Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Economic optimization of resource-constrained project scheduling: a two-phase metaheuristic approach

Angela H. L. CHEN[†1,2], Chiuh-Cheng CHYU[1]

([1]*Department of Industrial Engineering and Management, Yuan Ze University, Taiwan 320, Taoyuan*)

([2]*Department of Finance, Nanya Institute of Technology, Taiwan 320, Taoyuan*)

[†]E-mail: angela@saturn.yzu.edu.tw

Received Oct. 22, 2009; Revision accepted Dec. 7, 2009; Crosschecked May 14, 2010

Abstract: This paper deals with the problem of project scheduling subject to multiple execution modes with non-renewable resources, and a model that handles some of monetary issues in real world applications. The objective is to schedule the activities to maximize the expected net present value (NPV) of the project, taking into account the activity costs, the activity durations, and the cash flows generated by successfully completing an activity. Owing to the combinatorial nature of this problem, the current study develops a hybrid of branch-and-bound procedure and memetic algorithm to enhance both mode assignment and activity scheduling. Modifications for the makespan minimization problem have been made through a set of benchmark problem instances. Algorithmic performance is rated on the maximization of the project NPV and computational results show that the two-phase hybrid metaheuristic performs competitively for all instances of different problem sizes.

Key words: Memetic algorithm (MA), Branch and bound (B&B) algorithm, Net present value (NPV), Project scheduling problem
doi:10.1631/jzus.C0910633 **Document code:** A **CLC number:** C935; F205

1 Introduction

Project scheduling in its broadest sense is defined as an act of planning and prioritizing activities that need to be executed in an orderly sequence of operation for the project completion. This act must meet certain requirements, constraints, or objectives of a project, for example, increasing efficiency and capacity utilization, and increasing the profitability of an organization. In today's extremely competitive environment with the high cost of capital and the time value of investment, attention to the economic consequences of using different methods for scheduling becomes an important aspect of project management. The net present value (NPV), which can be evaluated by measuring the amount and timing of all cash flows occurring over the project makespan, is the most straightforward and unambiguous traditional valuation method in the discounted cash flow measurement methodology. Russell (1970) was the first to put this maximization of the NPV of the cash flows and use optimal procedures in scheduling a project under an unconstrained case. Many others (Russell, 1970; Grinold, 1972; Doersch and Patterson, 1977; Elmaghraby and Herroelen, 1990; Herroelen and Gallens, 1993; Dayanand and Padman, 1997; Etgar *et al*., 1997) have also emphasized economically optimal procedures for unconstrained problems.

To the best of our knowledge, for current approaches used to solve the max-NPV problem, several researchers (Russell, 1970; Grinold, 1972; Patterson *et al*., 1990; Shtub and Etgar, 1997; de Reyck and Leus, 2008) employed exact methods such as integer programming, linear programming, and branch-and-bound (B&B) algorithms. Other researchers solve this problem with heuristics; for example, Icmeli and Erengüç (1996) used a multi-pass heuristic procedure and a bounding algorithm. Padman *et al.* (1997) presented a greedy single-pass parallel scheduling algorithm with nine priority rules embedded to maximize

the NPV. Pinder and Marucheck (1996) presented heuristics that provide schedules with superior NPVs. As for the applications of metaheuristics in the max-NPV problem, Zhu and Padman (1999) applied a tabu search with a multi-heuristic start procedure for neighborhood generation and candidate selection schemes. Etgar *et al.* (1997) described a simulated annealing approach with different implementation strategies for the unconstrained case of time-dependent, contingent cash flows. Vanhoucke (2006) embedded an enhanced bi-directional scheme and a recursive forward/backward improvement method in his implementation of a meta-heuristic scatter search procedure to solve a single-mode resource-constrained project scheduling with discounted cash flows (RCPSPDC).

In this study, we apply a memetic algorithm (MA), which imitates cultural evolution and allows individuals to intentionally acquire, modify, and improve their memes (Digalakis and Margaritis, 2004). A number of publications have shown MAs to be very effective in many hard combinatorial optimization problems; examples of such are the traveling salesman problem (Moscato and Norman, 1992; Krasnogor and Smith, 2000; Merz and Freisleben, 2001; Tsai *et al.*, 2002), and the job shop scheduling problem (Yin, 2004). In a survey of the literature, however, one finds no MA applications in the resource-constrained project scheduling problem for maximizing NPV (Chen and Chyu, 2007; 2008).

2 Economic optimization model

The project scheduling problem considered in this paper for a single project assumes the non-preemptive multimode resource constrained max-NPV (Chen and Chyu, 2008). There are I activities in the project and M_i modes for activity i. The project has K^ρ renewable resource types and K^ν nonrenewable resource types. The available amounts of renewable resource type k for each period and nonrenewable resource type k for the entire project are R_k^ρ and R_k^ν, respectively. Each mode m of activity i has its unique duration (d_{im}), consumptions on renewable and nonrenewable resource type k (r_{imk}^ρ and r_{imk}^ν), and costs. As such, all the activities should be completed by a common due date (T). Over the course of completing the project, a series of cash flows occurs as inflows and outflows, both in installments. The cash flows are determined based on total resource cost (TRC) as follows:

$$\mathrm{TRC}=\sum_{i=1}^{I}\sum_{m=1}^{M_i}\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i}\left(\sum_{k=1}^{K^\rho}u_k^\rho r_{imk}^\rho+\sum_{k=1}^{K^\nu}u_k^\nu r_{imk}^\nu\right)\cdot x_{imt}. \quad (1)$$

Herein, EFT_i denotes the earliest finish time for activity i while LFT_i the latest finish time for activity i; u_k^ρ and u_k^ν are unit costs of renewable and nonrenewable resource type k, respectively; x_{imt}, the binary decision variable, equals one when activity i finishes at time t and executes with mode m; otherwise, x_{imt} becomes zero. The other cost (OC) refers to an unexpected cost, behaving as the uncertainty of the real life project. The OC is determined by a multiplier of certain percentage from a TRC. The cash outflow is the sum of the TRC and the OC; the cash inflow is calculated by an estimated percentage of profit earning from the cash outflow. Finally, the time value of money is considered through discounting the cash flows at a daily discount rate (α).

Within the following mathematical model, cash inflow (F_{im}^{in}) of activity i happens at $t-d_{im}$ (i.e., the beginning of an activity) while cash outflows (F_{im}^{out}) of activity i at t (i.e., at the end of an activity). The objective is to maximize the project's NPV as shown in Eq. (2):

$$\max\left(\sum_{m=1}^{M_i}\sum_{i=1}^{I}\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i}F_{im}^{\mathrm{in}}\mathrm{e}^{-\alpha(t-d_{im})}x_{im(t-d_{im})}-\sum_{m=1}^{M_i}\sum_{i=1}^{I}\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i}F_{im}^{\mathrm{out}}\mathrm{e}^{-\alpha t}x_{imt}\right) \quad (2)$$

subject to

$$\sum_{m=1}^{M_i}\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i}x_{imt}=1,\qquad i=1,2,\ldots,I, \quad (3)$$

$$\sum_{m=1}^{M_j}\sum_{t=\mathrm{EFT}_j}^{\mathrm{LFT}_j}tx_{jmt}\le\sum_{m=1}^{M_i}\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i}(t-d_{im})x_{imt},\quad j\in P_i;\ i=1,2,\ldots,I, \quad (4)$$

$$\sum_{i=1}^{I}\sum_{m=1}^{M_i}r_{imk}^\rho\cdot\sum_{q=\max\{t,\mathrm{EFT}_i\}}^{\min\{t+d_{im}-1,\mathrm{LFT}_i\}}x_{imq}\le R_k^\rho,\quad k=1,2,\ldots,K^\rho;\ t=1,2,\ldots,T, \quad (5)$$

$$\sum_{i=1}^{I}\sum_{m=1}^{M_i}r_{imk}^\nu\cdot\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i}x_{imt}\le R_k^\nu,\quad k=1,2,\ldots,K^\nu;\ t=1,2,\ldots,T, \quad (6)$$

$$\sum_{m=1}^{M_i}\sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i} tx_{imt} \le T, \quad i=1, 2, \ldots, I, \tag{7}$$

$$x_{imt} \in \{0,1\}, \ \forall i, \ m, \ t. \tag{8}$$

Eq. (3) ensures that each activity begins with one mode and at one time period only. Eq. (4) defines the precedence relationships in a set of direct predecessors of activity i, denoted as P_i. Eqs. (5) and (6) represent the renewable and nonrenewable resource utilization requirements and prevent resource overuse. Eq. (7) specifies that the project must be completed before due date T. Eq. (8) defines the decision variable (x_{imt}) as a binary one.

3 Two-phase metaheuristic

In this section, we first briefly outline the two-phase metaheuristic proposed for this problem. In the first phase, we implicitly or explicitly enumerate all feasible extensions of mode combinations by the B&B algorithm. Once a feasible mode assignment is found, it must go through a preliminary filtering test for its eligibility. For this filtering test, the selected mode set is arranged by a specified priority rule (PR) and local search (LS). Then, its makespan is compared with the specified threshold. When the current feasible mode set passes the test, that is, its makespan is less than the specified threshold, the second phase is activated. Otherwise, it will return to the B&B algorithm and look for the next feasible mode set. Once the second phase begins, the MA then derives a solution which specifies how each activity should be performed to achieve the maximum NPV. A detailed description of B&B and MA is provided in the next two subsections.

3.1 Branch-and-bound algorithm

Fig. 1a gives an overview on the structure of the B&B algorithm. The notion of mode assignment lists corresponds to the nodes of the search tree in our B&B algorithm. In the following example, we assume a total of 5 activities (excluding 2 dummies) and a total of 3 modes per each activity. The B&B algorithm begins with emanating three branches from mode 0 (or node 0) of activity 1 (or stage 1) to activity 2. Then, the branch emanates another three branches from mode 1 (node 1) of activity 2 (or stage 2). The same goes from mode 2 (node 2) and mode 3 (node 3) of activity 2, etc. Further details on some of its aspects are provided below.

For the selection of an activity pair to branch on, we select the activity pair $\{i, j\}$ based on an increasing numerical order. Then we select its corresponding modes also based on an increasing mode index (Fig. 1b). In other words, the algorithm begins to find the earliest feasible mode assignment for activities 1, 2, 3 and so on, in order. Nodes that represent precedence- and resource-feasible project networks and that are not fathomed by any node fathoming rule described later lead to a new branching.

However, for an activity whose mode cannot be assigned without violating some resource constraints, the algorithm backtracks to the most recently assigned activity and attempts to assign modes m+1, m+2, …, M_i, etc. Fig. 1c shows such occurrence at mode 1 in activity 5 (shown in dashed arrow). So the procedure backtracks to the previous level in the search tree and reconsiders the other modes pending at that level (e.g., mode 2 in activity 5). Note that, in our procedure, nodes can be fathomed when the precedence or resource constraints are violated, or its critical path method (CPM) time exceeds the target due date. The CPM value is determined by first calculating the shortest duration among modes of each activity and, as the branching begins, computing the lower bound on the network. Hence, for nodes are not fathomed, the process of assignment and backtracking continues until all activities are assigned a feasible mode, or until an attempt is made for all activities.

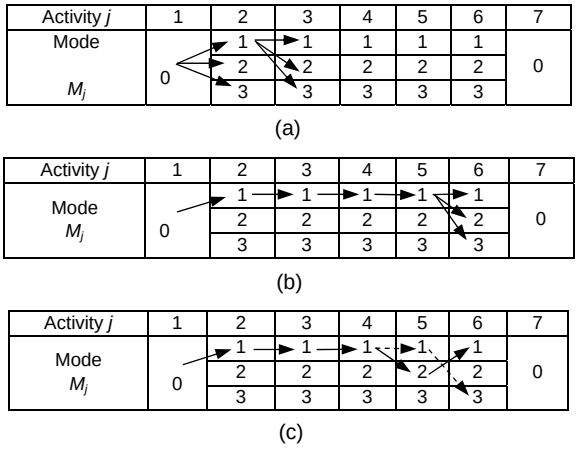

Fig. 1 The notion of (a) mode lists, (b) branching strategy, and (c) backtracking in the branch-and-bound algorithm
The solid arrow: a successful branching; the dotted arrow: an unsuccessful branching so a backtracking is needed

Meanwhile, nodes are fathomed under two conditions: (1) a resource conflict occurs where the resource requirement of the set of unscheduled activities exceeds the capacity of at least one type of non-renewable resources; (2) without considering resource constraints, the corresponding schedule computed by the CPM exceeds a given fixed deadline. When any one of the criteria above is reached, the current branching operation will stop and a new branch will be investigated. The entire B&B procedure terminates when the algorithm reaches the end stage.

3.2 Memetic algorithm

The first use of the term 'memetic algorithms' (MA) appeared in Holland (1975) and was defined as an evolution-based search augmented by hybridizing with one or more phases of local search procedures or by the use of problem-specific information. We show, in pseudo-code, the general mechanism for the MA below.

```
Procedure memetic algorithm
    Generate Initial_Pop;
    do Local-Search(i);
    while (not stopping criterion) do
        Convergence Check Pop(i);
        if convergence, do
            Mutate_Pop(i);
            Local_Search_Pop(i_c);
        else, do
            Recombine_Pop(i_a, i_b);
            Local_Search_Pop(i_c);
            Similarity_Test;
        end if
        Update Pop;
    end while
```

The algorithm begins by generating the initial population, and then a local search is applied to all members in population. After checking the convergence rate of the population, mutation is performed if the rate is greater than the threshold. Again, a local search is applied to the offspring generated in mutation operation. Otherwise, a certain level of diversity is maintained, recombination operation is employed, and a local search is implemented to improve the offspring. After that, the following steps continue until the terminating condition is reached.

In this implementation of the MA, five main components—the initial population, the solution representation and its decoding scheme, the selection, the recombination and the mutation, and the local search, are described as follows.

3.2.1 Population initialization

As is known, the generation of the initial population commonly involves random, multi-priority rules and sampling approaches. In an MA, the initial population is usually randomly generated; however, extensive testing showed that such population generation would lead to slow convergence. On the contrary, the initial population generated by user-specified priority rules makes a better algorithm. This is commonly referred to as the multi-priority rule approach which constructs each solution from a different priority rule. The sampling approach, on the other hand, considers only one priority rule and obtains different solutions by biasing the selection of the priority rule through a random device (Kolisch, 1996b). In this paper, we employed a combined procedure with one half of population by the weighted EFT rule and the other half by the weighted LFT rule. We consider the LFT rule because it is reported to be one of the best priority rules appearing in Alcaraz and Maroto (2001). On the other hand, we employed the EFT rule just to provide enough diversity of the initial population.

3.2.2 Solution representation and decoding scheme

For the solution representation in this study (Fig. 2), we encoded the feasible solution by the two arrays: one is a list of execution modes for the corresponding activities called the mode assignment (determined by the method, i.e., the B&B algorithm at the first phase) and the other is a precedence-feasible combination of activities called the activity list (where each position is occupied by an activity index, and each activity can randomly appear in any position of the list as long as it is after all its predecessors). Note that the first and last positions are for two dummy nodes (Fig. 2), so the corresponding modes are both denoted as 0. To construct the related project schedule, we consider two different decoding schemes, parallel list scheduling (PLS) and serial list

Sequence	0	1	…	*i*	…	*N*	*N*+1
Activity	0	…	…	*j*	…	…	*N*+1
Mode	0	…	…	*m*	…	…	0

Fig. 2 Activity list representation with corresponding modes

scheduling (SLS), to transform an activity list (AL) into a feasible schedule. According to Alcaraz and Maroto (2001), the forward pass and SLS predominantly performs better; more so, we have confirmed this observation from our preliminary testing. Thus, we decided to implement the forward serial list scheduling (FSLS). The necessary steps for implementing both decoding schemes can be referred to in Kolisch (1996a).

3.2.3 Selection operator

Selection, by all means, imitates the natural phenomenon of the fittest individuals dominating over weaker ones. Though several ways of implementing such selection mechanism exist, we have applied a common selection scheme, called tournament selection (Goldberg, 1989; Michalewicz, 1996). In a binary tournament selection, two individuals are randomly selected from the population as parent solutions and compete for survival. The offspring solution with the best fittest value will be added to the following generation. This way we can ensure a higher probability of surviving for the best individuals.

3.2.4 Recombination operator

Many different general and specific recombination operators have been proposed in the literature. In our study, we applied a modified order-based technique. In our modified order-based procedure, a number of activities are first randomly selected from one parent, and then the order of activities in the selected position is imposed onto the corresponding activities in the other parent. The remaining empty positions of the offspring are sequentially filled with the unselected activities according to the order of the second parent.

To illustrate the modified order-based recombination process, we refer to Fig. 3 showing the precedence relations of an example project network with the maximum resource capacity of 6 units, and Fig. 4 showing how an offspring is produced using the order-based genetic operator. In Fig. 4, activities 1, 2, and 5 are first selected from parent 1, and then these three activities are imposed onto the positions 3, 6, and 7 of the proto-offspring using the order of activities in parent 1. Lastly, the remaining activities are taken from parent 2 to fill the vacant positions in the offspring by following their orders in parent 2. Next, the offspring is checked for the precedence feasibility, and activities that violate their precedence relations are shifted leftward one by one to their first feasible positions.

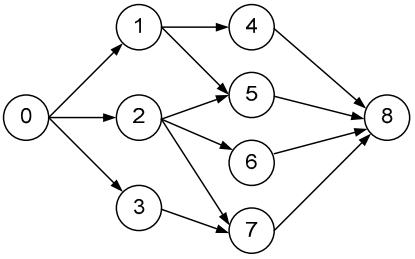

Fig. 3 Precedence relations of a project network example

Parent 1:	0	**1**	4	**2**	6	**5**	3	7	8
Parent 2:	0	3	**2**	7	6	**1**	**5**	4	8
Offspring:	0	3	**1**	7	6	**2**	**5**	4	8

Fig. 4 An illustration of the order-based recombination operator

From our example, there is a precedence violation between activities 2 and 6; thus activity 2 is moved leftward to the first feasible position, which is in front of activities 6 and 7 (Fig. 5). After a quick adjustment, a newly produced offspring {0, 3, 1, 2, 7, 6, 5, 4, 8} is formed. Finally, a similarity ratio test is performed to check if the offspring is acceptable. The similarity ratio is defined as the total number of activities having identical positions over the total number of non-dummy activities. The calculation of similarity ratios in this example turns out to be 1/7 to parent 1 and 3/7 to parent 2.

Proto-offspring:	0	3	**1**	7	6	**2**	**5**	4	8
Offspring:	0	3	**1**	**2**	7	6	**5**	4	8

Fig. 5 An illustration of the repairing strategy for infeasible offspring

3.2.5 Mutation operator

As the recombination implemented will lead to a quick loss of diversity, the mutation we use here will preserve the diversity of the population. We implement a traditional mutation strategy, called adjacent swapping when the population fails to pass the convergence check. To do so, two adjacent positions are first randomly picked. Then the alleles in these two adjacent positions swap their values while the precedence feasibility is sustained. An example of the

adjacent swapping mutation is illustrated in Fig. 6. Assume that two adjacent activities 5 and 6 are randomly selected and their positions in the activity list and corresponding modes are swapped. Since the outcome does not violate the precedence feasibility, no repair is needed. A mutation operation is applied to all members but the best one in the population. This can be considered as an Elitist strategy to preserve the best solution obtained.

Fig. 6 An illustration of the adjacent swap mutation operator

3.2.6 Local search

In MA, we use the local search operator to improve solutions in the configuration space before inserting into the population. From our preliminary testing, we applied several local searches to refine a constructed schedule to local optimum. For instance, the forward-backward improvement (FBI) begins with the backward pass to a feasible schedule. That is, the activities are rescheduled in decreasing order of their scheduled completion time and shifted to the latest feasible position in their forward free slack. Then, the forward pass is applied to the feasible schedule obtained by the backward pass. The activities here, however, are rescheduled in increasing order of their scheduled start time and shifted left to the earliest feasible position in their backward free slack. We can obtain further details of the FBI method in (Klein, 2000; Tormos and Lova, 2001; Valls *et al.*, 2004; 2005).

Another local search, the enhanced left move method, selects an activity at random and bundles up this activity with all of its direct predecessors. These activities are moved leftward together to the most front positions of the activity list. Then the two-swap method randomly selects the first activity from the activity list, and then determines the corresponding left- and right-shift limits. Between these two limits, the second activity is randomly selected for possible swapping. The procedure performs swapping when these two activities have different start time in the current schedule and follow their precedence constraints; otherwise, the two-swap method will begin the random selection again. As for the insertion method that randomly chooses an activity and finds the farthest left- and right-predecessors of this activity, the chosen activity is then inserted to the position where a resource conflict occurs. However, when no resource conflict occurs, another activity will be randomly chosen and the procedure repeats again.

4 Results and discussion

The implementations of the B&B and MA algorithms were coded in Visual C++. The experiments were run on an Intel Core 2 Quad CPU Q6600 2.4 GHz and 2 GB RAM. We have performed a series of computational experiments on our proposed two-phase hybrid metaheuristic approach and also on different solution procedures in order to examine and compare the performance effectively.

4.1 Test problems

We use randomly generated test problems from ProGen developed by Kolisch *et al.* (1995). These test problems with 2 renewable and 2 non-renewable resources cover activity sizes of 10, 12, 14, 16, and 18 (denoted as *j*10, *j*12, *j*14, *j*16, and *j*18 respectively). Also, durations for these modes in each activity are discrete values in [1, 10]. In each problem size, we consider 16 different problem parameter settings for instances *j*10 to *j*18.

The original ProGen dataset, however, does not contain data on cash flows for the activities such as unit costs and arbitrary project deadlines. Thus, additional data were generated to modify the original dataset. We considered 2 unit cost settings: in the first setting, the unit cost is a constant at 100 monetary units for both renewable and nonrenewable resources (denoted as instance set I); in the second setting, the unit cost is generated as independent realizations of a discrete uniform random variable on [100, 400] (denoted as instance set II). Hence, there are 80 instances for each cost distribution. In addition, the other cost (OC) is determined by a multiplier of 10%–20% from a total resource cost (TRC). The target deadline T for each problem instance is an integer randomly generated from the interval $[1.1T^*, 1.4T^*]$ where T^* is the best makespan. The experiment was performed at a discount rate of 0.1% per period and a profit margin of 10%, both being constant over the entire planning horizon.

4.2 Parameter calibration

The performance of metaheuristics is commonly influenced by the fine tune-up of the parameter values in the algorithms. The number of parameters needed, therefore, will determine the load of calibration experiments. To reduce the complexity of experiments, this study has removed two popular genetic algorithm (GA) parameters, probability of crossover and probability of mutation, from the design of MA. Instead, the operation of crossover and mutation is controlled by a new parameter, i.e., the convergence rate of the current population. Once the convergence rate is higher than the pre-specified threshold, the mutation operator will be activated; otherwise, the crossover operator will be employed. All the experiments are based on the following parameters: the convergence rate=0.8, the population size=20, and the stopping criterion of MA=300 evaluations unless some different settings are specified. One set of test instances ($j16$) consisting of 16 instances is employed for the parameter calibration, since it is considered as the medium-to-large instance among all sets. Both $j16$ sets from different unit cost settings will be tested. To better distinguish the effect of parameter settings, the best NPV value among the experiments is selected, and then the deviation from the best solution is calculated from the comparison.

In the first experiment of parameter calibration, three convergence rates 0.2, 0.5, and 0.8 are tested. Figs. 7a and 7b show the box plot of the deviation (in monetary unit) over different convergence rate settings in instance sets I and II, respectively. Two figures provide similar results. When the convergence rate=0.2, the deviation is the largest, and the solution quality improves with the increase of the convergence rate. Thus, we conclude that the convergence rate=0.8 is the best setting for the convergence rate in our experiment.

Another parameter in the proposed MA required to be investigated is the population size. Three population sizes, 10, 20, and 30, were tested to compare the performance. Figs. 8a and 8b illustrate the box plot of the deviation over different population sizes in 16 $j16$ instances of instance sets I and II, respectively. When the population size is equal to 10, the deviation fluctuates between 0 and 424 for instance set I and between 0 and 1290 for instance set II. On the other hand, when the population size is increased to 20, the solution quality is improved tremendously and the similar performance remains when the population size is set to 30. Therefore, the best population size can be concluded as either 20 or 30 (we choose 20 as the population size for the experiments thereafter).

The last parameter is related to the convergence performance of the MA. Four different values, 100, 200, 300, and 400, of the total number of evaluations for each feasible mode assignment in MA were tried. From Figs. 9a and 9b we can clearly observe that the solution quality improves when the number of evaluations increases and also the improvement stagnates while the number of evaluations reaches

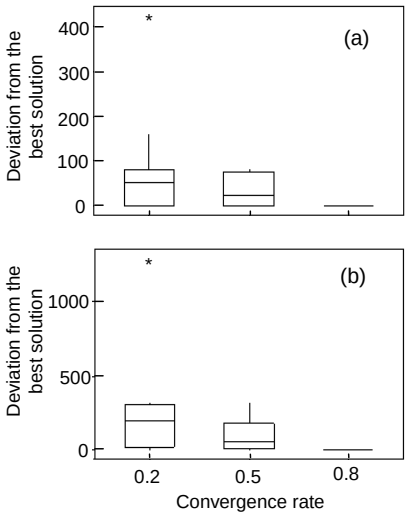

Fig. 7 Box plots of the deviation from the best solution over different convergence rates collected from 16 instances of $j16$ in (a) instance set I and (b) instance set II

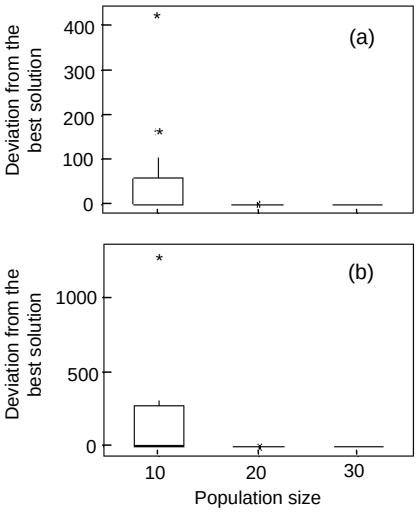

Fig. 8 Box plots of the deviation from the best solution over different population sizes collected from 16 instances of $j16$ in (a) instance set I and (b) instance set II

300. Therefore, to save the computational expense, we set the stopping criterion of MA as 300 evaluations for each feasible mode. And the experiments in the following section were based on the calibration results suggested here: convergence rate=0.8, population size=20, and the stopping criterion of MA=300 evaluations.

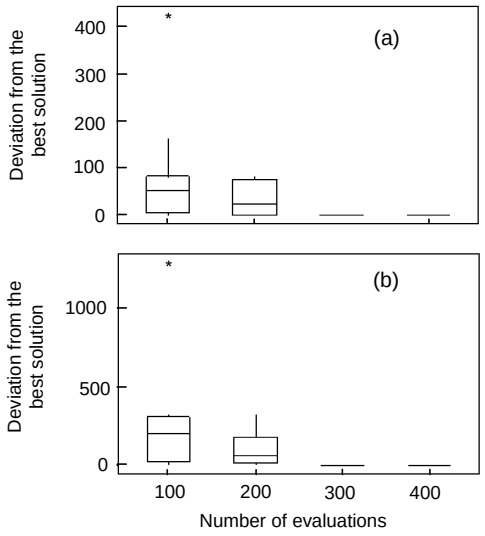

Fig. 9 Box plots of the deviation from the best solution over different numbers of evaluations collected from 16 instances of *j*16 in (a) instance set I and (b) instance set II

4.3 Design of the testing environments

We have designed our testing environment based on different solution methods in the first and second phases. Other than our proposed B&B algorithm used in the first phase, we implement mode selection rules (MSR) for performance comparison. Our literature survey has identified a few issues with respect to the multi-mode problem: first, there appears some common mode selection rules applied in makespan problems and mostly designed for renewable resources, not for nonrenewable resources (Boctor, 1993; Kolisch, 1996b; Salewski *et al.*, 1997; Buddhakulsomsiri and Kim, 2007); second, few mode selection rules in the makespan problem are applied in the NPV maximization problem; third, mode selection rules in maximizing NPV vary among studies and are problem-specific. Two mode selection rules, shortest feasible mode (SFM) and least resource proportion (LRP), are adopted directly from Boctor (1993). The other rules, including least product sum of resource and duration (LPSRD), least product sum of resource cost and duration (LPSRCD), least total resource usage (LTRU), least criticality ratio (LCR), the least ratio sum of non-renewable resource (LRS), least total resource cost (LTRC), least relative resource consumption (LRRC) and LRRC*, maximum ratio of the average remaining capacity (MRARC) and MRARC*, are either modified or developed to take into account the duration of each activity mode alternative and the cost of each resource type (refer to Chen and Chyu (2007) for further implementation details).

For the solution methods used in the second phase for activity scheduling, the priority rules were applied for performance comparison with the MA. They are well known in the makespan minimization problem and listed below: most immediate successors (MIS), most total successors (MTS), least non-related jobs (LNRJ), minimum slack first (MSLK), earliest finish time (EFT), latest finish time (LFT), greatest cumulative resource demand (GCUMRD), resource scheduling method (RSM), weighted resource utilization ratio and precedence (WRUP), activity-time (ACTIM), and resource over time (ROT). Further detailed implementation of the above rules can be found in (Davis and Patterson, 1975; Boctor, 1993; Klein, 2000). Overall, there are a total of 11 priority rules implemented in this study.

Fig. 10 shows the four testing environments in our experimental design: the first experiment implements the B&B algorithm in the first phase and the MA in the second phase; the second experiment applies the B&B algorithm in the first phase but the PR in the second phase; the third experiment implements the MSR in the first phase but the MA in the second phase; the fourth experiment applies the MSR in the first phase and the PR in the second phase. In order to demonstrate the effectiveness of our proposed hybrid metaheuristic, however, we also implemented the B&B algorithm in the first phase and the GA in the second phase as our fifth testing environment.

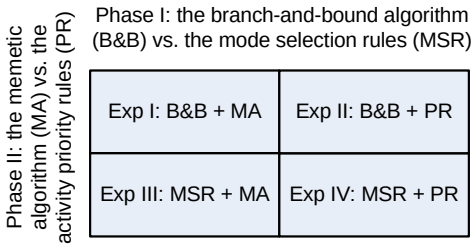

Fig. 10 Design for test environments

In order to measure the quality of the solution method under study, we define the percentage deviation from the best known as

$$\Delta_{\text{npv}} = \left| \frac{\text{npv}^{\text{best}} - \text{npv}^{\text{current}}}{\text{npv}^{\text{best}}} \right| \times 100\%,$$

where npv^{best} denotes the best known net present value obtained and $\text{npv}^{\text{current}}$ denotes the current net present value determined in each test environment.

Below we present the computational results and in each case the following numbers are shown:

avg. dev.: the average percentage deviation from the best solution known;

max. dev.: the maximum percentage deviation from the best solution known;

min. dev.: the minimum percentage deviation from the best solution known;

#infeasible: the number of instances for which the algorithm is not able to find a solution;

#optimal: the number of instances for which the algorithm finds a solution equal to the best solution known.

4.3.1 Performance on the B&B algorithm versus MSR

Tables 1–4 report the comparison results for Experiments II and IV under different unit costs. It reveals that the B&B algorithm at the first phase has successfully found more feasible modes and reduced the deviation from the best solution. When considering the percentage deviation, B&B+PR (Exp II) performs quite well over different problem sizes, but MSR+PR (Exp IV) can provide only comparable performance in *j*10 instances. When the problem size goes up, it is obvious that the B&B algorithm has shown its merit.

Similar observation can be found in Tables 5–8. Two variations of MA are compared here. The difference is how the mutation performs. In MA1, mutation is employed as regenerating the initial population; in MA2, adjacent swap is used as the mutation operator. Although MSR+MA (Exp III) performs better than MSR+PR (Exp IV), the drawback that fails to find feasible solutions still exists. By employing the B&B algorithm at the first phase, MA is always able to find feasible solutions to the end of the procedure and with much better NPV. Thus, we can conclude that both Exp I and II outperform Exp III and IV by employing the B&B algorithm at the first phase.

Table 1 The results of Experiments II and IV in deviation for unit cost at 100

Instance	avg. dev. (%)											
	Exp II	Exp IV										
		MIS	MTS	LNRJ	MSLK	EFT	LFT	GCUMRD	WRUP	RSM	ACTIM	ROT
*j*10	0.001	11.086	4.942	11.095	4.959	4.942	4.957	11.093	11.095	4.960	4.954	4.941
*j*12	0.000	24.549	24.447	24.439	30.076	24.442	24.448	24.414	24.403	24.416	30.040	24.387
*j*14	0.049	35.699	23.928	23.926	35.551	29.986	35.545	35.681	35.675	29.983	35.525	29.961
*j*16	0.012	19.293	19.201	19.204	19.200	24.445	19.199	24.437	19.180	24.443	19.198	24.435
*j*18	0.010	24.934	19.521	19.527	24.943	24.936	19.526	19.537	19.526	24.939	19.533	19.524
Instance	min. dev. (%)											
	Exp II	Exp IV										
		MIS	MTS	LNRJ	MSLK	EFT	LFT	GCUMRD	WRUP	RSM	ACTIM	ROT
*j*10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*j*12	0.000	2.387	2.340	2.293	2.247	2.200	2.153	2.106	2.059	2.013	1.966	1.919
*j*14	0.000	1.217	1.195	1.173	1.151	1.129	1.107	1.085	1.063	1.041	1.018	0.996
*j*16	0.000	1.276	1.255	1.276	1.276	1.264	1.255	1.276	1.276	1.287	1.278	1.207
*j*18	0.000	0.475	0.475	0.475	0.475	0.475	0.475	0.475	0.475	0.475	0.475	0.475

Exp II: B&B+PR; Exp IV: MSR+PR. B&B: branch-and-bound algorithm; PR: priority rule; MSR: mode selection rules. MIS: most immediate successors; MTS: most total successors; LNRJ: least non-related jobs; MSLK: minimum slack first; EFT: earliest finish time; LFT: latest finish time; GCUMRD: greatest cumulative resource demand; RSM: resource scheduling method; WRUP: weighted resource utilization ratio and precedence; ACTIM: activity-time; ROT: resource over time. The same for Tables 2–4

Table 2 The results of Experiments II and IV in deviation for unit cost at [100, 400]

Instance	avg. dev. (%)										
	Exp II	Exp IV									
		MIS	MTS	LNRJ	MSLK	EFT	LFT	GCUMRD	WRUP	RSM	ACTIM	ROT
*j*10	0.000	11.183	6.177	17.420	12.415	6.171	6.188	11.199	11.188	6.187	6.194	6.171
*j*12	0.016	19.013	19.003	19.004	30.621	19.005	24.567	19.000	18.989	18.999	30.611	18.980
*j*14	0.016	30.295	19.786	25.571	24.893	30.804	29.999	25.185	25.180	30.807	24.376	25.175
*j*16	0.011	19.039	19.060	19.063	19.057	24.787	19.058	24.773	19.038	24.787	19.063	24.780
*j*18	0.013	18.738	18.736	18.740	24.514	24.497	30.534	24.511	18.741	30.530	24.508	18.736
Instance	min. dev. (%)										
	Exp II	Exp IV									
		MIS	MTS	LNRJ	MSLK	EFT	LFT	GCUMRD	WRUP	RSM	ACTIM	ROT
*j*10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
*j*12	0.000	2.794	2.795	2.837	2.893	2.794	2.795	2.893	2.835	2.840	2.893	2.794
*j*14	0.000	2.646	2.646	2.646	2.646	2.646	2.646	2.646	2.646	2.646	2.646	2.646
*j*16	0.000	1.910	1.910	1.910	1.910	1.910	1.910	1.910	1.910	1.910	1.910	1.910
*j*18	0.000	3.080	3.077	3.087	3.114	3.053	3.114	3.114	3.114	3.114	3.114	3.076

Table 3 The results of Experiments II and IV in #infeasible for unit cost at 100

Instance	#infeasible										
	Exp II	Exp IV									
		MIS	MTS	LNRJ	MSLK	EFT	LFT	GCUMRD	WRUP	RSM	ACTIM	ROT
*j*10	0	1	0	1	0	0	0	1	1	0	0	0
*j*12	0	3	3	3	4	3	3	3	3	3	4	3
*j*14	0	5	3	3	5	4	5	5	5	4	5	4
*j*16	0	2	2	2	2	3	2	3	2	3	2	3
*j*18	0	3	2	2	3	3	2	2	2	3	2	2

Table 4 The results of Experiments II and IV in #infeasible for unit cost at [100, 400]

Instance	#infeasible										
	Exp II	Exp IV									
		MIS	MTS	LNRJ	MSLK	EFT	LFT	GCUMRD	WRUP	RSM	ACTIM	ROT
*j*10	0	1	0	2	1	0	0	1	1	0	0	0
*j*12	0	2	2	2	3	2	2	2	2	2	4	2
*j*14	0	4	2	3	3	4	4	3	3	4	3	3
*j*16	0	2	2	2	2	3	2	3	2	3	2	3
*j*18	0	2	2	2	3	3	4	3	2	4	3	2

Table 5 The results of Experiments I and III in deviation for unit cost at 100

Instance	avg. dev. (%)				min. dev. (%)			
	Exp I		Exp III		Exp I		Exp III	
	MA1	MA2	MA1	MA2	MA1	MA2	MA1	MA2
*j*10	0.000	0.000	10.264	10.261	0.000	0.000	0.000	0.000
*j*12	0.000	0.000	24.603	24.597	0.000	0.000	1.872	1.826
*j*14	0.000	0.000	23.888	23.885	0.000	0.000	0.974	0.952
*j*16	0.002	0.000	19.162	19.160	0.000	0.000	1.172	1.180
*j*18	0.001	0.000	19.508	19.507	0.000	0.000	0.475	0.475

Exp I: B&B+MA; Exp III: MSR+MA. B&B: branch-and-bound algorithm; MSR: mode selection rules; MA: memetic algorithm. In MA1, mutation is employed as regenerating the initial population; in MA2, adjacent swap is used as the mutation operator. The same for Tables 6–8

Table 6 The results of Experiments I and III in deviation for unit cost at [100, 400]

Instance	avg. dev. (%)				min. dev. (%)			
	Exp I		Exp III		Exp I		Exp III	
	MA1	MA2	MA1	MA2	MA1	MA2	MA1	MA2
*j*10	0.000	0.000	11.166	11.166	0.000	0.000	0.000	0.000
*j*12	0.000	0.000	18.817	18.815	0.000	0.000	2.778	2.778
*j*14	0.000	0.000	19.259	19.257	0.000	0.000	2.646	2.646
*j*16	0.001	0.018	19.182	19.025	0.000	0.000	1.910	1.910
*j*18	0.002	0.001	18.974	18.975	0.000	0.000	3.053	3.060

Table 7 The results of Experiments I and III in #infeasible for unit cost at 100

Instance	#infeasible			
	Exp I		Exp III	
	MA1	MA2	MA1	MA2
*j*10	0	0	1	1
*j*12	0	0	3	3
*j*14	0	0	3	3
*j*16	0	0	2	2
*j*18	0	0	2	2

Table 8 The results of Experiments I and III in #infeasible for unit cost at [100, 400]

Instance	#infeasible			
	Exp I		Exp III	
	MA1	MA2	MA1	MA2
*j*10	0	0	1	1
*j*12	0	0	2	2
*j*14	0	0	2	2
*j*16	0	0	2	2
*j*18	0	0	2	2

4.3.2 Performance on the MA versus PR and GA

Since B&B at the first phase has dominated MSR, this section deals with the comparison of different algorithms at the second phase, i.e., the comparison among PR, two MA variations, and GA. Tables 9 and 10 show the percentage deviation from the best objective function values for each problem size on different cost distributions. In Table 9 (unit cost at 100, i.e., instance set I), MA2 outperforms other competing algorithms in both average and maximum percentage deviations. When considering the case of unit cost at [100, 400] (instance set II) (Table 10), both MA variations provide competitive performance and beat PR and GA.

When taking the number of best solutions found into account, MA2 is an absolute winner in the case of unit cost 100 (Table 11) since MA2 is able to find the best solution in all 80 instances no matter what size it is. Also, the performance of both MA variations is more stable than that of PR and GA as summarized in both Tables 11 and 12. When the problem size arises, the number of best solutions found by PR and GA decreases. To sum up, MA2 has successfully found 157 best solutions out of 160 instances.

Table 9 Comparison in average and maximum deviations for unit cost at 100

Instance	avg. dev. (%)			
	PR	MA1	MA2	GA
*j*10	0.001	0.000	0.000	0.002
*j*12	0.000	0.000	0.000	0.000
*j*14	0.049	0.000	0.000	0.001
*j*16	0.012	0.002	0.000	0.007
*j*18	0.010	0.001	0.000	0.011
Instance	max. dev. (%)			
	PR	MA1	MA2	GA
*j*10	0.013	0.000	0.000	0.013
*j*12	0.001	0.001	0.000	0.001
*j*14	0.762	0.000	0.000	0.015
*j*16	0.048	0.014	0.000	0.048
*j*18	0.069	0.011	0.000	0.066

Table 10 Comparison in average and maximum deviations for unit cost at [100, 400]

Instance	avg. dev. (%)			
	PR	MA1	MA2	GA
*j*10	0.000	0.000	0.000	0.000
*j*12	0.016	0.000	0.000	0.005
*j*14	0.016	0.000	0.000	0.137
*j*16	0.011	0.001	0.018	0.002
*j*18	0.013	0.002	0.001	0.009
Instance	max. dev. (%)			
	PR	MA1	MA2	GA
*j*10	0.000	0.000	0.000	0.000
*j*12	1.067	0.002	0.000	0.074
*j*14	0.048	0.000	0.006	1.136
*j*16	0.036	0.007	0.286	0.028
*j*18	0.070	0.016	0.016	0.128

Table 11 Comparison in #optimal for unit cost at 100

Instance	#optimal			
	PR	MA1	MA2	GA
*j*10	15	16	16	14
*j*12	15	15	16	15
*j*14	14	16	16	13
*j*16	10	14	16	11
*j*18	10	14	16	11

Table 12 Comparison in #optimal for unit cost at [100, 400]

Instance	#optimal			
	PR	MA1	MA2	GA
*j*10	16	16	16	16
*j*12	12	14	16	14
*j*14	14	16	15	11
*j*16	11	13	14	15
*j*18	10	14	15	14

4.3.3 Discussion on the number of feasible modes

The computational complexity of the multi-mode problem increases explosively with the raise of the problem size. From Table 13, the number of possible mode assignments in an instance can go up to 387 420 489 when the 18 activities are under consideration. To have a better understanding of the test instances, we collected the number of feasible modes found by the B&B algorithm over instances (Table 14). Although the numbers in Table 14 are not as enormous as the ones in Table 13, there are still over 1.4 million feasible modes, and over 9 million feasible modes on average in *j*16 and *j*18 instances respectively. For most instances, the proposed B&B+MA algorithm is able to solve it within a few seconds. Considering the complexity of the problems, B&B+MA does provide an effective and efficient approach for the multi-mode scheduling problem with both nonrenewable and renewable resource constraints.

Table 13 Number of maximum possible mode assignments on different problem sizes

Instance	Maximum	Instance	Maximum
*j*10	59 049	*j*16	43 046 721
*j*12	531 441	*j*18	387 420 489
*j*14	4 782 969		

Table 14 Statistics on the number of feasible modes over instances

Instance	Minimum	Maximum	Average
*j*10	58	14 619	3187
*j*12	37	97 408	27 597
*j*14	4186	1 624 537	399 166
*j*16	745	9 062 785	1 474 323
*j*18	73 335	44 345 658	9 145 742

5 Conclusions

In this paper, we have presented a model and algorithms for multi-mode resource-constrained project scheduling to maximize the expected net present value when both renewable and nonrenewable resources are present. Though this problem is NP-hard, we have introduced and formulated a generic model for solving optimally scheduling problem. Also, we developed the two-phase hybrid metaheuristic: using a branch-and-bound algorithm to solve the mode assignment problem in the first phase; then, by transforming a multi-mode case into a single-mode problem, the second phase was activated and the memetic algorithm was applied to achieve good quality solutions. Finally, for a comprehensive design of experiment, the proposed two-phase hybrid metaheuristic was compared with different approaches, mode selection rules and activity priority rules. They are found to be good in quality with respect to several performance measurements—#optimal, #infeasible, avg. dev., min. dev., and max. dev. Further research aimed at investigating the functional relationship between mode assignment and cash flow scheduling problems may be of interest.

References

Alcaraz, J., Maroto, C., 2001. A robust genetic algorithm for resource allocation in project scheduling. *Ann. Oper. Res.*, **102**(1-4):83-109. [doi:10.1023/A:1010949931021]

Boctor, F.F., 1993. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *Int. J. Prod. Res.*, **31**(11):2547-2558. [doi:10.1080/00207549308956882]

Buddhakulsomsiri, J., Kim, D.S., 2007. Priority rule-based heuristic for multi-mode resource constrained project scheduling problems with resource vacations and activity splitting. *Eur. J. Oper. Res.*, **178**(2):374-390. [doi:10.1016/j.ejor.2006.02.010]

Chen, A.H.L., Chyu, C.C., 2007. On Maximizing the Net Present Value of a Project: a Comparison of Mode Selection Rules and Activity Priority Rules. Proc. Conf. of the Operations Research Society of Taiwan, No. 7040615967.

Chen, A.H.L., Chyu, C.C., 2008. A Memetic Algorithm for Maximizing Net Present Value in Resource-Constrained Project Scheduling Problem. Proc. IEEE Congress on Evolutionary Computation, p.2401-2408.

Davis, E.W., Patterson, J.H., 1975. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Manag. Sci.*, **21**(8):944-955. [doi:10.1287/mnsc.21.8.944]

Dayanand, N., Padman, R., 1997. On modeling payments in projects. *J. Oper. Res. Soc.*, **48**(9):906-918. [doi:10.1057/palgrave.jors.2600440]

de Reyck, B., Leus, R., 2008. R&D project scheduling when activities may fail. *IIE Trans.*, **40**(4):367-384. [doi:10.1080/07408170701413944]

Digalakis, J., Margaritis, K., 2004. Performance comparison of memetic algorithms. *Appl. Math. Comput.*, **158**(1):237-252. [doi:10.1016/j.amc.2003.08.115]

Doersch, R.H., Patterson, J.H., 1977. Scheduling a project to maximize its present value: a zero-one programming approach. *Manag. Sci.*, **23**(8):882-889. [doi:10.1287/mnsc.23.8.882]

Elmaghraby, S.E., Herroelen, W.S., 1990. The scheduling of activities to maximize the net present value of projects. *Eur. J. Oper. Res.*, **49**(1):35-49. [doi:10.1016/0377-2217(90)90118-U]

Etgar, R., Shtub, A., LeBlanc, L.J., 1997. Scheduling projects to maximize net present value: the case of time-dependent, contingent cash flows. *Eur. J. Oper. Res.*, **96**(1):90-96. [doi:10.1016/0377-2217(95)00382-7]

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA, p.412.

Grinold, R.C., 1972. The payment scheduling problem. *Nav. Res. Log. Q.*, **19**(1):123-136. [doi:10.1002/nav.3800190110]

Herroelen, W.S., Gallens, E., 1993. Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value. *Eur. J. Oper. Res.*, **65**(2):274-277. [doi:10.1016/0377-2217(93)90340-S]

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University Michigan Press, Ann Arbor, MI, p.90-109.

Icmeli, O., Erengüç, S.S., 1996. A branch-and-bound procedure for the resource-constrained project scheduling problem with discounted cash flows. *Manag. Sci.*, **42**(10):1395-1408. [doi:10.1287/mnsc.42.10.1395]

Klein, R., 2000. Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects. *Eur. J. Oper. Res.*, **127**(3):619-638. [doi:10.1016/S0377-2217(99)00347-1]

Kolisch, R., 1996a. Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. *Eur. J. Oper. Res.*, **90**(2):320-333. [doi:10.1016/0377-2217(95)00357-6]

Kolisch, R., 1996b. Efficient priority rules for the resource-constrained project scheduling problem. *J. Oper. Manag.*, **14**(3):179-192. [doi:10.1016/0272-6963(95)00032-1]

Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. *Manag. Sci.*, **41**(10):1693-1703. [doi:10.1287/mnsc.41.10.1693]

Krasnogor, N., Smith, J., 2000. A Memetic Algorithm with Self-Adaptive Local Search: TSP as a Case Study. Proc. Genetic and Evolutionary Computation Conf., p.987-994.

Merz, P., Freisleben, B., 2001. Memetic algorithms for the traveling salesman problem. *Compl. Syst.*, **13**(4):297-345.

Michalewicz, E., 1996. Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.). Springer-Verlag, Berlin, p.15-22.

Moscato, P., Norman, M.G., 1992. A 'Memetic' Approach for the Traveling Salesman Problem: Implementation of Computational Ecology on Message Passing Systems. *In*: Valero, M., Onate, E., Jane, M., *et al*. (Eds.), Parallel Computing and Transputer Applications. IOS Press, Amsterdam, p.187-194.

Padman, R., Smith-Daniels, D.E., Smith-Daniels, V.L., 1997. Heuristic scheduling of resource-constrained projects with cash flows. *Nav. Res. Log.*, **44**(4):365-381. [doi:10.1002/(SICI)1520-6750(199706)44:4<365::AID-NAV6>3.3.CO;2-H]

Patterson, J.H., Slowinski, R., Talbot, F.B., Weglarz, J., 1990. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *Eur. J. Oper. Res.*, **49**(1):68-97. [doi:10.1016/0377-2217(90)90121-Q]

Pinder, J.P., Marucheck, A.S., 1996. Using discounted cash flow heuristics to improve project net present value. *J. Oper. Manag.*, **14**(3):229-240. [doi:10.1016/0272-6963(96)00003-4]

Russell, A.H., 1970. Cash flows in networks. *Manag. Sci.*, **16**(5):357-373. [doi:10.1287/mnsc.16.5.357]

Salewski, F., Schirmer, A., Drexl, A., 1997. Project scheduling under resource and mode identity constraints: model, complexity, methods, and application. *Eur. J. Oper. Res.*, **102**(1):88-110. [doi:10.1016/S0377-2217(96)00219-6]

Shtub, A., Etgar, R., 1997. A branch and bound algorithm for scheduling projects to maximize net present value: the case of time dependent, contingent cash flows. *Int. J. Prod. Res.*, **35**(12):3367-3378. [doi:10.1080/002075497194129]

Tormos, P., Lova, A., 2001. A competitive heuristic solution technique for resource-constrained project scheduling. *Ann. Oper. Res.*, **102**(1/4):65-81. [doi:10.1023/A:1010997814183]

Tsai, H.K., Yang, J.M., Kao, C.Y., 2002. Solving Traveling Salesman Problems by Combining Global and Local Search Mechanisms. Proc. Congress of Evolutionary Computation, p.1290-1295.

Valls, V., Ballestin, F., Quintanilla, S., 2004. A population-based approach to the resource constrained project scheduling problem. *Ann. Oper. Res.*, **131**(1-4):305-324. [doi:10.1023/B:ANOR.0000039524.09792.c9]

Valls, V., Ballestin, F., Quintanilla, S., 2005. Justification and RCPSP: a technique that pays. *Eur. J. Oper. Res.*, **165**(2): 375-386. [doi:10.1016/j.ejor.2004.04.008]

Vanhoucke, M., 2006. A scatter Search Procedure for Maximizing the net Present Value of a Project under Renewable Resource Constraints. Vlerick Leuven Gent Working Paper Series 2006/40.

Yin, A., 2004. A New Neighborhood Structure for the Job Shop Scheduling Problem. Proc. 3rd Int. Conf. on Machine Learning and Cybernetics, **4**:2098-2101.

Zhu, D., Padman, R., 1999. A metaheuristic scheduling procedure for resource-constrained projects with cash flows. *Nav. Res. Log.*, **46**(8):912-927. [doi:10.1002/(SICI)1520-6750(199912)46:8<912::AID-NAV3>3.0.CO;2-C]