Available online at www.sciencedirect.com
SCIENCE DIRECT®
Int. J. Production Economics 96 (2005) 175–187
international journal of production economics
www.elsevier.com/locate/dsw

A bi-objective model for robust resource-constrained project scheduling

M.A. Al-Fawzan[a,*], Mohamed Haouari[b]

[a] King Abdulaziz City for Science and Technology, P.O. Box 6086, Riyadh 11442, Saudi Arabia
[b] Ecole Polytechnique de Tunisie, BP 743, 2078 La Marsa, Tunisia

Received 5 March 2003; accepted 1 April 2004

Available online 20 July 2004

Abstract

A common problem which arises in project management is the fact that the planned schedule is often disrupted by several uncontrollable factors like additional time that might be required for rework and correction of detected defects. As a result, project managers are often unable to meet the promised completion dates. It is therefore vital to take into account such possible disruptions and their potential negative consequences at the project schedule design stage. In this paper, we address the issue of designing a project schedule which is not only short in time, but also less vulnerable to disruptions due to reworks and other undesirable conditions. To that aim, we introduce the concept of schedule robustness and we develop a bi-objective resource-constrained project scheduling model. We consider the objectives of robustness maximization along with makespan minimization. We develop a tabu search algorithm in order to generate an approximate set of efficient solutions. Several variants of the algorithm are tested and compared on a large set of benchmark problems.

Keywords: Resource-constrained project scheduling; Robust scheduling; Multi-objective combinatorial optimization; Tabu search

1. Introduction

Project scheduling deals with the allocation of scarce resources to a set of interrelated activities that are usually directed toward some major output and require a significant period of time to perform. Since the early days of operations research, project scheduling has attracted an ever growing attention. This interest is largely motivated by its great practical importance in diverse industries for designing new production processes, developing new products, setting up new facilities, and implementing new information systems, to cite just a few applications. The two well-known network techniques CPM and PERT, were both developed in the late 1950s to determine the minimum completion time of a project in such a way that the precedence relationships among activities are satisfied. However, in most practical situations the activities of a project require several additional resources. These resources may include manpower, machines, equipment, and capital budget (Blazewicz et al., 1986). Ignoring the

*Corresponding author. Fax: +996-1-481-3991.
E-mail address: mfawzan@kacst.edu.sa (M.A. Al-Fawzan).

0925-5273/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2004.04.002

resource constraints may result in an infeasible schedule with several resource conflicts. During the last 20 years, Resource Constrained Project Scheduling Problems (RCPSPs) have generated a great amount of research in the operations research literature. Solving a RCPSP means providing an optimal project schedule that not only satisfies the precedence constraints but also provide a feasible allocation of the required resources (i.e., resolve resource conflicts). Traditionally, two main categories of resource constraints are distinguished:

- *Renewable resources*: a specified amount of the resource is *continuously* available during the planning horizon (examples are machines and manpower).
- *Non-renewable resources*: a specified amount of the resource is available for the *entire* planning horizon (an example is the capital budget).

Moreover, in many practical situations, each activity may be performed according to several alternatives called *modes*. Each mode being characterized by a specific time duration and resource consumption.

In this paper, we address the single-mode RCPSP with renewable resources. Formally, this problem may be defined as follows: a project consists of a set J of n interrelated activities $1, \ldots, n$. There are precedence constraints which specify that activity i cannot be performed until all its predecessors P_i have been finished. There are K renewable resources. A constant amount of R_k units of resource k is continuously available from time zero onwards. The duration of activity i is p_i units of time. During this time period a constant amount of r_{ik} units of resource k is occupied. Preemption is not allowed. The objective is to determine starting (or finishing) times for the activities in such a way that a performance criterion is optimized, the precedence constraints are satisfied, and at each unit of time the total resource demand does not exceed the resource availability for each resource type.

We will not attempt to review the important RCPSP literature, since very recently, Demeulemeester and Herroelen (2002) edited an excellent handbook which is entirely devoted to this challenging topic. In addition, Brucker et al. (1999) provide an extensive review of the RCPSP literature, quoting more than 200 relevant papers. They discuss both exact and heuristic recent approaches, and they present a new notation and classification scheme, similar to the one traditionally used in the machine scheduling literature. Herroelen et al. (1998) present an exhaustive survey of exact algorithms (branch and bound procedures) for several variants of the RCPSP. An exhaustive review of heuristic algorithms is presented in Kolisch and Hartmann (1998). Most of these heuristics are extensively evaluated in Hartmann and Kolisch (2000). Finally, Herroelen et al. (1997) present a survey paper exclusively focusing on project scheduling problems with discounted cash flows. Moreover, the reader can refer to the comprehensive monograph of Kolisch (1995).

However, it is worth noting that the considerable effort devoted hitherto to modeling and solving RCPSP has been almost exclusively focusing on three optimization criteria. The optimization criterion which has been the most extensively considered so far is the makespan ($C_{\max}$) minimization, which is defined as the total time elapsed between the start and the end of the project. A second optimization criterion, which has also been widely used is the net present value (NPV) maximization. This latter criterion is appropriate for capturing the monetary aspects of project management when important levels of cash flows (expenditures and/or payments) are present. Finally, the third optimization criterion which has also attracted attention is the cost minimization (CM). This objective includes the case where activities may be performed in several modes resulting in different costs and/or resource consumption.

In this paper, we introduce the concept of schedule robustness and we investigate a bi-objective resource-constrained project scheduling model. It is worth noting that although project scheduling problems are often inherently multi-criteria decision problems, the literature on multi-objective project scheduling is surprisingly scant. Indeed, as far as we know, Slowinski (1989) is the first author who explicitly placed the RCPSP in a

multi-objective framework. Recently, Hapke et al. (1998) presented a two-stage interactive search procedure aimed at handling a very general class of multiple-criteria project scheduling problem. In addition, Viana and de Sousa (2000) implemented two metaheuristics to solve a multi-mode RCPSP with three objectives that are minimized: the makespan, the weighted lateness of activities, and the violation of resource constraints (thus, they considered "soft" resource constraints).

The rest of the paper is organized as follows. In Section 2, we present a bi-objective model for generating robust schedules for the RCPSP. In Section 3, we present a Tabu Search heuristic for our model. Several variants of the algorithm are compared on a series of benchmark problems. The computational results are reported in Section 4. In the final section, we present our conclusions and discuss future research.

2. A model for robust project scheduling

A common drawback to most deterministic project scheduling models is that they assume that the activities will be carried out in ideal conditions and that the proposed schedule could be implemented as planned. However, in practice, several uncontrollable factors may result in delaying the project completion time and increasing the cost. Among these undesirable factors is the imperfect quality in the achievement of some activities, which may cause additional time for rework and correction of the detected defects. It is worth recalling that rework is considered by several authors as the primary cause of project schedule disturbance (Cooper, 1993; Lewis, 1998, p. 265). Other factors which disrupt likewise a project schedule are the delay caused by unreliable deliveries from suppliers and unreliable subcontractors who are unable to meet the promised schedule. Therefore, a challenging issue in project scheduling is the design of a feasible schedule which is not only short but also which could be ideally achieved as intended (or more reasonably, slightly delayed) even if undesirable conditions occur.

In this paper, we address this issue and we propose the development of a resource-constrained project scheduling model with two objectives: makespan minimization and robustness maximization. We define the robustness of a schedule, as its ability to cope with "small" increases in the time duration of some activities that may result from uncontrollable factors (i.e. with a limited effect on the completion time of the project). More precisely, our proposed model aims at constructing schedules that are not only short but also that can be effectively implemented (as planned) without assuming that the activities will be carried out in ideal conditions. Thus, the problem is modeled as a bi-objective combinatorial optimization problem. It is worth noting that our model is in accordance with the expectations of the project planners as surveyed in Icmeli-Tukel and Rom (1997).

We propose to measure the robustness of a given schedule as follows. Define the free slack s_i as the amount of time that an activity i ($i = 1, \ldots, n$) can slip without delaying the start of the very next activity and while maintaining resource feasibility. Thus, the robustness of a schedule is defined as the total sum of the free slacks. Hence, the robustness of a schedule is understood as its ability to be maintained even in the case of some undesired events influencing the realization of particular activities.

Formally, the objective of our model, which we call the Bi-objective Resource Constrained Project Scheduling Problem (BRCPSP), is to determine starting times t_i for the activities $i = 1, \ldots, n$ in such a way that:

- at each time t, the total resource demand does not exceed the resource availability for each resource type,
- the given precedence constraints are satisfied,
- the makespan $C_{\max} = \mathrm{Max}_{i=1,\ldots,n}(t_i + p_i)$ is minimized, and
- the robustness $\Omega = \sum_{i=1}^{n} s_i$ is maximized.

2.1. A motivating example

Consider the example given by Icmeli-Tukel and Rom (1997). For convenience, we reproduce in

Fig. 1 the network graph. The numbers of the nodes are the activity numbers. The two numbers at the top of each node represent the duration (in days) and the resource requirement, respectively. There is one resource type and the availability of that resource is four units per time period.

Consider the two schedules S^1 and S^2 that are depicted in Figs. 2 and 3, respectively.

Both of S^1 and S^2 have a makespan $C_{\max} = 7$. However, the computed robustness (i.e. the sum of slacks $\sum_{i=2}^{5} s_i$) of S^1 and S^2 are 0 and 4, respectively. Now suppose that under unexpected circumstances some of the activities of the project did not meet the customer's expectations (i.e. specifications). Hence, they need additional rework in order to comply with those specifications. The additional work for each activity are displayed in Table 1.

According to the last changes, both of S^1 and S^2 have to be rescheduled as shown in Figs. 4 and 5, respectively.

We observe that although S^1 and S^2 were expected to have the same makespan, after rework S^1 was delayed by 3 days whereas S^2 (which has a largest robustness) was delayed by one day only. Hence, one may reasonably expect that by maximizing the robustness of the schedule it became less sensitive to unexpected delays or any additional work which might be required.

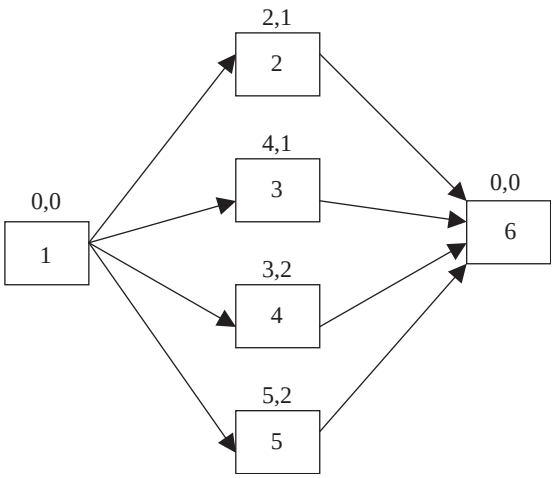

Fig. 1. Example network.

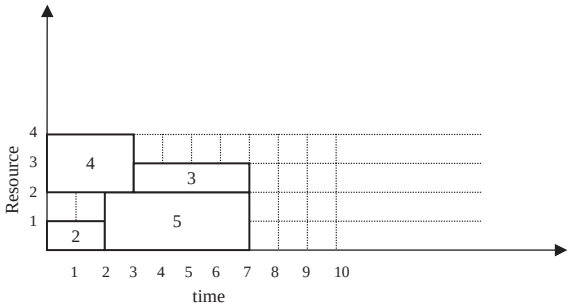

Fig. 2. Graphical representation of S^1.

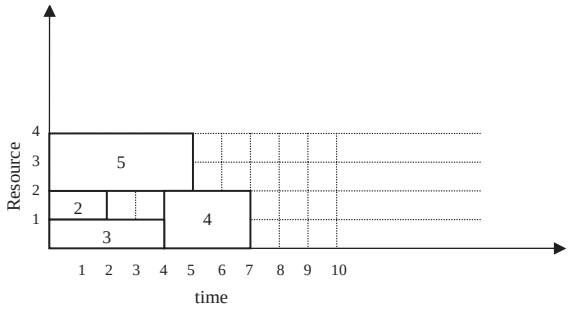

Fig. 3. graphical representation of S^2.

Table 1
Additional work (days)

Activity	Additional work
2	1
3	1
4	0
5	2

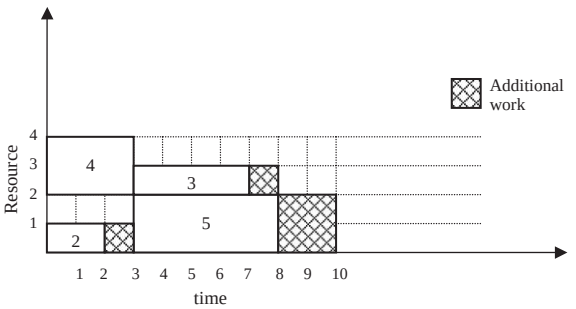

Fig. 4. Graphical representation of S^1 with additional work.

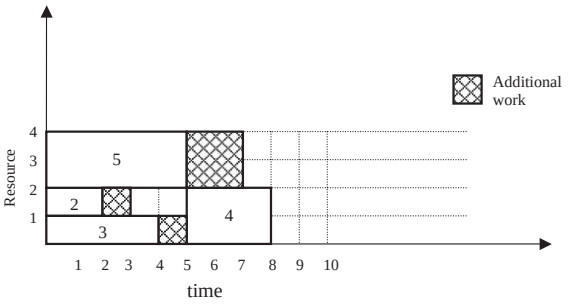

Fig. 5. Graphical representation of S^2 with additional work.

It is worth noting that alternative approaches, dealing with uncertainty in the realization of activities, are the traditional probabilistic approach, the use of fuzzy numbers (Slowinski and Hapke, 1999), and the Critical Chain Scheduling and Buffer Management (CC/BM) of Goldratt (1997). In particular, this latter has recently emerged as one of the most popular approaches to project management. Basically, CC/BM accounts explicitly for duration uncertainty and focuses on the key tasks and resources (through the reduction or elimination of multitasking). For an in-depth analysis of the strengths and weaknesses of this approach, the reader is referred to Herroelen and Leus (2001).

In the following, we present a solution strategy for the BRCPSP.

3. A multi-objective tabu search algorithm for the BRCPSP

The last decade witnessed an increasing interest in research dealing with multi-objective combinatorial optimization (MOCO) problems. For a survey, the reader is referred to Ehrgott and Gandibleux (2000). So far, three types of strategies have been proposed to solve MOCO problems:

(i) Enumerate the entire set of efficient solutions through exact methods like Branch-and-Bound (B&B) or Dynamic Programming (DP). It is worth recalling that a solution is said to be efficient (or non-dominated) if it is impossible to improve it with respect to all of the objectives without violating any constraint. These exact methods are very scarce in the literature and most of them are designed for specific problems. For instance, the B&B algorithm of Visèe et al. (1998) provides the whole set of efficient solutions of the bi-objective knapsack problem, and the DP algorithm of Klamroth and Wiecek (2000) generates all the non-dominated solutions of different complex knapsack models. As it is widely known for single objective (NP-hard) combinatorial optimization problems, exact methods can only solve very small sized MOCO problems. In addition, the cardinality of the efficient set of solutions usually increases not only with the problem size but also with the number of criteria.

(ii) It is also possible to approximate the set of efficient (or non-dominated) solutions by means of constructive heuristics (see for instance Chattopadhyay and Momoh, 1999), or metaheuristics such as Genetic Algorithms (Murat and Ishibuchi, 1997), Tabu Search (Ben Abdelaziz et al., 1999), and Simulated Annealing (Ulungu et al., 1999).

(iii) The third strategy is to cooperate with the Decision Maker (DM) in order to obtain his *preferred solutions*. This process is called an *interactive method*. Interactive methods include either exact methods or heuristic methods since at each step, the procedure has to search for some efficient (or non-dominated) solutions in order to present them to the DM. This approach is particularly designed for the case where the set of efficient solutions is large. Teghem et al. (2000), for example, proposed a simulated annealing based interactive method in order to help a DM to make a choice between the approximate set of efficient solutions. This method was experimented on the knapsack and the assignment multi-objective problems.

In the sequel, we present an algorithm for the BRCPSP based on the second strategy, and aimed at providing an approximate set of non-dominated schedules. More precisely, we propose the adaptation of the principles of TS to the BRCPSP. Since

we are concerned with a multi-objective framework, the problem may have more than one solution forming the efficient set. The adaptation of TS to this context generates an approximation of this set. Moreover, because of the combinatorial aspect of the BRCPSP, the search must generate both supported and unsupported efficient solutions (i.e. those efficient solutions that belong to a facet or the relative interior of the polyhedron of feasible solutions, respectively). It is worth noting that TS has been successfully applied by many authors for the solution of the classical single objective RCPSP by Pinson et al. (1994), Lee and Kim (1996), Baar et al. (1999), and Nonobe and Ibaraki (2002).

We organize the presentation of our MOTS algorithm by describing successively the schedule representation, the neighborhood structure, the selection scheme, the tabu list attributes and the aspiration criterion.

3.1. Schedule representation

A schedule π is represented as an ordered list $L = (j_{(1)}, \ldots, j_{(n)})$ of the n activities. For the sake of simplicity, only precedence feasible lists are considered (a list is said to be precedence feasible if for each activity a we have $\arg(a) > \max_{b \in P_a}\{\arg(b)\}$) Given a precedence feasible list L, a corresponding precedence and resource feasible schedule can be built by using a Schedule Generation Scheme (SGS). As pointed out by Kolisch and Hartmann (1998), SGS is the core of most heuristics for the RCPSP. This algorithm starts from scratch and constructs in n steps a *feasible* schedule, so that at step i ($i = 1, \ldots, n$) activity $j_{(i)}$ is selected and scheduled at the earliest precedence and resource feasible start time. For that purpose, we use the following Forward Recursion Procedure.

Notation

$\tilde{R}_k(t)$	amount of resource k available at time t
Pred_j	set of immediate predecessors of activity j
S_j^e	earliest start time of activity j
C_j^e	earliest completion time of activity j
U	set of unscheduled activities
$\{0\}$	dummy activity with zero processing time, no resource consumption, and which precedes all the activities
EC	$\{C_j^e : j \notin U\}$
$[C_{(q)}]_{q=0,n}$	list of the elements of EC ranked in a non-decreasing order

SGS—forward recursion procedure

Initialization: $U = J$, EC $= \{0\}, q = 0$, $C_{(0)} = 0$
For $i = 1$ to n do
  Select the activity $j = j_{(i)}$
  Set $U = U \backslash \{j\}$
  For all $t \in$ EC do
    For $k = 1, \ldots, K$ do
      Compute $\tilde{R}_k(t)$
  $S_j^e \coloneqq \mathrm{Max}_{i \in \mathrm{Pred}_j}\{C_i^e\}$
  Let q be the index with $S_j^e = C_{(q)}$
  While ($\forall\ t \in [S_j^e, S_j^e + p_j]$,
    $\forall k = 1, \ldots, K$, $\tilde{R}_k(t) \geqslant r_{jk}$) is False do
    $q \coloneqq q + 1$
    $S_j^e \coloneqq C_{(q)}$
  End (while)
  $C_j^e \coloneqq S_j^e + p_j$
  EC $=$ EC $\cup \{C_j^e\}$, Update C
End (for i)

This algorithm provides the earliest feasible start and completion times for the given list. The makespan of the project is $C_{\max} = \mathrm{Max}_{j=1,\ldots,n}\{C_j^e\}$. However, in order to compute the free slacks of the activities, we need to compute the latest feasible completion times C_j^l ($j = 1, \ldots, n$). Hence, the free slack of an activity j is $s_j = C_j^l - C_j^e$ ($j = 1, \ldots, n$).

At this point, it should be made clear that the latest completion times are computed in such a way that for each activity j ($j = 1, \ldots, n$) the following properties hold:

(P1) During the time interval $[S_j^e, C_j^l]$ the K resources *are continuously available with an amount* $\geqslant r_{jk}$.

(P2) It is possible to complete activity j at time C_j^l (but not at time $C_j^l + 1$) without delaying the start of the very next activity (i.e. each successor could start at its earliest start

time that was calculated by the forward recursion).

Hence, if activity j starts at S_j^e but, for some uncontrollable factor, actually requires $p_j + \varepsilon_j$ units of time (with $0 \leqslant \varepsilon_j \leqslant s_j$), instead of the planned p_j units of time, then the schedule remains feasible and no successor would be delayed.

In order to compute the latest finish times, we consider the earliest completion times that were calculated with the forward procedure and we construct an ordered list of the n activities $L' = (u_{(1)}, u_{(2)}, \ldots, u_{(n)})$, where activities are ranked in a non-decreasing order of their earliest completion time (note that this list is different from L). We use a backward procedure so that at step i ($i = n, \ldots, 1$) activity $u_{(i)}$ is included in the partial schedule and scheduled at the latest precedence and resource feasible completion time while satisfying properties (P1) and (P2). Before providing a formal description of the procedure, we extend the previous notation:

S_j^l	latest start time of activity j
Succ_j	set of immediate successors of activity j
$\{n+1\}$	dummy activity with zero processing time, no resource consumption, and which is preceded by all the activities
ES	$\{S_j^e : j \notin U\}$
$[S_{(p)}]_{p=1,n+1}$	list of the elements of ES ranked in a non-increasing order

SGS—backward recursion procedure

Initialization: $U = J$, $\text{ES} = \{C_{\max}\}$, $q = 1, S_{(1)} = C_{\max}$.
For $i \coloneqq n$ down to 1 do
  $j = u_{(i)}$
  Set $U = U \backslash \{j\}$
  For all $t \in \text{ES}$ do
    For $k = 1, \ldots, K$ do
      Compute $\tilde{R}_k(t)$
  $C_j^l \coloneqq \text{Min}_{h \in \text{Succ}_j}\{S_h^e\}$
  Let q be the index with $C_j^l = S_{(q)}$
  While ($\forall t \in [S_j^e, C_j^l]$, $\forall k = 1, \ldots, K$, $\tilde{R}_k(t) \geqslant r_{jk}$) is False do
    $q \coloneqq q + 1$
    $C_j^l \coloneqq S_{(q)}$
  End (while)
  $S_j^l \coloneqq C_j^l - p_j$
  $\text{ES} = \text{ES} \cup \{S_j^e\}$, Update S
End (for i)

3.2. Neighborhood structure

At a given iteration, let π^0 be the current feasible schedule with a corresponding precedence feasible priority list $L(\pi^0) = (j_{(1)}, \ldots, j_{(n)})$. We generate N neighbors by randomly selecting N feasible moves (N is a problem parameter which denotes the neighborhood size). A move requires two adjacent activities $(j_{(k)}, j_{(k+1)})$ to be swapped in the list $L(\pi^0)$. Only precedence feasible moves are considered. More precisely, activities $j_{(k)}$ and $j_{(k+1)}$ are eligible for swapping if the following conditions hold:

(C1) activity $j_{(k)}$ is not a predecessor of $j_{(k+1)}$ ($j_{(k)} \notin P_{j_{(k+1)}}$) and
(C2) in the schedule π^0, activity $j_{(k)}$ starts strictly before activity $j_{(k+1)}$ ($S_{j_{(k)}}^e < S_{j_{(k+1)}}^e$).

Condition (C1) ensures that the new list is precedence feasible. In addition, if condition (C2) is satisfied then swapping the positions of $j_{(k+1)}$ and $j_{(k)}$ might affect the makespan.

Obviously, considering only lists which are compatible with precedence constraints reduce the computational burden, since in the Forward Recursion Procedure the activities are simply scheduled according to the ordering that is specified by the corresponding list.

3.3. Selection strategy

A precedence-feasible list of the n activities L_0 is built and used to generate an initial schedule. Assume that π^0 is the current feasible schedule. For each neighbor schedule π^u ($u = 1, \ldots, N$), compute the makespan ($C_{\max}$) and the robustness ($\sum_{i=1}^{n} s_i$). Let $z_M(u)$ and $z_R(u)$ denote the corresponding values, respectively. Consider the aggregation function defined as

$$z_\lambda(u) = \lambda z_M(u) - (1 - \lambda) z_R(u)$$

where $0 \leqslant \lambda \leqslant 1$. At a given iteration, we select as a new current solution the neighbor schedule π^{u^*} such that $z_\lambda(u^*) = \min_{u=1,\ldots,N} z_\lambda(u)$. The weight λ plays an important role since it aims to reach a particular supported efficient solution and then generate its efficient neighborhood, so that the interior of the feasible set is explored in order to get unsupported efficient solutions. Since non-supported, non-dominated points are typically not far from the non-dominated frontier, the generated set is expected to be a good sample of the whole set of efficient solutions. In our implementation, we used the set of weights $W = \{\lambda_v = v/s : v = 0, 1, \ldots, s\}$, where s is a non-negative integer parameter denoted hereafter by the *number of steps*. Therefore, our MOTS algorithm consists in running $s + 1$ times a single objective TS, each with a particular value of the weight λ. Note that, the particular values of λ_s and λ_0 tend to provide the approximate solutions of the minimum makespan and the maximum robustness, respectively.

3.4. Tabu list attributes and aspiration criterion

Since a move consists of selecting a feasible pair $(j_{(k)}, j_{(k+1)})$, then we store in the tabu list the last performed moves. In order to override the tabu list when there is a good move, we implement the following aspiration criterion: a tabu move is accepted if it produces a solution which outperforms the best solution obtained so far. This comparison is performed with respect to the aggregation function.

3.5. Approximation of the efficient set

An approximate set AE_v of potentially efficient solutions of the single objective problem defined by the weight λ_v ($v = 0, 1, \ldots, s$), is obtained in the following way: the set AE_v is initialized with the initial schedule. At each step of the TS, the generated neighborhood is scanned and the set of potentially efficient solutions AE_v is updated. This operation consists in removing dominated schedules (a schedule π^x dominates a schedule π^y if $z_\mathrm{M}(x) \leqslant z_\mathrm{M}(y)$ and $z_\mathrm{R}(x) > z_\mathrm{R}(y)$ or $z_\mathrm{M}(x) < z_\mathrm{M}(y)$ and $z_\mathrm{R}(x) \geqslant z_\mathrm{R}(y)$). If after a fixed number of iterations ($\mathrm{Iter}_{\max}$) there is no improvement of AE_v then the algorithm stops. An approximation AE of the efficient set E is obtained after dropping the dominated schedules from the union set $\bigcup_{v=0,\ldots,s} \mathrm{AE}_v$.

3.6. Synthesis of the MOTS algorithm

Initialization
  Construct a precedence-feasible list L_0 and generate an initial schedule π^0
  $\mathrm{AE} = \emptyset$, $\mathrm{TL} = \emptyset$ (TL denotes the tabu list)
Iterative step
  For $v = 0$ to s do
    $\mathrm{AE}_v = \{\pi^0\}, \lambda_v = \frac{v}{s}$
  Repeat
  Select randomly N moves (non-tabu moves and/or satisfying the aspiration criterion)
    Construct the set of neighbors $N(\pi^0) = \{\pi^u : u = 1, \ldots, N\}$
    Compute for each π^u the makespan $z_\mathrm{M}(u)$ and the robustness $z_\mathrm{R}(u)$
    $\mathrm{AE}_v = \mathrm{AE}_v \cup N(\pi^0)$
    Remove from AE_v all dominated schedules
    Select π^{u^*} such that $z_{\lambda_v}(u^*) = \min_{u=1,\ldots,N}(\lambda_v z_\mathrm{M}(u) - (1 - \lambda_v) z_\mathrm{R}(u))$
    $\pi^0 = \pi^{u^*}$
  Update the tabu list TL
  Until ($\mathrm{Iter}_{\max}$ iterations are performed with no improvement of AE_v)
  $\mathrm{AE} = \mathrm{AE} \cup \mathrm{AE}_v$
  End (for)
Remove from AE all dominated schedules

4. Computational experiments

4.1. Test problems

We tested our approach on the benchmark problem set *j*30 generated with PROGEN, which is a problem generator developed by Kolisch et al. (1995). This problem set is specifically designed for the RCPSP, and it consists of 480 instances with $n = 30$ activities and $K = 4$ resources.

4.2. Parameters selection

Our MOTS algorithm allows for different choices of five parameters: the tabu list size TL, the number of iterations before stopping $\text{Iter}_{\max}$, the number of steps s, the neighborhood size N, and the aggregation function. We performed some preliminary experiments and we found no evidence that the variation of the two first factors (namely, TL and $\text{Iter}_{\max}$) plays any significant role. Hence, based on these findings, we fixed the tabu list size to $\text{TL} = 7$, and the number of iterations before stopping to $\text{Iter}_{\max} = 0.5n$ (where n represents the number of activities). Moreover, we assigned two levels for each of the three remaining factors in the following way:

- Number of steps: $s = 10$ and 20.
- Neighborhood size: $N = \lceil \theta n \rceil$, with $\theta = 0.3$ and 0.6 (n is the number of activities).
- Aggregation function AF : $z_\lambda(u)$ and $w_\lambda(u) = \lambda w_{\text{M}}(u) + (1-\lambda) w_{\text{R}}(u)$ where

  $w_{\text{M}}(u)$: relative improvement of the makespan $= (z_{\text{M}}(0) - z_{\text{M}}(u))/z_{\text{M}}(0)$ ($z_{\text{M}}(0)$ is the makespan of the current solution).

  $w_{\text{R}}(u)$: relative improvement of the robustness $= (z_{\text{R}}(u) - z_{\text{R}}(0))/z_{\text{R}}(0)$ ($z_{\text{R}}(0)$ is the robustness of the current solution).

Note that when $w_\lambda(.)$ is used as the aggregation function, then at each step, the best neighbor is the schedule π^{u^*} such that $w^\lambda(u^*) = \max_{u=1,N} w^\lambda(u)$. The use of $w_\lambda(.)$ is motivated by the observation that very often the robustness is much larger than the makespan, and, as a consequence, it strongly dominates it in the aggregation function $z_\lambda(.)$. This drawback is avoided by considering only relative improvements.

After combining the three factors, eight different variants V_h ($h = 1, \ldots, 8$) are obtained. The characteristics of these variants are displayed in Table 2.

Table 2
Characteristics of the eight variants

Variant	Number of steps	Neighborhood size	Aggregation function
V_1	10	0.3	$z_\lambda(.)$
V_2	20	0.3	$z_\lambda(.)$
V_3	10	0.6	$z_\lambda(.)$
V_4	20	0.6	$z_\lambda(.)$
V_5	10	0.3	$w_\lambda(.)$
V_6	20	0.3	$w_\lambda(.)$
V_7	10	0.6	$w_\lambda(.)$
V_8	20	0.6	$w_\lambda(.)$

4.3. Performance measure

The eight variants were coded in Basic and compiled with the Quick Basic 4.5 compiler. All the computational experiments were carried out on a Pentium II 350 MHz Personal Computer. In order to analyze the relative performance of the variants, all of the 480 instances of the set $j30$ were successively solved by the eight variants. For each instance i, we computed the approximate non-dominated set AE_{ih} and the CPU time t_{ih}, obtained after running the variant V_h ($h = 1, \ldots, 8$).

It is worth noting, that the problem of selecting the best variant has no simple answer. Indeed, in MOCO, and contrary to single objective combinatorial optimization, there is no straightforward basis of comparison of heuristics, and addressing the issue of designing a reliable measure of performance is by itself a challenging problem. In this context, an interesting approach developed by Czyzak and Jaszkiewicz (1998) provides an estimate of the distance of the approximate non-dominated set to the exact one (denoted by R). In the case of the BRCPSP, this latter measure is based on the distance measure $d(\sigma, R)$ between a feasible schedule σ and the non-dominated set R and defined as

$$d(\sigma, R) = \min_{\mu \in R} \left\{ \max \left(\frac{|C^\sigma_{\max} - C^\mu_{\max}|}{\Delta(C_{\max})}, \frac{|\Omega^\sigma - \Omega^\mu|}{\Delta(\Omega)} \right) \right\}, \quad (9)$$

where $C^x_{\max}$ and Ω^x represent the makespan and the robustness of a given schedule x, respectively, and $\Delta(C_{\max})$ and $\Delta(\Omega)$ represent the range of the makespan and the robustness in the non-dominated set R, respectively (the reader may have noticed that it is implicitly assumed that the non-dominated set is *not* a singleton). Hence, a measure

of the quality of a given approximate set AE_{ih} may be defined as

$$d(AE_{ih}, R) = \frac{1}{|AE_{ih}|} \sum_{\sigma \in AE_{ih}} d(\sigma, R). \quad (10)$$

This measure offers the advantage of providing a reliable estimate of the proximity of a heuristic set to the non-dominated frontier. However, since in our experiments the exact non-dominated set is unknown, we have to replace in (9) and (10) the exact set R by the approximate set AE_i obtained after removing the dominated schedules from the union set $\bigcup_{h=1,8} AE_{ih}$. Unfortunately, the drawback of this measure is that it is not appropriate for the case where the approximate set AE_i is a singleton which contains just one approximate ideal point (and in our experiments we found relatively many such cases).

An alternative measure of performance, that we used in our experiments, and which was introduced by Ulungu (1993), is based on the following ratio:

$$\varepsilon_{ih} = \frac{|AE_{ih} \cap AE_i|}{|AE_i|}. \quad (11)$$

Although this latter ratio is insensitive to the "closeness" to the approximate non-dominated set, it provides a measure of the "*contribution*" of variant h to the approximate non-dominated set AE_i. Consequently, if variant h dominates (is dominated) by the other variants, then ε_{ih} will be close to 1 (0). Moreover, it can be used in all cases, including the case where AE_i is a singleton.

Table 3 presents the results obtained with each of the eight variants. For each variant h ($h = 1, \ldots, 8$) the sum $\varepsilon_h = \sum_{i=1}^{480} \varepsilon_{ih}$, as well as the mean CPU times t_h in seconds.

Table 3 shows that the best performance is obtained with a large number of steps, a large neighborhood size, and the aggregation function $w_\lambda(.)$.

Not surprisingly, we observe that the CPU time increases (almost linearly) as s and/or θ increase. However, statistical testing (namely, the so-called *paired comparison design*) strongly supports the argument that the aggregation function has no significant impact on the running time.

4.4. Robustness analysis

A very desirable algorithmic characteristic of a heuristic is its *robustness*. This robustness may be defined as the ability of the heuristic to generate an approximate solution which is not sensitive to the variation of the seeds. In order to analyze the robustness of our best variant (namely, V_8), we selected randomly 50 instances from the set *j*30. Each instance i was run 10 times with different seeds, while keeping all the parameters fixed. For each instance i ($i = 1, \ldots, 50$), we denote AE_{ik} the approximate non-dominated set generated after run k ($k = 1, \ldots, 10$), and AE_i the approximate non-dominated set which results from the union of the sets AE_{ik}. Then, we computed the ratio ε_{ik} using (11) and the mean value ε_i (computed over the 10 runs). Clearly, the distribution of the series (ε_i) provides a good idea on the robustness of the approach. Hence, a large value of ε_i (i.e. close to 1) means that, *on average*, a large percentage of the set of non-dominated schedules (AE_i) is generated in each run of instance i, regardless of the used seed. On the contrary, a small value of ε_i (i.e. close to 0) indicates that there is a significant difference between the approximate sets obtained with different seeds.

In Fig. 6, we report the results obtained with the fifty instances (the instances are numbered by increasing ε_i).

Fig. 6 indicates that the approach is fairly robust. Indeed, we found that the mean value of the (ε_i) is 0.54.

Table 3
Performance results

Variant	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
ε_h	253.58	279.93	269.92	300.34	265.61	295.66	289.13	322.78
t_h	12.1	20.7	23.0	40.2	12.3	20.5	23.6	40.2

4.5. Analysis of the cardinality of AE

It is of primary importance in multi-objective modeling to have an estimate of the cardinality of the approximate set of efficient solutions. Indeed, the approach may not be appropriate if the number of suggested efficient solutions is extremely large. In this case, an interactive method must be used in order to remove some efficient solutions according to the decision maker preferences. However, in our computational experiments, we observe that in most cases |AE| is of moderate size. Indeed, we found that (for the set $j30$) 120 instances have only one single approximate non-dominated solution. Table 4 provides the empirical distribution of the cardinality of the approximate efficient set obtained over the set $j30$. The mean (maximum) cardinality is 10.69 (30).

4.6. Performance on makespan minimization

The approximate non-dominated set AE obtained after running the MOTS algorithm can be used to derive a good approximate solution to the (single objective) RCPSP with minimum makespan objective function. For that purpose, it suffices to scan the set AE and to select the schedule π^{u^*} with minimum makespan. Hence, the approach consists in running $s+1$ times a tabu search algorithm, each time considering as an optimization criterion the single objective $w_v(.)$ obtained through the aggregation of the relative improvement of makespan *and* the robustness:

$$w_v(u) = \frac{v}{s} w_M(u) + \left(1 - \frac{v}{s}\right) w_R(u), \quad v = 0, \ldots, s.$$

We used this strategy and compared the minimal makespan schedule provided by the MOTS algorithm with the optimal makespan. For that purpose, we ran the variant V_8 on the entire set $j30$ (480 instances). The optimal values are taken from Demeulemeester and Herroelen (1997). We found that 387 instances out of 480 (80.63%) were solved exactly, and that the average percentage deviation from optimal solution is 0.48%. The maximal deviation being 7.89%. We compared our results with those obtained by three specialized algorithms, namely:

- two tabu search algorithms (TS1 and TS2) both developed by Baar et al. (1999) and
- a genetic algorithm (GA) devised by Hartmann (1998).

The results of the comparison are presented in Table 5.

The results indicate that even though our algorithm is not specifically designed for makespan minimization, it compares rather favorably with special purpose algorithms. However, as expected, the mean computing time is comparatively very long for MOTS. This is due to two main reasons. Firstly, running MOTS is equivalent to solving $(s+1)$ times a single objective RCPSP, and secondly, at each iteration the robustness of each neighbor schedule is evaluated, which is time consuming.

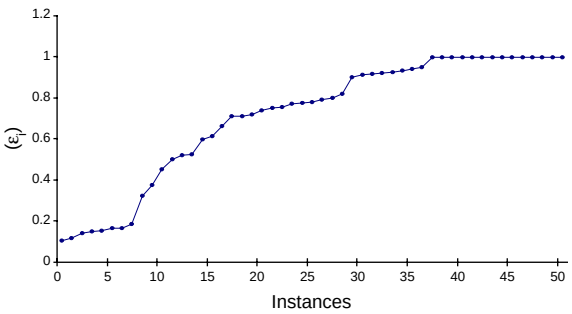

Fig. 6. Analysis of the robustness.

Table 4
Empirical distribution of |AE| for the set $j30$

|AE|	1	2–10	11–17	18–21	22–30
Frequency (%)	25.00	24.80	26.25	17.70	6.25

5. Conclusion

In this paper, we investigated a bi-objective resource-constrained project scheduling problem. Two objectives were considered: makespan minimization and robustness maximization. The importance of the first objective is obvious in the present context of global competition which compels companies to complete the project in a minimum time. The second objective, which is

Table 5
Performance of the MOTS algorithm on the makespan minimization

	TS1	TS2	GA	MOTS
Problems solved to optimality	347	389	Not available	387[a]/329[b]/373[c]
Average deviation (%)	0.7	0.4	0.54[b]/0.25[c]	0.48[a]/1.93[b]/0.62[c]
Maximal deviation (%)	7.5	6.9	Not available	7.89[a]/19.35[b]/8.62[c]
Average CPU time (s)	2[d]	15[d]	Not available	42[a]/23[b]/41[c]

[a] This result was obtained using the stopping criterion described in Section 3.5.
[b] This result was obtained after 1000 evaluations.
[c] This result was obtained after 5000 evaluations.
[d] This time was obtained on a Sun Ultra 2 workstation.

equivalent to the maximization of a sum of free slacks, is a measure of the ability of a schedule to be completed within the promised date even in the case of some undesired events influencing the realization of particular activities.

A multi-objective tabu search algorithm has been devised for solving the bi-objective resource constrained project scheduling problem. Several variants of the algorithm were compared using statistical design of experiments. The best variant was thoroughly tested on a large sample of 480 benchmark problems and it proved to be robust and to generate in most cases a moderately sized set of efficient solutions. Moreover, its performance on the single objective makespan minimization is comparable to special purpose tabu search algorithms.

For future research, at least two issues are worth investigating. Firstly, it would be interesting to generalize the present model to include non-renewable resources as well as multiple execution modes for each activity. Secondly, it appears that the number of efficient solutions increases with the number of activities. Thus, an extension which deserves to be investigated is to embed the present work in an interactive strategy which aims to reduce the number of proposed schedules.

Acknowledgements

The authors would like to thank two anonymous referees for their valuable suggestions that have led to several improvements.

References

Baar, T., Brucker, P., Knust, S., 1999. Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Dordrecht, pp. 1–18.

Ben Abdelaziz, F., Krichen, S., Chaouachi, J., 1999. A hybrid heuristic for multiobjective knapsack problems. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Dordrecht, pp. 205–212.

Blazewicz, J., Cellary, W., Slowinski, R., Weglarz, J., 1986. Scheduling under resource constraints–Deterministic models. Annals of Operations Research 7.

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

Chattopadhyay, D., Momoh, J., 1999. A multiobjective operations planning model with unit commitment and transmission constraints. IEEE Transactions on Power Systems 14, 1078–1084.

Cooper, K.G., 1993. The rework cycle: benchmarks for the project manager. Project Management Journal 24, 17–21.

Czyzak, P., Jaszkiewicz, A., 1998. Pareto simulated annealing? A metaheuristic for multiobjective combinatorial optimization. Journal of Multi-Criteria Decision Analysis 7, 34–47.

Demeulemeester, E., Herroelen, W., 1997. New benchmark results for the resource-constrained project scheduling problem. Management Science 43, 1485–1492.

Demeulemeester, E., Herroelen, W., 2002. Project scheduling: A Research handbook. Kluwer Academic Publishers, Boston.

Ehrgott, M., Gandibleux, X., 2000. A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum 22, 425–460.

Goldratt, E.M., 1997. Critical Chain. The North River Press Publishing Corporation, Great Barrington.

Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45, 733–750.

Hartmann, S., Kolisch, R., 2000. Experimental state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127, 394–407.

Hapke, M., Jaszkiewicz, A., Slowinski, R., 1998. Interactive analysis of multiple-criteria project scheduling problems. European Journal of Operational Research 107, 315–324.

Herroelen, W., Leus, R., 2001. On the merits and pitfalls of critical chain scheduling. Journal of Operations Management 19, 559–577.

Herroelen, W.S., Van Dommelen, P., Demeulemeester, E.L., 1997. Project network models with discounted cash flows a guided tour through recent developments. European Journal of Operational Research 100, 97–121.

Herroelen, W., De Reyck, B., Demeulemeester, E.L., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research 25, 279–302.

Icmeli-Tukel, O., Rom, W.O., 1997. Ensuring quality in resource constrained project scheduling. European Journal of Operational Research 103, 483–496.

Klamroth, K., Wiecek, M.M., 2000. Dynamic programming approaches to the multiple criteria knapsack problem. Naval Research Logistics 47, 57–76.

Kolisch, R., 1995. Project Scheduling Under Resource Constraints, Efficient Heuristics for Several Problem Cases. Physica-Verlag, Wurzburg.

Kolisch, R., Hartmann, S., 1998. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Handbook on Recent Advances in Project Scheduling. Kluwer, Dordrecht, to appear.

Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41 (10), 1693–1703.

Lee, J.K., Kim, Y.D., 1996. Search heuristics for resource-constrained project scheduling. Journal of Operational Research Society 47, 678–689.

Lewis, J.P., 1998. Mastering Project Management. McGraw Hill, New York.

Murat, T., Ishibuchi, H., 1997. Performance of MOGA for flow shop scheduling problems. In: Kuriynis (Ed.), Proceedings of the 14th Conference on Production Research, Osaka Institute of Technology, Japan, pp. 498–501.

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Ribeiro, C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, Dordrecht, pp. 557–588.

Pinson, E., Prins, C., Rullier, F., 1994. Using tabu search for solving the resource-constrained project scheduling problem. Working paper, Universitè Catholique de l'Ouest, Angers.

Slowinski, R., 1989. Multiobjective project scheduling under multiple-category resource constraints. In: Slowinski, R., Weglarz, J. (Eds.), Advances in Project Scheduling. Elsevier, Amsterdam.

Slowinski, R., Hapke, M., 1999. Scheduling Under Fuzziness. Springer, Berlin.

Teghem, J., Tuyttens, D., Ulungu, E.L., 2000. An iteractive method for multi-objective combinatorial optimization. Computers and Operations Research 27, 621–634.

Ulungu, E.L., 1993. Optimisation combinatoire multicritére: Dètermination de l'ensemble des solutions efficaces et mèthodes interactives. Ph.D. dissertation Faculté Polytechnique de Mons, Belgium. Unpublished.

Ulungu, E.L., Teghem, J., Fortemps, P., Tuyttens, D., 1999. MOSA method: A tool for solving MOCO problem. Journal of Multi-Criteria Decision Analysis 8, 221–236.

Viana, A., de Sousa, J.P., 2000. Using metaheuristics in multiobjective resource constraints project scheduling. European Journal of Operational Research 120, 359–374.

Visèe, M., Teghem, J., Pirlot, M., Ulungu, E.L., 1998. An interactive heuristic method for multi-objective knapsack problem. Journal of Global Optimization 12, 139–155.