.93	83.74	99.93	142.94	69.68	100.00
	15%	328.22	77.27	100.00	323.67	65.32	100.00
	20%	140.15	78.98	100.00	146.57	62.78	100.00
#Act	25	194.05	67.13	99.58	190.05	85.83	100.00
	50	289.57	79.53	99.54	285.36	54.28	100.00
	75	149.66	89.46	99.07	153.51	70.69	99.98
	100	201.62	89.81	98.77	208.08	61.13	99.68
OS	0.25	161.83	84.35	98.75	161.09	76.40	99.91
	0.50	203.33	81.43	99.11	207.53	68.26	99.83
	0.75	260.60	76.08	99.86	257.26	59.31	100.00
RC	0.25	333.43	79.35	99.03	65.50	82.83	100.00
	0.50	139.87	81.02	99.15	338.33	62.09	99.74
	0.75	153.61	81.44	99.55	224.44	59.06	100.00
RU	2	248.57	78.34	99.06	248.18	73.44	99.83
	4	169.16	82.86	99.42	169.18	62.55	100.00
Overall		208.71	71.97	99.24	209.33	67.99	99.91

Table 7: Comparative computational results 12,500 schedules

#Act	AvTime (s)	%Limit
25	1.48	0.00
50	5.05	0.00
75	11.58	0.00
100	23.42	0.47
Overall	10.28	0.12

Table 8: Computation times 12,500 schedules

Although the improvement of our method in table 7 is smaller than the one in tables 5 and 6, it is important to take into account that our algorithm on average only takes 10.28 seconds, see table 8, whereas Gu et al. (2013) always use 5 minutes. We furthermore employ a computer with a slower processor to test our method. As such, based on both tables 7 and 8 it can be concluded that our proposed scheduling method and GA2 are not only faster but also provide better results than the methodology of Gu et al. (2013). In particular, the number of generated schedules is set to 12,500 since this can be seen as the cut–off point in terms of number of schedules where our GA2 outperforms the CP–LR of Gu et al. (2013).

7 Conclusions

In this paper we discussed the RCPSPDC with payments at activities' completion times. We proposed a new scheduling technique which makes use of rules to move activities. These rules focus on maximizing project NPV by delaying sets of activities with a negative cumulative NPV or advancing those with a positive NPV. An important part of our methodology is the construction of sets of activities which have to be moved together and can be built in two ways. A first method evaluates the predecessors and successors of an activity and determines whether they should be included in the set or not. A second method focusses on the neighbouring activities in the project schedule which means the activities under consideration need not be precedence related. A penalty function was also included for the schedules infeasible with respect to the project deadline. Each step of our algorithm has been extensively tested and two variants of a genetic algorithm were proposed. Our final procedure was compared with the best known results from literature and was shown to perform considerably better.

Acknowledgements

We would like to thank Dr. Hanyu Gu for providing us with more detailed results of the method of Gu et al. (2013).

Funding

We acknowledge the support provided by the 'Bijzonder Onderzoeksfonds' (BOF) for the project with contract number BOF12GOA021. We also acknowledge and appreciate the thesis funding awarded by PMI Belgium in 2012 for the preparatory study prior to this research.

References

Baroum, S. and Patterson, J. (1996). The development of cash flow weight procedures for maximizing the net present value of a project. *Journal of Operations Management*, 14:209–227.

Blazewicz, J., Lenstra, J., and Rinnooy Kan, A. (1983). Scheduling subject to resource constraints: notation, classification and complexity. *Discrete Applied Mathematics*, 5:11–24.

Brucker, P., Drexl, A., Möhring, W., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41.

Debels, D., De Reyck, B., Leus, R., and Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta–heuristic for project scheduling. *European Journal of Operational Research*, 169:638–653.

Debels, D. and Vanhoucke, M. (2007). A decomposition–based genetic algorithm for the resource–constrained project scheduling problem. *Operations Research*, 55(3):457–469.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818.

Gu, H., Schutt, A., and Stuckey, P. (2013). A Lagrangian relaxation based forward-backward improvement heuristic for maximising the net present value of resource-constrained projects. *Lecture Notes in Computer Science*, 7874:340–346.

Gu, H., Stuckey, P., and Wallace, M. (2012). Maximising the net present value of large resource-constrained projects. *Lecture Notes in Computer Science*, 7514:767–781.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207:1–14.

Herroelen, W., Demeulemeester, E., and De Reyck, B. (1999). A classification scheme for project scheduling. In Weglarz, J., editor, *Project Scheduling - recent models, algorithms and applications. International Series in Operations Research and Management Science*, number 14, pages 77–106. Boston: Kluwer Academic Publishers.

Herroelen, W., Van Dommelen, P., and Demeulemeester, E. (1997). Project networks with discounted cash flows: A guided tour through recent developments. *European Journal of Operational Research*, 100:97–121.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

Icmeli, O. and Erengüç, S. (1996). A branch and bound procedure for the resource-constrained project scheduling problem with discounted cash flows. *Management Science*, 42(10):1395–1408.

Kimms, A. (2001). 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.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Rules and computation. *European Journal of Operational Research*, 90:320–333.

Li, K. and Willis, R. (1992). An interative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, 56:370–379.

Mika, M., Waligóra, G., and Weglarz, J. (2005). Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *European Journal of Operational Research*, 164:639–668.

Russell, R. (1986). A comparison of heuristics for scheduling projects with cash flows and resource restrictions. *Management Science*, 32:1291–1300.

Schutt, A., Chu, G., Stuckey, P., and Wallace, M. (2012). Maximising the net present value for resource-constrained project scheduling. *Lecture Notes in Computer Science*, 7298:362–378.

Selle, T. and Zimmermann, J. (2003). A bidirectional heuristic for maximizing the net present value of large-scale projects subject to limited resources. *Naval Research Logistics*, 50:130–148.

Smith-Daniels, D. and Aquilano, N. (1987). Using a late-start resource-constrained project schedule to improve project net present value. *Decision Sciences*, 18:617–630.

Ulusoy, G., Sivrikaya-Şerifoğlu, F., and Şahin, S. (2001). Four payment models for the multi-mode resource constrained project scheduling problem with discounted cash flows. *Annals of Operations Research*, 102:237–261.

Valls, V., Ballestín, F., and Quintanilla, S. (2004). A population–based approach to the resource–constrained project scheduling problem. *Annals of Operations Research*, 131:305–324.

Valls, V., Quintanilla, S., and Ballestín, F. (2003). Resource–constrained project scheduling: A critical activity reordering heuristic. *European Journal of Operational Research*, 149:282–301.

Vanhoucke, M. (2006). An efficient hybrid search procedure for various optimization problems. *Lecture Notes in Computer Science*, 5482:13–24.

Vanhoucke, M. (2009). A genetic algorithm for net present value maximization for resource constrained projects. *Lecture Notes in Computer Science*, 5482:13–24.

Vanhoucke, M. (2010). A scatter search procedure for maximizing the net present value of a resource-constrained project with fixed activity cash flow. *International Journal of Production Research*, 48(7):1983–2001.

Vanhoucke, M., Demeulemeester, E., and Herroelen, W. (2001). On maximizing the net present value of a project under renewable resource constraints. *Management Science*, 47(8):1113–1121.

Yang, K., Tay, L., and Sum, C. (1995). A comparison of stochastic scheduling rules for maximizing project net present value. *European Journal of Operational Research*, 85:327–339.

Zhu, D. and Padman, R. (1999). A metaheuristic scheduling procedure for resource-constrained projects with cash flows. *Naval Research Logistics*, 46:912–927.