Maximising the Net Present Value of Project Schedules Using CMSA and Parallel ACO

Dhananjay Thiruvady[1(✉)], Christian Blum[2], and Andreas T. Ernst[1]

[1] School of Mathematical Sciences, Monash University, Melbourne, Australia
{dhananjay.thiruvady,andreas.ernst}@monash.edu
[2] Artificial Intelligence Research Institute (IIIA-CSIC),
Campus of the UAB, Bellaterra, Spain
christian.blum@iiia.csic.es

Abstract. This study considers the problem of resource constrained project scheduling to maximise the net present value. A number of tasks must be scheduled within a fixed time horizon. Tasks may have precedences between them and they use a number of common resources when executing. For each resource, there is a limit, and the cumulative resource requirements of all tasks executing at the same time must not exceed the limits. To solve this problem, we develop a hybrid of Construct, Merge, Solve and Adapt (CMSA) and Ant Colony Optimisation (ACO). The methods are implemented in a parallel setting within a multi-core shared memory architecture. The results show that the proposed algorithm outperforms the previous state-of-the-art method, a hybrid of Lagrangian relaxation and ACO.

Keywords: Project scheduling · Net present value
Construct, Merge, Solve & Adapt · Ant Colony Optimisation

1 Introduction

Resource constrained project scheduling is a problem that has been investigated for many years. Due to the complexity of variants of the problem, solving instances with just 100 tasks is still challenging. The details of different project scheduling problems vary, but the basic elements require a number of tasks to be completed with an objective related to the completion time or some value of the tasks. In recent times, maximizing the *net present value* (NPV) of a project has received attention. Each task has a cash flow that may be positive or negative. The aim is to maximize the net present value of the profit, that is, the sum of the discounted cumulative cash flows of the tasks [7,18,22–24].

Numerous variants of project scheduling can be found in the literature; see [6] for an early review. All problems considered in [6] consist of tasks, precedences between them, limited shared (renewable) resources and deadlines. The problems are solved with a number of different methods, including heuristics, local

M. J. Blesa Aguilera et al. (Eds.): HM 2019, LNCS 11299, pp. 16–30, 2019.
https://doi.org/10.1007/978-3-030-05983-5_2

search and branch & bound. A discussion on details of methods for solving these problems, including exact approaches, heuristics and meta-heuristics, is provided in [10]. Among meta-heuristics, simulated annealing, genetic algorithms, and tabu search have been applied. Furthermore, heuristic and exact approaches for project scheduling with time windows are discussed in [17].

A closely related problem is resource constrained job scheduling (RCJS) [21], with the main difference being in the objective, i.e., minimising the total weighted tardiness of the tasks. To solve this problem, hybrids of mixed integer programming (MIP) decompositions and meta-heuristics [20], constraint programming and ant colony optimisation (ACO) [21], and parallel implementations of these methods [8,19] have been successfully applied.

For project scheduling with the NPV objective, various heuristic and exact approaches can be found in the literature. Problem instances with up to 98 tasks are solved in [7], with an ACO approach that outperforms other meta-heuristics such as a genetic algorithm, tabu search and simulated annealing. In [18], the authors show that ACO can be effective for a similar problem with up to 50 tasks. A scatter search heuristic described in [23] was shown to be more effective than exact approaches based on branch & bound methods for the same problem [24]. In [13], a hybrid of constraint programming and Lagrangian relaxation is considered which is able to find good feasible solutions easily for this problem. A hybrid of Lagrangian relaxation and ACO [22] was shown to be particularly effective when run in parallel [5].

In this study, we consider resource constrained project scheduling (RCPS) maximising the NPV [14]. This problem is henceforth simply denoted by RCPS-NPV. The problem considers several tasks with varying cash flows, precedences between some of them and a common deadline for all of them. There are also common renewable resources, of which the tasks require some proportion. We solve this problem considering a hybrid of two techniques: (1) Construct, Merge, Solve & Adapt (CMSA) and (2) ACO. Hereby, within the CMSA framework, a parallel implementation of ACO (PACO) is iteratively used to generate solutions for being used within CMSA.

CMSA is a rather recent, generic MIP-based metaheuristic that has shown to be effective on several problems including: the minimum common string partition and minimum covering arborescence problems [4], the repetition-free longest common subsequence problem [3], and unbalanced common string partition [2]. While this method has not yet been tested extensively on a wide range of problems, the results from the initial studies are very promising. At each iteration, CMSA requires a set of promising solutions to be generated by some randomized technique. These solutions are then used to build a reduced sub-instance with respect to the tackled problem instance. This sub-instance is then solved by a complete technique. In the context of problems that can be modelled as MIPs, a general-purpose MIP solver may be used for this purpose. As mentioned above, ACO is used in the case of this paper for generating the set of solutions per iteration. ACO is a meta-heuristic that uses the principles underlying the foraging behavior of ants. This technique has shown to be effective on a number of prob-

lems [12]. Project scheduling variants have also been solved with ACO [7,16,18]. The RCPS problem with makespan as the objective was considered, for example, in [16]. Moreover, an ACO for project scheduling where the tasks can execute in multiple modes is described in [7]. In [18], a problem similar to that of [23], but with fewer tasks, was tackled.

The paper is organized as follows. Section 2 provides a MIP model of the RCPS-NPV. Section 3 discusses CMSA and PACO. Section 4 details the experiments conducted and the results obtained from these experiments, before conclusions are provided in Sect. 5.

2 The RCPS-NPV Problem

The RCPS-NPV problem can be formulated as follows. A set $\mathcal{J}$ of n tasks is given, with each task $i \in \mathcal{J}$ having a duration d_i, a cash flow cf_{it} at each time period $t \geq 1$ (which may be positive or negative), and an amount of resource of each type k, r_{ik}, that it requires. While the cash flow of a task may vary over its duration, we can simply calculate the total net value c_i that the task would contribute to the project if it was completed at time 0. From this we can compute the discounted value using discount factor $\alpha > 0$ for start time s_i as $c_i\, e^{-\alpha(s_i+d_i)}$, where d_i is the duration of task i. Note that the formula $e^{-\alpha\, t}$ for discounting is equivalent to the commonly used function $1/(1+\bar{\alpha})^t$ for a suitable choice of α. Let P be the set of precedences P between tasks. We will write $i \rightarrow j$ or $(i,j) \in P \subseteq \mathcal{J} \times \mathcal{J}$ to denote that the processing of task i must complete before the processing of j starts.

Given k resources and their limits—that is, $R_1, \ldots, R_k$—the cumulative use of resources by all tasks executing at the same time must satisfy these resource limits. There is also a common deadline for all the tasks, δ, representing the time horizon for completing the project. Without such a deadline, tasks with negative cash flow and no successors would never be completed. Given the objective of maximizing the NPV, the problem can be stated as follows:

$$\max \quad \sum_{i \in \mathcal{J}} c_i\, e^{-\alpha(s_i+d_i)} \tag{1}$$

$$\text{S.T.} \quad s_i + d_i \leq s_j \qquad \forall\ (i,j) \in P \tag{2}$$

$$\sum_{i \in S(t)} r_{im} \leq R_m \qquad \forall\ m = 1, \ldots k \tag{3}$$

$$0 \leq s_i \leq \delta - d_i \qquad \forall\ i \in \mathcal{J} \tag{4}$$

Hereby, set $S(t)$ consists of tasks executing at time t. The NPV objective function (1) is non-linear and neither convex nor concave, making this problem challenging to solve. Constraints (2) enforce the precedences. Constraints (3) ensure that all the resource limits are satisfied. Constraints (4) require that the deadline is satisfied.

Like in the studies by [14] and [22], the deadline is typically not tight, as the aim is not to minimize the makespan. The available slack means that negative-valued tasks can be scheduled later, thereby slightly increasing the NPV.

2.1 MIP Model

A MIP model for the RCPS-NPV (see also [14]) can be defined as follows: Let $V := \{x_{it} \mid i = 1, \ldots, n \text{ and } t = 1, \ldots, \delta\}$ be a the set of binary variables, where x_{it} takes value 1 if task i completes at time t. The objective is to maximise the NPV:

$$\max \quad \sum_{i \in \mathcal{J}} \sum_{t=1}^{\delta} c_i \, e^{-\alpha t} \, x_{it} \tag{5}$$

$$\text{S.T.} \quad \sum_{t=1}^{\delta} x_{it} = 1 \qquad \forall \, i \in \mathcal{J} \tag{6}$$

$$\sum_{t=1}^{\delta} t \, x_{jt} - \sum_{t=1}^{\delta} t \, x_{it} \geq d_j \qquad \forall \, (i,j) \in P \tag{7}$$

$$\sum_{i=1}^{n} \sum_{\hat{t}=t}^{t+d_i-1} r_{ik} \, x_{i\hat{t}} \leq R_m \qquad \forall \, m = 1, \ldots, k, \; t \in \{1, \ldots, \delta\} \tag{8}$$

$$x_{it} \in \{0, 1\} \qquad \forall \, x_{it} \in V \tag{9}$$

Equation (5) maximises the NPV. Constraints (6) ensure that all tasks are completed exactly once. Constraints (7) ensure that the precedences are satisfied. Constraints (8) require that all the resource limits are satisfied.

For an exact MIP solver this would not be expected to be the most efficient formulation. For the CMSA algorithm described below, an important characteristic of this MIP formulation is that solutions have a low density of non-zeros, just one per activity. Furthermore, instead of solving the full model, in CMSA we only solve small subproblems with a significantly reduced subset of the possible time points. During preliminary experiments it was shown that such type of MIPs can be easily solved by applying a general-purpose MIP solver.

3 The Proposed Algorithm

In this section, we provide the details of our implementation of CMSA. This algorithm learns from sets of solutions which are recombined with the aid of a general-purpose MIP solver. In this study we use ACO, implemented in a parallel setting to ensure that diversity is achieved efficiently as described in [21], to generate these sets of solutions for CMSA at each of its iterations.

3.1 Construct, Merge, Solve and Adapt

Algorithm 1 presents our implementation of CMSA for the tackled RCPS-NPV problem. As input the algorithm takes (1) a problem instance, (2) the number of solutions (n_s), (3) a total computation time limit (t_{total}), (4) a computation

time limit per MIP-solver application at each iteration (t_{iter}), and (5) a maximum age limit (a_{max}). CMSA keeps, at all times, a subset V' of the complete set of variables V of the MIP model (see previous section for the definition of V). Moreover, a solution S in the context of CMSA is a subset of V. A solution $S \subseteq V$ is called a valid solution iff assigning value one to all variables in S and zero to the remaining variables in $V \setminus S$, results in a valid RCSP-NPV solution.

The objective function value of a solution S is henceforth denoted by $f(S)$.

Algorithm 1. CMSA for the RCPS-NPV problem

```
1: INPUT: RCPS-NPV instance, n_s, t_total, t_iter, a_max
2: Initialisation: V' := ∅, S^bs := ∅, a_jt := 0 ∀ x_jt ∈ V
3: while time limit t_total not expired do
4:     for i = 1, 2, ..., n_s do                    # note that this is done in parallel
5:         S_i := GenerateSolution()
6:         V' := V' ∪ {S_i}
7:     end for
8:     S^ib ← Apply_ILP_Solver(V',S^bs)
9:     if f(S^ib) > f(S^bs) then S^bs := S^ib end if
10:    Adapt(V', S^bs)
11: end while
12: OUTPUT: S^bs
```

The algorithm works as follows. First, in line 2, the initial subset of variables (V') is initialized to the empty set. The same is done for the best-so-far solution S^{bs}.[1] Moreover, the age value a_{jt} of each variable $x_{jt} \in V$ is initialized to zero. The main algorithm now executes between Lines 3–11 until the time limit (t_{total}) has expired. At each iteration of CMSA, the following actions are taken. First, a number of n_s solutions is generated in a randomized way, that is, the *Construct* phase of CMSA is executed. In principle, any randomized method to generate feasible solutions is acceptable. However, here we use ACO (see Sect. 3.2). In particular, n_s ACO colonies are run in parallel, where one colony is seeded with the best-so-far solution S^{bs}. The remaining colonies are seeded with a random solution. This method ensures that good parts of the search space and also sufficiently diverse parts of the search space are covered.

Once the solutions have been produced, the variables they contain (remember that these are the variables with value one in the corresponding MIP solutions) are added to V' in line 6 (*Merge* phase of CMSA). Based on V', a reduced MIP model is generated (see below for a detailed description) and then solved by applying a general-purpose MIP solver in function Apply_ILP_Solver(V',S^{bs}) (see line 8 of Algorithm 1). This phase is the *Solve* phase of CMSA. Note that the MIP solver is warm-started with the best-so-far solution S^{bs}. Finally, after potentially updating the best-so-far solution S^{bs} with solution S^{ib} obtained as

[1] Note that, for consistency, the objective function value of an empty solution is defined as $-\infty$.

output from the MIP solver, the *Adapt* phase of CMSA is executed in function $\mathsf{Adapt}(V', S^{bs})$ (see line 10) as follows. All variables from V' which are not present in S^{bs} have their age values incremented. The age values of those variables, whose corresponding age values have passed $a_{\max}$, are re-initialized to zero. Moreover, they are removed from V'. Finally, the output of CMSA is solution S^{bs}.

The Restricted MIP Model. The restricted MIP model that is solved in function $\mathsf{Apply\_ILP\_Solver}(V', S^{bs})$ of Algorithm 1 is obtained from the original MIP model outlined in Sect. 2 as follows. Instead of considering all variables from V, the model is restricted to the variables from V'. In this way the model becomes much smaller and can be relatively easily solved. However, note that the search space of such a restricted model is only a part of the original search space. Only when V' is very large (this may happen when n_s is large and the solution construction is very much random) the restricted MIP may not be solved within the given time. As mentioned above, a time limit (t_{iter}) is used to ensure the solver always terminates sufficiently quickly even if it does not find an optimal solution to the restricted MIP within the given time.

It should be noted that this Merge phase is guaranteed to always produce a solution that is at least as good as any of the solutions that has contributed to V'. This is because (1) all variables that take value one in solution S^{bs} are always present in V', and (2) solution S^{bs} is used for warm-starting the MIP solver. Hence the merge phase provides strong intensification. Diversification relies entirely on the solution generation mechanism. In previous papers (see, for example, [2]), solution generation was based on randomized greedy heuristics. It is expected that using ACO here with a guided—respectively, biased—random solution generation will both reduce the number of variables in the restricted MIPs (as solutions are expected to be of higher quality and, therefore, probably with more parts in common), and will assist the method to better explore the neighborhood of the best-so-far solution. The high-quality results that we will present in Sect. 4 seem to support this hypothesis. However, a deeper comparison between a more randomized solution construction and the construction of solutions by parallel ACO colonies is mandatory for future work.

3.2 Parallel Ant Colony Optimisation

ACO was proposed in [11] to solve combinatorial optimisation problems. The inspiration of these algorithms is the ability of natural ant colonies to find short paths between their nest and food sources.

For the purpose of this work, we use the ACO model for the resource constrained job scheduling problem originally proposed in [21]. This approach was extended to a parallel method in a multi-core shared memory architecture by [5]. For the sake of completeness, the details of the ACO implementation are provided here.

A solution in the ACO model is represented by a permutation of all tasks (π) rather than by the start times of the tasks. This is because there are potentially

Algorithm 2. ACO for the RCPS-NPV problem

1: **input**: An RCPS-NPV instance, $\mathcal{T}$, π^{bs} (optional)
2: Initialise π^{bs} (if given as input, otherwise not)
3: **while** termination conditions not satisfied **do**
4: **for** $j = 1$ to n_{ants} **do**:
5: $\pi^j :=$ ConstructPermutation($\mathcal{T}$)
6: ScheduleTask(π^j)
7: **end for**
8: $\pi^{ib} := \arg\min_{j=1,\ldots,n_{\text{ants}}} f(\pi^j)$
9: $\pi^{bs} :=$ Update(π^{ib})
10: PheromoneUpdate($\mathcal{T}$, π^{bs})
11: $cf :=$ ComputeConvergence($\mathcal{T}$)
12: **if** $cf =$ true **then** *initialise* $\mathcal{T}$ **end if**
13: **end while**
14: **output:** π^{bs} (converted into a CMSA solution)

too many parameters if the ACO model is defined to explicitly learn the start times of the tasks. Given a permutation, a serial scheduling heuristic (see [22]) can be used to generate a resource and precedence feasible schedule consisting of starting times for all tasks in a well-defined way. This is described in Sect. 3.3 below. For the moment it is enough to know that a CMSA solution S (in terms of a set of variables) can be derived from and ACO solution π in a well-defined way. As in the case of CMSA solutions, the objective function value of an ACO solution π is denoted by $f(\pi)$.

The pheromone model of our ACO approach is similar to the one used by [1], that is, the set of pheromone values ($\mathcal{T}$) consist of values τ_{ij} that represent the desirability of selecting task j for position i in the permutations to be built. Ant colony system (ACS) [12] is the specific ACO-variant that was implemented.

The ACO algorithm is shown in Algorithm 2. An instance of the problem and the set of pheromone value $\mathcal{T}$ are provided as input. Additionally, a solution (π^{bs}) can be provided as input which serves the purpose of initially guiding the search towards this solution. If no solution is provided, S^{bs} is initialised to be an empty solution.

The main loop of the algorithm at lines 3–13 runs until a time or iteration limit is exceeded. Within the main loop, a number of solutions (n_{ants}) are constructed (ConstructPermutation($\mathcal{T}$)). Hereby, a permutation π is built incrementally from left to right by selecting, at each step, a task for the current position $i = 1, \ldots, n$, making use of the pheromone values. Henceforth, $\hat{\mathcal{J}}$ denotes the tasks that can be chosen for position i, that is, $\hat{\mathcal{J}}$ consists of all tasks (1) not assigned already to an earlier position of π and (2) whose predecessors are all already scheduled. In ACS, a task is selected in one of two ways. A random number $q \in (0, 1]$ is generated and a task is selected deterministically if $q < q_0$. That is, task k is chosen for position i of π using:

$$k = \arg\max_{j \in \hat{\mathcal{J}}} \tau_{ij} \tag{10}$$

Otherwise, a probabilistic selection is used where job k is selected according to:

$$P(\pi_i = k) = \frac{\tau_{ik}}{\sum_{j \in \hat{\mathcal{J}}} \tau_{ij}} \tag{11}$$

After each step, a local pheromone update is applied to the selected task k at position i:

$$\tau_{ik} := \tau_{ik} \times \rho + \tau_{min} \tag{12}$$

where $\tau_{min} := 0.001$ is a limit below which a pheromone value does not drop, so that a task k may always have the possibility of being selected for position i.

After the construction of n_{ants} solutions, the iteration-best solution π^{ib} is determined (line 8) and the global best solution π^{bs} is potentially updated in function $\mathsf{Update}(\pi^{ib})$: $f(\pi^{ib}) > f(\pi^{bs}) \Rightarrow \pi^{bs} := \pi^{ib}$). Then, all pheromone values from $\mathcal{T}$ are updated using the solution components from π^{bs} in function $\mathsf{PheromoneUpdate}(\pi^{bs})$:

$$\tau_{i\pi(i)} = \tau_{i\pi(i)} \cdot \rho + \delta \tag{13}$$

where $\delta = 0.01$ is set to be a (small) reward. The value of the evaporation rate—that is, $\rho = 0.1$—is the same as the one chosen in the original study [22].

Finally, note that [5] showed different ways of parallelising ACO, where the obvious parallelisation involves building solutions concurrently. For two reasons we chose not to use this method for CMSA, but rather to run multiple colonies. The first, and most significant, is that multiple solutions obtained by a single colony can often consist of very similar components, especially when ACO is close to convergence. Second, the cores available to the algorithm are not fully utilised when the other components of ACO are executing (e.g. the pheromone update). Hence, in our CMSA algorithm we use n_s colonies in parallel, which ensures diversity (especially when initialised with different random solutions) and fully utilises the available cores.

3.3 Scheduling Tasks

As mentioned before, given a feasible permutation π of all tasks, a feasible solution in which the starting times of all tasks are explicitly given can be derived in a well-defined way. We briefly discuss this method in the following. For complete details of this procedure, we refer the reader to [22].

Consider a permutation π. In the given order, sets of tasks are selected where each set consists of a task and its successors. Each set is either net positive-valued or net negative-valued, that is, the total NPV of tasks in the set lead to positive or negative values. Those sets which are positive valued are greedily scheduled at the beginning of the horizon, satisfying the precedences and resources.[2] Tasks in the sets that are negative-valued are scheduled starting at the end of the horizon and working backwards while tasks remain. The motivation for this method is

[2] Tasks are selected which have no preceding tasks first. This is followed by selecting dependent tasks that a free to be scheduled. This continues until all tasks have been scheduled.

that positive-valued tasks being scheduled early lead to increasing the NPV, and scheduling negative-valued tasks at the end of the horizon leads to the least decrease in the NPV.

4 Experiments and Results

CMSA was implemented in C++ and compiled with GCC-5.2.0. Gurobi 8.0.0[3] was used to solve the MIPs and ACO's parallel component was implemented using OpenMP [9]. Monash University's Campus Cluster, MonARCH,[4] was used to carry out all experiments. Each machine of the cluster provides 24 cores and 256 GB RAM. Each physical core consists of two hyper-threaded cores with Intel Xeon E5-2680 v3 2.5 GHz, 30 M Cache, 9.60 GT/s QPI, Turbo, HT, 12C/24T (120 W).

We used the same instances as [14]. These problem instances were originally obtained from the PSPLIB [15].[5] These instances are characterized by the number of tasks $\{30, 60, 90, 120\}$, the resource factor $\{0.25, 0.5, 0.75, 1.0\}$, the network complexity $\{1.5, 1.8, 2.1\}$, and the resource strength. Instances with 30, 60 and 90 tasks have resource strengths from the range $\{0.2, 0.5, 0.7, 1.0\}$ while the instances with 120 tasks have resource strengths from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. The resource factor indicates the proportion of the total amount of each resource that is used, on average, by a task. The resource strengths specify how scarce the resources are with low values indicating tight resource constraints. The network complexity indicates the proportion of precedences, where large values imply a large number of precedences. The benchmark set consists, for each combination of the four parameters indicated above, of 10 instances. This means that there are 480 instances with 30, 60, and 90 tasks, and 600 instances with 120 tasks. In total the benchmark set contains 2040 instances.[6]

The study by [14] was used as a guide to obtain the deadlines and cash flows. For each task, the latest start time (l_i) is determined as $l_j \geq d_i \ \forall i \rightarrow j$, and then $\delta := 3.5 \times max_j l_j$. A task's cash flow c_i is generated uniformly at random from $[-500, 1000]$. The discount rate is determined as $\alpha := \sqrt[52]{1 + 0.05} - 1$.

For all runs we allowed the use of 5 cores, for both the MIP solver (Gurobi) and the ACO component (resulting in 5 parallel colonies). The parameters settings for each individual colony were the same as those used in [22]. Additionally, the current state-of-the-art algorithm from [22], which is a hybrid between Lagrangian Relaxation and ACO (henceforth denoted as LRACO), was re-run for all the instances to allow a fair comparison. Moreover, note that ACO also used 5 cores in parallel.

[3] http://www.gurobi.com/.
[4] https://confluence.apps.monash.edu/display/monarch/MonARCH+Home.
[5] https://www.sciencedirect.com/science/article/abs/pii/S0377221796001701.
[6] Gurobi can solve most of the problem instances with 30 tasks (the optimal solutions are provided in PSPLIB), a number of instances with 60 tasks (<60%), and a very small proportion of the instances with 90 tasks (<2%). None of the instances with 120 tasks can be solved with Gurobi.

The tuning of the parameters related to CMSA is described in Appendix A. Both algorithms were run once on each problem instance and allowed 15 min of wall-clock time.

4.1 Comparison: CMSA Versus LRACO

Table 1 shows a summary of the comparison between LRACO and CMSA. Averaged over the all the instances with the same number of tasks, the table provides gap ($\frac{UB-LB}{UB}$) for each algorithm and the number of times (out of 480, resp. 600) each one finds the best solution (*# best*). Moreover, the last table column contains the factor of how many more best solutions were found by CMSA relative to LRACO. For example, in the case of instances with 30 tasks, CMSA found 20.92 times more best solutions than LRACO.

For CMSA, the results are reported as the gap between the CMSA lower bound and the upper bound of LRACO (since we do not have a valid upper bound from CMSA). We see that CMSA performs—on average—better than LRACO for each problem size. Additionally, the number of best solutions found shows that CMSA is significantly more effective. Interestingly, the factor (last table column) drops with an increase in the number of tasks (from 20.92 for 30 tasks compared to 2.92 for 120 tasks). This means that, while CMSA is very strong for rather small problem instances, it still significantly outperforms LRACO on large problem instances. The drops of the performance of CMSA with growing problem size, might be related to the number of iterations that can be performed within the allowed computation time. More specifically, CMSA can generally perform more than 100 iterations within the allowed CPU time for instances with 30 tasks, but generally less than 20 iterations for 120 task problems.

These results are visualized in Fig. 1. The boxplots in this graphic show the improvement of CMSA over LRACO in percent ($\frac{(CMSA_i - LR_i)*100}{LR_i}$, $\forall i \in I$)[7] for subsets of the complete set of instances. A value above 0.00 indicates that CMSA

Table 1. Average gaps of LRACO and CMSA to the upper bound obtained by LRACO computed as $\frac{UB-LB}{UB}$. The number of instances for which an algorithm finds the best solution is shown in column *# best*, while *Factor* indicates how many times more best solutions were found by CMSA relative to LRACO.

Tasks	LRACO			CMSA			Factor
	Mean	SD	# best	Mean	SD	# best	
30	0.0189	0.0605	12	0.0180	0.0692	251	20.92
60	0.0148	0.0125	25	0.0133	0.0118	369	14.76
90	0.0147	0.0105	59	0.0136	0.0103	355	6.02
120	0.0176	0.0099	153	0.0165	0.0105	446	2.92

[7] I is the set of instances.

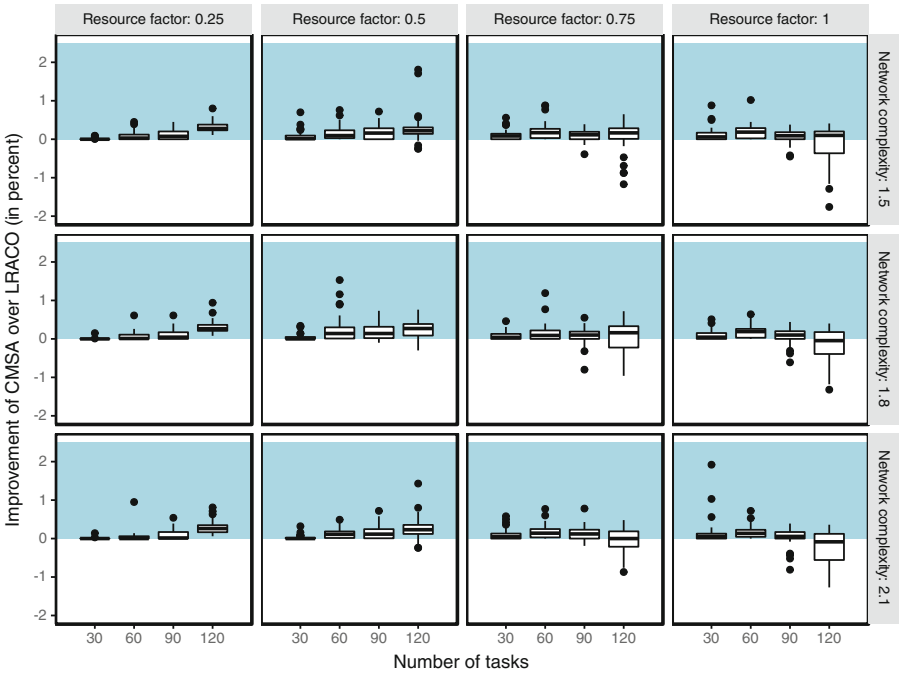

Fig. 1. The improvement of CMSA over LRACO (in percent) for subgroups of instances (as determined by the network complexity and the resource factor). Note that CMSA improves over LRACO whenever a data point has a positive value (that is, when it is located in the area with light blue background). (Color figure online)

performs better than LRACO for the respective instance, while a value below 0.0 indicates the opposite.

In can be observed that for instances with resource factors 0.25 and 0.5, CMSA generally improves over LRACO. However, with growing instance size and resource factor, this advantage of CMSA diminishes. In fact, for instances with 120 tasks and a resource factor of 1.0, LR-ACO seems to slightly outperform CMSA. Note also that a single instance of 30 tasks was removed from this graphic, as the gap was very large in favor of CMSA ($gap = 2.98$). This instance is interesting since it consists of many more negative valued tasks than other instances, leading to solutions with large negative NPVs. In this case CMSA is able to handle the negative tasks much more effectively than LRACO.

Table 2 shows an even more detailed breakdown of the results with respect to the four problem instance parameters (number of tasks, network complexity, resource factor and resource strength). In addition to the observations mentioned earlier, there are a few cases where large differences between the algorithms can be seen. For instances with 30 tasks, with a high resource factor ($RF = 1.0$) and a relatively high resource strength ($RS = 0.7$), the difference between CMSA and

Table 2. A breakdown of the results with respect to instance parameters (number of tasks, NC - network complexity, RF - resource factor, RS- resource strength). Gap% shows the same information as the boxplots in Fig. 1 (percentage improvement of CMSA over LRACO).

		30 tasks		60 tasks		90 tasks		120 tasks	
		Gap%	SD	Gap%	SD	Gap%	SD	Gap%	SD
NC	1.5	0.08	0.14	0.16	0.19	0.12	0.17	0.17	0.54
	1.8	0.06	0.11	0.17	0.24	0.11	0.20	0.10	0.39
	2.1	0.07	0.20	0.13	0.16	0.11	0.19	0.07	0.38
RF	0.3	0.01	0.02	0.07	0.13	0.11	0.13	0.30	0.15
	0.5	0.05	0.10	0.18	0.24	0.19	0.18	0.25	0.30
	0.8	0.10	0.13	0.18	0.21	0.11	0.18	0.09	0.57
	1.0	2.61	27.20	0.18	0.18	0.05	0.21	−0.19	0.47
RS	0.1	-	-	-	-	-	-	−0.05	0.80
	0.2	0.16	0.19	0.32	0.26	0.13	0.30	0.24	0.17
	0.3	-	-	-	-	-	-	0.03	0.44
	0.4	-	-	-	-	-	-	0.14	0.25
	0.5	0.08	0.20	0.21	0.16	0.21	0.13	0.20	0.14
	0.7	2.52	27.21	0.08	0.07	0.11	0.08	-	-
	1.0	0.00	0.00	0.00	0.00	0.00	0.00	-	-

LRACO is very high. On the other side, when the instances consist of relatively trivial constraints (high resource strength, i.e., RS = 1.0), both algorithms are nearly equal. Interestingly, for a number of very hard instances, LRACO outperforms CMSA (RF = 1.0, RS = 0.1). As mentioned above, this is likely to be due to the insufficient number of iterations completed the CMSA within the time-limit for such large problems.

5 Conclusion

In this study, we investigate a parallel hybrid of CMSA and ACO for resource constrained project scheduling under the maximization of the net present value. The proposed hybrid algorithm has several key characteristics. First, it makes use of a general purpose MIP solver in an iterative way, in order to solve reduced subinstances of the original problem instances. Second, it makes an efficient use of resources in parallel. This is achieved by running a number of ACO colonies in parallel leading to a diverse solutions space in the considered reduced subinstances. The results show that the hybrid CMSA approach is efficient at solving the problem, and improves upon LRACO, which is currently the state-of-the-art method in the literature.

This study has shown that if a problem can be efficiently modeled both with ACO and with MIP techniques, a generic hybrid can be developed that can

solve such a problem efficiently. We are currently working in this direction by developing a generic heuristic that can be applied to a range of problems in a very similar fashion. Furthermore, given that multi-core architectures are readily available nowadays, the parallelisation is straightforward.

A further line of investigation is related to developing a tighter coupling between ACO and CMSA. Since the MIP leads the search towards local optima, the solution information could be used to further assist ACO. For example, the solutions found by the MIPs could be used to generate or update the pheromone matrices at each iteration.

A key component of CMSA is defining an efficient MIP model. In this study, a straightforward MIP model has been used, however, the results could significantly improve with more efficient models. For example, the cumulative model proposed in [22] is more efficient, but requires a different way of extracting solution information. For this purpose, we are developing an alternative method labeled *Merge search*.

Acknowledgements. This research was supported in part by the Monash eResearch Centre and eSolutions-Research Support Services through the use of the MonARCH HPC Cluster.

A CMSA Parameter Selection

In order to determine the parameter settings of CMSA for the experiments, a subset of the problem instances were selected and used for testing. The parameters of interest are the MIP time limit (we considered either 60 or 120 s), the number of ACO iterations (500, 1000 and 2000 iterations) and the maximum age limit (3, 5 and 10). The instances selected consist of 60 (small) and 120 (large) tasks, with resource factors of 0.5 (low-medium) and 1.0 (high), resource strengths of 0.5 (low-medium) and 1.0 (high). Each run was given 5 ACO colonies (and hence 5 cores) and 15 min of wall-clock time.

Table 3. Number of best solutions found by varying parameters (MIP time limit - MIP TL., ACO iterations - ACO iter. and Age limit - Age).

Param.	Value	60 tasks	120 tasks
MIP TL.	60	**4**	**11**
	120	2	1
ACO Iter.	500	**4**	4
	1000	0	4
	2000	2	4
Age	3	**3**	4
	5	1	**7**
	10	**3**	1

Table 3 shows the number of instances for which the best solution was found with the respective settings. We consider the MIP time limit first, choose the best option, then select the best value for the number of ACO iterations, choose the best option, and finally select between the age limits. For the MIP time limit, 60 s is best for 60 and 120 tasks. Comparing iterations for 60 tasks, 500 is best and we use the same value for 120 tasks observing that there is no advantage. The best age limit for 120 tasks is 5, but because there is no obvious advantage for 60 tasks, we pick 3 in order for the built MIPs to be smaller and the MIP solver, therefore, faster.

References

1. den Besten, M., Stützle, T., Dorigo, M.: Ant colony optimization for the total weighted tardiness problem. In: Schoenauer, M., et al. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 611–620. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45356-3_60
2. Blum, C.: Construct, merge, solve and adapt: application to unbalanced minimum common string partition. In: Blesa, M.J., et al. (eds.) HM 2016. LNCS, vol. 9668, pp. 17–31. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39636-1_2
3. Blum, C., Blesa, M.J.: Construct, merge, solve and adapt: application to the repetition-free longest common subsequence problem. In: Chicano, F., Hu, B., García-Sánchez, P. (eds.) EvoCOP 2016. LNCS, vol. 9595, pp. 46–57. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30698-8_4
4. Blum, C., Pinacho, P., López-Ibáñez, M., Lozano, J.A.: Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. Comput. Oper. Res. **68**, 75–88 (2016)
5. Brent, O., Thiruvady, D., Gómez-Iglesias, A., Garcia-Flores, R.: A parallel lagrangian-ACO heuristic for project scheduling. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 2985–2991 (2014)
6. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: notation, classification, models, and methods. Eur. J. Oper. Res. **112**, 3–41 (1999)
7. Chen, W.N., Zhang, J., Chung, H.S.H., Huang, R.Z., Liu, O.: Optimizing discounted cash flows in project scheduling - an ant colony optimization approach. IEEE Trans. Syst. Man Cybern. Part C **40**(1), 64–77 (2010)
8. Cohen, D., Gómez-Iglesias, A., Thiruvady, D., Ernst, A.T.: Resource constrained job scheduling with parallel constraint-based ACO. In: Wagner, M., Li, X., Hendtlass, T. (eds.) ACALCI 2017. LNCS (LNAI), vol. 10142, pp. 266–278. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51691-2_23
9. Dagum, L., Menon, R.: OpenMP: an industry-standard API for shared-memory programming. IEEE Comput. Sci. Eng. **5**(1), 46–55 (1998)
10. Demeulemeester, E., Herroelen, W.: Project Scheduling: A Research Handbook. Kluwer, Boston (2002)
11. Dorigo, M.: Optimization, learning and natural algorithms. Ph.D. thesis, Dip. Elettronica (1992)
12. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
13. Gu, H., Schutt, A., Stuckey, P.J.: A lagrangian relaxation based forward-backward improvement heuristic for maximising the net present value of resource-constrained

projects. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 340–346. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38171-3_24
14. Kimms, A.: Maximizing the net present value of a project under resource constraints using a lagrangian relaxation based heuristic with tight upper bounds. Ann. Oper. Res. **102**, 221–236 (2001)
15. Kolisch, R., Sprecher, A.: PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. Eur. J. Oper. Res. **96**(1), 205–216 (1997)
16. Merkle, D., Middendorf, M., Schmeck, H.: Ant colony optimization for resource-constrained project scheduling. IEEE Trans. Evol. Comput. **6**(4), 893–900 (2000)
17. Neumann, K., Schwindt, C., Zimmermann, J.: Project Scheduling with Time Windows and Scarce Resources: Temporal and Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions, vol. 508. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24800-2
18. Show, Y.Y.: Ant colony algorithm for scheduling resource constrained projects with discounted cash flows. In: Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalain, China, pp. 176–180. IEEE (2006)
19. Thiruvady, D., Ernst, A.T., Singh, G.: Parallel ant colony optimization for resource constrained job scheduling. Ann. Oper. Res. **242**(2), 355–372 (2016)
20. Thiruvady, D., Singh, G., Ernst, A.T.: Hybrids of integer programming and ACO for resource constrained job scheduling. In: Blesa, M.J., Blum, C., Voß, S. (eds.) HM 2014. LNCS, vol. 8457, pp. 130–144. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07644-7_10
21. Thiruvady, D., Singh, G., Ernst, A.T., Meyer, B.: Constraint-based ACO for a shared resource constrained scheduling problem. Int. J. Prod. Econ. **141**(1), 230–242 (2012)
22. Thiruvady, D., Wallace, M., Gu, H., Schutt, A.: A lagrangian relaxation and ACO hybrid for resource constrained project scheduling with discounted cash flows. J. Heuristics **20**(6), 643–676 (2014)
23. Vanhoucke, M.: A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. Int. J. Prod. Res. **48**(7), 1983–2001 (2010)
24. Vanhoucke, M., Demeulemeester, E., Herroelen, W.: On maximizing the net present value of a project under renewable resource constraints. Manag. Sci. **47**(8), 1113–1121 (2001)