Cost optimization in the (S-1,S) lost sales inventory model with multiple demand classes

Cost Optimization in the $(S-1,S)$ Lost Sales Inventory Model with Multiple Demand Classes

A.A. Kranenburg*, G.J. van Houtum
Department of Technology Management,
Technische Universiteit Eindhoven,
Eindhoven, The Netherlands

September 29, 2005

Abstract

For the $(S-1,S)$ lost sales inventory model with several demand classes that have different lost sale penalty cost, three accurate and efficient heuristic algorithms are presented that, at a given base stock level, aim to find optimal values for the critical levels, i.e., values that minimize inventory holding and penalty cost.

Keywords: inventory, multiple demand classes, customer differentiation, rationing, lost sales

1 Introduction

In this paper, we study the $(S-1,S)$ lost sales inventory model with multiple demand classes that have different penalty cost values. A penalty is incurred if a demand is not fulfilled from stock, and it can represent e.g. contractual penalty cost, (additional) cost of emergency transportation and/or loss of goodwill of a customer. Within this setting, we aim to minimize the total inventory holding and penalty cost, and we distinguish between different classes by introducing critical levels. A demand from a certain class is only fulfilled if the physical stock is above the critical level for this class.

The problem of multiple demand classes has been introduced by Veinott [13]. He also introduced the concept of critical level policies. After that, the problem has been studied in a number of variants, and these papers can be distinguished in two different streams of research.

* *Corresponding author.* Department of Technology Management, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. Phone: +31 40 247 2637, Fax: +31 40 246 4531, E-mail address: a.a.kranenburg@tue.nl.

The first stream studies the structure of the optimal policy. Within this stream, there are interesting studies that derive the optimality of critical level policies for single-item models with multiple customer classes. Topkis [12] considers a periodic-review model with generally distributed demand and zero lead time. In that situation, the optimal critical levels are dependent on the remaining time in a period. Ha [7] has studied a continuous-review model with Poisson demand processes, a single exponential server for replenishments, and lost sales. He derives the optimality of critical level policies, and in this situation both the base stock levels and critical levels are time-independent. De Véricourt, Karaesmen, and Dallery [14] studied the same model as Ha but with backordering of unsatisfied demand, and obtained the same results.

The second stream consists of studies that consider evaluation and optimization within a given class of policies. Within this stream, there are interesting contributions by Dekker, Hill, Kleijn, and Teunter [3], Melchiors, Dekker, and Klein [10], and Deshpande, Cohen, and Donohue [4]. Dekker et al. [3] derive exact and heuristic procedures for the generation of an optimal critical level policy for a continuous-review model with multiple customer classes, Poisson demands, ample supply, and lost sales. For the case with two customer classes, Melchiors et al. [10] extend this work for fixed quantity ordering. In this model, the fixed order quantity, the base stock level, and a single critical level are optimized in order to minimize the sum of fixed ordering, inventory holding, and lost sales cost. Deshpande et al. [4] consider a similar model but with backordering of unsatisfied demand. In that situation one also must decide in which order backordered demands are satisfied, which leads to additional complications. For further contributions, see the references in the above papers.

In this paper, we study a single-item, continuous-review model with multiple customer classes, Poisson demand processes, lost sales, and ample supply. The ample supply represents that the supplier can deliver as much as desired within a given replenishment lead time. We limit ourselves to the class of critical level policies with time-independent critical and base stock levels. Critical levels are easy to explain in practice, and the results on optimal policies in the first stream of research suggest that this class is at least close-to-optimal (and maybe even optimal under exponential replenishment leadtimes). Under the given class of critical level policies, our problem is to optimize $|J|$ critical levels and one base stock level simultaneously, where J is the set of demand classes. By a projection of the $(|J|+1)$-dimensional total cost function on the dimension of the base stock level and the definition of appropriate convex lower and upper bound functions, the full $(|J|+1)$-dimensional optimization problem may be reduced to solving a series of $|J|$-dimensional subproblems. Here, each $|J|$-dimensional subproblem concerns the optimization of the $|J|$ critical levels at a given base stock level. This reduction has been described by Dekker et al. [3]. By this reduction and applying enumeration for the $|J|$-dimensional subproblems, an exact solution method is obtained for the full problem. Enumeration, however, is expensive from a computational point of view, especially when the number of demand classes is larger than two. Therefore, there is a clear need for efficient

heuristics for the $|J|$-dimensional subproblems.

The main contribution of this paper is that three heuristic algorithms are presented for the $|J|$-dimensional subproblems. An extensive computational experiment is performed, and, surprisingly, in this experiment *all three algorithms always end up in an optimal solution*. The size and settings for the computational experiment give us a reason to believe that the algorithms are either very good (i.e., find an optimal solution in many cases) or exact (i.e., always find an optimal solution). We conjecture the latter, but for practical application, even if our conjecture does not hold, we still have obtained an important result. Computation times of the three heuristic algorithms are small, and far less than of explicit enumeration. So, the three heuristics are accurate and efficient. The three heuristics directly lead to three accurate and efficient heuristics for the full $(|J|+1)$-dimensional problem, and that may be key to obtain efficient solution methods for real-life, multi-item spare parts problems with multiple demand classes and aggregate fill rate or availability constraints. These problems may be solved by a Dantzig-Wolfe decomposition framework, under which multiple instances of a single-item problem with inventory holding and penalty cost have to be solved as column generation subproblems. Our algorithms may be used for these column generation subproblems in all but the last iteration, while an exact algorithm may be used in the last iteration. That avoids the use of an expensive exact method in all iterations and decreases the order of computation time considerably without losing the property that the Dantzig-Wolfe method is exact; see the companion paper [9].

The paper is organized as follows. In Section 2, the model is described, followed by the derivation of a few elementary properties in Section 3. Next, the exact method for the full $(|J|+1)$-dimensional problem is briefly described in Section 4. After that, in Section 5, we present our three efficient heuristic algorithms for the $|J|$-dimensional subproblem, and we show that these algorithms always lead to an optimal solution in an extensive computational experiment. In Section 6, we apply these algorithms in the original full problem and we compare them to a heuristic described by Dekker et al. [3]. Finally, we conclude in Section 7.

2 Model

Consider an item (or stock-keeping unit, SKU) that is demanded by a number of demand classes or customer groups. Let J denote the set of demand classes, with $|J| \geq 1$. For each class $j \in J$, demands are assumed to occur according to a Poisson process with constant rate m_j (> 0). If an item is not delivered to class j upon request, the demand is lost and a penalty cost p_j (> 0) is to be paid. Classes are numbered $1, 2, \ldots, |J|$ and such that $p_1 \geq p_2 \geq \ldots \geq p_{|J|}$.

The item's stock is controlled by a continuous-review critical level policy. This means that the total stock is controlled by a base stock policy with base stock level S $(\in \mathbb{N}_0 := \mathbb{N} \cup \{0\})$, and that

there is a critical level c_j ($\in \mathbb{N}_0$) per class $j \in J$, with $c_1 \leq \ldots \leq c_{|J|} \leq S$. The ordering for the critical levels is assumed because of the opposite ordering in the penalty cost parameters. We call this ordering of the critical levels the *monotonicity constraint*. A critical level policy is denoted by vector (c, S), with $c := (c_1, \ldots, c_{|J|})$. If a class j demand arrives at a moment that the physical stock is *larger than* c_j, then this demand is satisfied. Otherwise, the demand is lost. At and below level c_j, physical stock can be seen as stock that is reserved for more important classes (having a lower index and higher shortage penalty cost). (Intuitively, it seems optimal to choose $c_1 = 0$ as there is not a more important class than class 1; in Section 3, we derive this formally.) For ease of notation, we define $c_0 \equiv 0$ and $c_{|J|+1} \equiv S$. Replenishment lead times are i.i.d. with mean t (following an arbitrary distribution). Holding cost per unit per unit time are h. Without loss of generality, we assume that holding cost is also charged for items in replenishment (see also Remark 1 at the end of this section).

Let $\beta_j(c, S)$ denote the fill rate for class $j \in J$ under critical level policy (c, S), i.e., the expected percentage of requests by class j that will be delivered. An expression for $\beta_j(c, S)$ can be derived as follows. If the number of parts in the pipeline is $k \in \{0, \ldots, S\}$, and thus the number of parts in the physical stock is $S - k$, then the demand rate equals $\mu_k = \sum_{j|k<(S-c_j)} m_j$, $k \in \{0, \ldots, S-1\}$. Our inventory model can be described by a closed queueing network with S customers and two stations: (i) an ample server with mean service time t, which represents the pipeline stock; (ii) a load-dependent, exponential, single server with first-come first-served service discipline, which represents the physical stock. The service rates of the load-dependent server are given by the μ_k. This network belongs to the class of so-called BCMP networks and thus has a product-form solution; see Baskett et al. [2]. By applying the theory of [2], we find that the steady state probabilities q_k for having k parts in the pipeline are equal to:

$$\begin{aligned} q_k &= \left\{ \prod_{i=0}^{k-1} \mu_i \right\} \frac{t^k}{k!} q_0, \quad k \in \{1, \ldots, S\}, \\ q_0 &= \left(\sum_{k=0}^{S} \left\{ \prod_{i=0}^{k-1} \mu_i \right\} \frac{t^k}{k!} \right)^{-1}, \end{aligned} \tag{1}$$

with the convention that $\prod_{i=0}^{k-1} \mu_i = 1$ for $k = 0$ (this result also follows from Gnedenko and Kovalenko [6], pp. 250–252). By these steady state probabilities, we obtain the fill rates:

$$\beta_j(c, S) = \sum_{k=0}^{S-c_j-1} q_k, \qquad j \in J, \tag{2}$$

with the convention that this sum is empty if $S - c_j - 1 < 0$ (i.e., $\beta_j(c, S) = 0$ if $c_j = S$). Notice that $1 \geq \beta_1(c, S) \geq \ldots \geq \beta_{|J|}(c, S) \geq 0$.

The objective is to minimize the average inventory holding and penalty cost per time unit. The

average cost of a policy (c, S) is

$$C(c, S) := hS + \sum_{j \in J} p_j m_j (1 - \beta_j(c, S)).$$

Our optimization problem is a nonlinear integer programming problem and is stated as follows:

$$\begin{array}{lll} (P) & \min & hS + \sum_{j \in J} p_j m_j (1 - \beta_j(c, S)) \\ & \text{subject to} & c_1 \leq \ldots \leq c_{|J|} \leq S, \\ & & c_j \in \mathbb{N}_0, j \in J, \quad S \in \mathbb{N}_0. \end{array}$$

An optimal policy for Problem (P) is denoted by (c^*, S^*) and the corresponding optimal cost is $C(c^*, S^*)$.

For the situation with a fixed base stock level $S \in \mathbb{N}_0$, let Problem $(P(S))$ denote the problem of finding the critical levels such that the average cost $C(c, S)$ is minimized. Problem $(P(S))$ is stated as:

$$\begin{array}{lll} (P(S)) & \min & hS + \sum\limits_{j \in J} p_j m_j (1 - \beta_j(c, S)) \\ & \text{subject to} & c_1 \leq \ldots \leq c_{|J|} \leq S, \\ & & c_j \in \mathbb{N}_0, j \in J. \end{array}$$

An optimal policy for Problem $(P(S))$, $S \in N_0$, is denoted by $(c^*(S), S)$ and the corresponding optimal cost is $C(c^*(S), S)$. Obviously, $(c^*(S^*), S^*) = (c^*, S^*)$. Note that in Problem $(P(S))$, the holding cost term hS constitutes a constant factor. It is included in the formulation of Problem $(P(S))$, however, to ease later reference.

Remark 1 *We assume that holding cost is charged for both items in stock and in the replenishment pipeline. If we only charge holding cost for items in stock, the holding cost expression would be* $hS - \sum_{j \in J} htm_j \beta_j(c, S)$ *since, according to Little's law, the expected number of parts in the pipeline is* $\sum_{j \in J} tm_j \beta_j(c, S)$*. The holding cost expression in this case can be rewritten as* $hS + \sum_{j \in J} htm_j (1 - \beta_j(c, S)) - \sum_{j \in J} htm_j$*. From this, it follows that the problem of finding an optimal policy at penalty cost parameters* p_j*,* $j \in J$*, with no holding cost charged for pipeline inventory, corresponds to a problem where holding cost is charged for pipeline inventory and the penalty cost parameters are set as* $\hat{p}_j := p_j + ht$*,* $j \in J$*. The cost of both problems will differ a constant factor* $\sum_{j \in J} htm_j$*. Notice that the assumed monotonicity of the penalty cost parameters is not violated by this transformation.*

3 Elementary properties

In this section, we derive a few elementary properties. Let e_j denote a row vector of size $|J|$ with the j-th element equal to 1 and all other elements equal to 0. The following monotonicity properties

hold for the fill rates when comparing policy $(c + e_j, S)$ to policy (c, S).

Lemma 1 *For any $j \in J$ with $c_j < c_{j+1}$,*

(i) $\sum_{k\in J} m_k\beta_k(c + e_j, S) < \sum_{k\in J} m_k\beta_k(c, S)$,

(ii) $\beta_k(c + e_j, S) > \beta_k(c, S)$, $k \in J$, $k \neq j$,

(iii) $\beta_j(c + e_j, S) < \beta_j(c, S)$.

Proof: Let $j \in J$ with $c_j < c_{j+1}$. Parts (i)-(iii) are obtained as follows:

(i) Theorem 1 of Dekker et al. [3] implies that $C^{hold}(c + e_j, S) > C^{hold}(c, S)$, with $C^{hold}(c, S) := \sum_{k=0}^{S} h(S - k)q_k$. $C^{hold}(c, S)$ represents holding cost for items that are not in the pipeline. As stated in our Remark 1, $C^{hold}(c, S)$ is also given by $hS - \sum_{k\in J} htm_k\beta_k(c, S)$, and thus $\sum_{k\in J} m_k\beta_k(c + e_j, S) < \sum_{k\in J} m_k\beta_k(c, S)$.

(ii) Lemma 1 in Dekker et al. [3] states that under policy $(c+e_j, S)$, the steady state probabilities for states $0, \ldots, S-c_j-1$ are strictly larger than those under policy (c, S), and that the steady state probabilities for states $S - c_j, \ldots, S$ are strictly smaller. For classes $k > j$, it follows that $\beta_k(c + e_j, S) > \beta_k(c, S)$ because in both policies the same states are *included* in the calculation of $\beta_k(c+e_j, S)$ and $\beta_k(c, S)$ (see (2)), and all these states have larger probabilities under policy $(c + e_j, S)$. For classes $k < j$, it follows that $\beta_k(c + e_j, S) > \beta(c, S)$ because in both policies the same states are *excluded* in the calculation of $\beta_k(c+e_j, S)$ and $\beta_k(c, S)$, and all these states have smaller probabilities under policy $(c + e_j, S)$.

(iii) By parts (ii) and (i), respectively, we find that

$$m_j(\beta_j(c + e_j, S) - \beta_j(c, S)) < \sum_{k\in J, k\neq j} m_k(\beta_k(c, S) - \beta_k(c + e_j, S)) < 0,$$

and thus $\beta_j(c + e_j, S) < \beta_j(c, S)$.

(Q.E.D.)

Lemma 1 states that, if allowed, increasing the critical level for demand class j with one unit leads to a lower fill rate for class j itself and to higher fill rates for all other demand classes. Further, a decrease is obtained for the weighted sum $\sum_{k\in J} m_k\beta_k(\cdot)$. By Lemma 1, we can prove that for all demand classes with the highest penalty cost parameter, the optimal critical levels are equal to 0, both within Problem $(P(S))$ and Problem (P). This is stated in Lemma 2. This lemma implies that all optimal critical levels are equal to 0 if the penalty cost parameter is the same for all demand classes.

Lemma 2 *Let $k = \max\{j \in J | p_j = p_1\}$. It holds that $c_j^*(S) = 0$ for all $j \leq k$ and $S \in \mathbb{N}_0$, and that $c_j^* = 0$ for all $j \leq k$.*

Proof: Let $k = \max\{j \in J | p_j = p_1\}$ and let policy (c, S) be such that $c_j > c_{j-1}$ for some $j \leq k$ (as $c_0 = 0$, this inequality reduces to $c_1 > 0$ if $j = 1$). It holds that

$$\begin{aligned} C(c,S) &= hS + \sum_{i \in J} p_i m_i (1 - \beta_i(c,S)) \\ &= hS + p_1 \sum_{i \in J} m_i (1 - \beta_i(c,S)) - \sum_{i \in J, i > k} (p_1 - p_i) m_i (1 - \beta_i(c,S)), \end{aligned}$$

and similarly for $C(c - e_j, S)$. By Lemma 1, $\sum_{i \in J} m_i \beta_i(c,S) < \sum_{i \in J} m_i \beta_i(c - e_j, S))$ and $\beta_i(c,S) > \beta_i(c - e_j, S)$ for all $i > k$, and thus via the above expression for $C(c,S)$ we find that $C(c,S) > C(c - e_j, S)$. In other words, policy (c,S) is suboptimal. This implies the properties for optimal critical levels, both for Problem $(P(S))$ and Problem (P). (Q.E.D.)

4 Exact method for Problem (P)

A method to solve Problem (P) exactly has been described by Dekker et al. [3]. This method exploits convex lower and upper bound functions for the function $C(c^*(S), S)$. First, a lower bound S_l is determined for the optimal base stock level. Next, starting at this lower bound, Problem $(P(S))$ is solved by enumeration for increasing values of S, until a stopping criterium is met that implies that an optimal policy has been found. In this section, the method is summarized. For proofs, the reader is referred to Dekker et al. [3]. Notice that our notation and our indices of critical levels deviate from theirs. Furthermore, notice that they do not account holding cost for items in pipeline, but as we stated in Remark 1, this is equivalent to the same problem with a transformation of penalty cost parameters.

An upper bound for the function $C(c^*(S), S)$ is obtained by taking all critical levels equal to 0 for each S. This gives the upper bound function $C_u(S) = C(0, S)$. This function is convex, which follows from the convex behavior of the Erlang loss probability for all $S \in N_0$; for a proof of this latter result, see Appendix (Dekker et al. [3] refer to Smith [11], who indeed claim that it can be seen by algebraic manipulation that the Erlang loss probability is convex, but Smith [11] does not prove it). A lower bound function for $C(c^*(S), S)$ is obtained by replacing all penalty cost parameters p_j by the lowest penalty cost parameter $p_{|J|}$ in Problem $(P(S))$. Under an optimal policy for this modified problem all critical levels are zero (cf. Lemma 2). The resulting cost are denoted by $C_l(S)$, and also for this function the convexity follows from the convexity of the Erlang loss probability.

The exact algorithm for Problem (P) is as follows. First, define S' as a minimizing point for the upper bound function $C_u(S)$. Next, S_l is defined as the lowest $S \in \mathbb{N}_0$ for which $C_l(S) \leq C_u(S')$. This S_l is a lower bound for the optimal base stock level S^*. Then, for $S = S_l$, Problem $(P(S))$ is solved using explicit enumeration, and the current best solution is set equal to $(c^*(S), S)$. Hereafter, S is increased by one, $(P(S))$ is solved using explicit enumeration, and the current best solution is adapted if $(c^*(S), S)$ provides a better one. This is done repeatedly until $C_l(S+1) > C_l(S)$ and $C_l(S+1)$ is larger than or equal to the current best solution (these two conditions imply that for any base stock level greater than the current base stock level no better solution can be found than the current best solution). At this point, the current best solution constitutes an optimal solution for Problem (P).

5 Algorithms for Problem $(P(S))$

The exact method uses enumeration to solve Problem $(P(S))$ for multiple values of S and thus is time consuming, in particular for problems with 3 or more demand classes. Therefore, in this section, we describe and test fast heuristics for Problem $(P(S))$. The heuristics that we consider are (a kind of) local search algorithms. We test the accuracy in an extensive computational experiment, and we find that all three heuristics produce an optimal solution in all instances. Unfortunately, there is no proof of exactness for any of these heuristics.

Before we formulate the heuristics, we show the typical behavior of the function $C(c, S)$ for a fixed S. In Figure 1, the costs $C(c, S)$ are shown for an example with $|J| = 3$ demand classes. In this example, the critical level for class 1 is fixed at 0, as that is known to be optimal (by Lemma 2). We see in this figure that $-C(c, S)$ is unimodal in c_3 for any fixed c_2, and vice versa. This means that the sign of the first order difference of the cost terms changes at most once. If it changes, it changes from minus into plus. In all examples that we considered in detail, we observed this unimodal behavior.

Intuitively, the observed unimodality increases the chances for local search type algorithms to find an optimal solution, but a guarantee cannot be given. Proving that a local search algorithm is exact for an optimization problem with a multi-dimensional, discrete state space is known to be very hard. Under a continuous solution space, local search would be exact for convex functions, but proving convexity can be difficult. The property that a local minimum is a global minimum is not obtained for direct extensions of the concept of convexity to discrete spaces, but it is obtained for *multimodular functions*, as introduced by Hajek [8]; see Altman et al. [1] for a whole theory based on multimodularity. This concept seems not applicable to our problem, however (the cost function in Figure 1 for the example with $|J| = 3$ demand classes seems not multimodular).

c2 \ c3	0	1	2	3	4	5	6	7	8	9	10	11
0	13.234	12.498	12.026	11.766	11.729	12.002	12.732	14.056	15.974	18.201	20.129	21.070
1		11.834	11.583	11.470	11.532	11.869	12.641	13.993	15.927	18.164	20.096	21.039
2			11.575	11.465	11.528	11.867	12.640	13.992	15.926	18.163	20.095	21.039
3				12.086	11.943	12.147	12.831	14.126	16.026	18.243	20.165	21.105
4					13.762	13.369	13.666	14.714	16.461	18.590	20.470	21.397
5						17.625	16.563	16.748	17.966	19.792	21.524	22.405
6							25.154	22.735	22.377	23.307	24.604	25.351
7								37.769	33.272	31.916	32.123	32.536
8									56.041	49.394	47.191	46.885
9										78.953	71.660	69.920
10											103.13	98.460
11												121.00

Figure 1: Cost $C(c, S)$ as function of c_2 and c_3 for $c_1 = 0$ and $S = 11$ at input parameters $|J| = 3$, $m_1 = m_2 = m_3 = 1$, $t = 1$, $h = 1$, $p_1 = 10000$, $p_2 = 100$, $p_3 = 10$.

5.1 Description of the heuristics

In this subsection, we formulate the three heuristic algorithms for Problem $(P(S))$. Algorithms 1 and 2 are rather straightforward algorithms. Algorithm 3 is similar to a heuristic of Dekker et al. [3] for Problem (P) (their Algorithm 3). The difference with their heuristic is that we assume a constant base stock level (and thus solve Problem $(P(S))$), while they also incorporate optimization of the base stock level. The algorithms are as follows (in all three algorithms the critical levels for the demand classes $1, \ldots, k$, with $k = \max\{j \in J | p_j = p_1\}$, are fixed at 0 as that is optimal, cf. Lemma 2):

Algorithm 1 Start with an arbitrary choice for c_j, $j \in J$, $j > k$, that satisfies the monotonicity constraint. Define the neighborhood of this current policy (c, S) as all policies that still satisfy the monotonicity constraint and that have critical levels that differ at most one from the corresponding critical level in the original policy. This gives at most $3^{|J|-k} - 1$ neighbors, but usually much less because of the monotonicity constraint. If the cost of the cheapest neighbor is smaller than the cost of the current solution, then select this neighbor and set this policy as the current solution, and repeat the process of evaluating all neighbors for this new policy. Otherwise, stop and take the current solution as the solution found by the algorithm.

Algorithm 2 Start with an arbitrary choice for c_j, $j \in J$, $j > k$, that satisfies the monotonicity constraint. For $i = |J|$, find $c_i \in \{c_{i-1}, \ldots, c_{i+1}\}$ with the lowest cost, at fixed values of the other critical levels, and change c_i accordingly (recall that $c_{|J|+1} = S$). Repeat this optimization for one critical level at a time for $i = |J| - 1$ down to $k + 1$. After that, optimize again for $i = |J|$. Continue this iterative process until for none of the i-values $(\in \{k + 1, \ldots, |J|\})$ an improvement is found. This is the solution found by the algorithm.

Algorithm 3 Start with all critical levels equal to zero. Next we start iteration 1. We first consider increasing $c_{|J|}$ by one (if allowed by the monotonicity constraint), and accept this increase if it has lower cost than the current solution. Then, the critical levels $c_{|J|-1}$ down to c_{k+1} are increased with 1 (one critical level at a time), and each time an increase of a critical level is accepted if lower cost are obtained. If $c_{|J|}$ was increased with 1 in this first iteration, then we execute another interation, and so on. The process stops as soon as $c_{|J|}$ has not been increased in an iteration (but the last iteration is executed completely, i.e., it is possible that some of the critical levels c_j, $k < j < |J|$, are increased in the last iteration, while $c_{|J|}$ stayed at the same level). The policy found at the end of the last iteration is the solution found by the algorithm.

5.2 Computational experiment

We test the performance of the Algorithms 1, 2, and 3 in an extensive computational experiment. In this experiment, the settings are chosen as described in Table 1. As seen in this table, we study instances with different numbers of classes. For demand rates, we have 5 values, small and large, and with difference factors of 2, 5, 10, 20, 50, and 100. We study any combination of these demand rates. At $|J|$ classes, this implies $5^{|J|}$ choices. Both the mean replenishment lead time and the holding cost parameter are fixed at 1, which we may do w.l.o.g.. For the penalty cost parameters, we have varied both the relative weight compared to the (fixed) holding cost parameter and the ratio between the penalty cost parameters for different classes. For the penalty cost parameters, we have 9 different settings, independent of $|J|$. If $|J| < 5$, only the first $|J|$ values are used. From Table 1, we obtain 225 instances with 2 classes, 1125 instances with 3 classes, and 28125 instances with 5 classes. The number of instances studied increases with the number of classes. This is in line with our intention, since if an algorithm would fail to reach an optimal solution, it is generally more likely that this occurs in situations with more degrees of freedom and therefore it is recommendable to study this kind of instances more thoroughly.

For each choice of settings from Table 1, we study values for base stock level S from 1 up to a value that depends on the settings. This maximum base stock level is determined as the smallest S for which it holds that $\beta_j(c, S) \geq 1 - 10^{-12}$, for all $j \in J$, where c is taken equal to $c = (0, \ldots, 0)$. We have chosen to use this variable number of base stock levels since in this way, for all parameter settings from Table 1, we study as many different base stock levels as relevant. Notice that, by Lemma 1, for larger base stock levels than the maximum base stock level, the cost decrease of applying rationing (i.e., using non-zero critical levels) is at most $10^{-12} \sum_{j \in J, j \neq |J|} p_j m_j$ compared to a pure base stock policy (with all critical levels zero). For all parameter settings we use, this expression is always less than 10^{-6}, and thus the excluded cases are not really of interest. Maximum base stock levels vary between 7 and 67. On average, the number of base stock levels considered

Parameter	**Values**
$\|J\|$	2, 3, 5
m_j	0.05, 0.1, 0.5, 1, 5
$(p_1, p_2, p_3, p_4, p_5)$	(5000, 4000, 3000, 2000, 1000)
	(50, 40, 30, 20, 10)
	(0.5, 0.4, 0.3, 0.2, 0.1)
	(16000, 8000, 4000, 2000, 1000)
	(160, 80, 40, 20, 10)
	(1.6, 0.8, 0.4, 0.2, 0.1)
	(10000, 1000, 100, 10, 1)
	(100, 10, 1, 0.1, 0.01)
	(1, 0.1, 0.01, 0.001, 0.0001)
t	1
h	1

Table 1: Parameter settings for numerical experiment (for penalty cost values, only values $p_1, \ldots, p_{|J|}$ are used)

per choice of the other parameters, is 30, and the total number of instances is 883845.

For all 883845 instances, we compared the solutions obtained by Algorithms 1, 2, and 3 to the optimal solution obtained by explicit enumeration. Algorithms 1 and 2 were evaluated with different starting points $c = (0, \ldots, 0)$ and $c = (0, S, \ldots, S)$, respectively denoted by (i) and (ii). Additionally, for $|J| = 3$, we considered starting point $c = (0, 0, S)$, and for $|J| = 5$, we considered starting point $c = (0, 0, 0, S, S)$. This additional starting point is denoted by (iii). For Algorithm 3, the starting point is always $c = (0, \ldots, 0)$. The results of the numerical experiment are given in Table 2. Notice that the number of instances considered at starting point (iii) for Algorithms 1 and 2 is smaller than 883845, since the instances with two classes do not have this additional starting point. In Table 2, the second column (the number of instances for which the algorithm did find an optimal solution) denotes the number of instances for which the cost of the obtained solution was within $10^{-12}\%$ of the solution found by enumeration (i.e., $(C(c(S), S) - C(c^*(S), S))/C(c^*(S), S) \leq 10^{-12}$). Mostly, exactly the same solution was found, but in some cases (number given between parentheses), the obtained policy was slightly different from the one found by enumeration. In those cases, the cost differences between both solutions were smaller than or of the same size as the numerical precision in the calculation of the cost, and thus both may be considered as optimal. (The maximum absolute difference, i.e. the maximum value for $C(c(S), S) - C(c^*(S), S)$, was $2.64 \cdot 10^{-11}$.)

As we see, *Algorithm 1 (with different starting points), Algorithm 2 (with different starting points), and Algorithm 3, all result in an optimal solution in all instances in the computational*

Algorithm	**Numbers of instances**		**Times (in milliseconds)**	
	optimum found	**no optimum found**	**average**	**maximum**
Enumeration	883845		95	7050
Algorithm 1 (i)	883845 (129)	0	0	30
Algorithm 1 (ii)	883845 (2432)	0	3	50
Algorithm 1 (iii)	879624 (1186)	0	1	31
Algorithm 2 (i)	883845 (828)	0	0	11
Algorithm 2 (ii)	883845 (2635)	0	0	11
Algorithm 2 (iii)	879624 (592)	0	0	11
Algorithm 3	883845 (899)	0	0	11

Table 2: Results from numerical experiment for Problem ($P(S)$)

experiment. This can be seen from the third column in Table 2, where the numbers of instances are indicated for which each of the methods did not find an optimal solution. As discussed above, there is no proof for the exactness of (one of) these algorithms. However, based on the size and results of the numerical experiment, we believe that Algorithms 1, 2, and 3 always will find an optimal solution, i.e., *we conjecture that Algorithms 1, 2, and 3 are exact for Problem* ($P(S)$).

Especially for Algorithm 3, it is interesting that it always ends up in an optimal solution, since the number of policies considered in this algorithm is much lower than under the other two algorithms, and the search procedure does give the impression that a specific underlying structure is present with respect to cost as a function of the critical levels.

With respect to computation time, we can report the following. We have run our numerical experiment on a Pentium 4 PC, with algorithms implemented in Delphi 7.0. Per instance, the computation time was measured in milliseconds (i.e., in integer numbers of milliseconds). Average and maximum computation times are given in Table 2. The listed computation times show that each of the proposed algorithms strongly outperforms explicit enumeration.

6 Algorithms for Problem (P)

For Problem (P), an exact method was presented in Section 4. In the exact method, Problem ($P(S)$) has to be solved multiple times, and this is done by explicit enumeration. However, Section 5 shows that Problem ($P(S)$) can be solved more efficiently by either one of the proposed heuristic algorithms. In this section, we replace the explicit enumeration for Problem ($P(S)$) by the algorithms from Section 5, show that also in this setting the algorithms perform well, and compare them to a heuristic formulated by Dekker et al. [3]. This heuristic is the only good heuristic for Problem (P) that is available in the literature. Below, we describe the algorithms for Problem (P)

in Subsection 6.1, and we evaluate their performance in Subsection 6.2.

6.1 Description

The Algorithms 1, 2, and 3 for Problem $(P(S))$ can be plugged into the exact method for Problem (P), thus replacing the explicit enumeration. The resulting algorithms are called **Algorithms 4, 5, and 6**, where now for Algorithms 4 and 5 the choice for the starting point is fixed at $c = (0, \ldots, 0)$. The heuristic by Dekker et al. [3] (p. 605, Algorithm 3) is denoted as Algorithm 7 and works as follows (again the critical levels for the demand classes $1, \ldots, k$, with $k = \max\{j \in J | p_j = p_1\}$ are fixed at 0):

Algorithm 7 Start with all critical levels equal to zero and determine the optimal base stock level for these given critical levels. Next we start iteration 1. We first consider increasing $c_{|J|}$ by one (if allowed by the monotonicity constraint), and optimize the base stock again. We accept this increase in $c_{|J|}$ if it has lower cost than the current solution. Then, the critical levels $c_{|J|-1}$ down to c_{k+1} are increased with 1 (one critical level at a time), and each time an increase of a critical level is accepted if lower cost are obtained. Also here, in each step the base stock level is optimized. If $c_{|J|}$ was increased with 1 in this first iteration, then we execute another interation, and so on. The process stops as soon as $c_{|J|}$ has not been increased in an iteration. The policy found at the end of the last iteration is the solution found by the algorithm.

We obtained the code from Dekker et al., and used that code to ensure that we had the same code as they used in their experiments.

6.2 Computational experiment

We implemented Algorithms 4, 5, 6, and 7 in Delphi 7.0 to test the performance of the algorithms. We use the test instances as mentioned in Table 1. The difference with the experiment in Section 5 is that in the current section, base stock level S is part of the optimization. In total we study 29475 instances: 225 instances with 2 classes, 1125 instances with 3 classes, and 28125 instances with 5 classes.

For all instances, we compared the solutions obtained by Algorithms 4, 5, 6, and 7 to the optimal solution obtained by the exact method. The results of the computational experiment are given in Table 3. The second column in this table (the number of instances for which the algorithm did find an optimal solution) denotes the number of instances for which exactly the same solution was found as in the exact method. The third column denotes the number of instances for which the algorithm did not find an optimal solution.

Algorithm	Numbers of instances		Times (in milliseconds)	
	optimum found	no optimum found	average	maximum
Exact Method	29475		86	8812
Algorithm 4	29475	0	0	11
Algorithm 5	29475	0	0	11
Algorithm 6	29475	0	0	11
Algorithm 7	28735	740	0	11

Table 3: Results from numerical experiment for Problem (P)

As we expected, Algorithms 4, 5, and 6 perform well. *They result in an optimal solution for all considered instances, but require much less computation time than the exact method.* The heuristic of Dekker et al., however, does not find an optimal solution in 2.5% of the cases. The relative difference between the cost of the obtained solution and the cost of the optimal policy, $(C(c,S) - C(c^*,S^*))/C(c^*,S^*)$, in these 740 instances is 25% on average and 313% in the worst case. Although Dekker et al. did not claim optimality, they observed that their heuristic always found an optimal solution in their numerical experiment. Our experiment clearly shows that their heuristic can perform poorly. Recall that Algorithm 3 and 7 show similarity, since the only difference is that in Algorithm 7 the base stock level is variable while it is fixed in Algorithm 3. Apparently, incorporation of the base stock level as variable in the algorithm is not always a good choice.

Remark 2 *Dekker et al. conjecture that for each option (increase of one of the critical levels), in the optimization of the base stock level, it suffices to consider S (the current value) and $S-1$. We found the following counterexample for this conjecture: $|J| = 2$, $m_1 = m_2 = 1$, $t = 14$, $h = 1$, $p_1 = 10000$, and $p_2 = 100$. In this example, for $c_2 = 0$, the best choice for S is 48, while for $c_2 = 1$, the best choice for S is 46. (Recall that Dekker et al. do not incur holding cost for items in the pipeline but that this is just a transformation in the penalty cost parameters, see Remark 1).*

7 Conclusion

In this paper, an $(S-1,S)$ lost sales inventory model with priority demand classes and controlled by critical level policies has been considered. We distinguished two problems. In Problem (P), the base stock level and critical levels are optimized in order to obtain minimal inventory holding and penalty cost. In a subproblem, Problem $(P(S))$, the base stock level is fixed and only the critical levels are optimized. The main contribution of this paper consists of the formulation of three heuristic algorithms for Problem $(P(S))$ that, in an extensive computational experiment, all three

always end up in an optimal solution. Our algorithms are much faster than explicit enumeration, and thus are an welcome alternative for enumeration in an exact algorithm for Problem (P) in which Problem $(P(S))$ has to be solved many times as subproblem.

The size and settings for the numerical experiment give us reason to believe that the algorithms are either very good (i.e., find an optimal solution in many cases) or exact (i.e., find an optimal solution always). We conjecture the latter, but for practical application, even if our conjecture does not hold, we still have obtained an important result. In multi-item spare parts inventory problems with customer differentiation, Problem (P) is a column generation subproblem in a Dantzig-Wolfe decomposition framework. This subproblem has to be solved multiple times. Even if our algorithms are not exact, a large reduction in computation time is obtained without losing optimality in the Dantzig-Wolfe method. This is done by using one of our algorithms for the column generation subproblem instead of explicit enumeration in all but the last iteration in Dantzig-Wolfe decomposition; see the results in the companion paper [9].

References

[1] E. Altman, B. Gaujal, A. Hordijk, Discrete-Event Control of Stochastic Networks: Multimodularity and Regularity, Springer, 2003.

[2] F. Baskett, K.M. Chandy, R.R. Muntz, F.G. Palacios, Open, closed, and mixed networks of queues with different classes of customers, Journal of the Association for Computing Machinery 22 (1975) 248–260.

[3] R. Dekker, R.M. Hill, M.J. Kleijn, R.H. Teunter, On the $(S-1,S)$ lost sales inventory model with priority demand classes, Naval Research Logistics 49 (2002) 593–610.

[4] V. Deshpande, M.A. Cohen, K. Donohue, A threshold inventory rationing policy for service-differentiated demand classes, Management Science 49 (2003) 683–703.

[5] L. Dowdy, D. Eager, K. Gordon, L. Saxton, Throughput concavity and response time convexity, Information Processing Letters 19 (1984) 209–212.

[6] B.V. Gnedenko, I.N. Kovalenko, Introduction to Queueing Theory, Israel Program for Scientific Translations, 1968.

[7] A.Y. Ha, Inventory rationing in a make-to-stock production system with several demand classes and lost sales, Management Science 43 (1997) 1093–1103.

[8] B. Hajek, Extremal splittings of point processes, Mathematics of Operations Research 10 (1985) 543–556.

[9] A.A. Kranenburg, G.J. van Houtum, Service differentiation in multi-item spare parts inventory systems, Working Paper, Technische Universiteit Eindhoven, 2005.

[10] P. Melchiors, R. Dekker, M.J. Kleijn, Inventory rationing in an (s,Q) inventory model with lost sales and two demand classes, Journal of the Operational Research Society 51 (2000) 111–122.

[11] S.A. Smith, Optimal inventories for an $(S-1,S)$ system with no backorders, Management Science 23 (1977) 522–528.

[12] D.M. Topkis, Optimal ordering and rationing policies in a nonstationary dynamic inventory model with n demand classes, Management Science 15 (1968) 160–176.

[13] A.F. Veinott, Optimal policy in a dynamic, single product, non-stationary inventory model with several demand classes, Operations Research 13 (1965) 761–778.

[14] F. de Véricourt, F. Karaesmen, Y. Dallery, Optimal stock allocation for a capacitated supply system, Management Science 48 (2002) 1486–1501.

Appendix

In this appendix, it is proved that $C_u(S) = C(0,S)$ is convex in S. To show that, it suffices to show that $\beta_1(0,S)$ is concave as a function of $S \in \mathbb{N}_0$.

Under policy $(0,S)$, the behavior of our inventory system is identical to the behavior of a closed queueing network with S customers and two stations: (i) an ample server with mean service time t; (ii) an exponential, load-independent single server. We are interested in the throughput of the network. Because the steady-state distribution for the number of jobs per station and the throughput are independent of the distribution of the service times at the ample server, in the rest of this proof we may assume w.l.o.g. that the service times at the ample server are exponential. The ample server then may be seen as a load-dependent, exponential, single server, with a service rate that is linear with the number of jobs. Hence, we can apply a result of Dowdy et al. [5]. They prove that the throughput in a closed queueing network with one load-independent, exponential, single server and one load-dependent, exponential, single server, whose load-dependent service rates are nondecreasing and concave (of which our linearly increasing service rate is a special case), is concave as a function of the number of customers in the network. It is easily verified that in our network the throughput is equal to $\beta_j(0,S)\sum_{j\in J} m_j$. The property that $\beta_j(0,S)\sum_{j\in J} m_j$ is concave in S implies that $\beta_j(0,S)$ is concave as a function of $S \in \mathbb{N}_0$.