Drexl, Andreas

Scheduling of project networks by job assignment

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 247

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Drexl, Andreas (1990) : Scheduling of project networks by job assignment, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 247, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/161993

Nr. 247

Scheduling of Project Networks by Job Assignment

Andreas Drexl

Juni 1990

Andreas Drexl, Institut für Betriebswirtschaftslehre,
Christian–Albrechts–Universität zu Kiel, Olshausenstraße 40, D–2300 Kiel 1

A recurring problem in project management involves the allocation of scarce resources to the individual jobs comprising the project. In many situations such as audit scheduling, the resources correspond to individuals (skilled labour). This naturally leads to an assignment type project scheduling problem, i.e. a project has to be processed by assigning one of several individuals (resources) to each job. In this paper we consider the nonpreemptive variant of a resource–constrained project job–assignment problem, where job durations as well as costs depend upon the assigned resource. Regarding precedence relations as well as release dates and deadlines, the question arises, to which jobs resources should be assigned in order to minimize overall costs. For solving this time–resource–cost–tradeoff problem we present a hybrid branch and bound / dynamic programming algorithm with a (rather efficient Monte Carlo type) heuristic upper bounding technique as well as various relaxation procedures for determining lower bounds. Computational results are presented as well.

(PROJECT MANAGEMENT – RESOURCE CONSTRAINTS; AUDIT SCHEDULING; GENERALIZED ASSIGNMENT PROBLEM; BRANCH AND BOUND; DYNAMIC PROGRAMMING; MONTE CARLO HEURISTIC)

1. Introduction

The scheduling problems considered here deal with determining when jobs should be processed, given limited availabilities of resources as well as a limited number of time periods. The words job and project will be used throughout the paper to denote two levels of aggregation. A project consists of a set of jobs, i.e. we only consider the job level and the project level.

Traditional resource–constrained project scheduling approaches [4], [5], [22], [24] have been restricted to the case in which each job may be performed in only one predefined way. More recently efforts have been made to formulate and solve the more general preemptive project scheduling problem where job durations are functions of consumed resources [1]. Meanwhile efforts have been documented in [8], [15], [16], [23] regarding the formulation and solution of a variety of nonpreemptive project scheduling problems where job durations are discrete functions of job performance modes.

In this paper we consider a variant of the discrete nonpreemptive multi–mode resource–constrained scheduling problem, where the resources are substitutional, i.e. may be assigned alternatively to the jobs: Each job must be scheduled requiring only one of the doubly–constrained resources (limited per period and total availability). Job

durations as well as costs depend upon the assignment of each resource to a job. Which job should be scheduled by which resource at minimum overall costs regarding precedence relations as well as release dates and deadlines?

Although a number of different objectives may be incorporated in our model, this paper focuses on the time–resource–cost tradeoff variant, where job costs (durations) appear in the objective function (constraints). One of the reasons is, that the resulting model is highly suitable for dealing with audit scheduling problems [2], [3], [6]. In addition to audit scheduling, this model is appropriate for other man/machine project scheduling problems.

2. Model Assumptions

Before giving the mathematical programming formulation, assumptions describing our model are stated more precisely:

- One project has to be finished within the planning horizon T (deadline), divided into T periods of equal length (for the generalization to the multi–project case see section 7).
- The project consists of a set of jobs j = 1,2,...,J and precedence relations between some of these jobs are specified in advance. The set of immediate predecessors of job j is denoted by $\mathcal{V}_j$.
- There are K doubly – constrained resources, each resource being available with one unit per period; total number of periods in which resource k is available is D_k .
- Each job must be scheduled by one resource; each resource can only schedule one job per period; jobs once initiated cannot be interrupted (preempted) by another job.
- Scheduling job j by resource k requires d_{jk} time periods; the corresponding scheduling costs are c_{jk} units.

The *notation* used in the following sections is summarized in Table 1.

3. Formulation of the Cost Minimization Problem

We get a bound on the maximum time interval, within which job j may be scheduled by any resource k , by taking $\delta_j := \min\{d_{jk} \mid k=1,2,...,K\}$ and performing a traditional critical path analysis in order to reduce the number of binary variables in the following formulation.

TABLE 1

Definitions and Notation

Symbol (Listed Alphabetically)	Definition / Notation
c_{jk}	Total costs of scheduling job j by resource k
d_{jk}	Duration of scheduling job j by resource k
δ_j	$\min\{d_{jk} \mid k = 1,2,...,K\}$
D_k	Total capacity of resource k (in periods)
DL_k	Left–over capacity of resource k
ES_j	The critical path earliest start time of job j
EF_j	The critical path earliest finish time of job j
S0	Set of jobs currently unscheduled
S1	Set of jobs currently scheduled
SC	Set of candidate jobs
j	A specific job, j = 1,2,...,J
J	Number of jobs
(j,k)	Denotes assignment of resource k to job j
k	A specific resource, k = 1,2,...,K
K	Number of doubly–constrained resources
LF_j	The critical path latest finish time of job j
LS_j	The critical path latest start time of job j
ST_j	Slack time of job j
t	A specific period, t = 0,1,...,T
T	Planning horizon (deadline)
ν_j	The set of immediate predecessors of job j
x_{jkt}	Job j is assigned to resource k and finished in time period t (binary variable)
Z(.)	Objective function value
Z*	Optimal objective function value
$\underline{Z}$, $\bar{Z}$	Lower, upper bound

More precisely, we determine the critical path earliest start time ES_j of job j by using δ_j for all j = 1,2,...,J. Analogously we determine the critical path latest finish time LF_j of each job with respect to T by using δ_j once more. Thus [ES_j , LF_j] represents *a bound on the maximum time interval*, within which job j can be scheduled without violating deadline. Defining variables

$$x_{jkt} := \begin{cases} 1, & \text{execution of job j by resource k is finished in time period t} \\ 0, & \text{otherwise} \end{cases}$$

we can formulate a *binary program* using the general framework given in [17] as follows:

$$\text{Minimize } Z(\underline{x}) = \sum_{j=1}^{J} \sum_{k=1}^{K} c_{jk} \sum_{t=ES_j+d_{jk}}^{LF_j} x_{jkt} \tag{1}$$

Subject to

$$\sum_{k=1}^{K} \sum_{t=ES_j+d_{jk}}^{LF_j} x_{jkt} = 1 \qquad \text{for all j} \tag{2}$$

$$\sum_{k=1}^{K} \sum_{t=ES_h+d_{hk}}^{LF_h} t \cdot x_{hkt} \leq \sum_{k=1}^{K} \sum_{t=ES_j}^{LF_j - d_{jk}} (t - d_{jk})\, x_{jkt} \qquad \text{for all j and } h \in \mathcal{V}_j \tag{3}$$

$$\sum_{j=1}^{J} \sum_{q=t}^{t+d_{jk}-1} x_{jkq} \leq 1 \qquad \text{for all k, t} \tag{4}$$

$$\sum_{j=1}^{J} \sum_{t=ES_j+d_{jk}}^{LF_j} d_{jk}\, x_{jkt} \leq D_k \qquad \text{for all k} \tag{5}$$

$$x_{jkt} \in \{0,1\} \text{ for all j, k, t} \tag{6}$$

In (1) $\underline{x}$ represents the vector of all binary variables, $Z(\underline{x})$ represents the objective function value for any feasible vector $\underline{x}$ with respect to (2)–(6). (2) are job completion constraints. (3) ensures that all precedence relations are maintained. (4) and (5) represent per period and total availability resource constraints, respectively.

4. Historical Treatment of the Problem

First of all, it should be stressed that in the presence of time and resource restrictions there seems to be no way of successfully using conventional scheduling rules [11], [12], [20], [21], [25] in order to definitely find a (hopefully good) feasible solution (see section 7). In fact, we even don't know in advance whether any feasible solution exists at all. Furthermore, in our formulation (1)–(6) the number of variables and constraints grows rapidly with increasing problem size. Thus general 0–1 programming approaches are of only limited importance.

Up to now three approaches for solving (1)–(6) have been suggested: The first one, presented in [2], defines in the first phase additional precedence relations which guarantee, that in no period more than K jobs are competing for scarce resources. In the second phase a binary program is formulated, which allows for assigning resources and determining start times of jobs (without overlapping). The main drawbacks of this approach are the unsystematic way of generating additional precedence relations and the effort needed to solve the binary program by standard methods (although the number of variables is reduced by a factor of 5 to 10, compared with (1)–(6)). The second approach uses set – partitioning techniques [18]. One determines in the first phase partial feasible schedules for each of the K resource types. Therefore it is necessary to reduce the feasible time interval of any job according to the inequality $LF_h \leq ES_j$ for all h, j with $h \in \mathcal{V}_j$. In phase two, one formulates and solves a set – partitioning problem. The main drawback of this approach is that the number of variables (columns) of the set – partitioning problem is growing exponentially with increasing problem size. Although both approaches use optimization techniques, they do not guarantee that they will determine the optimum solution to a problem even with "infinite time". The third one, presented in [6], is an enumerative type of optimization algorithm. In section 6 we will outline this algorithm.

5. Monte Carlo Heuristic

Traditional scheduling rules for heuristically constructing feasible solutions (and thus determining upper bounds $\bar{Z}$ for the unknown optimum objective function value Z^*) quite often do not find an (existing) feasible solution in the presence of time and resource restrictions (see section 7 below). Therefore a more sophisticated stochastic (Monte Carlo) scheduling method like the following should be used in this context.

Adopting an operating scheme similar to the one described in [12] for traditional scheduling rules, jobs currently unscheduled are selected as candidates, if all predecessors

have been scheduled, and if the earliest start time is less than or equal to the time t of the simulation clock. Starting with t:=0 the simulation clock is increased successively. More formally we obtain the set of candidates SC as follows:

$$S0 := \{j \mid \text{job } j \text{ currently is unscheduled}\}$$

$$S1 := \{j \mid \text{job } j \text{ currently is scheduled}\}$$

$$SC := \{j \in S0 \mid ES_j \leq t, \forall_j \in S1\}$$

Denoting with AR the set of resources which are available in t (assigned to a job only until t–1; enough left–over capacity), the following opportunity costs may be calculated:

$$\mu_{jk} := \max_{q \in AR} c_{jq} - c_{jk} \quad \text{for all } j \in SC \text{ and } k \in AR$$

μ_{jk} compares the costs of scheduling job j by resource k with the worst–case consequence if k would be unavailable. In this sense it seems to be appropriate to take μ_{jk} in order to decide which available resource should be assigned to which candidate job.

In the following we will take μ_{jk} (slightly modified) as stochastic assignment probabilities. Taking μ_{jk} as defined above all resources with highest scheduling costs would get an assignment probability of zero – a misleading consequence in the case of scarce resources and tight times. In order to overcome this deficiency we set

$$\mu_{min} := \min \{\mu_{jk} > 0 \mid \text{for all } j \in SC \text{ and } k \in AR\}$$

and calculate

$$\sigma_{jk} := (\mu_{jk} + \mu_{min})^{\alpha} \text{ for all } j \in SC \text{ and } k \in AR \qquad (7)$$

Taking σ_{jk} as stochastic assignment probabilities we get an arbitrarily large range of stochastic scheduling rules for $\alpha \geq 0$.

It should be noted that the above choice of μ_{min} is not crucial for the behavior of the algorithm. Alternatively we could take (without affecting algorithmic performance substantially) a parameter $\beta > 0$ (e.g. $\beta = 1$) which should be "small" compared with the μ_{jk} . The adjective "small" is the motivation for taking μ_{min} .

In general, one does not know in advance the tightness of resources and dates; therefore one should start with an appropriate high value of α (e.g. $\alpha = 2.0$) in order to hopefully get a near–optimum solution. Some trials fail in attempting to construct a feasible solution α should then be decreased and the solution process should be repeated. Table 6 explains more about the sensitivity of the solution process to the exponential weight α .

Many possibilities exist for stochastically assigning resources to jobs. For example a linear sum assignment problem [10] can be formulated taking σ_{jk} as cost coefficients, starting an optimization algorithm and stopping before reaching optimality. Due to the unavailability of appropriate stopping criteria for assignment algorithms, we chose a pragmatic way of successively assigning resources to jobs: We randomly assign $k \in AR$ to $j \in SC$, update both sets and the stochastic assignment probabilities (7), assign randomly once more etc. as long as either AR or SC is (or both are) empty. Then we increase t by the minimum number of time periods such that both sets become nonempty and start the random assignment process once more.

Formally the *stochastic construction method (STOCOM)* may be described as follows using DL_k (left–over capacity of resource k) and AD_k (availability date of resource k) as additional symbols:

1. $t := 0$; $DL_k := D_k \; \forall$ k; $AD_k := 0 \; \forall$ k; $S1 := \phi$; $S0 := \{j \mid \forall \; j\}$.

2. Determine ES_j and LF_j by traditional critical path analysis (using δ_j).

3. $SC := \{j \in S0 \mid ES_j \leq t, \forall_j \in S1\}$; if $SC = \phi$ then goto 7;
   $AR := \{k \mid AD_k \leq t, DL_k > 0 \; \forall \; k\}$; if $AR = \phi$ then goto 6.

4. Calculate σ_{jk} according to (7) $\forall \; j \in SC, \forall \; k \in AR$ (if $d_{jk} > DL_k$ then set $\sigma_{jk} := 0$); if $\sigma_{jk} = 0 \; \forall$ j and k then goto 6.

5. Chose $\gamma \in SC$ and $\pi \in AR$ randomly with probability proportional to σ_{jk} ; if $t + d_{\gamma\pi} > LF_\gamma$ then STOP (no feasible solution found); $DL_k := DL_k - d_{\gamma\pi}$; $S0 := S0 - \gamma$; $S1 := S1 \cup \gamma$; $AD_\pi := d_{\gamma\pi}$; store the partial feasible schedule; update ES_j for all successors of γ; if all jobs have been scheduled then STOP (feasible solution found); goto 3.

6. $\tau := \max \{0, \min \{AD_k > 0 \mid \forall \; k \text{ with } DL_k > 0\}\}$; if $\tau = 0$ then STOP (no feasible solution found); $AD_k := \max \{0, AD_k - \tau\} \; \forall$ k; $t := t + \tau$; update $ES_j \; \forall \; j \in S0$; goto 3.

7. $\tau := \min \{ES_j \mid j \in S0, \forall_j \in S1\}$; $AD_k := \max \{0, AD_k - (\tau - t)\} \; \forall$ k; $t := \tau$; goto 3.

The procedure either stops with a feasible solution, if all jobs have been assigned, or at a point, where no further assignments are possible. In both cases one should make some restarts at $t := 0$ (see section 7) in order to hopefully get feasible (near–optimum) solutions.

STOCOM has been implemented rather efficiently (see section 7) using the following data structures: The precedence relations are stored in a forward manner (successors) as well as in a backward manner (predecessors), both as node oriented lists. The sets SC as well as AR are represented by cyclic linked lists. Thus all algorithmic instructions can be realized by a few simple operations.

In some cases it may be interesting to restrict the set of candidate jobs to those elements of SC with minimum slack time. Denoting with LS_j the latest start time (calculated analogously to LF_j) and with $ST_j := LS_j - ES_j$ the corresponding slack time we get an alternative job candidate set as follows:

$$STC := \{j \in SC \mid ST_j = \min \{ST_\gamma \mid \gamma \in SC\}$$

Using STC instead of SC may – in some cases – reveal an algorithmic variant with a better performance (see section 7).

Conceptually STOCOM may be interpreted as a stochastic generalization of Vogel's method to the transportation problem, which uses "regrets" in a deterministic way.

6. Exact Algorithm

The algorithm is an enumerative type of branch and bound method. It simultaneously decides about job–sequencing (which job should preceed others?) and resource–assignment (which resource should be assigned to which job ?). Beginning with all jobs being unassigned ($x_{jkt} = 0$ for all j, k, t) the algorithm starts by selecting one job as a candidate for being scheduled as early as possible by one of the resources, setting the corresponding variable $x_{jkt} := 1$. The algorithm always builds precedence and resource feasible partial schedules (solutions). "Partial" indicates that not all jobs have currently been scheduled (corresponds to "$\leq$" instead of "=" in (2)). Scheduling jobs is equivalent to augmenting the partial feasible solution. Enumeration is done in a LIFO–implicit way, i.e. partial feasible schedules are augmented as long as neither precedence/resource infeasibilities occur nor lower bounds exceed the upper bound; in both cases backtracking occurs.

This enumeration scheme is similar to one proposed by other researchers [15], [16], [23] for solving discrete multi–mode resource–constrained project scheduling problems. Preliminary computational experiences with this scheme (without additional features) have been rather discouraging. So we incorporate particular upper (section 5) and lower bounding procedures, dynamic programming features as well as preprocessing techniques in order to accelerate convergence. We now outline all these components; for a detailed description see [6].

6.1 Lower Bounding Procedures

Let us assume that in any stage of the enumeration process a partial feasible schedule, denoted by the set S1 (and the complementary set S0), is given. In order to decide whether this (actual or current) partial schedule may yield a feasible schedule with an objective function value, which is better than the one known so far $\bar{Z}$, we may use one feasibility lower bound and four ways of calculating lower bounds $\underline{Z}$ for the (unknown) optimum objective function value as described in the following.

Feasibility Lower Bound (FLB)

Calculating the actual total resource consumption and adding a lower bound for the additional resource requirements for scheduling currently unscheduled jobs (sum of $\min \{d_{jk} \mid k=1,2,...,K\}$ for all $j \in S0$) yields a feasibility lower bound.

Optimality Lower Bound 1 (OLB1)

Adding a lower bound for the additional scheduling costs necessary for scheduling currently unscheduled jobs (sum of $\min \{c_{jk} \mid k=1,2,...,K\}$ for all $j \in S0$) to the objective function value of the current partial feasible solution yields a lower bound $\underline{Z}$ for the optimum objective function value.

Optimality Lower Bound 2 (OLB2)

OLB2 may be derived by relaxing (omitting) constraints (3) and (4) as well as K–1 constraints in (5). This leads to a (modified) continuous knapsack problem and thus we get another lower bound $\underline{Z}$ for the optimum objective function value.

Optimality Lower Bound 3 (OLB3)

OLB3 may be derived by relaxing constraints (4) and (5). The resulting model determines those additional costs, which are – regarding the start time t of that job in the current partial feasible schedule, which has been scheduled as the last one – necessary for scheduling the current unscheduled jobs without violating project deadline. OLB3 requires in some cases – compared with OLB1 – "faster" (and more "expensive") resources in order to satisfy project deadline. Thus OLB3 is stronger than OLB1.

Computational experiments were performed by solving the subproblems in an enumerative way. This indeed reduces the search tree drastically – with little additional computational effort. So we propose to slightly relax once more in order to get "good" bounds more quickly. This can be done heuristically by determining a lower bound for the project crashing costs as follows: Perform a traditional critical path analysis for the currently unscheduled jobs $j \in S0$ using job durations corresponding to the least expensive resource; look whether or not some jobs must be accelerated in order to satisfy project deadline. If duration reduction is necessary, we determine that critical job $j \in S0$,

for which the ratio "cost increase / time saving" regarding the resource with minimum additional costs is minimal. Job j yields the relative least expensive project crashing possibility. Thus, conceptually this *project crashing procedure* may be considered as a bottleneck heuristic.

Optimality Lower Bound 4 (OLB4)

OLB4 can be derived by relaxing constraints (3) and (4). This leads to a model which is well–known to be the generalized assignment problem (GAP) [9], [14], [19]. The GAP is known to be NP–hard [9]. Indeed it would take substantial CPU – time to solve hundreds or thousands of GAPs even of smaller dimensions to optimality, especially in the case of scarce resources. So we propose to calculate a lower bound $\underline{Z}$ for the optimum objective function value of the GAP by the multiplier adjustment method of [9] without incorporating it into a branch and bound scheme.

With respect to computational effort necessary to obtain these bounds we propose to calculate them in the order FLB, OLB1, OLB2, OLB3 and OLB4.

6.2 Dynamic Programming Features

Simultaneously enumerating job sequencing and resource assigning decisions as outlined above has the following disadvantage: If candidate jobs are assigned to distinct resources at immediate succeeding nodes of the search tree, then they all may start at the same time period. In other words: None of these assigned jobs causes a delayed start of one of the other jobs due to resource conflicts. In this situation it is only necessary to investigate one of several sequencing decisions. This situation essentially corresponds to the "collapsing tree" behaviour of dynamic programming algorithms for the travelling salesman problem [13].

With respect to this observation, it would be desirable to solve (1)–(6) by dynamic programming. Due to explosive growth of storage requirements, this is impractical even for very small problems. In order to make this observation (at least partially) useful within our branch and bound algorithm, we proceed as follows (without explicitly formulating a recursive equation for the subproblem under consideration): In any node of the search tree we identify non–conflicting assignments of resources to candidate jobs and branch in such a way that only one of these partial feasible schedules will be examined.

More formally let $SK_\nu(m)$ $\nu=1,2,...$ be sets of tupels (j,k) each such that for each pair of tupels (β,γ) and (ξ,π) $\beta \neq \xi$ and $\gamma \neq \pi$. Each of these sets corresponds to an assignment of m distinct candidate jobs to distinct resources. We arbitrarily order the tupels in each set and evaluate only this sequence of job–resource assignments.

In order to demonstrate the advantage of this shortcut examine the example of Table 2. In t = 0 we have SC := {1, 2, 14, 15}. For m = 2 there are 39 distinct sets $SK_\nu(2)$ with $\nu = 1,2,...,39$. For m = 3 there are 51 distinct sets $SK_\nu(3)$ with $\nu = 1,2,...,51$. For m = 4 there are 20 distinct sets $SK_\nu(4)$ with $\nu = 1,2,...,20$. Obviously only tupels (j,k) with $d_{jk} < \infty$ have to be considered ($c_{jk} = d_{jk} = \infty$ means that job j must not be scheduled by resource k). Within each of these sets only one of the m! possible job resource assignment sequences has to be considered, thus saving the evaluation of m!–1 job resource assignment sequences within each of these sets.

Conceptually this procedure yields a hybrid branch and bound / dynamic programming algorithm, which combines the advantages of both approaches. It should be noted that the implementation of this procedure is computationally highly involved and indeed requires a lot of "administrative" instructions – with the reward of reducing the number of nodes of the search tree explicitly evaluated drastically.

6.3 Preprocessing Techniques

It has been shown by other researchers [23] that the order in which jobs / resources are considered for augmenting partial feasible schedules, has some influence on algorithmic performance. For comparative purposes we will investigate four ways of sorting jobs:

ES	Sort all jobs in non–decreasing order of earliest start times; ties are broken by mimimum slack.
EF	Sort all jobs in non–decreasing order of earliest finish times; ties are broken by mimimum slack.
LS	Sort all jobs in non–decreasing order of latest start times; ties are broken by mimimum slack.
LF	Sort all jobs in non–decreasing order of latest finish times; ties are broken by mimimum slack.

These four criteria used for presorting of jobs are traditional critical path times, calculated on the basis of δ_j for all j.

With respect to the cost minimization objective function only "cost–sorting" of the resources is considered in the following way:

> Sort all the resources which may be assigned to one job in non–decreasing order of scheduling costs. Perform this sorting for each job.

Cost–sorting is combined with each of the four job–sorting criteria. Job– and cost–sorting is done in a preprocessive way before starting the optimization part of the algorithm.

We have compared these preprocessing techniques on small problems and we found that they performed similarly with a possible small advantage to ES. So the latter will be used in the following.

7. Computational Results

The algorithms have been coded in FORTRAN 77 and implemented on an IBM 3090–200. Table 2 presents the precedence structure, job durations and job costs of an audit scheduling test example taken from the literature [2], [18].

This example corresponds to a multi–project generalization of (1)–(6) with project–specific release dates R_i as well as project–specific deadlines T_i with R_i = (0, 7, 20, 0, 15) and T_i = (20, 30, 30, 20, 30) for project i = 1,2,...,5. In Table 2 the separation of the 20 jobs to the five projects is indicated by dotted lines. The realease dates and the deadlines may be represented implicitely through the bounds computed in section 3, to reduce the number of variables.

Though being relatively small the example requires 497 binary variables and 207 constraints in terms of (1)–(6).

Table 3 presents sample results concerning the example of Table 2 for a wide range of total availability resource restrictions D_k ranging from 12 to 22 ($\forall k$). For $D_k \geq 22$ the optimum objective function value equals 5065. For $D_k = 12$ no feasible solution could be determined. Thus it is supposed that no feasible solution exists for $D_k \leq 12$.

The procedure of [2] determines solutions with (bad) objective function values 5390, 5225 and 5135 for D_k = 15, 20 and 25 respectively (with corresponding computation times of 175, 31 and 3100 sec on a UNIVAC 1108). The set – partitioning approach [18] produces 5150, 5070 and 5070 for D_k = 15, 20 and 25 respectively (with computation times of 37.6, 41.3 and 47.8 sec on a CDC CYBER 172). Thus for $D_k = 25$ the set – partitioning approach could not find and verify the optimum solution.

Columns two and three of Table 3 present objective function values of the best feasible solutions determined with STOCOM using SC and STC, where in both cases for $\alpha = 2.0$, $\alpha = 1.5$, $\alpha = 1.0$ and $\alpha = 0.5$ the heuristic has been applied 50 times. STC seems to be slightly superior as a rule for selecting job candidates due to looking at urgency of jobs. Regarding computational requirements (constructing $4 \cdot 50 = 200$ schedules takes 86 milliseconds on average – which corresponds to 0.43 milliseconds per schedule) as well as quality of solutions (percentage deviation from optimum objective function value never exceeds 3% substantially – whenever a feasible solution has been found at all), the heuristic seems to be very effective.

TABLE 2

An Example from Literature

j	ν_j	d_{jk}						c_{jk}					
1	ϕ	7	∞	6	∞	4	4	280	∞	300	∞	400	380
2	ϕ	∞	6	∞	3	∞	3	∞	270	∞	270	∞	285
3	{1, 2}	∞	∞	6	4	∞	3	∞	∞	300	360	∞	285
4	{1, 2}	5	∞	4	∞	∞	∞	200	∞	200	∞	∞	∞
5	{3, 4}	∞	7	7	∞	∞	4	∞	315	350	∞	∞	380
6	ϕ	8	7	∞	∞	3	∞	320	315	∞	∞	300	∞
7	ϕ	∞	∞	7	4	∞	∞	∞	∞	350	360	∞	∞
8	{6, 7}	7	7	∞	∞	5	3	280	315	∞	∞	500	285
9	{6, 7}	∞	∞	5	3	∞	∞	∞	∞	250	270	∞	∞
10	{8, 9}	8	8	∞	∞	3	∞	320	360	∞	∞	300	∞
11	ϕ	∞	∞	4	2	∞	2	∞	∞	200	180	∞	190
12	ϕ	3	∞	3	∞	1	∞	120	∞	150	∞	100	∞
13	{11, 12}	∞	5	5	3	∞	∞	∞	225	250	270	∞	∞
14	ϕ	∞	∞	∞	5	5	∞	∞	∞	∞	450	500	∞
15	ϕ	6	∞	5	∞	∞	∞	240	∞	250	∞	∞	∞
16	{15}	∞	4	∞	2	2	∞	∞	180	∞	180	200	∞
17	ϕ	5	5	4	∞	∞	∞	200	225	200	∞	∞	∞
18	{17}	∞	∞	5	3	∞	3	∞	∞	250	270	∞	285
19	{17}	5	5	∞	∞	2	3	200	225	∞	∞	200	285
20	{18,19}	∞	∞	4	3	∞	∞	∞	∞	200	270	∞	∞

The last five columns of Table 3 give some information about the relative effectiveness of the bounding procedures. Each entry represents the ratio "number of times, when the bounding procedure causes backtracking" divided by the "total number of executions of the corresponding procedure". The effectiveness of FLB and OLB1 is (not surprisingly) opposite with respect to a varying degree of capacity scarceness. The other bounding procedures are relatively (in)effective regarding all capacity restrictions. OLB2 and OLB4, which were initially thought to be "sensitive" with respect to scarce resources,

TABLE 3

Experimental Results for the Example of Table 2

D_k	SC	$\bar{Z}$ STC	Z*	CPU–sec	FLB	OLB1	OLB2	OLB3	OLB4
12	∞	∞	∞(!?)	>2500.0	0.33	0.00	0.00	0.00	0.08
13	∞	∞	5240	937.18	0.16	0.08	0.11	0.00	0.14
14	5310	5310	5150	389.27	0.11	0.03	0.14	0.33	0.14
15	5310	5255	5150	216.01	0.09	0.03	0.14	0.31	0.10
16	5125	5125	5120	157.86	0.03	0.25	0.06	0.30	0.00
17	5165	5115	5100	64.39	0.01	0.25	0.14	0.22	0.01
18	5110	5100	5070	11.74	0.00	0.26	0.18	0.33	0.03
19	5090	5080	5070	32.59	0.00	0.26	0.23	0.38	0.05
20	5080	5090	5070	33.76	0.00	0.27	0.27	0.47	0.06
21	5090	5080	5070	39.23	0.00	0.29	0.32	0.42	0.08
22	5080	5080	5065	43.54	0.00	0.53	0.04	0.26	0.02

are ineffective as well due to not "reaching" optimum objective function values increasing with decreasing capacities. It should be mentioned, that these ratios highly depend on the order in which the bounding procedures are performed. Indeed, the "weak and fast" bound OLB2 reduces the ratios especially of the "tight and slow" GAP bound OLB4.

The results of Table 3 demonstrate that even small problems can only be solved to optimality with the optimization procedure in the case of plentiful resources. With resources becoming scarce, the required computation time increases drastically even for problems with slightly more than a dozen jobs: The bounding procedures are not able to "compensate" for the increase in the optimum objective function value. Thus, it is quite clear that the main benefit of the exact approach is to provide benchmark tests for small problems in order to (hopefully) establish the quality of results obtainable with the Monte Carlo heuristic STOCOM.

Table 4 compares the quality of solutions of STOCOM with the results obtained by the following traditional deterministic scheduling rules [12], [23]:

SOF — Schedule the job with minimum scheduling costs regarding all available resources (analogous to shortest operation first).

MAXMIN	Schedule the job whose minimal costs of being scheduled by the least expensive available resource are maximal.
MINSLK	Schedule the job whose actual slack time is minimal.
FCFS	Schedule the job which arrived first in the set of candidate queue SC or STC (first come – first served).

In all cases the tie breaker is the (minimum) resource or job number.

Columns 1 and 2 of Table 4 provide the number of jobs J as well as the number of doubly – constrained resources K. The measure of resource scarcity MRS (column 3) is defined as follows:

$$MRS = \sum_{k=1}^{K} D_k \; / \; \sum_{j=1}^{J} \delta_j$$

In the example of Table 2 MRS ranges from 1.1 ($D_k = 12 \; \forall \; k$) to 2.0 ($D_k = 22 \; \forall \; k$). Note that none of the above deterministic scheduling rules is able to find a feasible solution for the example of Table 2 for $D_k \leq 15$ ($\forall k$) (corresponding to MRS $\leq$ 1.4). MRS seems to be an appropriate measure for the evaluation of heuristics.

For each combination of J, K and MRS 5 test problems have been generated randomly with c_{jk} and d_{jk} being uniformely distributed in [0;100]. The entries #1 / #2 of Table 4 correspond to the number of times, when the appropriate heuristic was able to find a feasible solution (#1) and to the number of times when the solution determined with the heuristic was equal to the best one found by anyone of the heuristics (#2). Once more STOCOM has been applied 50 times for α = 2.0, 1.5, 1.0 and 0.5 using SC as well as STC.

As can be seen the determinsitic rules were not able to find the best known solutions in the case of MRS $\leq$ 1.5 for anyone of the test examples. Moreover these rules could find a feasible solution for MRS $\leq$ 1.5 in strictly less than 50% of the instances. In the case of MRS = 2.0 the deterministic rules often produced feasible solutions, but for most examples they did not succeed in determining the best ones. In contrast to this poor behavior STOCOM performed very well regardless of using SC or STC especially for MRS $\geq$ 1.5; for MRS = 1.0 STOCOM was able to find feasible solutions in more than 50% of the instances. Once more STC seems to be slightly superior.

Of course STOCOM may not be "fairly" compared with the deterministic scheduling rules, considering $4 \cdot 50 = 200$ applications of the former and only one of the latter. Nevertheless Table 4 drastically shows the following: The deterministic scheduling rules surprisingly behave rather poor regarding the unability of finding feasible solutions even in the case of plentiful resources.

TABLE 4
Results of STOCOM and the Deterministic Scheduling Rules

J	K	MRS	SOF	MAXMIN	MINSLK	FCFS	STOCOM SC	STOCOM STC	STOCOM CPU
50	5	1.0	0/0	1/0	0/0	0/0	3/2	2/2	
		1.5	2/0	3/0	1/0	2/0	5/3	5/5	
		2.0	5/1	5/0	5/2	5/1	5/5	5/5	5.2
	10	1.0	1/0	0/0	0/0	0/0	2/2	2/2	
		1.5	1/0	1/0	0/0	2/0	5/5	5/4	
		2.0	5/0	5/2	5/1	5/0	5/5	5/5	7.8
100	5	1.0	0/0	0/0	0/0	0/0	2/1	1/1	
		1.5	0/0	0/0	1/0	1/0	5/3	5/5	
		2.0	5/2	5/1	5/0	5/0	5/5	5/5	49.8
	10	1.0	0/0	0/0	0/0	0/0	2/2	3/2	
		1.5	2/0	0/0	1/0	1/0	5/5	5/4	
		2.0	5/0	5/2	5/1	5/1	5/5	5/5	77.1
200	5	1.0	0/0	0/0	1/0	0/0	4/4	4/4	
		1.5	1/0	3/0	2/0	0/0	5/4	5/5	
		2.0	5/1	5/0	5/1	5/1	5/5	5/5	286.4
	10	1.0	0/0	0/0	0/0	0/0	3/2	3/3	
		1.5	0/0	2/0	0/0	3/0	5/4	5/5	
		2.0	5/1	5/1	5/1	5/1	5/5	5/5	470.5

TABLE 5
Comparison of STOCOM with the Optimization Procedure

J	K	MRS	CPU–sec (Opt.) MIN	MEAN	MAX	STOCOM (STC) NOPT	DMN%	DMX%	NOFEAS
10	3	1.00	0.06	0.14	0.37	1	0.9	3.7	1
		1.25	0.03	0.06	0.18	3	0.4	2.1	0
		1.50	0.00	0.01	0.04	7	0.2	1.2	0
		1.75	0.01	0.02	0.05	8	0.0	0.3	0
		2.00	0.02	0.03	0.08	8	0.0	0.2	0
	6	1.00	0.13	0.37	1.09	2	0.7	3.9	2
		1.25	0.08	0.24	0.76	4	0.5	3.4	1
		1.50	0.00	0.03	0.08	6	0.3	1.5	0
		1.75	0.02	0.06	0.13	9	0.0	0.4	0
		2.00	0.05	0.09	0.17	8	0.0	0.1	0
20	3	1.00	26.3	213.5	986.5	1	0.6	2.8	0
		1.25	12.0	56.2	209.8	2	0.5	3.1	1
		1.50	2.1	5.7	51.6	4	0.4	1.7	0
		1.75	2.2	8.1	43.1	5	0.4	0.9	0
		2.00	3.2	14.9	37.5	8	0.0	0.2	0
	6	1.00	137.0	801.0	>2500	3	0.4	3.2	1
		1.25	97.1	217.4	1700	3	0.4	2.9	0
		1.50	5.3	13.5	64.9	4	0.3	1.8	0
		1.75	6.9	18.7	55.2	8	0.2	0.9	0
		2.00	13.7	51.0	87.6	9	0.0	0.1	0

The last column of Table 4 reports the average computation times in milliseconds of STOCOM needed for one execution. As can be seen STOCOM does not require more than half a second of CPU – time per execution even for large problems with 200 jobs and 10 doubly – constrained resources.

Table 5 compares STOCOM with the optimization algorithm. Columns 1 to 3 describe problem characteristics. For each combination of J, K and MRS 10 test problems have been generated randomly as indicated above. Columns 4 to 6 report the minimum (MIN), the average (MEAN), and the maximum (MAX) CPU – sec required by the optimization algorithm. As can be seen the behavior of the algorithm shown earlier in Table 3 is rather typical; e.g. the computation times are increasing rapidly with decreasing resource availability.

The last four columns of Table 5 provide the number of times, in which STOCOM (using STC; 200 executions once again) determined optimum solutions (NOPT) in each problem class, the average (DMN%) and the maximum (DMX%) percentage deviation of objective function values from the optimum one as well as the number of times in which STOCOM found no feasible solution provided one could be determined with the exact algorithm (NOFEAS). (Percentage deviations have been calculated taking only those examples for which a feasible solution could be determined with STOCOM provided a finite optimum has been found with the exact approach.) STOCOM performs rather good: Within each category in at least one case the optimum has been determined. The mean (maximum) deviation does not exceed 1% (4%). Only in the case of scarce resources (MRS $\leq$ 1.25) STOCOM did not succeed in finding an existing feasible solution for some data.

Table 6 demonstrates the sensitivity of STOCOM to α. A problem with J = 20, K = 5 as well as MRS = 1.5 has been generated randomly and solved repeatedly using STC until 100 feasible solutions had been found. Using α = 2.0 the number of trials (executions of the algorithm) was 119, where 84 (feasible) solutions had a deviation from the optimum objective function value in the range [0%,2%], 11 in the range (2%,4%] and 5 in the range (4%,6%].

The results of Table 6 (which are rather typical also for other problem instances treated) indicate that with increasing α

- more trials are necessary in order to generate a prespecified number of feasible solutions and
- the percentage deviations of the objective function values from the optimum one are decreasing.

Thus we observe that the probability of finding a feasible solution is decreasing with increasing α but the quality of solutions is increasing with increasing α.

TABLE 6

Variation of the parameter α

α	#trials	percentage deviation					
		0–2	2–4	4–6	6–8	8–10	10–12
2.0	119	84	11	5	–	–	–
1.5	115	53	25	13	9	–	–
1.0	111	16	49	21	7	7	–
0.5	105	9	14	19	42	12	4

In summary of these observations a general advice for an appropriate choice of α may be given as follows: For a given data set one should start with a rather high α (say $\alpha = 2.0$), perform some hundreds of trials (depending on the problem size and on the amount of available computer resources), reduce α as indicated above and so on.

8. Summary and Conclusions

This paper has presented an assignment–type, resource–constrained, project scheduling model, which incorporates many of the features being important with respect to project management in practice. For the solution of this model, a new type of stochastic scheduling heuristic as well as a hybrid branch and bound / dynamic programming algorithm has been presented.

With respect to the computational experience, the stochastic assignment heuristic seems to be a highly suitable method for approximately solving large time–resource–cost–tradeoff problems of the type considered in this paper. The fundamental idea of the heuristic is, to resolve conflicts between jobs (which are competing for scarce resources) by Monte–Carlo methods on the basis of opportunity costs. This idea is a quite general one. It should be worthwhile to investigate its usefulness for solving scheduling problems where a job can use more than one resource [7] and for other hard combinatorial problems approximately, too.

The exact algorithm is essentially based on lower bounding procedures corresponding to several well–known (sub)problems such as the knapsack problem and the generalized assignment problem. This algorithm enables us to solve smaller problems to optimality within a reasonable amount of computation time.

Acknowledgement

The author is indebted to two anonymous referees for their helpful comments on earlier versions which improved readibility of the paper substantially.

References

[1] Blażewicz, J.; W. Cellary; R. Słowiński and J. Węglarz, *"Scheduling Under Resource Constraints – Deterministic Models"*, Basel 1986 (Annals of Operations Research, Vol. 7).

[2] Bolenz, G. and R. Frank, "Das Zuordnungsproblem von Prüfern zu Prüffeldern unter Berücksichtigung von Reihenfolgebedingungen – Ein Lösungsansatz der binären Optimierung", *Zeitschrift für betriebswirtschaftliche Forschung*, Jg. 29 (1977), pp. 427–447.

[3] Chan, K.H. and B. Dodin, "A Decision Support System for Audit–Staff Scheduling with Precedence Constraints and Due Dates", *The Accounting Review*, Vol. LXI (1986), pp. 726–733.

[4] Christofides, N; R. Alvarez–Valdes and J.M. Tamarit, "Project Scheduling with Resource Constraints: A Branch and Bound Approach", *European Journal of Operational Research*, Vol. 29 (1987), pp. 262–273.

[5] Davis, E.W. and J.H. Patterson, "A Comparison of Heuristic and Optimum Solutions in Resource–Constrained Project Scheduling", *Management Science*, Vol. 21 (1975), pp. 944–955.

[6] Drexl, A., *"Planung des Ablaufs von Unternehmensprüfungen"*, Stuttgart 1990.

[7] Drexl, A. and J. Gruenewald, "Nonpreemptive Multi – Mode Resource – Constrained Project Scheduling", in preparation.

[8] Elmaghraby, S.E., *"Activity Networks: Project Planning and Control by Network Models"*, New York 1977.

[9] Fisher, M.L.; R. Jaikumar and L.N. Van Wassenhove, "A Multiplier Adjustment Method for the Generalized Assignment Problem", *Management Science*, Vol. 32 (1986), pp. 1095–1103.

[10] Jonker, R. and A. Volgenant, "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems", *Computing*, Vol. 38 (1987), pp. 325–340.

[11] Kurtulus, I.S. and E.W. Davis, "Multi–Project Scheduling: Categorization of Heuristic Rules Performance", *Management Science*, Vol. 28 (1982), pp. 161–172.

[12] Kurtulus, I.S. and S.C. Narula, "Multi–Project Scheduling: Analysis of Project Performance", *IIE Transactions*, Vol. 17 (1985), pp. 58–66.

[13] Lawler, E.L. and D.E. Wood, "Branch – and – Bound Methods: A Survey", *Operations Research*, Vol. 14 (1966), pp. 699–719.

[14] Martello, S. and P. Toth, "An Algorithm for the Generalized Assignment Problem", in: J.P. Brans (Ed.), *Operational Research '81*, North–Holland, Amsterdam 1981, pp. 589–609.

[15] Patterson, J.; R. Słowiński; B. Talbot and J. Węglarz, "An Algorithm for a General Class of Precedence and Resource Constrained Scheduling Problems", in [21, pp. 3–28].

[16] Patterson, J.; R. Słowiński; B. Talbot and J. Węglarz, "Computational Experience with a Backtracking Algorithm for Solving a General Class of Resource Constrained Scheduling Problems", to appear in *European Journal of Operational Research.*

[17] Pritsker, A.A.B.; W.D. Watters and P.M. Wolfe, "Multiproject Scheduling With Limited Resources: A Zero–One Programming Approach", *Management Science*, Vol. 16 (1969), pp. 93–108.

[18] Rohde, M., "*Das Set–Partitioning–Problem: Wirtschaftliche Anwendungen und Algorithmen*", Diss., FU Berlin, 1978.

[19] Ross, G.T. and R.M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem", *Mathematical Programming*, Vol. 8 (1975), pp. 91–103.

[20] Russell, R.A., "A Comparison of Heuristics for Scheduling Projects with Cash Flows and Resource Restrictions", *Management Science*, Vol. 32 (1986), pp. 1291–1300.

[21] Słowiński, R., Węglarz, J. (Eds.): *"Advances in Project Scheduling"*, Amsterdam 1989.

[22] Stinson, J. P.; E. W. Davis and B. M. Khumawala, "Multiple Resource–Constrained Scheduling Using Branch and Bound", *AIIE Transactions*, Vol. 10 (1978), pp. 252–259.

[23] Talbot, F.B., "Resource – Constrained Project Scheduling With Time–Resource Tradeoffs: The Nonpreemptive Case", *Management Science*, Vol. 28 (1982), pp. 1197–1210.

[24] Talbot, F. B. and J. H. Patterson, "An Efficient Integer Programming Algorithm With Network Cuts for Solving Resource–Constrained Scheduling Problems", *Management Science*, Vol. 24 (1978), pp. 1163–1174.

[25] Wiest, J.D., "A Heuristic Model for Scheduling Large Projects With Limited Resources", *Management Science*, Vol. 13 (1967), pp. B–359 – B–377.