Open access • Journal Article • DOI:10.1287/MNSC.37.12.1590

Scheduling of Project Networks by Job Assignment — Source link

Andreas Drexl

Institutions: University of Kiel

Published on: 01 Dec 1991 - Management Science (INFORMS)

Topics: Resource allocation, Generalized assignment problem, Project management, Scheduling (computing) and Scheduling (production processes)

Drexl, Andreas

Working Paper — Digitized Version

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

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

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	$\min\{d_{jk} / k = 1,2,\ldots,K\}$
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, $j = 1,2,\ldots,J$
J	Number of jobs
(j,k)	Denotes assignment of resource k to job j
k	A specific resource, $k = 1,2,\ldots,K$
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, $t = 0,1,\ldots,T$
T	Planning horizon (deadline)
V_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)
$Z(\cdot)$	Objective function value
Z^*	Optimal objective function value
$\underline{Z}, \bar{Z}$	Lower, upper bound

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 $LF_h \le ES_j$ for all h, j with $h \in V_j$. 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:

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

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

$$SC := \{j \in S0 \,/\, ES_j \le t, V_j \in S1\}$$

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

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

and calculate

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

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. $t := 0$; $DL_k := D_k \; \forall k$; $AD_k := 0 \; \forall k$; $S1 := \emptyset$; $S0 := \{j \mid \forall j\}$.
2. Determine ES_j and LF_j by traditional critical path analysis (using δ_j).
3. $SC := \{j \in S0 \mid ES_j \leq t, V_j \in S1\}$; if $SC = \emptyset$ then goto 7; $AR := \{k \mid AD_k \leq t, DL_k > 0 \; \forall k\}$; if $AR = \emptyset$ 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. $\tau := \min\{ES_j \mid j \in S0, V_j \in S1\}$; $AD_k := \max\{0, AD_k - (\tau - t)\} \; \forall k$; $t := \tau$; 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:

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

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 "≤" 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].

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

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 $SK_\nu(m)$ $\nu=1,2,\ldots$ be sets of tupels (j,k) each such that for each pair of tupels (β,γ) and (ζ,π) $\beta\neq\zeta$ 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.

TABLE 2

An Example from Literature

d_{jk} c_{jk} j V_j

6 ϖ 4 4 280 ϖ 300 ϖ 400 380 1 ø 7 ϖ 6

? ϖ 3 ϖ 270 ϖ 270 ϖ 285 2 ø ϖ 6 ϖ 3

3 ϖ ϖ 300 360 ϖ 285 3 {1, 2} ϖ ϖ 6 4 ϖ

200 ϖ 200 ϖ ϖ ϖ 4 {1, 2} 5 ϖ 4 ϖ ϖ ϖ

ϖ 315 350 ϖ ϖ 380 5 {3, 4} ϖ 7 7 ϖ ϖ 4

315 ϖ ϖ 300 ϖ 6 ø 8 7 ϖ ϖ 3 ϖ 320

350 360 ϖ ϖ 7 ø ϖ ϖ 7 4 ϖ ϖ ϖ ϖ

ϖ 500 285 8 {6, 7} 7 7 ϖ ϖ 5 3 280 315 ϖ

270 ϖ ϖ 9 {6, 7} ϖ ϖ 5 3 ϖ ϖ ϖ ϖ 250

300 ϖ 10 {8, 9} 8 8 ϖ ϖ 3 ϖ 320 360 ϖ ϖ

190 11 ø ϖ ϖ 4 2 ϖ 2 ϖ ϖ 200 180 ϖ

12 ø 3 ϖ 3 ϖ 1 ϖ 120 ϖ 150 ϖ 100 ϖ

13 {11, 12} ϖ 5 5 3 ϖ ϖ ϖ 225 250 270 ϖ ϖ

14 ø ϖ ϖ ϖ 5 5 ϖ ϖ ϖ ϖ 450 500 ϖ

15 ø 6 ϖ 5 ϖ ϖ ϖ 240 ϖ 250 ϖ ϖ ϖ

16 {15} ϖ 4 ϖ 2 2 ϖ ϖ 180 ϖ 180 200 ϖ

17 ø 5 5 4 ϖ ϖ ϖ 200 225 200 ϖ ϖ ϖ

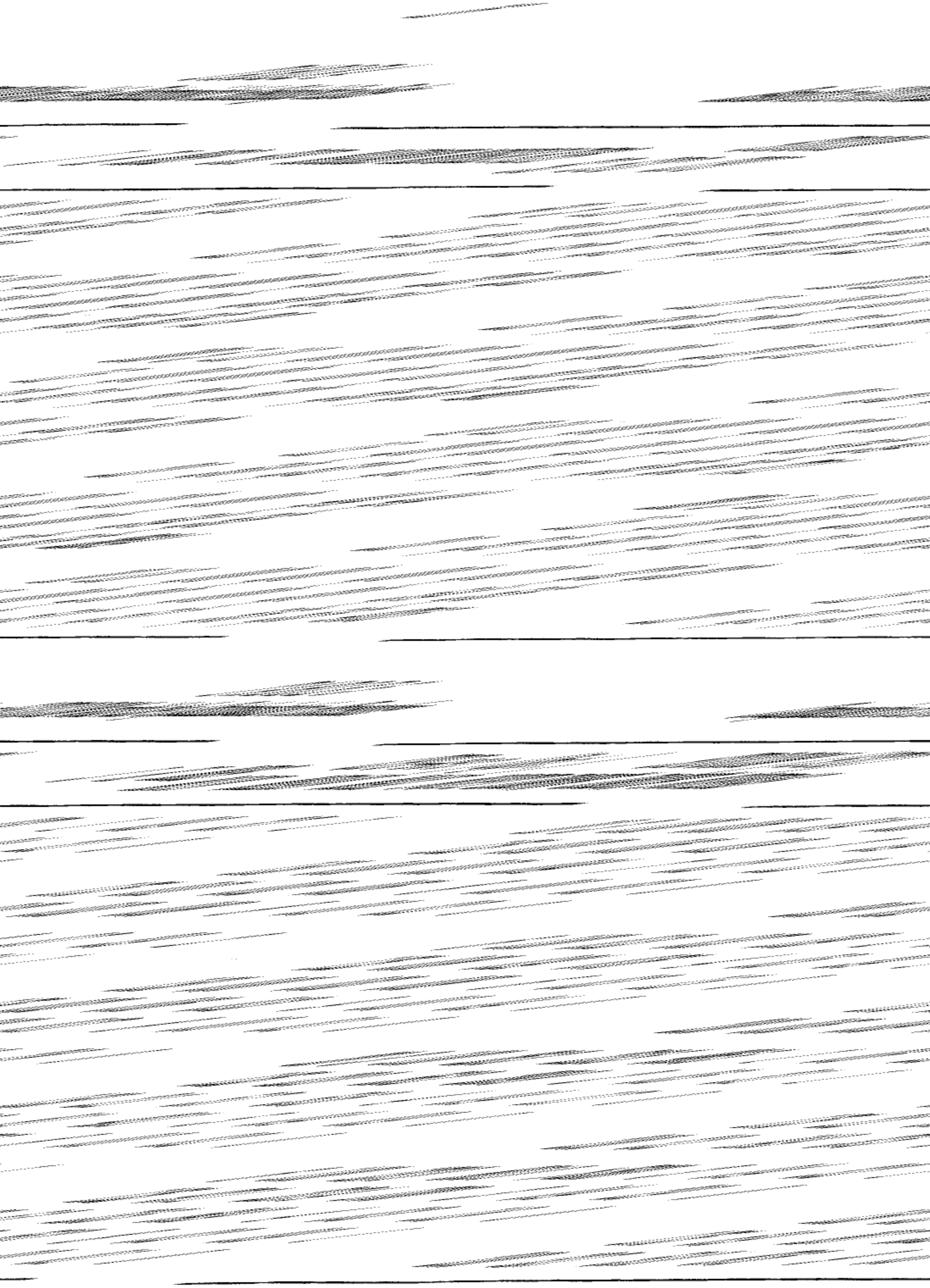
{17} ϖ ϖ 5 3 ϖ 3 ϖ ϖ 250 270 ϖ 285 18

{17} 5 5 ϖ ϖ 2 3 200 225 ϖ ϖ 200 285 19

{18,19} ϖ ϖ 4 3 ϖ ϖ ϖ ϖ 200 270 ϖ ϖ 20 {

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

[illegible]



[illegible]

[illegible]

[illegible]