KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Economics and Applied Economics

Efficient and effective solution procedures for order acceptance and capacity planning

Jade Herbots, Willy Herroelen and Roel Leus

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0720

Efficient and Effective Solution Procedures for Order Acceptance and Capacity Planning

Jade Herbots • Willy Herroelen • Roel Leus*

Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Belgium

{Jade.Herbots ; Willy.Herroelen ; Roel.Leus}@econ.kuleuven.be

This paper investigates dynamic order acceptance and capacity planning under limited regular and non-regular resources. Our goal is to maximize the profits of the accepted projects within a finite planning horizon. The way in which the projects are planned affects their payout time and, as a consequence, the reinvestment revenues as well as the available capacity for future arriving projects. In general, project proposals arise dynamically to the organization, and their actual characteristics are only revealed upon arrival. Dynamic solution approaches are therefore most likely to obtain good results. Although the problem can theoretically be solved to optimality as a stochastic dynamic program, real-life problem instances are too difficult to be solved exactly within a reasonable amount of time. Efficient and effective heuristics are thus required that supply a response without delay. For this reason, this paper considers both 'single-pass' algorithms as well as approximate dynamic-programming algorithms and investigates their suitability to solve the problem. Simulation experiments compare the performance of our procedures to a first-come, first-served policy that is commonly used in practice.

Keywords: approximate dynamic programming, order acceptance, capacity planning, simulation, multi-project.

1 Introduction

In this paper, we examine the order-acceptance and capacity-planning decision facing multi-project organizations upon project arrival. *Capacity planning* determines the allocation of the available (regular and non-regular) aggregate resources to the candidate projects, while

*Corresponding author

order acceptance is concerned with the accept/reject decision of these projects. In practice, companies tend to accept all project proposals with a positive net present value (NPV) and plan them on a first-come, first-served (FCFS) basis, without consideration of future arrivals. In this paper, we present algorithms for dynamic order acceptance and capacity planning that perform better in terms of the generated profits from the accepted projects. Our algorithms are particularly relevant for environments in which a scarce resource acts as a single static bottleneck and where at least rudimentary information about the work content of the proposed and future projects is available. Examples of such environments are MTOs (manufacture-to-order) with a single static bottleneck resource (e.g. [10]), construction environments and maintenance projects (e.g. [4]).

In our setting, we consider capacity at an aggregate level, while the regular per-period capacity is limited. Additionally, non-regular capacity units can be allocated at specific per-unit costs. Project proposals arrive one by one, and a decision on acceptance and capacity planning needs to be made without delay. At arrival, detailed problem characteristics are not yet available; nevertheless, the company has information on the main project characteristics, that is to say revenue, workload and due date. If a project is accepted, its payoff is received upon completion and from this point on reinvestment revenues are reaped. The way the projects are planned affects their payout time and as a consequence, the reinvestment revenues, as well as the available capacity for future arriving projects. We assume that the company has forecasts for the main features of the incoming projects. An extensive analysis of the underlying problem characteristics is given in [7], together with a stochastic dynamic-programming (SDP) algorithm that solves the problem to optimality. However, since SDP suffers from *the curse of dimensionality* (term suggested by Bellman [2]), heuristic methods are required that solve real-life problem instances within a reasonable amount of time.

Our work adheres to different research domains, one of which is *revenue-based capacity management*, which studies the problem of satisfying customer demand with limited resources while maximizing the company's revenue and profitability [1]. Our work is also related to *dynamic portfolio planning*, which involves the selection and prioritization of dynamically arriving projects. The existing work is relatively scarce, although there has been a growing interest in recent years ([9], [10]). Articles [1] and [6] use *simulation* to compare different order-acceptance strategies. For a more elaborate survey of the project selection literature, we refer to [7].

The remainder of this paper is structured as follows. Section 2 provides a detailed problem

statement. In Section 3 we describe our first-come, first-served policy, which accepts all projects that contribute to the company's profits on a FCFS basis. A more sophisticated algorithm, which takes information from crucial problem characteristics into account, is presented in Section 4. In Section 5, we investigate the suitability of approximate dynamic-programming (ADP) algorithms [3] to solve the problem. In Section 6, we report on the results of the simulation experiments performed to compare the performance of our solution procedures. Finally, in Section 7, we provide a summary and conclusions.

2 Problem description

We express both the project workload as well as the capacity available in the organization in discrete *capacity units* (e.g. man-hours). A single capacity unit belonging to the work content of a project is referred to as a *work package*. In our model, each project k consists of an aggregated workload, expressed as a discrete number p_k of work packages. An accepted order can only be executed between its release time r_k and the project's due date d_k, which is regarded here as a deadline. This implies that due dates cannot be exceeded and thus orders for which the due date cannot be met, must be rejected. The payoff of a work order, denoted as y_k, is generated immediately upon its completion. We assume that all these revenues can be reinvested at a fixed interest rate $i \geq 0$.

In what follows, we use the terms 'order' and 'project' to refer either to a *Request for Quotation* (RFQ) or to a request for execution of an order at a *given* price. An RFQ is an invitation for suppliers, through a bidding process, to bid on a specific product or service. Since our solution methods determine profit thresholds below which prices lead to rejection, price setting and order acceptance at a fixed price can be treated similarly. In both cases, we assume that order acceptance results in a fixed-price contract (see e.g. [8]). Our models are developed from the viewpoint of one individual bidder, and decisions are made without consideration of competitors.

The stream of incoming order arrivals is the main source of uncertainty in dynamic order acceptance. When a company has to make an accept/reject decision, it has at its disposal only rudimentary information about the project in question and forecasts of the main characteristics of the future incoming projects (e.g. based on sales-force polling). We discretize the planning horizon into T periods or time buckets (e.g. weeks or months). Additionally, we introduce the concept of a *stage*, which is the time interval between two consecutive project

arrivals: a new stage starts every time a new project arrives. The number of projects arriving sequentially within the planning horizon T (and hence, the number of stages) is N. At first, we assume N and the interarrival times to be fixed, in Section 6.2.3 we study the impact of exponential interarrival times.

The stream of arriving offers is represented as $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N$. It was explained earlier that the main characteristics of a project proposal k are its payoff y_k, a positive workload p_k and a deadline d_k. From a given positive maximum time lag l_k allowed for realizing the project, one can easily derive the deadline by adding the stage's release time r_k to the maximum time lag: $d_k = r_k + l_k$. In our model, a decision needs to be made regarding order $\mathbf{w}_1$, while estimates about the characteristics of the stream of future order arrivals are captured as follows. Each $\mathbf{w}_k = (p_k, y_k, l_k)$ is assumed to be an independent realization of a random variable $\mathbf{W}$. The amount of possible values for the project characteristics can be limited by defining a discrete number of order types. A project k belongs to an order type q, with $q \in Q$, when its characteristics have the values $\tilde{p}_q, \tilde{y}_q$ and $\tilde{l}_q$, which represent the order type's workload, payoff and maximum time lag, respectively. The probability that $\mathbf{w}_k$ $(k = 2, \ldots, N)$ will take on the value $(\tilde{p}_q, \tilde{y}_q, \tilde{l}_q)$ is given by $Pr[q]$.

At the start of each stage k, we decide upon order acceptance and capacity allocation of project k, with stage 1 being the first stage. In this paper we consider only one resource type, which is taken to represent the *bottleneck* resource of the company. We assume that the amount of *regular* capacity units is the result of a long-term strategic decision that cannot be revised within the time horizon considered in our planning framework. In contrast, the amount of *non-regular* capacity units can be altered as a result of working overtime, hiring temporary labor or outsourcing. We count the available capacity units in every time period by means of a *capacity profile*, which is a vector

$$\mathbf{x}_k = (\mathbf{m}_k, \mathbf{s}_k), \text{ with } \begin{cases} \mathbf{m}_k &= (m_{r_k+1}, \ldots, m_T) \\ \mathbf{s}_k &= (s_{r_k+1}, \ldots, s_T) \end{cases} \tag{1}$$

where k represents the stage number, m_t is the number of available regular capacity units in time period t and s_t the maximum number of non-regular capacity units that can be hired during time period t. The cost per unit of consumed non-regular capacity is c, whereas the actual utilization of regular capacity does not give rise to incremental costs. In stage k, $\mathbf{x}_k$ only reflects resource availability from time r_k, because all unused capacity units before r_k have 'perished'. We call a specific capacity unit *current* if it perishes in the following stage, otherwise it is referred to as a *future* capacity unit. In the remainder of this article, we

refer to *perishable resources* (cfr. [11]); the scheduling literature sometimes uses the term *renewable resources* (see, for instance [5]).

Upon arrival, the organization can choose whether to reject or to accept the project according to any feasible order plan. An *order plan* is an allocation of capacity units to the different work packages of a project; it is *feasible* if the total workload of the project is covered and if all work packages are planned between the stage's release time and the project's deadline. We represent an order plan j for arrival k as a vector $\mathbf{a}_k^j = (\mathbf{a}_{km}^j, \mathbf{a}_{ks}^j)$, where $\mathbf{a}_{km}^j$ and $\mathbf{a}_{ks}^j$ have the same dimension as $\mathbf{m}_k$ and $\mathbf{s}_k$, and count the number of (regular and non-regular) capacity units that are allocated to project k in each relevant time period. In case the company rejects the offer, no capacity is reserved and no further action is taken until the next project arrival. This rejection decision cannot be withdrawn. We associate a 'degenerate' order plan $\mathbf{a}_k^0 = \mathbf{0}$ (the null vector) with rejection, and we let symbol $\mathcal{A}_k$ represent the set of all feasible order plans augmented with $\mathbf{a}_k^0$.

3 First-come, first-served policy

The FCFS policy is conceived as a two-phase algorithm that determines an order plan during the first phase, while the second phase decides upon acceptance of the project. *Forward planning* is applied when building an order plan. This planning method allocates the available regular capacity units from the earliest possible time periods first, given the release time of the project. Only when these resources do not suffice, non-regular capacity is used. We demonstrate the forward planning method on capacity profile $\mathbf{x}_1$ shown in Figure 1. Since the grey units represent capacity units that were allocated earlier, $\mathbf{x}_1 = (\mathbf{m}_1, \mathbf{s}_1) = ((0,1,2,2,2),(0,1,1,1,1))$, for a problem horizon $T = 5$ periods. The construction of a 'forward' order plan for projects with workload of either two, three or four work packages and a deadline equal to time 3, proceeds as follows. In accordance with the forward planning method, regular capacity units are allocated as much as possible, so that allocating two work packages results in order plan $((0,1,1,0,0),(0,0,0,0,0))$. When one additional work package requires allocation, we plan an extra regular capacity unit from period 3, leading to order plan $((0,1,2,0,0),(0,0,0,0,0))$. Since the project deadline is 3, no additional regular capacity units can be allocated and thus when four work packages are considered, the earliest available non-regular unit is appointed $((0,1,2,0,0),(0,1,0,0,0))$. Acceptance of all project proposals with a positive contribution constitutes the accept/reject

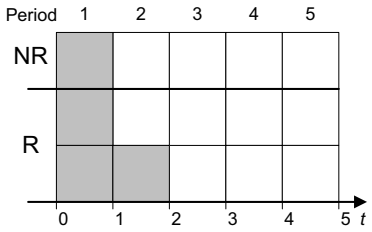


Figure 1: Capacity profile with regular (R) and non-regular (NR) capacity units.

rule for the FCFS policy. In the next section, a more involved threshold formula is devised that comprises information from crucial problem characteristics such as the free capacity and the expected future incoming workload and revenue.

4 Threshold policy

Like the FCFS policy, the threshold policy consists of both a planning and an acceptance phase. For a project k, the order plan is again the result of forward planning, but the acceptance decision is now based on a threshold value, calculated as described in pseudo-code in Algorithm 1. The algorithm builds on the intuition that a heavier load of the system allows for a more selective acceptance policy and vice versa.

Algorithm 1 allocates s^* non-regular capacity units and determines a threshold for a forward plan with realization time t^*. It initially ranks the order types q in increasing order of the revenue per work package. Next, it determines q^*, which is a boundary order type: orders with smaller per-work-package revenues are not accepted. The boundary order type is chosen by comparing a parameter δ with the occupation of the system. As a result, δ can be interpreted as the occupation up to which order types are accepted. If $\delta = \infty$, the threshold policy behaves like the FCFS policy. In the opposite case ($\delta < \infty$), the threshold value equals the number of allocated future capacity units ($p_k - nrcurrent$) of the project k, multiplied by the revenue per capacity unit of order type q^*. The value of δ is determined through simulation (this issue is further discussed in Section 6.1).

The occupation of the system is the ratio of the expected load (*expload*) from future arriving orders and the available capacity (*available*) of the company (both regular and non-regular), within the planning window of the forward plan. This time window is chosen to lie between the release time r_{k+1} of the next arrival and the realization time t^* of the forward plan, because current capacity and project arrivals beyond the completion time are less relevant for the decision-making process. Orders arrive at a rate λ per period. The expected

load is a weighted sum of the expected loads of the subset of order types with highest value of revenue per work package (via q^*). The weight $\tilde{w}$ of an order type lies between 0 and 1; it equals 1 if the deadline of the order type is smaller than the realization time of the forward plan, since in this case, the order type's workload can be planned entirely within our period of interest. On the other hand, when the order type's deadline exceeds the realization time of the forward plan, its workload is only partially added.

Algorithm 1 Threshold calculation.

Rank order types q in increasing order of $\tilde{y}_q/\tilde{p}_q$;
$nrstages = \lambda(t^* - r_{k+1})$;
$q^* = 1$;
$nothreshold = false$;
repeat
  $expload = 0$;
  Compute $available$;
  for $j = 1, \ldots, nrstages$ **do**
    for $q = q^*, \ldots, nrordertypes$ **do**
      if $r_k + j/\lambda + \tilde{l}_q > t^*$ **then**
        $\tilde{w} = \tilde{l}_q/(t^* - (r_k + j/\lambda))$;
      else
        $\tilde{w} = 1$;
      end if
      $expload = expload + \tilde{w} \cdot Pr[q] \cdot \tilde{p}_q$;
    end for
  end for
  if $q^* = 1$ and $expload/available <= \delta$ **then**
    $nothreshold = true$;
  end if
  $q^* = q^* + 1$;
until $expload/available <= \delta$ or $q^* = nrordertypes + 1$
$q^* = q^* - 1$;
if $nothreshold = false$ **then**
  $threshold = (p_k - nrcurrent) \cdot (\tilde{y}_{q^*}/\tilde{p}_{q^*})$;
end if
if $threshold < cs^*$ **then**
  $threshold = cs^*$;
end if

5 Approximate dynamic programming

In Section 5.1 we show how the problem discussed in this paper can theoretically be solved to optimality by a stochastic dynamic program (SDP). The dynamic-programming approach has the advantage that it can easily deal with many different problem characteristics (varying due dates, hiring non-regular capacity units,...). A downside to the high flexibility is that our SDP, as is typical for dynamic programming, also suffers from *the curse of dimensionality*. We counterpart the high computational complexity by developing an approximate dynamic program (ADP) in Section 5.2. The ADP model we apply is a rollout algorithm that employs our threshold policy as initial policy.

5.1 Stochastic dynamic programming

We first describe the SDP model proposed in [7]. The reward in stage k for order plan $\mathbf{a}_k^j$ and offer $\mathbf{w}_k$ is determined as:

$$g_k(\mathbf{a}_k^j, \mathbf{w}_k) = \begin{cases} 0 & \text{if } j = 0, \\ y_k(1+i)^{(T-t_j^*)} - cs_j^* & \text{otherwise,} \end{cases} \tag{2}$$

where s_j^* equals the number of allocated non-regular capacity units in $\mathbf{a}_k^j$. t_j^* refers to the realization time (and hence, the pay-out time) of an accepted project which is planned according to $\mathbf{a}_k^j$. As a result, $T - t_j^*$ is the period for which the company receives reinvestment revenues.

The backward SDP algorithm consists of iteratively solving the following recursion:

$$\begin{cases} f_N(\mathbf{x}_N) &= \max\limits_{\mathbf{a}_N^j \in \mathcal{A}_N} \{g_N(\mathbf{a}_N^j, \mathbf{w}_N)\} \\ f_k(\mathbf{x}_k) &= \max\limits_{\mathbf{a}_k^j \in \mathcal{A}_k} \{g_k(\mathbf{a}_k^j, \mathbf{w}_k) + E[f_{k+1}(\mathbf{x}_{k+1})]\} \quad \text{if } k \neq N, \end{cases} \tag{3}$$

with

$$\mathbf{x}_{k+1} = v(\mathbf{x}_k - \mathbf{a}_k^j) \tag{4}$$

and $E[\cdot]$ the expectation operator. The perishing function v transforms a vector $\mathbf{x}_k$ into a vector $\mathbf{x}_{k+1}$ from which the perished capacity units are removed; $\mathbf{x}_{k+1}$ represents the state or capacity profile after implementing order plan $\mathbf{a}_k^j$. If $j = 0$, $\mathbf{x}_{k+1}$ equals $v(\mathbf{x}_k)$.

In Eq. (3), $f_k(\mathbf{x}_k)$ is the maximum expected reward that can be earned during stages $k, k+1, \ldots, N$ given that the initial state corresponds with $\mathbf{x}_k$. This recursion captures the essential idea of dynamic programming, namely to split the evaluation of every decision in

two parts: (1) the immediate reward and (2) the expected future rewards reachable from this action. These future rewards are captured by the *continuation value* $f_{k+1}(\mathbf{x}_{k+1})$, also referred to as the *reward-to-go*, or *cost-to-go* in the context of minimization problems.

5.2 Rollout algorithm

Within dynamic programming, a *policy* is the rule by which we select the next action. Through a so-called *policy improvement step*, an initial policy μ can be upgraded to policy $\bar{\mu}$. To this aim, we consider every possible control $\mathbf{a}_k^j \in \mathcal{A}_k$ and determine the next action, given the initial state $\mathbf{x}_k$, based on a *greedy policy*:

$$\begin{aligned} \bar{\mathbf{a}}_k^j &= \arg\max_{\mathbf{a}_k^j \in \mathcal{A}_k} \{g_k(\mathbf{a}_k^j, \mathbf{w}_k) + E[\tilde{f}_{k+1}^{\mu}(\mathbf{x}_{k+1})]\} \quad \text{if } k \neq N, \\ \bar{\mathbf{a}}_N^j &= \arg\max_{\mathbf{a}_N^j \in \mathcal{A}_N} \{g_N(\mathbf{a}_N^j, \mathbf{w}_N)\}, \end{aligned} \tag{5}$$

with $\tilde{f}_{k+1}^{\mu}$ the *approximate reward-to-go* under policy μ. This entails that the expected future rewards are approximated using a suboptimal policy (in this paper, the FCFS policy or the threshold policy) and calculated either analytically or, in our case, by Monte-Carlo simulation. From many statistically independent sample-state trajectories, $E[\tilde{f}_{k+1}^{\mu}]$ is determined as the average of the rewards over the different simulation runs. The approximate-DP technique is based on the hypothesis that if $\tilde{f}_{k+1}^{\mu}$ is a good approximation of the real value of f_{k+1}, then the greedy policy in Eq. (5) is close to optimal. Under these circumstances, $\bar{\mathbf{a}}_k^j$ approximates the optimal action, which is in most cases computationally overly demanding to calculate. The resulting heuristic is referred to as a rollout (RO) algorithm. In this paper we look at a *single stage* or *one-step lookahead* RO algorithm, since at a given state $\mathbf{x}_k$ ($k \neq N$), we find the optimal decision for a one-stage problem with one-stage reward $g_k(\mathbf{a}_k^j, \mathbf{w}_k)$ and termination payoff $E[\tilde{f}_{k+1}^{\mu}(\mathbf{x}_{k+1})]$.

The viability of a rollout algorithm highly depends on the available computation time for decision-making and thus, calculating $\bar{\mathbf{a}}_k^j$. For the algorithm to be applicable in practice, it is required that the Monte-Carlo simulations and the calculation of $\bar{\mathbf{a}}_k^j$ comply with the real-time constraints of the problem. Replacing the reward-to-go with an approximate value strongly reduces its solution time. Nevertheless, the number of possible actions may be so large that even the rollout algorithm requires too much CPU time.

This paper considers two techniques to speed up the calculation. The first method consists of lowering the number of runs and reducing the length of the Monte-Carlo simulation

that is used to determine the expected continuation value. We will further discuss this issue in Section 6.2.2, which deals with the technical details of our computational experiments. The second speed-up technique replaces $\mathcal{A}_k$ in Eq. (5) with a subset $\tilde{\mathcal{A}}_k$. Obviously, this may have a negative effect on the solution quality. A smart selection of feasible order plans is thus required.

In the remainder of this section, we examine which order plans, apart from the forward plan, are most likely to improve profits. We consider two cases; in the first one, maximum time lags are shortened (Section 5.2.1) and in the second case, reinvestment revenues can be reaped (Section 5.2.2).

Because of their specific structure, rollout algorithms are particularly suitable for solving hard trade-off problems, such as the trade-off between incurring higher non-regular capacity costs and losing reinvestment revenues due to longer execution times. These trade-offs need not necessarily occur between order plans of the same project. We could, in a similar way, examine the trade-off between two simultaneously arriving projects, or even consider accepting both projects together, based on a rollout algorithm. Obviously, adding order plans to the decision set results in an augmentation of the execution time. Since a considerable number of simulation runs is required to obtain an accurate estimation of the reward-to-go, the computation time mounts strongly with the number of considered order plans.

5.2.1 Shortened maximum time lags

When the maximum time lags of the incoming projects are shortened, the planning flexibility is reduced and as a result, the rate of *implicit selection* is increased. By implicit selection we refer to the process were one or more project types become unplannable in practice, because of the specific structure of the problem. So, shortening the maximum time lags may increase the rate of implicit selection, which may have a negative effect on the expected profits. In some cases, this process can be counteracted by extending the decision set with additional order plans.

As an example, we present a case with *rush orders*; these are order types with higher payoffs in combination with smaller time lags than other order types. Since a rush order has a relatively short maximum time lag, it often becomes unplannable when we apply forward planning. To overcome these planning difficulties, we insert alternative order plans into the action set $\tilde{\mathcal{A}}_k$. When a non-rush-order proposal arrives, we *virtually plan* a rush order before we schedule the non-rush order forwardly, so that capacity is available if a rush order should

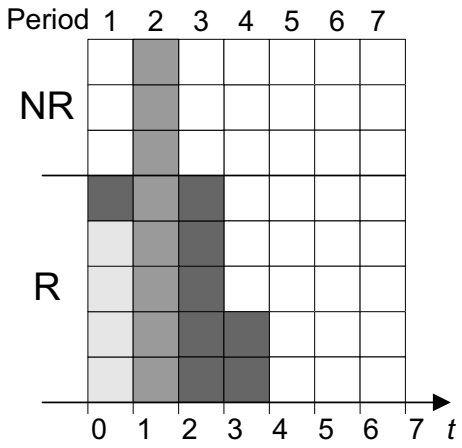


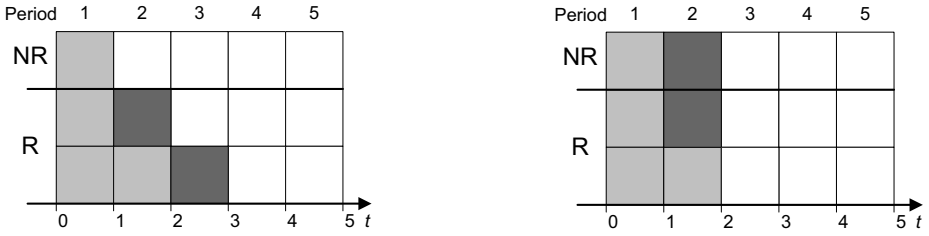
Figure 2: Capacity profile with regular (R) and non-regular (NR) capacity units, showing allocated units (light grey), a virtual plan (medium grey) and an alternative order plan (dark grey).

arise during the following stages. An alternative order plan should be built for every rush order type in Q. Virtual planning allocates the workload *backwardly*, so starting with the latest possible capacity units; first to the non-regular units and subsequently to the regular capacity. The rollout algorithm implicitly trades off the increased risk of perished capacity, due to the deviation from the early plan, against the risk of having to reject projects because of insufficient available resources.

As an illustration, we consider an incoming arrival stream consisting of two order types A and B. Order type A has a probability of arrival equal to 0.7, a workload of 8, a revenue of 16 and a maximum time lag of 4. The rush order type B has the same workload but a higher payoff ($y_B = 24$), and a shorter maximum time lag equal to 1. Figure 2 visualizes capacity profile $\mathbf{x}_1 = ((1,5,5,5,5,5,5),(3,3,3,3,3,3,3))$, the four previously allocated regular capacity units are indicated in light grey. The forward plan for project proposal $\mathbf{w}_1$ of type A is $\mathbf{a}_1^1 = ((1,5,2,0,0,0,0),(0,0,0,0,0,0,0))$. To construct the alternative order plan, we first build a virtual plan (in medium grey) for order type B. Since the release time r_2 of our next arrival equals 1 and order type B possesses a maximum time lag $l_B = 1$, the time window for the virtual plan equals period 2. The alternative order plan $\mathbf{a}_1^2 = ((1,0,5,2,0,0,0),(0,0,0,0,0,0,0))$ is now planned on the capacity units in dark grey.

5.2.2 Reinvestment revenues

When the interest rate is larger than 0, shorter project realization times create additional reinvestment revenues. When these benefits exceed the costs of extra non-regular capacity units, the company's profits can be increased by replacing regular capacity units with non-regular capacity units. This can be incorporated in our rollout algorithm by extending the decision set with an alternative order plan for which the workload is relocated from the regular capacity units in the period preceding the realization time to earlier non-regular capacity units. Following this logic, Figure 3(b) shows the alternative order plan $\mathbf{a}_1^2 = ((0,1,0,0,0),(0,1,0,0,0))$ for the forward plan $\mathbf{a}_1^1 = ((0,1,1,0,0),(0,0,0,0,0))$ in Figure 3(a).



(a) $\mathbf{a}_1^1$ (b) $\mathbf{a}_1^2$

Figure 3: Capacity profile with regular (R) and non-regular (NR) capacity units, showing allocated units (light grey) and an order plan (dark grey).

6 Computational experiments

The performance of the presented algorithms is evaluated through simulation experiments. Section 6.1 discusses the experimental setup, the results are analyzed in Section 6.2.

6.1 Experimental setup

We compare our algorithms based on the average profits from independent simulation runs; the performance of these algorithms is affected by several factors. The impact of the factors in a practical setting, as well as their values during the computational experiments, are discussed here. Clearly, the usefulness of project selection strategies highly depends on the relation between the incoming workload and the capacity of the company. In case of excess capacity, for instance, there is little need for project selection. This is captured by the

first factor, the *estimated system occupation* ρ, which is the ratio of the expected arriving workload over the available capacity (during the length of the simulation run); that is

$$\rho = \sum_{q \in Q} \frac{Pr[q] \cdot \tilde{p}_q \cdot \lambda}{M + S}, \tag{6}$$

with λ the arrival rate of incoming projects, and M and S the number of regular and non-regular capacity units available within the entire simulation, respectively. Since project selection becomes relevant only for over-demanded organizations (i.e., companies that have insufficient capacity to execute all proposed projects), we can limit our analysis to values of ρ equal to or larger than one. The value of the the estimated system occupation is varied from a low of 1 to a high of 3, $\rho \in \{1, 2, 3\}$.

The *arrival rate* λ equals the number of project arrivals within a period. For now, the interarrival times are assumed to be deterministic. In Section 6.2.3, we abandon this restriction and study the impact of exponential interarrival times on the performance of our algorithms. λ is set to 0.5, 1 and 2, denoting low, medium and high arrival rates, which offers us a considerable range for testing the algorithms. Obviously, the magnitude of λ must be interpreted in combination with the length of a period.

Next, we focus on the diversity of the order types that constitute the arrival stream. The *standard deviation σ of the gain per work package* $(\tilde{y}_q/\tilde{p}_q)$ of every order type q in Q, serves to measure the effect of diversity on the performance of the algorithms. We consider three values for σ, low (0.7), moderate (1.3) and high (2.1), constructed as follows. We allow for four different gain-per-work-package values, namely 1, 2, 3 and 4, which respectively characterize the order types 1, 2, 3 and 4. When order types 2 and 3 constitute the arrival stream and have equal arrival probabilities (0.5), the resulting standard deviation is low (0.7), since the order types are very similar in terms of gain per workload. In a second setting, all four existing order types make up the arrival stream with equal probability (0.25), so that the standard deviation equals 1.3. The highest value for the standard deviation, 2.1, is attained when order type 1 and 4 arrive with probability 0.5; in this case, the arrival stream is maximally diversified. The construction of the arrival streams is such that the weighted average gain of a work package equals 2.5.

To verify how our algorithms behave in a broad variety of environments, we have determined three settings for *workload distribution*, referred to as '*p*-dis.', with values A, B and C. In each setting, the workload is spread differently over the order types. For workload distribution A, the workload is equally distributed over the incoming projects, hence $\tilde{p}_q$ is

the same for every $q \in Q$. In this setting, the total workload from attractive and less attractive arriving projects is the same. Secondly, for workload distribution B the workload is negatively correlated with the gain-to-workload ratio. Here, the projects with the highest reward per workload possess the smallest workload. Since most of the incoming workload stems from unattractive projects, the company is often obliged to accept inferior projects to fill up the capacity. Finally, for workload distribution C, the workload is determined proportionally to the gain per workload. This entails that the most attractive projects are also the largest projects. From the company's point of view, this setting appears to be the most attractive, since small, inferior projects can often be refused with a low risk of wasted capacity.

The cost of one unit of non-regular capacity, c, is set to either a moderate value (1) or to a high value (2.5), where the latter equals the weighted average gain of a work package. In general, we look at systems with a high planning flexibility, which translates into a ratio of the workload to the maximum time lag equal to one. In Section 5.2.1, we discussed the case where the planning flexibility was reduced. An overview of the factors varied during the simulation experiments is given in Table 1.

Table 1: Factors of the simulation experiments.

factor	name	values
ρ	estimated system occupation	1, 2, 3
λ	arrival rate	0.5, 1, 2
σ	standard deviation of the gain per work package	0.7, 1.3, 2.1
p-dis.	workload distribution	A, B, C
c	cost of non-regular capacity	1, 2.5

We apply a simulation length of 1000 periods to ensure convergence of the results. The capacity profile contains eight available capacity units in every period, five of which are regular and three non-regular. Our computational experiments are performed on a computer with a 1GHz Pentium III processor, the algorithms are coded in Microsoft Visual C++. We have removed the capacity in the first ten periods, for otherwise the system would most likely accept all early arrivals due to the large amount of current capacity. This would create a bias from the first incoming arrivals. The parameter δ, required to calculate the threshold value in Algorithm 1, is determined for each factor setting separately through simulation with a known seed. A different seed value is used to obtain our simulation results. The number of simulation replications is equal to 250, which ensures that the variety of scenarios

is sufficiently large. Our statistics issue from one-tailed paired tests, using simulation results from runs with the same seed; the significance level is 1%.

6.2 Experimental results

This section reports the experimental results of the algorithms presented in this paper. First, we compare our initial policies, namely the threshold policy and the FCFS policy. Subsequently, the different rollout algorithms are evaluated and finally, we investigate the suitability of the presented algorithms when dealing with exponential interarrival times.

6.2.1 Performance of the initial policies

The results in Table 2 show the average percentage increase in the objective function (the maximum reward obtained over all project arrivals) when changing from the threshold policy to the FCFS policy for the test sets with the indicated factor values. The asterisks indicate results where no statistically significant (at the 1%-level) difference between the algorithms could be identified. A beneficial effect of the threshold policy is observed for all tested cases.

Clearly, the potential gains from the threshold policy tend to vary with the structure of the problem. When we compare the different workload distributions A, B and C, the smallest improvements are realized for workload distribution C; here, the projects with the highest gain per workload also have the highest total workload. This eventuates in fairly good results for the FCFS policy, due to the fact that when some large projects are accepted, little room is left for small, less attractive projects. This situation, in which unattractive projects cannot be fitted into the capacity profile, as a result of the composition of the arrival stream, will be referred to as *self-selection*. Its effect increases for higher costs of non-regular capacity, since under these circumstances, low-payoff offers can no longer be allocated to non-regular capacity. In practice, these offers will only be accepted in case of excess capacity. When the workload is spread equally, as for workload distribution A, or disproportionally to the gain per work package, as for workload distribution B, self-selection no longer takes place, so that the performance of the FCFS policy worsens and as a result, larger percentage improvements are achieved by the threshold policy.

Another striking trend is the large improvement increase between $\rho = 1$ and $\rho = 2$. The augmentation from $\rho = 2$ to $\rho = 3$ is less prominent. Apparently, an estimated system occupation of 1 does not allow for strong selectivity. However, once a certain level of expected

Table 2: Average percentage increase in the objective function when changing from the FCFS policy to the threshold policy, with $c = 1$ and $c = 2.5$.

			Average percentage increase $c = 1$			Average percentage increase $c = 2.5$		
ρ	σ	p-dis.	$\lambda = 0.5$	$\lambda = 1$	$\lambda = 2$	$\lambda = 0.5$	$\lambda = 1$	$\lambda = 2$
1	0.7	A	0.06	0.06	0.03	7.46	3.90	11.88
		B	1.10	1.37	1.71	9.81	8.72	6.99
		C	0.00*	0.00*	0.00*	0.12	0.08	0.08
	1.3	A	0.05	0.07	0.01	24.98	24.05	11.67
		B	7.21	6.01	2.89	27.86	30.44	31.43
		C	0.00*	0.00*	0.00*	0.24	0.13	0.00*
	2.1	A	0.00*	0.31	0.25	0.00*	28.48	12.7
		B	8.67	8.65	0.25	34.34	3.31	36.55
		C	0.02	0.00*	0.00*	0.44	0.29	0.20
2	0.7	A	18.95	19.63	19.59	29.96	29.99	28.59
		B	18.78	15.20	22.14	53.12	41.13	58.21
		C	0.14	0.04	0.05	0.20	0.03	0.00*
	1.3	A	39.76	40.70	39.44	57.71	59.84	58.79
		B	89.63	92.04	95.13	140.71	126.46	102.47
		C	5.46	5.65	6.08	8.17	7.94	8.15
	2.1	A	68.83	61.29	61.13	46.84	88.82	89.37
		B	69.13	213.31	210.01	105.32	188.40	124.03
		C	25.34	0.03	0.00*	14.31	0.07	0.00*
3	0.7	A	22.67	23.23	23.50	32.13	31.96	32.12
		B	33.06	30.15	30.51	78.43	80.69	81.18
		C	0.00*	0.11	0.01	0.30	0.16	0.00*
	1.3	A	55.09	55.09	55.71	76.10	76.52	53.11
		B	121.33	122.01	124.63	178.83	185.25	173.86
		C	11.14	12.21	13.19	16.20	16.88	16.88
	2.1	A	70.93	70.12	70.67	92.54	93.73	34.24
		B	332.73	350.81	357.57	250.86	263.67	212.23
		C	0.44	0.24	0.00*	0.67	0.27	0.00*

system occupation is obtained, a further load increase generates diminishing marginal returns. Exceptions to this general trend are found for workload distribution C in combination with a maximum standard deviation and an arrival rate of 0.5. Here, the improvements for $\rho = 2$ are 25.34% and 14.31% for $c = 1$ and $c = 2.5$, respectively; these values are much larger than 0.44% and 0.67% for the case where $\rho = 3$. Although the absolute values for the two policies for $\rho = 3$ are larger than for $\rho = 2$, their relative differences diminish. The reason can be found in the specific structure of the particular factor values. For the three values of the expected system load, self-selection takes place; as a result, the FCFS policy selects more or less the same projects as the threshold policy. For $\rho = 2$, however, the threshold policy allocates fewer non-regular capacity units, resulting in a higher percentage increase. For the heavily loaded system ($\rho = 3$), the regular capacity is totally exhausted, so that it is not possible to diminish the allocation of non-regular capacity. Other deviations from the general trend are related in a similar way to the specific problem structure.

Finally, we note that for the fully ($\rho = 2$ and $\rho = 3$) loaded systems, the average percentage increase mounts with σ. We again note that for workload distribution C and the maximum standard deviation, the improvement from the threshold policy is rather moderate. As we already pointed out, this is a result of the composition of the arrival stream. In general, a larger diversification of the order types will augment the benefits from project selection, in case no self-selection takes place.

For the tested cases, no linear relations between the profits and the model factors could be revealed. This is due to the high impact of the specific problem structure on the algorithmic performance. Nevertheless, it was established that the impact of our threshold heuristic was statistically significant in combination with the forward planning approach. Since the specific problem structure has a large impact on the algorithmic performance, it is possible that other planning methods may be more appropriate in other environments. For this reason, we investigate the performance of rollout algorithms in the next section.

6.2.2 Performance of the rollout algorithms

We previously hinted at the possibility of speeding up the rollout algorithm by lowering the number of runs and reducing the length of the Monte-Carlo simulation that estimates the expected reward-to-go (these are not the simulation parameters discussed in Section 6.1). We now study the effect of these parameters on the algorithmic performance for a particular choice of factor values (from the set discussed in Section 6.1) with a low speed of convergence.

Therefore, we investigate the lowest arrival rate (0.5) and thus the smallest number of arrivals within the length of the simulation. Additionally, we choose $\rho = 2, \sigma = 1.3, c = 1$ and p-dis. B, since this leads to a high variance in the simulation results. Figure 4 shows how the average total profits evolve when the simulation parameters of the continuation value are altered. The abscissa represents the horizon length, which is the maximum number of periods over which the expected reward-to-go is simulated (we never consider a horizon that exceeds the length of the original simulation run, i.e., 1000). The three lines in the first and second graph represent respectively, the average CPU time per simulation and the average profits when the total number of simulation runs (Sim) is varied (Sim $\in \{10, 25, 50\}$). The graphs demonstrate that a Sim-value of 10 and a horizon length below 200 are too low to obtain convergence. By keeping Sim high (equal to 50), we are able to shorten the horizon length to 200 and considerably lower the CPU time, without significant quality loss in the output. As a result, these values are used when applying the rollout algorithm.

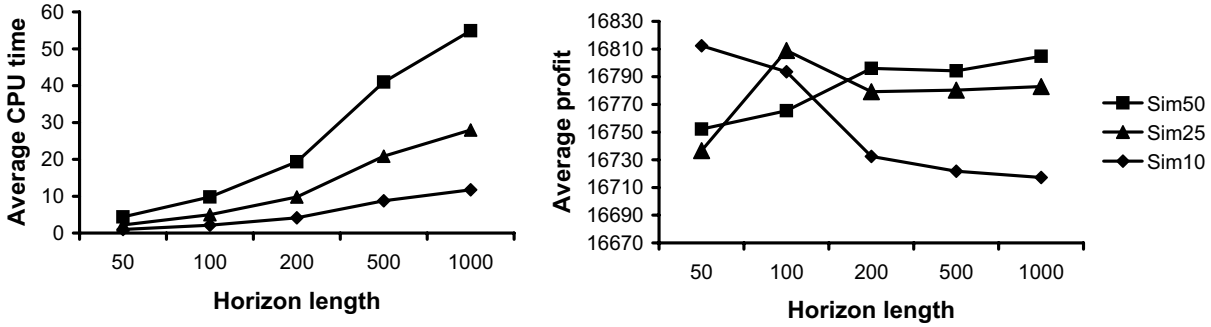


Figure 4: Average profits and CPU times for varying horizon lengths and number of simulations.

In Section 5.2, we mentioned two cases for which the application of a rollout algorithm seemed very promising. We are now able to quantify the benefits of the suggested rollout algorithms. For the rush-order case presented in Section 5.2.1, we compare the results of the rollout algorithm to the threshold policy, when the estimated system occupation is 1 and the arrival rate 0.5. For this special case, the rollout algorithm obtained an average profit of 11038 over 250 simulations while applying the alternative order plan in on average 85% of the cases in which it was generated. The result of the threshold policy was 10737.8; a one-tailed paired tests, using simulation results from runs with the same seed points out that this difference is highly significant (at a 1% significance level).

As for the case of reinvestment revenues (discussed in Section 5.2.2), the results are

gathered in Table 3 for an interest rate equal to 0.01. The numbers in the first column show the average percentage improvement in the objective function (the maximum reward obtained over all project arrivals) when changing from the threshold policy to the described rollout algorithm. The asterisks indicate results where no statistically significant (at the 1%-level) difference between the algorithms could be identified. The second column contains information on the order-plan choice; it displays the number of times the alternative order plan was applied; the number of accepted orders for which an alternative order plan was generated is shown between parentheses. In the third column, we find the average CPU time per simulation, expressed in seconds. Since the computation times for the threshold policy are negligible, we have omitted them. Although the execution times of the rollout algorithm are much higher, the decision time for acceptance and planning of a single arrival remains low enough for the algorithm to be practically applicable. When simulating the results in Table 3, the rewards of the accepted projects were adapted to $(y_k - cs_j^*)(1+i)^{(T-t_j^*)}$, which differs slightly from the reward function in Eq. (2). This change is necessary since the original reward function is intended to be used for modest problem horizons, common for tactical decisions. When the length of the simulation period is as high as required for our simulations, the cost of the non-regular capacity becomes negligible compared to the amount of the reinvestment revenues.

The results in Table 3 show that the rollout algorithm is able to achieve significant performance improvements. For three of the factor settings, the threshold policy obtained better average profits. We conjecture that, for these cases, the simulations of the expected reward-to-go were not accurate enough to make the best decisions.

6.2.3 Exponential interarrival times

We investigate the performance of our algorithms when interarrival times are no longer deterministic but exponential. In this case, the original SDP becomes a difficult imperfect state information problem [3], for which the number of arrivals and their arrival times are not known in advance. Nevertheless, our algorithms remain applicable. The FCFS policy and the threshold policy are kept unchanged, while the orders in the simulation now arrive with exponential interarrival times determined by λ, the arrival rate. Analogously, the expected reward-to-go used in the rollout algorithms is also the result of a simulation with exponential interarrival times. Figure 5 compares the average percentage increase in the objective function (the maximum reward obtained over all project arrivals) when changing

Table 3: Comparison of the rollout algorithm and threshold policy for $c = 1$ and $i = 0.01$.

			$\lambda = 0.5$			$\lambda = 1$		
ρ	σ	p-dis.	%-imp.	order plan	CPU(s)	%-imp.	order plan	CPU(s)
1	0.7	A	0.00*	0(493)	10.50	0.00*	3(3)	24.39
		B	5.14	86(256)	9.15	1.74	112(509)	31.32
		C	3.53	150(283)	8.30	3.47	190(539)	27.65
	1.3	A	9.89	1(493)	18.29	0.59	4(988)	42.15
		B	7.74	77(256)	11.48	6.62	137(515)	39.73
		C	0.09*	181(284)	8.13	-0.45	206(556)	29.10
	2.1	A	0.06	63(248)	2.80	0.78	6(8)	24.25
		B	2.81	50(244)	9.09	0.33	105(496)	31.28
		C	-1.35	160(267)	6.74	-3.48	125(538)	20.29
2	0.7	A	1.26	5(246)	21.74	0.44	7(415)	77.02
		B	4.55	1(115)	14.25	13.52	76(423)	52.83
		C	1.98	11(145)	4.44	0.52	4(205)	15.10
	1.3	A	1.04	6(203)	35.83	1.00	12(418)	147.67
		B	1.84	75(224)	17.41	4.13	128(368)	100.20
		C	3.50	28(251)	13.87	2.24	10(299)	53.53
	2.1	A	1.22	21(208)	22.51	0.86	18(415)	97.07
		B	1.58	12(146)	16.10	9.85	15(261)	41.87
		C	1.12	15(146)	4.85	0.77	9(234)	15.25
3	0.7	A	1.74	10(164)	17.23	0.56	3(205)	55.80
		B	2.56	8(118)	27.39	11.00	8(190)	76.13
		C	5.25	13(94)	3.18	0.52	17(121)	9.43
	1.3	A	2.11	13(151)	51.40	2.06	9(301)	222.71
		B	9.38	22(165)	45.26	11.79	21(299)	142.43
		C	9.72	19(102)	15.55	3.75	37(197)	95.96
	2.1	A	1.06	14(144)	16.97	0.78	9(232)	58.38
		B	3.61	21(188)	32.51	4.01	18(365)	92.49
		C	0.95	13(95)	3.15	0.37	19(147)	9.97

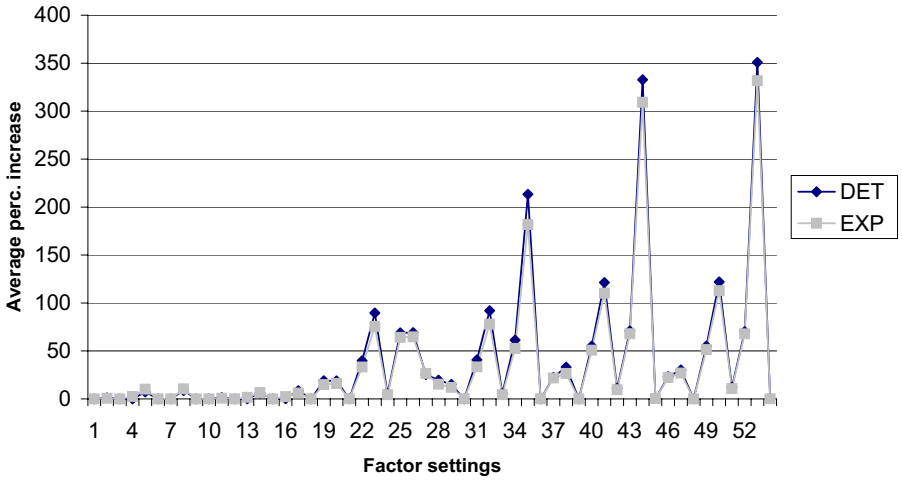


Figure 5: Average percentage increase in the objective function when changing from the FCFS policy to the threshold policy for deterministic (DET) and exponential (EXP) interarrival times for $c = 1$.

from the FCFS policy to the threshold policy for deterministic (DET) and exponential (EXP) interarrival times for $c = 1$. Figure 6 compares the average percentage increase in the objective function when changing from the threshold policy to the rollout algorithm for reinvestment revenues for deterministic (DET) and exponential (EXP) interarrival times for $c = 1$ and $i = 0.01$. The vertical axes list the different factor settings, while the horizontal axes show the average percentage increase. The latter are ranked as in Table 3: the first nine results are for $\rho = 1$ and $\lambda = 0.5$, followed by the results for $\rho = 1$ and $\lambda = 1$ and so on.

The results in Figure 5 reveal that the average percentage increase when changing from the FCFS policy to the threshold policy for the deterministic and the exponential arrival times follows the same trend for each factor setting: the correlation coefficient between both data streams is 0.999. For a high estimated system occupation ($\rho = 2$ or $\rho = 3$), the threshold policy results in a higher percentage increase in case of deterministic interarrival times.

The results in Figure 6 show that the average percentage increase when changing from the threshold policy to the rollout algorithm for reinvestment revenues for the deterministic and the exponential arrival times behaves similarly as in Figure 5: the correlation coefficient is equal to 0.813.

These results clearly demonstrate that the performance of both the threshold policy and the rollout algorithm is robust enough to be applied in environments with exponential interarrival times.

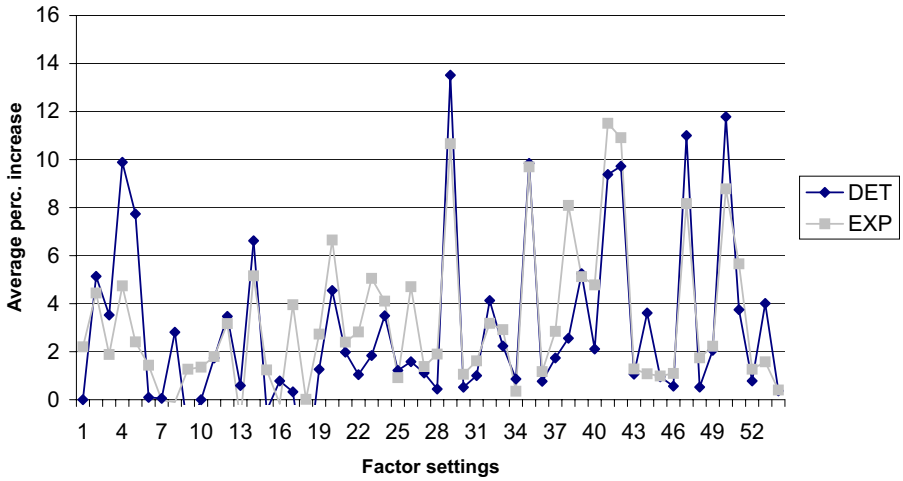


Figure 6: Average percentage increase in the objective function when changing from the threshold policy to the rollout algorithm for reinvestment revenues for deterministic (DET) and exponential (EXP) interarrival times for $c = 1$ and $i = 0.01$.

7 Conclusions

In this paper, we have investigated order acceptance and capacity planning in an over-demanded multi-project organization that aims to maximize its profits. We have stressed the importance of integrating order acceptance and capacity planning in order to be able to live up to competitive deadlines and reduce the sometimes excessive use of highly expensive non-regular capacity. Although the problem can theoretically be solved to optimality through a stochastic dynamic program, real-life problem instances are too difficult to be solved exactly within a reasonable amount of time. For this reason, this paper investigates heuristic algorithms that efficiently and effectively solve the problem and supply a response without delay. We have first presented a FCFS policy, which accepts all projects with a positive contribution on a FCFS basis. Next, we have considered a threshold policy that applies forward planning in combination with a threshold heuristic to solve the acceptance and planning decision. Subsequently, we have developed several approximate dynamic-programming algorithms, more specifically a number of rollout algorithms. These powerful algorithms provide us with a versatile tool to make decisions. Our rollout algorithms are able to integrate complicated trade-offs in the decision process, both between order plans of one project as well as between order plans of different projects. Simulation experiments were used to compare the performance of the threshold policy and the rollout algorithms to the FCFS policy.

We have established that without reinvestment revenues and with sufficient planning flexibility, the threshold policy performs very well. When the planning flexibility is reduced, ad hoc rollout algorithms will most likely do better. We have described such an algorithm for the case where companies are confronted with rush orders. In this case, the rollout algorithm is able to trade off the benefits from planning short-lagged, high-profit projects against the risk of perished capacity units. In the case of reinvestment revenues, a significant profit improvement can be obtained by applying a rollout algorithm that trades off the benefits of additional reinvestment revenues against the cost of extra non-regular capacity. Finally, we have demonstrated the suitability of our algorithms for environments where project offers arrive with exponential interarrival times.

References

[1] C. Akkan. Finite-capacity scheduling-based planning for revenue-based capacity management. *European Journal of Operational Research*, 100:170–179, 1997.

[2] R. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NJ, 1957.

[3] D.P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific, 2005.

[4] R. De Boer. *Resource-constrained multi-project management - A hierarchical decision support system*. PhD thesis, University of Twente, Enschede, the Netherlands, 1998.

[5] E.L. Demeulemeester and W.S. Herroelen. *Project scheduling - A research handbook*. Kluwer Academic Publishers, Dordrecht, 2002.

[6] M.J. Ebben, E.W. Hans, and F.M. Olde Weghuis. Workload based order acceptance in job shop environments. *OR Spektrum*, 27:107–122, 2005.

[7] J. Herbots, W. Herroelen, and R. Leus. Dynamic order acceptance and capacity planning within a multi-project environment. Technical Report KBI 0614, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Belgium, 2006.

[8] W. Herroelen, P. Van Dommelen, and E. Demeulemeester. Project network models with discounted cash flows: A guided tour through recent developments. *European Journal of Operational Research*, 100:97, 1997.

[9] C.H. Loch and S. Kavadias. Dynamic portfolio selection of NPD programs using marginal returns. *Management Science*, 48(10):1227–1241, 2002.

[10] T.C. Perry and J.C. Hartman. Allocating manufacturing capacity by solving a dynamic, stochastic multiknapsack problem. Technical Report ISE 04T-009, Lehigh University, PA, 2004.

[11] L.R. Weatherford and S.E. Bodily. A taxonomy and research overview of perishable-asset revenue management: Yield management, overbooking and pricing. *Operations Research*, 40(5):831–844, 1992.