Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 187 (2008) 1238–1250
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times

Marek Mika [a], Grzegorz Waligóra [a,*], Jan Węglarz [a,b]

[a] *Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland*
[b] *Poznań Supercomputing and Networking Center, Wieniawskiego 17/19, 61-713 Poznań, Poland*

Received 29 December 2005; accepted 8 June 2006
Available online 15 November 2006

Abstract

In this paper, a multi-mode resource-constrained project scheduling problem with schedule-dependent setup times is considered. A schedule-dependent setup time is defined as a setup time dependent on the assignment of resources to activities over time, when resources are, e.g., placed in different locations. In such a case, the time necessary to prepare the required resource for processing an activity depends not only on the sequence of activities but, more generally, on the locations in which successive activities are executed. Activities are non-preemptable, resources are renewable, and the objective is to minimize the project duration. A local search metaheuristic—tabu search is proposed to solve this strongly NP-hard problem, and it is compared with the multi-start iterative improvement method as well as with random sampling. A computational experiment is described, performed on a set of instances based on standard test problems constructed by the ProGen project generator. The algorithms are computationally compared, the results are analyzed and discussed, and some conclusions are given.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Project scheduling; Multi-mode resource-constrained project scheduling problem; Setup time; Tabu search

1. Introduction

In the classical resource-constrained project scheduling problem to minimize the makespan (RCPSP) given is a set of non-preemptable activities which have to be executed using some scarce resources. Furthermore, precedence constraints are defined between activities, i.e. no activity can be started before all its predecessors are finished. In certain situations it is possible to execute each activity in one of several alternative modes which usually represent relations between the used and consumed resources, and the duration of the activity. In such a case, the resulting problem is called the multi-mode resource-constrained project scheduling problem (MRCPSP). The objective is to find an assignment of modes to activities, as well as precedence- and resource-feasible starting times of all activities such that the project duration (the makespan) is minimized. The problem is known to be strongly

* Corresponding author. Tel.: +48 61 8790790; fax: +48 61 8771525.
E-mail addresses: marek.mika@cs.put.poznan.pl (M. Mika), grzegorz.waligora@cs.put.poznan.pl (G. Waligóra), jan.weglarz@cs.put.poznan.pl (J. Węglarz).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.06.069

NP-hard. For a comprehensive survey on project scheduling see Brucker et al. (1999) or Demeulemeester and Herroelen (2002), and a review of recent models, algorithms and applications can be found in Węglarz (1999).

In several papers, extensions of the RCPSP including setup times have been considered where, generally, a setup time is a time necessary to prepare the required resource for processing an activity. There are two types of setup times considered in the literature: *sequence-independent setup times* and *sequence-dependent setup times*. In the first case (Kolisch, 1995), setup times depend only on the activity and the resource on which the activity will be processed, but do not depend on the sequence of activities on a particular resource. This case is typical for situations when, e.g., a machine has to be set up for processing some activity but the time needed for this setup is fixed and independent of which activity has been processed previously on this machine. In the second case (Neumann et al., 2002; Schwindt, 2005) setup times depend not only on the activity and the resource, but also on the sequence of activities processed on this resource. For example, if there are three activities i, j, k such that i, j are direct predecessors of k, which are to be processed on the same resource, the setup time needed for processing activity k may be different, depending on whether activity k is executed directly after activity i (i.e. the sequence is (j, i, k)) or directly after activity j (i.e. the sequence is (i, j, k)). Such situations are typical, e.g., in industry processes, where processing units, like reactors or filters, have to be cleaned after the completion of certain activities. In general, the cleaning time will be larger when switching from a low- to a high-quality product then vice versa. Setup times of this type also occur in the context of research and development projects, where the resources represent staff of the same skill working jointly on specific tasks in different interdisciplinary teams, and each task requires a certain lead time necessary to the training of the staff. Since the training of a person moving from one task to another depends on the difference between the two tasks, the lead times and thus the setup times between tasks are sequence-dependent (Neumann et al., 2002).

However, a more general situation is also possible, where setup times do not depend only on the sequences of activities on particular resources, but—more generally—on the assignment of resources to activities over time. It means that not only activities and their sequences on resources affect setup times, but also the fact on which resources predecessors of particular activities have been scheduled. In such a case, the setup times are not just sequence-dependent, but they are *schedule-dependent*. For example, in the environment of a computational grid, where tasks of a workflow application are assigned resources from different locations, the setup time of a resource (i.e. a time needed for the resource to prepare it for processing a task) depends on which grid nodes (resources) the successive tasks are executed, since the transmission times of required data between these tasks will be different depending on the tasks and the locations. Similar examples from other areas can be also given. In such a case, we will call the resulting problem the multi-mode resource-constrained project scheduling problem with schedule-dependent setup times (MRCPSP-SST) (Mika et al., 2003).

The paper is organized as follows. In Section 2, we present a detailed description of the considered problem and all its parameters. In Section 3, an application of the tabu search metaheuristic is described, and in Section 4, the approaches of multi-start iterative improvement and random sampling are discussed. Section 5 is devoted to the computational experiment and the analysis of the results. Finally, some conclusions and directions for further research are given in Section 6.

2. Problem formulation

In this section, formal definitions of all the parameters of the considered problem are described. The section starts with the presentation of practical examples of the occurrence of our problem. Then the classical resource-constrained project scheduling problem (RCPSP), as well as its extensions to the multi-mode version (MRCPSP) and MRCPSP-SST will be discussed. Finally, the model of the considered problem is presented.

2.1. Problem overview

Assume that we have a classical situation when a set of activities are to be performed on a set of resources. The activities apply for the resources to be executed. Let us call the resources capable of executing different activities *multi-purpose resources*. Apart from multi-purpose resources, there may exist resources dedicated for particular activities, more precisely, resources needed to prepare multi-purpose resources for executing activities. These are resources which the activities do not apply for, they are only necessary for executing particular activities on multi-purpose resources. They can be available from the beginning, or they can be produced by some activities. Let us call such resources *setup-required resources* since they are, in fact, needed for setting up multi-purpose resources for executing activities.

As an example, let us consider a problem of scheduling workflow applications on a computational grid. The problem consists in assigning grid resources to tasks of a workflow job across multiple administrative domains. Workflow applications can be viewed as complex sets of precedence-related various transformations (tasks) performed on some data. The precedence constraints between tasks usually follow from data flow between them, i.e. data files generated by one task are needed to start another task. These data files can be treated as setup-required resources since a node (resource unit) is prepared for processing a task only when all input files required for this task are stored on the local discs of this node. As a result, setup-required resources (data files) are only used for preparing multi-purpose resources (nodes) for executing tasks of a workflow, but tasks do not compete for data files—it is known in advance which data files are dedicated to which tasks, and these files can not be replaced by any other data files. Now, as it is easy to see, the transmission times of these files depend on which nodes the files are transferred between, and the transmission times define the times of multi-purpose resource preparation, i.e. setup times. In other words, the transmission times (setup times) depend not only on the node to which the files are transferred, but also on the nodes where they are located, i.e. on the locations where the preceding tasks have been executed and from which they will be transferred. In consequence, setup times in this problem do not depend only on the sequences of tasks on particular resource units, but—more generally—on the assignment of nodes to tasks over time, so they are schedule-dependent. For more detailed description of the presented example see Mika et al. (2003).

Thus, schedule-dependent setup times occur when setup-required resources have to be transported or transmitted from one location to another in order to prepare a resource unit in the latter location for executing an activity. The setup-required resources may be available in some locations from the beginning, or they can be an outcome of executing some activities, usually preceding activities, mostly direct predecessors. In consequence, the times needed for providing the setup-required resources (i.e. setup-times) depend on where (in which location, i.e. on which resource unit) the activities producing those resources have been executed, and when. Thus, the setup times clearly depend on the schedule, and therefore we call them *schedule-dependent*. Note, that the considered situation cannot be modelled by means of generalized precedence constraints. Time lags are associated with activities, whereas setup times with resources. In our case, setup times depend on locations, and it is unknown in advance which activity will be executed in which location. Thus, the best we could do is to take a precedence constraint and define the minimal time lag as the largest transmission time of a setup-required resource from among all existing locations to the location of the direct successor. In consequence, schedules of poor quality would be generated, since the actual setup time could be much smaller, and it will be known just after the location assignment is made.

Examples from other areas can be given as well. For instance, an enterprise has several geographically distributed plants and makes products according to the assembly-to-order, make-to-order, or engineer-to-order scenarios. Resources of the enterprise are distributed among the plants, so that resources of a particular type may not be available in each plant. In consequence, it is not possible to complete the whole production process within one plant, and specific plants make semi-finished products only which have to be transported between some plants. The semi-finished products are setup-required resources and are dedicated to particular production processes making final products. The transportation times can be then clearly treated as setup times, and depend on the whole production schedule of semi-finished and final products over the distributed plants.

2.2. Problem definition

Let us now define the problem mathematically. The classical resource-constrained project scheduling problem (RCPSP) can be defined as follows. The project consists of a set V of activities, a set K_R of resources, and a set E of precedence constraints between activities. The set K_R consists of R *renewable resources* which are used to execute all activities of the project. Renewable resources are available at any time in limited numbers of units, i.e. an available amount of such a resource is renewed from period to period. The number of units of renewable resource k, $k = 1, \ldots, R$, available at each time period is R_k. The set V consists of n non-preemptable and precedence-related activities that need to be executed using some resources. The precedence constraints between some pairs of activities are defined by relations of the type: $i \rightarrow j$, where $i \rightarrow j$ indicates that activity i must be finished before the start of activity j. The structure of the project is represented by a so-called *activity-on-node* (AoN) graph $G(V, E)$ which is a directed, acyclic, and transitively reduced graph, where the set of nodes V corresponds to the set of project activities, and the set of arcs $E = \{(i, j)\text{: } i, j \in V; i \rightarrow j\}$ represents the finish-to-start precedence constraints with zero minimum time lags. It is assumed that the nodes of graph G are topologically ordered, i.e. a node has always a higher number than all its predecessors. Moreover, it is assumed that there is exactly one starting and exactly one finishing node in the graph. If this condition is not fulfilled, some special additional nodes representing so-called *dummy activities* have to be inserted at the start and/or at the end of the graph. Each activity of the project is characterized by its processing time, resource requests, and precedence relations with other activities. The processing time (duration) of activity j is denoted by p_j. Activity j requires r_{jk} units of resource $k \in K_R$ in each period of its processing. The precedence relations of activity j with other activities may be defined by two sets: a set P_j of direct predecessors of activity j, and a set S_j of direct successors of activity j. Once started, an activity may not be interrupted, i.e. preemption is not allowed. It is assumed that all activities and resources are available from the start of the project. The objective is to find a schedule S that minimizes the project duration (makespan) satisfying all precedence and resource constraints. A schedule assigns a start time s_j to each activity $j \in V$. The RCPSP is strongly NP-hard, as a generalization of the well-known job shop problem (Błażewicz et al., 1983).

The classical RCPSP assumes that each activity can only be executed in a single way which is determined by a fixed duration and fixed resource requests. But there are many practical situations in which the duration of an activity can be decreased at the expense of providing additional resources. In such a situation, an activity may be executed in one of several modes. A mode represents alternative ways of execution of an activity, and is a combination of the activity duration and its resource requests. In such a case, the resulting problem is called the multi-mode resource-constrained project scheduling problem (MRCPSP) (Elmaghraby, 1977). Thus, the main difference between the MRCPSP and the RCPSP is such that for each activity $j \in V$, a set M_j of possible execution modes is defined. Moreover, in the further extensions of the MRCPSP two additional categories of resources: *non-renewable* and *doubly constrained* are introduced. Let us recall that the availability of a renewable resource is once defined and renewed from period to period. For non-renewable resources, total consumption of the resource units is limited for the entire project. In the case of doubly constrained resources, both total and per period availabilities are limited. However, under discrete resources, the doubly constrained resources need not be taken into account explicitly since they can be incorporated by properly enlarging the sets of the first two types of resources. The duration of activity $j \in V$ executed in mode $m_j \in M_j$ is denoted by p_{jm}. Moreover, activity $j \in V$ executed in mode $m_j \in M_j$ requires for its processing r_{jmk} units of renewable resource k, $k = 1,\ldots,R$. Each activity j, $j = 1,\ldots,n$, started in mode m_j, $m_j \in 1,\ldots,|M_j|$, must be completed in mode m_j without preemption. The objective of the MRCPSP is to find an assignment of modes to activities, as well as precedence- and resource-feasible starting times of all activities, such that project duration is minimized. The MRCPSP is also strongly NP-hard, as a generalization of the RCPSP. Moreover, for more than one non-renewable resource, the problem of finding a feasible solution of the MRCPSP is already NP-complete (Kolisch, 1995).

As it has been mentioned in Section 1, in certain situations the execution of an activity has to be preceded by some preliminary operations, performed in order to prepare all necessary resources for the processing of the activity. The time consumed by those additional operations is usually called a *setup time*. Apart from already mentioned sequence-independent and sequence-dependent setup times, another situation is possible in practice, where setup times depend generally on the schedule. In this so-called MRCPSP-SST, activities of a project have to be executed using several resources. At this point we consider renewable resources only, but non-renewable ones can be easily incorporated as well, exactly in the same way as they are involved in the classical MRCPSP. All units of each multi-purpose renewable resource have the same functionality with respect to the possibility of processing the activities. The sets of units of each multi-purpose resource are divided into several subsets that are scattered among different locations. The set of all locations is denoted by K_L. Each activity can be only processed in exactly one location. Assume that there are two precedence-related activities i and j (i.e. $i \rightarrow j$). Activity i is processed using some resources placed in location l_i, whereas activity j is executed using resources placed in another location l_j. If, for example, a semi-finished product being an output of activity i is simultaneously treated as material (input) for activity j, which means that it is required to execute activity j, then in order to prepare all required resources to execute activity j in location l_j, this semi-finished product has to be first transported from l_i to l_j. Thus, a setup-required resource (the semi-finished product) is transported between two locations in order to set up a multi-purpose resource. Of course, the transportation time (setup time) depends on the weighted distance between locations l_i and l_j. In such cases, the setup times (usually associated with precedence constraints) are schedule-dependent.

One can notice that the transportation phase could also be treated as an additional activity which might be executed in one of several modes. However, the number of such additional activities grows with the number of precedence constraints (the cardinality of set E), and for each such activity the number of possible execution modes grows exponentially with the number of locations L. Thus, it is impractical for large size problems because the number of decision variables of the mathematical model of this problem also grows exponentially, in contrast to the approach with schedule-dependent setup times where the number of decision variables does not grow with some additional transportation operations.

We assume that, in general, a setup time ST, i.e. a time needed to prepare a multi-purpose resource, can be a sum of three components: sequence-independent (SIT), sequence-dependent (SDT), and schedule-dependent (SST) setup times. As it has been mentioned, SIT depends only on the activity and the resources on which this activity will be executed, i.e. is independent of the sequence of activities and of the locations. SDT depends on the activity, resources used for the processing of this activity, and the sequence of activities executed using these resources, but does not depend on the locations. Finally, the setup time SST, depends not only on the activity, resources and the sequence of activities on these resources, but also on the locations where direct predecessors of the considered activity are executed. In general, each of all the three components could be a function.

Although our MRCPSP-SST model is a general one, in this paper we consider a situation where the SIT and SDT components do not affect setup time, i.e. there is no other setup needed apart from the one following from the transmission of setup-required resources. In other words, only schedule-dependent setup times occur, and they depend only on two precedence-related activities and two locations in which these activities are executed. For instance, in the grid example, the only setup needed for a node to execute a task is the transmission of the required files from other nodes. The transmission time of a file is directly proportional to the size of this file multiplied by the bandwidth between the two locations. Although the bandwidth can be considered as constant between a particular pair of locations, the size of the transmitted file is associated with the two activities between which it is transferred and, as a result, the whole setup time also depends on this pair of activities. Of course, in general, other setups could be necessary as well, and the components SIT and/or SDT could appear. However, since they have been already considered in the literature, we restrict this research to schedule-dependent setup-times only.

The MRCPSP-SST parameters are summarized in Table 1. All information on the project is assumed to be deterministic and known in advance. All numerical parameters of the project are assumed to be non-negative and integer valued.

Now, it is important to stress that if $i \rightarrow j$ (i.e. i is a direct predecessor of j), the following inequality must be satisfied:

$$s_j \geqslant C_i + \mathrm{ST}_{pr},$$

where $p = l_i$ and $r = l_j$ (symbols p and r are introduced only for avoiding sub-subscripts), and ST_{pr} is the setup times related to the pair of activities i, j executed in locations l_i, l_j. This means that no activity may be started before its predecessor is finished and the required setup time for preparing the corresponding resource has elapsed. If an activity has more then one direct predecessor, the above condition must be fulfilled for each of them. Of

Table 1
Notation for the MRCPSP-SST

Symbol	Definition
$G(V, E)$	Directed AoN graph representing the structure of the project
V	Set of activities
$n = \|V\|$	Number of activities
$i \rightarrow j$	Precedence constraint between activities i and j
E	Set of precedence constraints between activities
P_j	Set of direct predecessors of activity j
S_j	Set of direct successors of activity j
K_R	Set of renewable resources
$R = \|K_R\|$	Number of renewable resources
R_k	Number of units of renewable resource k available in each time period
M_j	Set of execution modes of activity j
m_j	Execution mode of activity j
p_{jm}	Processing time of activity j executed in mode m_j
r_{jkm}	Request for resource k by activity j executed in mode m_j
K_L	Set of resource locations
$L = \|K_L\|$	Number of resource locations
l_j	Location where activity j is executed
R_{kl}	Number of units of renewable resource k available in location l, $\sum_{l \in K_L} R_{kl} = R_k,\ k = 1, \ldots, R$
S	Schedule
s_j	Starting time of activity j
C_j	Completion time of activity j
ST	Setup time
SIT	Sequence-independent setup time
SDT	Sequence-dependent setup time
SST	Schedule-dependent setup time

course, the schedule-dependent setup time can also be equal to 0 in some situations, e.g. where successive activities i and j (such that $i \rightarrow j$) are executed in the same location ($l_i = l_j$), and no transportation or transmission time is needed for preparing resources for processing activity j.

The objective of the MRCPSP-SST is to find an assignment of modes and locations to activities, as well as precedence-, setup-, and resource-feasible starting times for all activities such that the project duration is minimized. The problem is also strongly NP-hard, as a generalization of the MRCPSP. Therefore, it is justified to apply local search metaheuristics, like tabu search, to attack the problem. In the next section, our application of tabu search for the considered MRCPSP-SST is presented.

3. Tabu search

Tabu search (TS) is a metaheuristic based on neighbourhood search with overcoming local optimality. It was originally developed by Glover (1986, 1989, 1990), and a comprehensive report of the basic concepts and recent developments was given by Glover and Laguna (1997).

3.1. Solution representation

A feasible solution is represented by three n-element lists. The first one is a precedence-feasible permutation of activities, in which each activity j, $j = 1, \ldots, n$, has to occur after all its predecessors and before all its successors. This structure is called the *activity list* (AL). The second one is a list of execution modes for all activities and is called the *mode assignment* (MA). The jth element of this list defines the execution mode of activity j. This representation has been used in many previous algorithms for the MRCPSP. However, in our case another list is also needed which defines the location where an activity is executed. Thus, the third list is a list of locations for all activities, where jth element of this list defines the location of activity j. This structure is called *location assignment* (LA).

Let us stress the relation between the MA and LA lists of a solution. MA defines the execution modes of all activities and, as a result, their resource requests. If the location defined by LA for any activity is not capable to fulfil the resource requests of this activity processed in the mode defined by MA, i.e. the number of available units of at least one resource is not sufficient to execute the considered activity in this location, then such a solution is infeasible. A solution represented by the lists AL, MA, and LA is feasible only if for all activities the numbers of available units of all resources in the locations defined by LA are greater or equal to the corresponding resource requests for the execution modes defined by MA. This feasibility condition will be used in the pre-processing procedure described in point 3.3.

Given a feasible solution represented by the three lists described above, the starting times of all activities are then defined by using a modified version of the serial SGS (schedule generation scheme) decoding rule (Kelley, 1963). Since the project duration criterion is a regular performance measure (Kolisch, 1995), i.e. a measure which is non-decreasing in activity completion times (Conway et al., 1967), we may use the serial SGS rule to construct the schedule. As a result of this procedure, only semi-active schedules are generated and there is no danger of omitting an optimal schedule.

The modification of the classical serial SGS consists in taking into account schedule-dependent setup times. As we have mentioned, a setup time depends on the pair of precedence-related activities, and on the locations in which they are executed. In consequence, whenever two successive precedence-related activities are performed in different locations, the setup time associated with this pair of activities executed in these locations must be added to the schedule between the completion of the first activity and the start of the second. In other words, the earliest possible moment of starting the succeeding activity is the finishing time of the preceding activity plus the properly calculated setup time. It is easy to see that the SGS with the described modification also generates semi-active schedules.

3.2. Example of a schedule

Let us consider an example of a feasible solution, and the resulting schedule obtained by using the serial SGS described above. For the example, we use a workflow job which is to be executed on a computational grid. The structure of the workflow is presented in Fig. 1, where tasks 1 and 10 are dummy.

There are two resources ($R = 2$): **R1** and **R2**, and two locations ($L = 2$): **L1** and **L2**. We assume that resource **R1** is available in location **L1**, and resource **R2** is available in location **L2**. Resource **R1** is available in 3 units ($R_1 = 3$), and resource **R2** in 2 units ($R_2 = 2$). We also assume that because of the sizes of the files, the transmission times within a given location cannot be neglected. The bandwidth within location **L1** is equal to 10 M/t, within location **L2** is equal to 5 M/t, and between these two locations is equal to 2 M/t, where M is a data unit and t is a time unit.

All data concerning tasks are presented in Tables 2 and 3. In Table 2 resource requests and durations of tasks are given. In Table 3 we present the sizes of files transferred between some pairs of tasks, where

Table 2
Example—task data

j	m_1			m_2		
	p_{j1}	r_{j11}	r_{j21}	p_{j2}	r_{j12}	r_{j22}
1	0	0	0	–	–	–
2	8	0	1	10	1	0
3	10	1	0	15	0	1
4	12	0	1	14	1	0
5	5	1	0	10	0	1
6	4	1	0	9	0	1
7	3	1	0	4	0	1
8	9	0	1	12	1	0
9	6	1	0	8	0	1
10	0	0	0	–	–	–

Table 3
Example—transfer files data

i	j	$\|F_{ij}\|$ (M)
2	5	40
3	5	16
3	6	10
4	6	30
4	7	25
5	8	25
6	8	30
7	8	60
7	9	20

$|F_{ij}|$ is the size of the file(s) that needs to be transmitted from task i to task j. In this example, we assume for simplicity that the tasks are not parallel computations, i.e. each task j requires in each mode exactly one resource unit (processor): $r_{jkm} = 1$.

Table 4 is the bandwidth matrix between locations **L1** and **L2**.

Let us finally assume that if precedence-related tasks i and j are executed on the same processor, then no file transmission is needed and the setup time is equal to 0. This is the case for the pairs of tasks: (2, 5), (4, 6), (4, 7) and (5, 8). The remaining five precedence relations result in some file transmis-

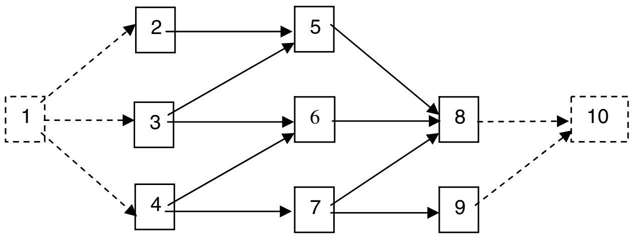

Fig. 1. The structure of a workflow.

Table 4
Example—bandwidth data

	L1	L2
L1	10 M/t	2 M/t
L2	2 M/t	5 M/t

sions, either within a location, e.g. tasks (6, 8), or between locations, e.g. tasks (3, 5).

Under all the above assumptions for the considered example, the schedule for a given feasible solution is presented in Fig. 2. Arrows represent schedule-dependent data transfers (or setups). Above the schedule, we show the feasible solution considered, i.e. lists AL, MA, and LA.

3.3. Pre-processing

In order to reduce the search space and adapt the project data to the implementation of tabu search, a slightly modified pre-processing procedure introduced by Sprecher et al. (1997) is used. After the execution of this procedure all inefficient modes, non-executable modes, and "incapable" locations are deleted.

An inefficient mode is a mode with processing time not shorter and, simultaneously, resource requests not greater than any other mode of the considered activity within a given location.

A non-executable mode is a mode in which a considered activity cannot be executed since there is no location capable to execute this activity in this mode (because of not enough resource units available). In other words, a non-executable mode is a mode for which all locations are incapable.

Additionally, for each activity we exclude all the locations that are not capable to execute the activity in any mode because of not enough resource units available, as it has been described in point 3.1. As a result, all incapable locations are deleted. Let us stress that the incapable locations are removed separately for each activity. Consequently, each activity has its own list of capable locations.

3.4. Objective function

The objective function is defined as the project duration for a particular feasible solution. This value is calculated during the execution of the modified serial SGS decoding rule, as the completion time C_n of the final activity.

3.5. Starting solution

A starting solution is generated in the following way:

- all activities on AL are set in an ascending order that follows from the ordering of nodes in the precedence relation graph,
- all modes on MA are set at 1,
- successive locations on LA are set at the first capable location for successive activities.

The procedure consisting of the first two points has been commonly used in local search algorithms for the MRCPSP before. The third point is needed

1	2	3	4	5	7	6	8	9	10	AL
1	1	1	1	2	2	2	1	1	1	MA
1	2	1	2	2	2	2	2	1	1	LA

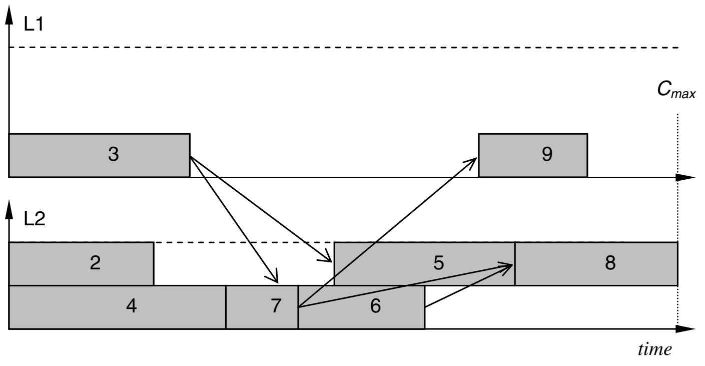

Fig. 2. A feasible solution and the schedule for the presented example.

in our problem on account of the extended representation.

3.6. Neighbourhood

Neighbours of the current solution are generated by swapping activities on AL, changing modes on MA, and changing locations on LA.

More precisely, there are three operators used in the neighbourhood generation mechanism:

- *activity swap*: swaps each two activities on the activity list that may be swapped without violating the precedence constraints; as a result, a set of neighbouring solutions is created, each of them differing from the current solution in two positions of AL;
- *mode change*: changes the execution mode of each activity to every other executable mode for this activity; as a result, a set of neighbouring solutions is created, each of them differing from the current solution in one position of MA;
- *location change*: changes the location of each activity to two neighbouring locations for this activity, i.e. to the previous and to the next location on the list of capable locations for this activity (provided that they exist); as a result, at most $2n$ neighbouring solutions are created, each of them differing from the current solution in one position of LA.

All the three operators are applied independently to the current solution to generate neighbours. It is easy to see that the above neighbourhood generation mechanism has the accessibility property (i.e. it permits to reach each feasible solution).

3.7. Tabu list

The tabu list is managed according to the *Tabu Navigation Method* (Skorin-Kapov, 1990). The tabu list (TL) is a queue of a given length. Whenever a move is performed, its reverse is added to TL in order to avoid going back to a solution already visited. The oldest existing move is removed from the front of the list (according to the FIFO policy). All moves existing on TL are tabu. However, the tabu status of a move may be cancelled according to the so-called aspiration criterion which allows the algorithm to move to a tabu solution (perform a tabu move) if this solution is better than the best found so far.

3.8. Stop criterion

The stop criterion has been defined as an assumed number of visited solutions, i.e. an assumed number of the objective function calculations, in order to assure a comparable computational effort for each search algorithm.

4. Multi-start iterative improvement and random sampling

4.1. Multi-start iterative improvement

An iterative improvement method searches the neighbourhood by performing improving moves only, so the final solution is a local optimum which may be different from the global one. In addition, this approach strongly depends on the starting solution chosen. However, if the method can start several times from different starting solutions, it can generate solutions of reasonable quality without any significant effort devoted to the search mechanism. We have chosen the multi-start iterative improvement (MSII) method also in order to assure a comparable computational effort to tabu search, by giving MSII the chance to visit the same number of feasible solutions.

Several various rules can drive the MSII search process, e.g. First Fit Strategy or Best Fit Strategy. In the former case, the first generated improving move is immediately accepted, in the latter case, the most improving move from a set of moves is chosen. We have applied the Best Fit Strategy in our implementation in order to make MSII work similarly to TS.

MSII uses several mechanisms defined for tabu search in the previous section. It starts from the same initial solution and uses the same neighbourhood generation mechanism. When there are no improving moves, it restarts with a random feasible solution, where AL is generated as described in point 3.5 but MA and LA are generated randomly.

4.2. Random sampling

Random sampling (RS) technique is a good technique to compare with, in order to prove an advantage of any more sophisticated method over solving a problem simply by generating a certain number of purely random feasible solutions. In our implementation, the random sampling technique generates an assumed number of feasible solutions in a random way, i.e. all the three lists: AL, MA, and LA are generated randomly. Of course, some restrictions during the generation have to be taken into account: AL must be precedence-feasible, MA must contain executable modes, and LA must contain capable locations for successive activities. The best solution generated over the whole process is the final solution found by RS.

5. Computational experiment

In this section, we present the results of a computational experiment concerning the implementations of the three search methods for the considered MRCPSP-SST. The implementations were coded and compiled in C++ and ran on an SGI ORIGIN 3800 machine with 160 RISC R12000 processors and 128 GFlops overall computing power, installed in the Poznań Supercomputing and Networking Center.

We used a set of benchmark problem instances generated using the project generator ProGen developed by Kolisch et al. (1995). The files with the mentioned instances, as well as the code of ProGen itself, are available in the project scheduling problem library PSPLIB from the ftp service of the University of Kiel (ftp://.bwl.uni-kiel.de/pub/operations-research/psplib). For more details concerning ProGen see Kolisch and Sprecher (1997).

We used benchmark sets for MRCPSP problems with 10, 12, 14, 16, 18, 20, and 30 non-dummy activities. All non-dummy activities may be executed in one of the three modes. The duration of activity j in mode m_j varies from 1 to 10. For each problem size a set of 640 instances was generated, but for some instances no feasible solution for the MRCPSP could be found and therefore these instances have been excluded from the PSPLIB library. In Table 5, we present the number of instances for which at least one feasible solution exists.

We have modified the PSPLIB instances in order to adapt them to our problem. Originally, there are two renewable and two non-renewable resources. We have taken into account renewable resources only which is quite justified in our case where resources are placed in different locations. However, this is not a limitation of our approach and non-renewable resources can be easily included as well. Moreover, in the PSPLIB instances the total numbers of available units for both renewable resources are given. It is not sufficient information in our problem where the total number of available units in each location must be given for both the resources. Therefore, we have assumed the minimal and maximal number of locations for each problem size, and then a random method of calculating the resource availabilities for each location, based on the resource availabilities given in the considered PSPLIB instance. More precisely, the minimal number of locations $L_{\min} = 2$, and the maximal number of locations depends on the problem size according to the formula:

$$L_{\max} = \lfloor n/3 \rfloor,$$

which means that it is equal to the greatest integer smaller or equal to $n/3$. For each problem size we have assumed up to 5 values of parameter $L \in [L_{\min}; L_{\max}]$. In Table 6 we present the number of locations for each number of activities.

Next, when the number of locations L is set, we calculate the resource availabilities of both the resources in each location by the following formula:

$$R_{kl} = \text{rand}[0.5; 1] \cdot R_k \quad k \in \{1, 2\}, l = 1, \ldots, L,$$

where rand[0.5; 1] is a random number from the interval [0.5; 1] and R_k, $k \in \{1, 2\}$, is the number of available units of renewable resource k given in a PSPLIB instance file. Notice that in this case the equality pointed in Table 1 that $\sum_{l \in K_L} R_{kl} = R_k$, $k = 1, \ldots, R$, $(R = 2)$ may not be satisfied, but this equality is defined for the general model and when using PSPLIB instances for the experiment the values of R_k are too small to divide them into several locations. Now—if, after calculating all the values of R_{kl} and performing the pre-processing procedure

Table 5
Number of instances

n	10	12	14	16	18	20	30
Number of instances	536	547	551	550	552	554	640

Table 6
Number of locations

n	10	12	14	16	18	20	30
Number of locations	2, 3	2, 3, 4	2, 3, 4	2, 3, 4, 5	2, 3, 4, 5, 6	2, 3, 4, 5, 6	2, 3, 5, 8, 10

described in point 3.3, it turns out that the list of capable locations for at least one activity is empty, we exclude this particular instance from considerations. As a result, for each problem size defined by the number of activities n and the number of locations L, the number of instances considered is less or equal to the one presented in Table 5. The final number of instances analyzed is given in Table 8 in column No.

Finally, for each pair of locations we have randomly generated a fixed transmission time (SST) as integer from the same interval as activity durations—[1; 10].

In order to ensure a comparable computational effort devoted to all the algorithms, the stop criterion has been defined as a number of solutions visited. In Table 7 we present the assumed number of visited solutions defining the stop criterion.

The results of the experiment are shown in Table 8. For each algorithm the following numbers are shown:

- #: the number of instances for which the algorithm found a solution equal to the best solution known (i.e. best solution found by any of the algorithms);

Table 7
Stop criterion

n	10	12	14	16	18	20	30
Number of solutions	120,000	144,000	168,000	192,000	216,000	240,000	360,000

Table 8
Results of the experiment

n	L	No.	TS				MSII				RS			
			#	ARD (%)	MRD (%)	CPU (seconds)	#	ARD (%)	MRD (%)	CPU (seconds)	#	ARD (%)	MRD (%)	CPU (seconds)
10	2	524	471	0.02	0.24	1.23	173	0.18	2.34	1.10	87	0.93	5.36	0.79
	3	529	477	0.02	0.21	1.36	190	0.17	3.75	1.21	64	1.07	6.78	0.92
12	2	537	489	0.03	0.32	1.67	175	0.25	1.99	1.56	85	1.11	10.23	1.13
	3	541	492	0.02	0.24	1.79	172	0.27	2.45	1.70	65	1.46	9.34	1.19
	4	543	497	0.02	0.25	1.96	181	0.34	5.11	1.82	62	1.93	9.98	1.29
14	2	539	490	0.04	0.31	2.23	183	0.33	4.23	2.14	81	2.56	12.34	1.33
	3	545	501	0.04	0.30	2.47	188	0.42	7.18	2.32	61	2.97	13.78	1.45
	4	546	501	0.03	0.25	2.65	199	0.41	6.98	2.52	54	3.83	15.77	1.67
16	2	540	503	0.03	0.39	3.56	197	0.45	5.58	3.50	74	4.02	17.99	2.28
	3	543	498	0.03	0.37	3.79	212	0.47	8.12	3.66	54	4.45	18.03	2.45
	4	547	510	0.02	0.36	4.08	225	0.56	7.96	3.72	50	5.19	22.23	2.99
	5	549	511	0.02	0.33	4.23	227	0.59	10.23	4.11	42	6.87	19.34	3.37
18	2	538	489	0.03	0.42	4.22	194	0.51	9.69	4.01	68	5.76	21.85	3.41
	3	540	494	0.02	0.39	4.36	207	0.53	10.92	4.12	53	6.13	25.67	3.55
	4	543	506	0.02	0.39	4.50	221	0.62	11.33	4.23	45	6.46	25.61	3.59
	5	548	515	0.02	0.35	4.72	224	0.69	11.95	4.39	43	6.39	28.12	3.66
	6	552	524	0.01	0.27	4.89	235	0.77	13.56	4.65	37	7.83	27.92	3.71
20	2	545	512	0.02	0.45	4.92	202	0.61	8.56	4.44	61	7.35	27.95	3.70
	3	545	514	0.02	0.44	5.11	209	0.66	10.04	4.58	48	8.11	28.21	3.82
	4	548	517	0.01	0.39	5.29	218	0.64	10.12	4.76	40	8.27	29.94	4.02
	5	550	528	0.01	0.35	5.41	224	0.79	13.46	4.92	38	8.87	28.67	4.13
	6	554	537	0.01	0.28	5.55	239	0.87	17.33	5.04	29	9.86	30.03	4.27
30	2	621	610	0.01	0.35	12.03	242	0.83	19.95	10.92	59	9.16	25.33	8.11
	3	625	613	0.01	0.27	12.69	249	0.88	22.17	11.16	48	9.28	29.29	8.30
	5	632	627	0.01	0.15	13.24	264	0.95	22.98	11.79	38	9.56	32.11	8.47
	8	638	637	0.00	0.06	13.97	278	1.07	24.34	12.54	37	9.79	33.03	8.67
	10	640	637	0.00	0.11	14.73	284	1.05	24.99	13.60	28	9.92	33.15	8.89

- ARD: the average relative deviation from the best solution known;
- MRD: the maximal relative deviation from the best solution known;
- CPU: the average computational time of the algorithm.

The results show that tabu search gives the best results, clearly outperforming multi-start iterative improvement and random sampling. For each problem size considered, the number of best solutions found by TS is the greatest, and the average and relative deviations are the smallest. This result is not surprising and confirms the expectations that intelligent search should get advantage over simple search algorithms like MSII or RS. Moreover, the advantage of TS grows when problem becomes more complex, i.e. with the number of activities, as well as with the number of locations when the number of activities is fixed. In each case, the number of best solutions found grows, and the ARD and MRD decrease. For 30 activities and larger number of locations (8 and 10), TS found the best solution in almost all instances, achieving the ARD practically equal to 0. The ARD for TS is generally very small and it keeps its small values over all problems tested. The MRD for TS can be a bit bigger but it does not exceed 0.5% for any problem size.

As far as MSII is concerned, its results are, in general, quite good – it finds a reasonable number of best solutions and the ARD just exceeds 1% for the largest problems. This is quite understandable taking into account that the algorithm is very similar to TS, using the same staring solution and neighbourhood generation mechanism. However, if the algorithm is stuck in several local optima of not that good quality, the deviation from the solution found by TS can be quite large. This is why the MRD for MSII is much bigger than for TS and reaches almost 25% for the largest problems.

As to RS, it performs worst from among the algorithms, and it gets worse with the number of activities and with the number of locations for a fixed *n*. This rather proves the fact that the problem is too complicated for a random algorithm to generate good results.

The computational times are ones to be expected—RS is the fastest, then MSII, and TS works the longest. The difference between TS and MSII in not that significant, which means that generating neighbours and calculating the objective function requires more computational effort than maintaining tabu list with simple data structures. It confirms the advantage of an intelligent local search in comparison with a simple greedy algorithm. The times obtained for RS are significantly shorter, which of course follows from the fact that it does not generate a neighbourhood, but it is still not acceptable in combination with the deterioration of the results produced.

Summarizing, the proposed TS implementation seems to be quite a promising algorithm for solving the defined problem, evidently outperforming simpler methods: MSII and RS. However, a more advanced approach, e.g. another metaheuristic, is needed to compare with tabu search in order to truly examine the efficiency of TS.

6. Conclusions

In this paper, the multi-mode resource-constrained project scheduling problem with schedule-dependent setup times has been considered, where a schedule-dependent setup time is defined as a setup time dependent on the assignment of resources to activities over time, when resources are, e.g., placed in different locations. In such a case, the time necessary to prepare the required resource for processing an activity depends on the locations in which successive activities are executed. The model of the problem has been presented, and practical examples have been discussed. An application of a local search metaheuristic—tabu search for the considered problem has been described. This metaheuristic has been compared with the multi-start iterative improvement method as well as with random sampling, on the basis of a computational experiment performed on a set of instances, obtained from standard test problems constructed by the ProGen project generator with some modifications.

The results of the computational experiment show that tabu search seems to be an efficient algorithm for solving the considered problem, clearly outperforming simple search algorithms: multi-start iterative improvement and random sampling. Nevertheless, for better evaluation of the proposed TS implementation, it has to be compared with more advanced methods, as well as with optimal solutions (or at least lower bounds) for smaller numbers of activities.

In the future we plan to implement other metaheuristics, e.g. simulated annealing, genetic algorithms, or scatter search, to the defined problem. Our implementation of SA is still one of the best metaheuristic approaches to the classical MRCPSP

(Józefowska et al., 2001). It would be interesting to examine its efficiency for the MRCPSP-SST and compare it with tabu search. It would be also useful to construct a branch-and-bound procedure in order to find optimal solutions. Comparing the results of the metaheuristics with optimal solutions will allow the full evaluation of their efficiency. Also other criteria can be taken into account, from among which financial criteria, like the maximization of the net present value, seem to have good practical justification.

Acknowledgement

This research has been supported by the Polish Ministry of Education and Science Grant No. 3T11F 025 28.

References

Błażewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling subject to resource constraints. Discrete Applied Mathematics 5, 11–24.

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

Conway, R.W., Maxwell, W.L., Miller, L.W., 1967. Theory of Scheduling. Addison-Wesley, Reading, Massachusetts.

Demeulemeester, E.L., Herroelen, W.S., 2002. Project Scheduling—A Research Handbook. Kluwer Academic Publishers, Boston.

Elmaghraby, S.E., 1977. Activity Networks: Project Planning and Control by Network Models. Wiley, New York.

Glover, F., 1986. Future path for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

Glover, F., 1989. Tabu search—part 1. ORSA Journal of Computing 1, 190–206.

Glover, F., 1990. Tabu search—part 2. ORSA Journal of Computing 2, 4–32.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Norwell.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Węglarz, J., 2001. Simulated annealing for multi-mode resource-constrained project scheduling problem. Annals of Operations Research 102, 137–155.

Kelley, J.E., 1963. The critical path method: resource planning and scheduling. In: Muth, J.F., Thompson, G.L. (Eds.), Industrial Scheduling. Prentice-Hall, pp. 347–365.

Kolisch, R., 1995. Project Scheduling under Resource Constraints—Efficient Heuristics for Several Problem Classes. Heidelberg, Physica.

Kolisch, R., Sprecher, A., 1997. PSPLIB—a project scheduling problem library. European Journal of Operational Research 96, 205–216.

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

Mika, M., Waligóra, G., Węglarz, J., 2003. A metaheuristic approach to scheduling workflow jobs on a grid. In: Nabrzyski, J., Schopf, J.M., Węglarz, J. (Eds.), Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers, Boston, pp. 295–318.

Neumann, K., Schwindt, C., Zimmermann, J., 2002. Project Scheduling with Time Windows and Scarce Resources. Springer, Berlin.

Schwindt, C., 2005. Resource Allocation in Project Management. Springer, Berlin.

Skorin-Kapov, J., 1990. Tabu search applied to the quadratic assignment problem. ORSA Journal of Computing 2, 33–45.

Sprecher, A., Hartmann, S., Drexl, A., 1997. An exact algorithm for project scheduling with multiple modes. OR Spektrum 19, 195–203.

Węglarz, J., 1999. Project Scheduling—Recent Models, Algorithms and Applications. Kluwer Academic Publishers, Dordrecht.