Inventory budget optimization: Meeting system-wide service levels in practice

P. Korevaar
U. Schimpel
R. Boedi

The work described in the literature on inventory and supply chain management has advanced greatly over the last few decades and now covers many aspects and challenges of applied supply chain management. In this paper we describe an approach that combines many of these academic aspects in a practical way to manage the spare parts logistics at a German automobile manufacturer. The basic problem is a single-echelon inventory problem with a system-wide service-level requirement and the possibility of issuing emergency orders. There exist two related optimization problems: One is to maximize the system-wide service level under the constraint of a given budget; the other is to minimize the budget for a given system-wide service level. The most important requirements and constraints considered are a detailed cost structure, different packaging sizes, capacity constraints, several storage zones, the decision whether or not to stock a product, stochastic lead times, highly sporadic demands, and the stability of the optimization result over time. Our approach has been implemented successfully in an automotive spare parts planning environment. The complete solution package integrates into the mySAP ERP®, the SAP Enterprise Resource Planning system, and APO 4.0, the SAP Advanced Planning and Optimization system. A detailed description of the model is given and results are presented.

Introduction

Most literature on inventory and supply chain management covers either single aspects of the supply chain in detail or more general scenarios with many simplifications. This paper describes an intermediate approach that covers the supply chain from central through regional storage to customers, and it models many practical obstacles and restrictions in detail. This is not a theoretical approach; it describes a real implementation that successfully manages the spare parts logistics of a German auto manufacturer.

The scenario we faced in the beginning was a serial multi-echelon environment with one central distribution center (CDC), one regional distribution center (RDC), and many customers who are supplied by the RDC. At a closer look, only the average service level to customers mattered: The replenishment policy at the CDC was not subject to further optimization because of dependencies with other RDCs that were out of scope for this project. This allowed us to reduce the optimization problem to a single-echelon environment with no disadvantage.

Thus, our basic problem can be classified as a single-echelon delivery structure with one specific multi-echelon aspect (the integrated stocking decision at the RDC level). The model allows for the following:

- Emergency orders between the CDC and the RDC.
- System-wide, customer-facing average service levels that have to be met, where the individual service levels per SKU (stock-keeping unit) can be optimized.
- Sparse and intermittent demand patterns.
- The decision whether or not to stock a SKU regionally.

Any customer demand that is not available from stock is delivered directly to the customer from the CDC at

0018-8646/07/$5.00 © 2007 IBM

an additional cost. There exist two related optimization problems:

1. Maximize the system-wide service level under the constraint of a given total cost budget.
2. Minimize the total cost budget for a given system-wide service level.

The setting at the automobile manufacturer required a detailed cost structure comprising fixed and variable costs for warehousing processes depending on the warehouse zone, and fixed and variable costs for handling and transportation of normal replenishment orders, planned rush orders, and unplanned back orders. Moreover, other factors were taken into account: different packaging sizes, capacity constraints, the decision whether to stock a SKU at the RDC, stochastic lead times, highly sporadic demand, and the stability of the optimization result over time.

This paper describes a heuristic solution to the outlined scenario. The full optimization process consists of two separate steps. In the first step, EOQ calculation (EOQ = economic order quantity), the optimal order quantity and optimal pack size are determined for each SKU. In the second step, budget optimization, it is decided whether or not each individual SKU should be stocked at the regional warehouse. If the decision is yes, the optimal *safety stock* (the planned stock level when a replenishment order enters the warehouse) and *reorder points* (the planned stock level at which a replenishment order is placed) are derived for normal and planned rush orders.

Our approach required the development of an optimization module, called the IBM Inventory *Budget Optimizer*, which was linked to the existing IBM Dynamic Inventory Optimization Solution (DIOS). The complete solution package, consisting of DIOS and the budget optimizer, has been successfully integrated into the SAP Enterprise Resource Planning system mySap ERP** and the SAP Advanced Planning and Optimization system APO 4.0 of the automobile manufacturer.

Business relevance and background

While restructuring the after-market logistics, a German automotive manufacturer decided to replace its internally developed ERP system with the standard mySAP ERP and SAP APO 4.0 systems, with the necessary adaptations and extensions. A detailed review of the planning capabilities of SAP APO 4.0 showed that a number of specific planning requirements could not be satisfied, and it became clear that upcoming releases of SAP APO would be unable to fill these gaps. As a result, the automobile manufacturer decided to extend its planning capabilities with specific planning components. IBM DIOS was well suited for this. In its standard form, DIOS was already able to fulfill more than 50 percent of the client's planning requirements. The missing functionality was developed specifically for this manufacturer and added to DIOS as a customer-specific plug-in. The plug-in serves two main tasks: EOQ calculation and budget optimization. The EOQ calculation is not discussed in detail in this paper. What is relevant here is that the EOQ serves as an input to the relationship between the safety stock and the service level, described in the section below on input parameters for the budget optimization.

Why needed in a SAP environment

A high-level comparison of functions shows that SAP APO is able to determine an EOQ and a safety stock for each stocked SKU. Therefore, it is justifiable to ask why DIOS and a specific plug-in are needed to provide these planning parameters instead. The answer is that a number of critical requirements cannot be met by SAP APO:

- Calculation of an optimal pack size using real cost. SKUs can be replenished in different pack sizes. For each pack size, other handling costs apply, since different pack sizes are stored in different warehouse regions with different handling costs. Therefore, the standard EOQ calculation requires various extensions to properly model these issues. SAP APO does not have such a detailed EOQ calculation.
- It must be decided whether each SKU should be stocked regionally in the RDC or only centrally in the CDC. Less than 30 percent of all required SKUs are stocked in the RDC. This decision, referred to as the *StockYN* decision, must be updated continuously, and a major issue is to keep available the optimal parts assortment that yields the required service levels at minimal cost. The stock decision for one SKU cannot be taken independently of other SKUs because of the system-wide cost and service-level targets. As soon as a SKU is removed from stock, the overall target service level is changed; as a result, the stock levels for other SKUs change because the system-wide service-level target must still be met. Thus, the stocking decision must be an integral part of the optimization process. SAP APO does not provide such an integral stocking decision. It has been shown that simple rules, such as *Stock when more than four picks have occurred in the last 12 months*, lead to solutions that are far from the cost optimum.
- Rush orders from an RDC to the CDC are placed for two reasons. The first is to deliver unavailable SKUs. In this case, an order has been placed by a dealer for a SKU that is not available in the RDC. A rush order

to the CDC then follows so that the dealer's request can be filled. These orders are called *back orders*. Second, rush orders are placed to prevent out-of-stock (*stockout*) situations in the RDC. This is common practice. Planners recognize that the stock levels are low and, although a normal order has already been placed, it is not expected to arrive in time. A rush order is then placed that will arrive before the normal order, thus increasing availability. These orders are called *planned rush orders*, or simply *rush orders*. From a cost perspective, these rush orders make sense because the safety stock levels in the RDC can be kept at a lower level and the target service levels can still be met. SAP APO does not offer any function that makes it possible to set a second reorder point, the *rush order reorder point* (ROROP). Like the StockYN decision, the ROROP setting cannot be computed independently for each SKU, nor independently of the safety stock setting, since both together determine the target service level. Therefore, the determination of the ROROP must be an integral part of the budget optimizer as well.

- For each SKU, a safety stock is required to ensure that its service level is reached. The relationship between the safety stock and the service level is nonlinear and depends on many variables, such as demand distribution, lead time, lead time variability, service-level type, the EOQ, and the demand. In practice, it is found that the demand distribution for most spare parts cannot be represented by the normally employed Gaussian or Poisson distributions. DIOS enables the use of all common types of demand distributions and takes into account all of the above dependencies. Implementing any textbook approach using predefined demand distributions can lead to dramatic inaccuracies in the relationship between safety stocks and service levels [1]. SAP APO offers only Gaussian and Poisson distributions. Details are explained below in the section on service-level relations. For each SKU, the relation is determined and passed to the budget optimizer.
- Service-level differentiation is an important means of cost reduction. For example, expensive SKUs may receive a lower service level, less expensive SKUs a higher one. The system-wide availability remains the same, but total costs go down. This is the heart of the budget optimizer module, from which the name follows. Though SAP APO makes it possible to assign service levels on an individual SKU basis, there are no optimization capabilities that automatically find the cost-optimal target service level for each SKU. The budget optimizer ensures that the target service level of each group is fulfilled, while it reduces the cost by leveraging the freedom of service-level differentiation.

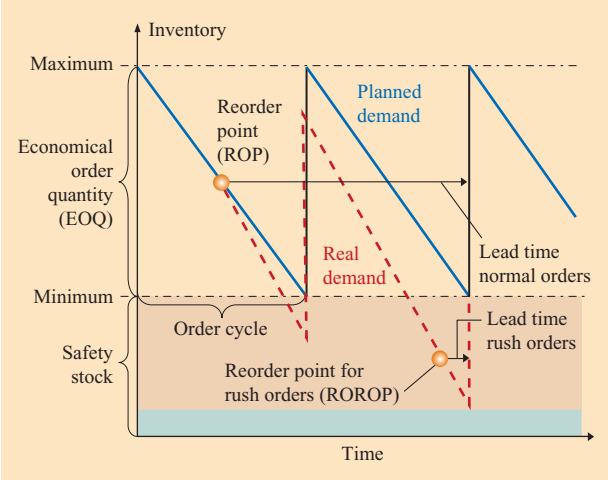

Figure 1

Interdependency among EOQ, safety stock, lead times, and reorder points. Minimum and maximum stock levels are not hard limits, but describe the planning targets without capacity constraints.

The relationships among all required inventory optimization variables are shown in **Figure 1**.

Market positioning

The need for inventory optimization is growing. As reported by the Aberdeen Group (see Figure 1 in [2]), many companies still use inferior techniques to plan their inventories. Several advanced planning system (APS) software vendors offer inventory optimization packages, some of which also offer budget optimization functionalities. The main vendors in this area are i2 Technologies, GainSystems, and ToolsGroup. Although all three offer software to optimize budgets, none explains in detail the algorithms that are being used and the kind of functionality that is implemented. Except for the information each company has itself published on the Web, we could find no independent evaluation in the literature, which makes it difficult to compile a detailed comparison. However, an analysis based on the information available on their Web sites indicates that the IBM Inventory Budget Optimization Solution provides a combination of capabilities that are not offered by any of these companies.

The i2 Service Budget Optimizer** [3] promises a rich functionality and is similar to the inventory budget optimization described in this paper. It performs a

full-cost optimization and differentiates service levels, still reaching the system-wide service-level targets. Key differentiators of the IBM Inventory Budget Optimizer are the integrated StockYN decision and the allowance for arbitrary demand distributions.

The GainSystems Inventory Chain Optimization (ICO) [4] allows for a StockYN decision and a service-level differentiation. However, the service-level differentiation is not an output of the optimization but an input entered by the planners.

ToolsGroup offers a set of white papers [5, 6] and can therefore be evaluated in somewhat more detail. Except for the StockYN decision, their software offers the same capabilities as the IBM Inventory Budget Optimizer.

Related work

The approach described in this paper unifies many aspects of supply chain management that are addressed mainly individually in the literature. Below we review the current literature on those single supply chain management topics that are significant for our model.

Multimodal replenishment

A key requirement in our project that has received considerable attention is multimodal replenishment, which was implemented as three modes: normal replenishment orders, planned rush orders, and unplanned back orders. Minner [7] distinguishes between two major categories of multimodal replenishment. One of the research streams assumes deterministic lead times and models emergency situations when the inventory is low. The far larger part of the research focuses on stochastic lead times and the statistical benefit of splitting orders, e.g., among several suppliers, to achieve a shorter effective lead time (the time until the first order arrives).

Deterministic lead times

Neuts [8] gives an optimal periodical replenishment policy with a critical stock level y^* and a fixed order quantity q. At the beginning of each period, q units are ordered using the slow delivery mode with a delay of one period. In addition, if the current stock is below the critical level y^*, an emergency order is triggered that immediately brings the stock back to y^*.

Scarf [9] describes the fundamental results of the very common (s, S) policy (when the stock level drops to s, replenish up to S), which requires a known demand distribution (which can change over time). It also assumes linear ordering, holding, and backlog costs and allows for fixed order costs. This standard (s, S) policy has been extended to dual-mode (normal and emergency) replenishment scenarios. For example, Veinott [10] describes a three-parameter policy (s, y^0, S) in which, at the beginning of each period, one or more orders are triggered if and only if the current stock x is below the critical level s. An emergency order of $\max(0, y^0 - x)$ units is immediately delivered, and a normal order with $[S - \max(y^0, x)]$ units is delivered one period later.

Fukuda [11] extends this approach to cases in which arbitrary delivery delays are allowed for normal and emergency orders. Whittemore and Saunders [12] describe conditions sufficient for ordering nothing by the normal or by the emergency channel, assuming arbitrary delivery times.

These and related approaches issue a normal and possibly an emergency order at the same point in time, which is in most cases induced by the periodical replenishment approach they follow. Triggering several orders at once is called *order splitting* in the literature, but it is usually related to stochastic lead times. Thus, the discussed approaches can be regarded as a conditional order-splitting policy with deterministic lead times, where the number of selected suppliers is not fixed but depends on the current stock level. In our case, these approaches are not suitable, primarily because of large transportation economies of scale and the wish of the customer for a more flexible approach.

Moinzadeh and Nahmias [13] extend the (s, Q) policy, in which a quantity Q is ordered whenever the current stock falls below s, to an approximately optimal (s_1, s_2, Q_1, Q_2) policy in which normal and emergency orders are not necessarily triggered at the same time. This approach assumes the usual linear cost factors, fixed ordering costs for both delivery modes, stochastic demand, and two arbitrary constant lead times $0 < \lambda_2 < \lambda_1$. Whenever the on-hand inventory falls below s_1, a normal order of Q_1 units is triggered and arrives after λ_1 time units. If at any later point in time the on-hand stock falls even below s_2, an emergency order of Q_2 units is triggered as long as its delivery (in λ_2 time units) will be prior to the arrival of the outstanding normal order. In addition to providing separate reorder points, this approach also differs from previous ones by considering only the on-hand inventory and not the total inventory position consisting of on-hand inventory plus ordered units minus backlogged items. Therefore, they have to restrict the number of open orders to one per type.

Moinzadeh and Nahmias [13] propose to first solve the simple (s_1, Q_1) inventory model and provide approximate formulas for the further procedure. Then an appropriate emergency reorder point s_2 and its order quantity Q_2 are found. Finally, they use s_1, s_2, and Q_2 to recalculate the normal order quantity Q_1. Their evaluation by extensive simulation shows that the reorder point s_1 in the dual-mode scenario is always lower than its counterpart in the regular (s, Q) policy for all investigated cases. Most benefits can be achieved if backlog costs are high. High fixed ordering costs lead to larger order quantities and

diminish the beneficial effect of this approach. Johansen and Thorstenson [14] extend this approach by using variable backlog costs in cases with Poisson distributed demand.

Stochastic lead times

Sculli and Wu [15] were among the first to show that splitting an order between two suppliers with independently normal distributed lead times reduces the reorder level and the buffer stock when compared with replenishment with only one supplier. Since then, many extensions and specializations have followed.

Ramasesh et al. [16] give a detailed comparison between sole sourcing and order splitting in the case of a regular (s, Q) replenishment policy with either uniformly or exponentially distributed lead times. They find that order splitting provides savings in holding and back-ordering costs, which increase if the demand volatility increases or the lead time distributions are skewed and have a long tail.

Furthermore, they divide the order-splitting approaches into two groups. First, macro studies analyze the effect of order splitting on the whole replenishment process, including the relation to the supplier. For example, a competition among the suppliers (producers) can lead to lower prices and better quality. Second, micro studies focus on the inventory perspective and savings induced by lower ordering, holding, and shortage costs.

Despite the attention given by the academic world to the concept of reducing lead time risk by order splitting, it has recently received considerable criticism by Thomas and Tyworth [17]. Regarding the micro focus, they argue that savings in holding and shortage costs are more than compensated for by increased ordering costs in reality. They see the gap between literature and reality mainly in the neglected transportation economies of scale and underestimated transporting costs. In a more macroscopic view, they question whether or not the savings from a reduced average cycle stock in one inventory are still valid or significant for the whole supply chain. Many approaches do not consider the in-transit inventory, which can lead to additional costs.

Consequently, Thomas and Tyworth suggest that future research should focus on other models of dual-mode replenishment, e.g., the cost performance differences in modes of transportation. With respect to our scenario, we can support many statements of Thomas and Tyworth, since the cost structure and practical replenishment constraints, such as transportation economies of scale, do not allow for an order-splitting approach.

In fact, the (s_1, s_2, Q_1, Q_2) approach of Moinzadeh and Nahmias [13], with its individual reorder point and order quantity for each order type (normal and emergency) and deterministic lead times, seemed most promising both to the experts at the customer and to us. Our model can be seen as a heuristic approach that extends the (s_1, s_2, Q_1, Q_2) model by stochastic lead times.

System-wide service level

Even though many companies apply the same target service level to all SKUs, they usually wish to achieve only a specific system-wide service level and do have some degree of freedom to set service levels for individual SKUs, which may be used to save costs. This requires the move from an isolated optimization for each SKU to an integrated and complex system-wide approach. Mitchell [18] developed an approximate model based on the generalized knapsack duality algorithm with fixed lead times, fixed and variable ordering and holding costs, and a periodic replenishment policy. By evaluating several 32-item inventories, he found substantial savings for the system approach for service levels of approximately 85 percent.

However, a system-wide approach still has to be feasible, even with tens of thousands of SKUs. Thonemann et al. [19] provide an easy-to-use approximation to quantify the benefits of a system approach when a system-wide service level should be reached, with minimal inventory costs under the assumption of $(S - 1, S)$ replenishment policies for each SKU, Poisson distributed demand, variable unit costs, and identical constant lead times. They look at the marginal increase of the service level per unit for each SKU. Starting from a minimum service level, they iteratively increase the stock level S by 1 for the SKU with the currently highest service-level increment per unit cost until the system-wide target service level is reached. Their analyses show that a system-wide approach benefits especially those inventories in which a small percentage of all units makes up the major part of the demand.

Hopp et al. [20] developed a heuristic that relates to us in many respects as it strives to cover a quite complex real-world inventory problem with many practical pitfalls and constraints. The inventory holds 30,000 SKUs, customer demand is random, and several supply sources exist. However, in their model they use a normal (r, Q) policy with only one supplier and assume constant lead times, Poisson distributed demand, and only fixed ordering costs. The optimization goal is to minimize the total inventory investment subject to constraints on the overall service level and the order frequency. Hopp et al. find simplified formulas for the inventory costs and the service level that enable a straightforward calculation of the two replenishment parameters r and Q once suitable Lagrange multipliers are available. Unfortunately, they could provide only numerical evaluations based on

subsets of the customer data because the model had not been operatively used by the company.

In our case, the overall service level of more than 100,000 SKUs has to be calculated, and a system-wide service level greater than 95 percent has to be reached. Moreover, additional complexity is encountered because assumptions about a specific demand distribution and constant lead times do not hold, and there is a need to incorporate a complex cost structure and emergency orders into the model.

Safety stock with sparse and intermittent demand

A primary challenge of spare parts replenishment is the sparse and intermittent demand for them. While this fact influences many facets of replenishment planning, it has a particularly strong impact on the accurate calculation of safety stock levels. Textbook approaches for calculating the relationship between service levels and safety stocks based on normal (Gaussian or Poisson) distributions generally cause the measured service levels to deviate significantly from the target service levels [1, 21].

Strijbosch et al. [22] compare two different (s, Q) policies in which the main difference lies in the modeling of the demand distribution during the lead time. Their simple approach assumes the demand during the lead time to be a normal distribution, while the advanced approach differentiates between no demand and positive demand during the lead time. The positive demand is modeled by means of a gamma distribution. By adapting the reorder point appropriately to the demand distribution parameter, the advanced model is able to match the desired service level in many situations in which the simple approach is not consistent and leads to stock levels that are too high.

Boedi and Schimpel [1] arrive at similar findings and give a more general model in which all of the SKUs in an inventory are clustered according to their normalized positive demand distribution. For each cluster, a best-fitting continuous distribution from a set of common distribution types is calculated. This cluster-wide distribution is rescaled for each SKU separately and allows for an individualized safety stock. Extensive evaluations with real data from ten different warehouses show that the average service level is much closer to its target value than for a model that assumes normal distributed demands. Moreover, the variance of the achieved service levels was significantly reduced. These effects become especially apparent for service levels greater than 95 percent.

Although these approaches have already been successfully applied, our customer's time constraint did not allow for such a calculation-intensive solution. In addition, the models given would have to be extended to fit the dual-mode replenishment scenario, which would interfere even more with the time constraint given by the customer. Hence, a less time-intensive simulation technique is applied to determine the mapping between service levels and safety stocks.

Stocking decision

A common problem for inventories in the spare parts business is the extremely low demand for the majority of SKUs; for some, the average demand can be less than one unit per year. Keeping only one unit in stock for those SKUs can, in total, lead to substantial inventory holding costs. Here the question arises, When is it more beneficial not to stock a SKU and instead have it delivered in a special supply mode? The fundamental paper by Croston [23] shows that, especially for a spare part with a high coefficient of variation in demand or in a situation in which unsatisfied demand is relatively inexpensive compared with holding cost, it is preferable not to stock this SKU. This effect is intensified by large time intervals between successive demands. Croston's model uses a periodic replenishment policy with holding and ordering costs and additional costs for nonsatisfied demand. There exist several extensions and modifications of this model. Our model is basically a modified version of this approach, extended by a more complex cost structure. Moreover, we make this stocking decision an integral part of the budget optimization, and not an independent decision for each SKU.

Service-level relations for normal and rush orders

Several preliminary steps are necessary to enable fast and efficient calculations by the core part of the optimization described below in the budget optimization section. These preparations are the subject of this section.

For each SKU, two relations are needed. One addresses the normal orders and how the service level relates to the safety stock. The other relation associates the service level with the reorder point for the planned rush orders. We do not assume certain demand patterns that would allow for an analytic solution because this could lead to very inferior results for spare parts, as discussed above. Instead we use the DIOS built-in simulation engine to generate these relations on the basis of the individual demand patterns for each SKU.

First we derive the input parameters that are essential for the budget optimization. We then describe the procedure for obtaining good approximations of the two service-level relations mentioned above with only limited numbers of simulations.

Input parameters for the budget optimization

A typical regional warehouse sells more than $n = 100{,}000$ different SKUs. These SKUs are divided into several

service-level groups. The objective is to reach a system-wide service level for each service-level group with minimal total cost, which is the sum of ordering, warehousing, and transportation costs. Using these goals and constraints, the budget optimizer optimally assigns an individual target service level sl_i and its associated total costs $c_i(sl_i)$ to each SKU $i \in \{1, \cdots, n\}$ and balances them with all other SKUs $j \in \{1, \cdots, n\}$. The same holds for a service-level maximization problem under a given budget constraint. In both cases it is essential to know the cost function $c_i(sl_i)$ within the possible range $(sl_{i\_\min}; sl_{i\_\max})$ of sl_i for each SKU $i \in \{1, \cdots, n\}$.

The total costs $c_i(sl_i)$ depend on the joint service level sl_i of normal and rush orders, which themselves depend on one hand on given parameters, such as demand and lead-time distributions, and on the other hand, on variable replenishment parameters, such as order quantities and reorder points. This issue is addressed now, as we translate the total costs into parameters related to the variable replenishment process.

While ordering costs depend only on the order quantity and the (forecasted) demand, the back-order and stock-holding costs are linked to the service level and thus to the safety stock.

In summary, the calculation of the total costs for each SKU is determined by four components: cost factors for ordering (or back-ordering) and stock-holding, order quantity, safety stock, and service level.

We assume that all cost factors are fixed exogenous variables and that the order quantity has been calculated beforehand on the basis of the fixed cost factors. Thus, all of these parameters are not subject to change and are forwarded directly to the budget optimizer as input parameters. The only variable parameters left are the safety stock and service levels for normal and rush orders. Their relations must be calculated prior to the budget optimization.

Service level and safety stock are related over the demand distribution D_{D} and the lead-time distribution D_{LT}. Moreover, the safety stock ss can be expressed by an equivalent reorder point p based on the average lead time μ_{LT} and the average demand μ_{D} by the equation $p = ss + \mu_{\mathrm{LT}} \cdot \mu_{\mathrm{D}}$. In other words, the expected minimal stock level (just before a new order arrives) is exactly ss.

While the parameters μ_{LT} and μ_{D} might be approximated from historic data, in practice no analytic formulas for the exact continuous distributions D_{D} and D_{LT} are known, and even suitable approximations to the demand distributions are difficult to calculate accurately. The complexity of the problem increases further when we consider two types of orders—normal and rush. A rush order differs from a normal order by different costs and by a different lead time distribution with a smaller average value. In such a dual-mode replenishment scenario, neither the joint service level nor the relation between safety stock and service level can be calculated analytically in a reasonable amount of time. The main challenge has been to find a fast and efficient algorithm that obeys two constraints: The calculation time should be less than 60 minutes for a typical warehouse (e.g., 100,000 SKUs), and the output data size should be small so that the result can be read rapidly by the budget optimizer in order to allow analysis on a regular desktop or laptop.

Algorithm to calculate the relationship between safety stock and service level

The key task of the algorithm is to relate the safety stock for normal and rush orders to their resulting joint service level for each individual SKU. Analytic formulas for the demand distributions are unknown; in principle they can be of arbitrary type and shape. A possible solution is to make analytical approximations of the demand distributions from the historic data. However, first experiments showed that the calculation of such approximations is too time-consuming. The same is true if the simulation is used to approximate the joint service level for all combinations of reorder points of normal and rush orders.

Thus, we decided to use two separate Monte Carlo simulations to determine the safety-stock and service-level pairs individually for normal and rush orders by using the historic demand data. Logically, both service-level components (for normal orders and for planned rush orders) contribute to the target service level reached by a SKU. The section below on the heuristic to combine normal and rush order service levels describes how the separate results are combined to achieve this joint service level.

Note that a safety-stock and service-level pair is completely equivalent to its associated reorder-point and service-level pair because of the one-to-one relationship between the reorder point and the safety stock of a SKU. We refer to both pairs as *result pairs*. The entire simulation process comprises three steps:

1. Simulation of the result pairs for normal orders.
2. Simulation of the result pairs for rush orders.
3. Calculation of the joint service level by a heuristic formula.

All simulations are performed by the DIOS built-in simulator. This simulation engine was developed by the authors and is used in DIOS to simulate the expected service levels on the basis of chosen replenishment parameters. In the next three subsections, we describe the simulation of a single result pair, show the control process that determines the next safety stock (reorder point) to be

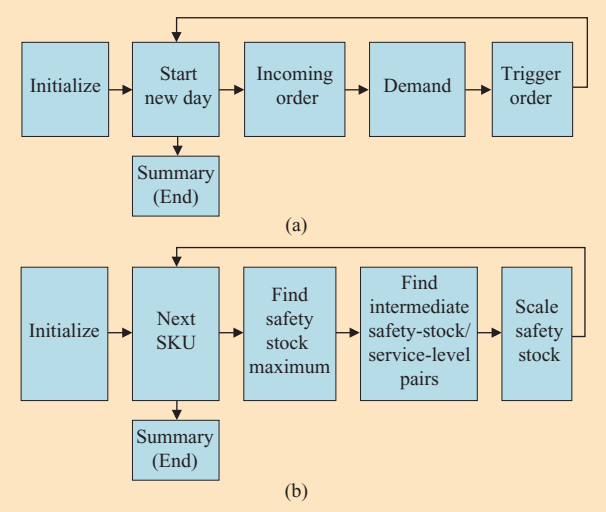

Figure 2

Replenishment simulation: (a) Process of simulating a replenishment policy; (b) control process for the replenishment simulation.

simulated, and show the calculation of the joint service level.

Replenishment simulation

Applying an inventory simulation requires knowledge about the warehouse replenishment policy. The standard policies implemented in DIOS are the two continuous (s, Q) and (s, S) policies and the periodical (T, S) policy. Moreover, it must be specified whether unsatisfied demand is backlogged or lost. The standard simulation process is shown in **Figure 2(a)**.

Initialization phase—During initialization, all parameters (e.g., order quantity, reorder point, service level, average lead time, and lead time fluctuation) are set. The initial stock is set to the average expected stock level ($safety\ stock + 0.5 \cdot order\ quantity$). To make the simulation as realistic as possible, outstanding orders are created. The latter is especially important for long lead times. Otherwise, the stockouts generated during the simulation until the arrival of the first order (which would then be triggered on the first day of the simulation) would lead to a very poor service level and a correspondingly high safety stock, which is unrealistic. For statistical reasons, the simulation horizon covers a period of five years on a daily basis, which leads to sufficiently stable service levels. The demand time series are repeated if the historical sales data covers less than five years.

Daily simulation phase—Whenever the simulated time exceeds five years, finalizing calculations are done as described below. Otherwise, a new day is started. For each simulation day, possible incoming orders are first added to the available stock; then the demand for this day is determined. The demand is next compared with the available stock, and possible stockouts and backlogs are remembered. Whenever the available stock plus the SKUs from outstanding orders falls on or below the reorder point, the succeeding step, *trigger order*, issues a new order according to the defined replenishment policy in the trigger order step, and an appropriate lead time is generated using the lead time and lead time fluctuation information.

Summary (end) phase—After the simulation has reached the five-year simulation horizon, the defined service level is calculated from the information about stockouts and total demand or replenishment cycles. There exist many different types of service levels (e.g., see [24, 25]), most of which are available in DIOS as well. The service level we use in this case is defined as the number of customer order lines that can be satisfied without delay. This is also referred to as the *order line fill rate*. The calculated service level is returned to the control process and associated with the reorder point used throughout this five-year simulation run.

The multistage simulation process described above is performed for each reorder point that has been determined by the control process.

Control process for the replenishment simulation

Recall that we have to determine the result pairs with each consisting of a safety stock level and its associated service level. These are used by the budget optimizer to quickly evaluate the effect that a changed reorder point (or safety stock) of a specific SKU has on the system-wide service level and total costs. Thus, a large sequence of possible result pairs must be provided for each single SKU.

The control process creates this sequence of result pairs for each SKU individually by determining the sequence of reorder points (or safety stocks) which serves as an input to simulate the appropriate service levels. From a theoretical view, the maximum safety stock $ss_{\max}$ for both order types has to be restricted to a maximum service level close to 100 percent ($sl_{\max} = 1 - \gamma$), with $0 < \gamma < 1$ when applying unbounded demand distributions. However, in practice, using discrete historical demand data, we can safely set $sl_{\max} = 100$ percent. Depending on further restrictions, there are several possibilities for the minimum value. Two intuitive examples are to simulate the minimal service level $sl_{\min}$ by setting the safety stock $ss_{\min} = 0$ or by setting the reorder point to 0 (equivalent to a nonpositive safety stock). In our approach, we set $ss_{\min} = 0$ for normal orders and the minimum reorder point to 0 for rush orders. Moreover, all safety stocks or reorder points are integer values.

The objective is to return the result pairs for the range $(ss_{\min}; ss_{\max})$. The process is shown in **Figure 2(b)**.

Initialize and next SKU—In the initialization, all SKUs are selected for which the result pairs should be determined. Moreover, the array of pairs is reset. The process terminates with a summary step after all SKUs have been processed. Otherwise, the result pairs of the next SKU are simulated.

Find safety stock maximum—The first two simulation runs for each SKU use a safety stock of $ss_{\min}$ and $ss_{\min}+1$, respectively. Intuitively, the service level should increase monotonically with an increasing safety stock. Whenever this is not the case (e.g., due to anomalies despite the long simulation horizon), the current service level sl_{new} is set to sl_{old}, the service level of the previous simulation run.

The safety stock is doubled, i.e., $ss_{\text{new}} = 2ss_{\text{old}}$, by each simulation run until $sl_{\max}$ is reached. For a safety stock value $ss > ss_{\min}+1$ there might exist a value $ss' < ss$ that also leads to $sl_{\max}$. Thus, we try to find the smallest safety stock $ss_{\min\_\max}$ that still leads to $sl_{\max}$. This is done by the well-known divide-and-conquer technique on the last interval $(ss_{\text{old}}, ss_{\text{new}})$, where $ss_{\text{new}} = 2ss_{\text{old}}$. When $ss_{\min\_\max}$ is found, the process moves on to the next step.

All result pairs $t_i = (ss_i, sl_i)$ are stored in the result array $T = (t_1, t_2, \cdots, t_m)$, where $I \in \{1, \cdots, m\}$, $t_1 = (ss_{\min}, sl_{\min})$, and $t_m = (ss_{\min\_\max}, sl_{\max})$. In our case, a significant percentage of all SKUs (i.e., >20 percent) reaches the maximum service level $sl_{\max}$ with a safety stock $ss_{\min\_\max} < 10$. This is quite common for the spare-parts business.

Find intermediate safety-stock/service-level pairs—To speed up calculations and make analysis feasible on a regular desktop or laptop, the number of result pairs should be small. Knowing that the service level is rising monotonically, we eliminate all pairs t_i with $I \in \{2, \cdots, m-1\}$ that can be expressed by their neighboring pairs t_{i-1} and t_{i+1} by linear interpolation plus or minus a small allowed absolute deviation ε.

Before pairs are eliminated, it has to be ensured that all consecutive pairs $t_i, t_j \in T$ with $i \in \{1, \cdots, m-1\}$ and $j = I+1$ appropriately represent the pairs $t_k \notin T$ in the interval (ss_i, ss_j) between them. Therefore, the service level sl_k of pair t_k with $ss_k = 0.5(ss_i + ss_j)$ is simulated. If sl_k lies outside the linear interpolation line $\pm\varepsilon$, the pair t_k is added to T between the entries t_j and t_i. In addition, a divide-and-conquer method is applied on the intervals (ss_i, ss_k) and (ss_k, ss_j) until their subintervals are appropriately represented or ss_i and ss_j are neighboring integers.

Once all intervals (ss_i, ss_j) are represented appropriately by their enclosing result pair $(t_i, t_j) \in T \times T$ with $j = i+1$, all result pairs $t_k \in T$ that lie within the ε-range around the regression line specified by their neighboring result pairs $t_{k-1}, t_{k+1} \in T$ are eliminated from T. In addition, all pairs $t_j \in T$ with $j \in \{2, \cdots, m\}$ are eliminated from T that have the same service level as their predecessor $t_{j-1} \in T$. The output of this step is a compacted result array T. Note that with sparse and intermittent demand, the safety-stock and service-level graph is much more a step function than a strict monotonic function.

Scale safety stock—An open problem is how to adjust the safety stock when the average historical demand differs significantly from the forecasted value. Under such a condition, the safety stock may not be sufficient for a strongly increasing demand, or it is much too high for a SKU at the end of its life cycle. This safety stock depends strongly on the variance of the future demand. For example, if the demand changes by a factor r, this can lead to two extreme cases: In the first case, the units of each single pick are scaled by r; in the second, the number of picks is scaled by r. Both cases require a completely different safety stock.

Because of the lack of insight (in practice) into the details of the demand change, we used the simplified approach to scale the safety stock for each SKU with the square root of r, which proved to give acceptable results.

Summary (*end*)—All compacted and scaled arrays containing the result pairs are written to a plain file and serve as input for the budget optimizer calculations.

This process over all SKUs is executed twice, first with the parameters of the normal order and then with the parameters of the rush order.

Heuristic to combine normal and rush order service levels

The budget optimizer retrieves the result pairs from the simulation and calculates the expected service level (by interpolation) separately for normal and rush orders. However, the question remains how the budget optimizer combines these two service levels, sl_{N} and sl_{R}, into an expected joint service level sl_{target}. Our heuristic to calculate the combined service level sl_{target} is described below and shown in **Figure 3**. It works acceptably in practice with the following assumptions and practical restrictions:

- The reorder point for the rush order is below the safety stock of the normal order.
- A rush order is triggered only if it arrives before the next normal order.
- A rush order cannot be triggered before a normal order has been issued.
- There are separate inbound order queues for normal and rush orders.
- The probability of triggering a rush order is uniformly distributed within the allowed time window (green area).

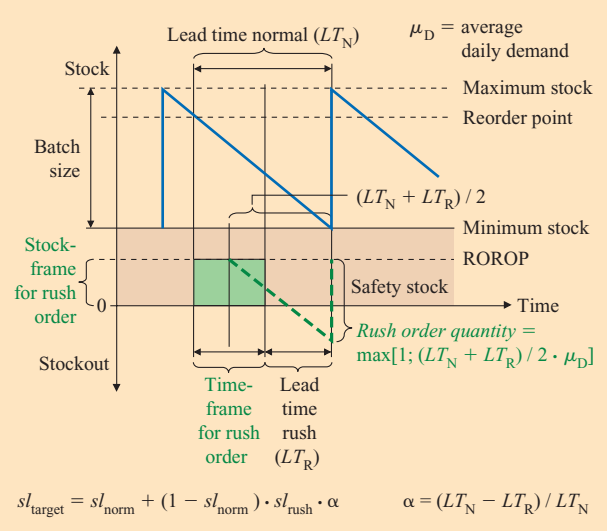

Figure 3

Heuristic used to approximate the joint service level.

Let us denote the event "rush order occurs" as *RO* and the event "stockout occurs" as *SO*. This leaves a time window $\Delta_R = LT_N - LT_R$ for triggering a rush order, where LT_N is the normal lead time and LT_R is the lead time of a rush order. Assuming a uniform distribution, the probability $p(RO)$ that a rush order will be triggered during the remaining lead time of the last normal order is

$$p(RO) = \frac{LT_N - LT_R}{LT_N}.$$

Thus, with a probability of $p(RO)$, the rush order will positively affect the stockout probability $p(SO)$ of the normal order. Knowing that $p(SO) = (1 - sl_N)$, where sl_N denotes the service level of the normal order, and that the positive effect of the rush order is related to its service level sl_R, the approximated joint service level sl_{target} can be given by

$$\begin{aligned} sl_{target} &= sl_N + (1 - sl_N) \cdot sl_R \cdot p(RO) \\ &= sl_N + (1 - sl_N) \cdot sl_R \cdot \frac{LT_N - LT_R}{LT_N}. \end{aligned}$$

The question remains, How many units should be delivered by a rush order? While there are many plausible explanations (e.g., considering ordering and storing costs), we use rush orders as a temporary support with the intention of avoiding short-term stockouts. Thus, in our interpretation, the rush order should simply cover the expected demand that will occur during the average (expected) time until the next normal order arrives.

Given the time window Δ_R with a uniform distribution for the point at which a rush order will be triggered leads to an average expected time until the arrival of the next outstanding normal order of $0.5 \cdot (LT_N - LT_R)$. Moreover, the minimum order quantity is 1, which is important for SKUs with very little demand. As a consequence, the rush order quantity is

$$\max(1, \lceil \mu_D \cdot 0.5 \cdot (LT_N - LT_R) \rceil).$$

Budget optimization

Given the optimal order quantities, corresponding pack size, and service-level relations for safety stock and ROROPs, the budget optimizer can perform two different tasks: It can minimize the budget for given group service levels, and it can maximize service levels for a given budget.

The first task is done automatically in the operational system implemented at the automobile manufacturer, where our system is used as a black-box optimizer. The second task is used in what-if analyses to determine how high the service level can be raised without overly increasing costs.

The budget comprises the sum of all of the costs listed in **Table 1**. This budget is part of the objective function to be minimized. The objective function further contains various penalties for not reaching certain goals (weak restrictions). These are penalties for not reaching the target service levels and penalties for the number of SKUs that change their StockYN status and their safety stock and rush order reorder points. The latter two are important for stabilizing the solution over time. This is described further in the section on solution stability.

Budget optimization influences

The goal of the budget optimization is to set an individual target service level SL_{target} for each SKU in such a way that system-wide service-level targets are met. The term *system-wide* can relate either to the warehouse as a whole or to disjoint groups of SKUs. In the latter case, each group has its own system-wide service-level target. The individual target service level SL_{target} is a combination of the safety stock and ROROP. The optimization is influenced by planning when the following parameters are set for each SKU:

1. StockYN decision: Whether or not to stock the SKU.
2. The safety stock (only for stocked SKUs).
3. The ROROP (only for stocked SKUs).

These parameters are changed in the optimization process described below. Each such change, like toggling the StockYN from "Stock" to "Do-not-Stock" for a SKU or increasing the safety stock from 1 to 3 for a SKU, is called a *mutation*.

Mutation 1: Changing the stock status

For some SKUs, the StockYN setting is fixed. This means either that the SKU cannot be stocked or that it must be stocked. In those cases, the budget optimizer is not allowed to change this parameter. The parameter is realized through the status variable StockYN for each SKU, which can take the following values:

- *YA* Currently stocked and can be changed by the optimizer.
- *YM* Currently stocked and cannot be changed by the optimizer.
- *NA* Currently not stocked and can be changed by the optimizer.
- *NM* Currently not stocked and cannot be changed by the optimizer.
- *EMP* A new SKU and the optimizer will determine its initial stock status.

The SKUs with *YM* and *NM* cannot change their StockYN status. This is a hard restriction in the optimization, which means that it will not be changed during the optimization. For SKUs changing their StockYN from *YA* to *NA* or from *NA* to *YA*, a penalty is added for stability reasons (see the section on solution stability below). This is not the case for SKUs with a stock status StockYN equal to *EMP*. For these SKUs, the status after the optimization will be *YA* or *NA*, but in neither case does a penalty occur.

The change of the StockYN status is one of the three mutations in the optimization process (see below). One of the budget optimizer options is "do not change the StockYN status" for any SKU. This is simply realized by not calling that mutation during the optimization process.

Mutation 2: Changing the safety stock

Before the budget optimizer is called, the relation between the safety stock and the service level is calculated for each SKU (see the previous section). The relation is nonlinear but monotonically increasing, so a higher safety stock automatically means a higher service level. In particular, the service level for a safety stock of zero units is known. This is the minimum service level of a SKU. On the other hand, the maximum safety stock is known for which the SKU has a 100 percent service level (according to the numerical simulation). The safety stock can take any integer value from 0 to this maximum. If a SKU has a minimum service-level requirement, the budget optimizer determines the minimum safety stock for which this minimum service level is reached and limits the safety-stock range for that SKU. Thus, this minimum service-level requirement is implemented as a hard restriction and is automatically fulfilled. Since a minimum service level can be kept only for stocked SKUs, for such a SKU the stock status is set to *YM* during the optimization, even if it was *NM*, so that a minimum service-level requirement can overwrite the "don't stock" requirement.

Table 1 Cost parameters used by the budget optimizer.

Budget cost component description	*Variable*
For optimal pack size: Fixed handling cost portion (per order line)	a_1
For optimal pack size: Variable handling cost portion (per order unit)	a_2
Extra handling cost for planned rush order (per order line)	c_{rush}
Extra handling cost for back order handling (per order line)	c_{back}
Transport costs per kg for normal orders	t_{normal}
Transport costs per kg for planned rush orders	t_{rush}
Transport costs per kg for back orders	t_{back}
Transport costs per m^3 for normal orders	tv_{normal}
Transport costs per m^3 for planned rush orders	tv_{rush}
Transport costs per m^3 for back orders	tv_{back}
Stock holding rate including capital interest rate	I
Warehouse space costs per m^3	c_{v}

The change of the safety stock is another mutation in the optimization process. This change is always allowed. However, an overall penalty that is applied to the sum of all safety-stock changes can be activated to limit the total number of safety-stock changes compared with the actual safety stock. This is implemented for stability reasons and is described further in the section on solution stability below.

Mutation 3: Changing the ROROP

As described above in the section on the algorithm for calculating the relationship between the safety stock and service level, the target service level for each SKU is calculated from one portion representing normal orders for which a safety stock is kept and a second portion representing planned rush orders for which a ROROP is set. Thus, changes in the ROROP will change the target service level for a SKU. Like the relation between safety stock and service level, a relation also exists between ROROP and service level. This relation is monotonically increasing as well. If the ROROP is not used at all, the target service level is due solely to the safety stock. If a ROROP is used, it can vary between zero and the safety

Table 2 Goals and constraints.

Set 1 (Meet service level)	
Setl_Gl	Total costs (budget) are minimized.
Setl_C1 W	Overall group service levels SL_{target} are met.
Setl_C2 W	Number of SKUs that change StockYN status is limited.
Setl_C3 W	Changes in safety stock and/or ROROP are limited.
Setl_C4 H	Minimum values for certain SKUs (SL_{min}) are met.
Setl_C5 H	ROROP is smaller than safety stock for all SKUs.
Setl_C6 H	SKUs with fixed StockYN status keep the status during optimization.
Set 2 (Meet budget)	
Set2_Gl	Overall service level SL_{target} is maximized.
Set2_Cl W	Total costs (budget) is reached.
Set2_C2 W	Number of SKUs that change StockYN status is limited.
Set2_C3 W	Changes in safety stock and/or ROROP are limited.
Set2_C4 H	Minimum values for certain SKUs (SL_{min}) are met.
Set2_C5 H	ROROP is smaller than safety stock for all SKUs.
Set2_C6 H	SKUs with fixed StockYN status keep the status during optimization.

stock of the normal order. Theoretically, this restriction does not have to be employed. However, for practical reasons (so that the people doing planning understand it clearly), this has been set as a hard restriction. Changing the ROROP is the third mutation in the optimization process. It makes sense only for stocked SKUs. When the safety stock is changed and the ROROP is larger than the safety stock, it is automatically reduced. The ROROP can also be set to −1 during the optimization, which means that no ROROP is used. (This is different from a ROROP of 0, because in that case, a rush order is placed as soon as the stock level reaches 0.)

Optimization process

Metaheuristic approach

Budget optimization is a nonlinear optimization problem, as the relation between the safety stock and service level is highly nonlinear. As such, linear or mixed-integer programming is not suitable for these types of optimization problems. Instead, an evolution-type optimization algorithm, the threshold-accepting algorithm, was chosen (see [26]). This general-purpose optimization algorithm was developed by the IBM Science Center in Heidelberg in the late 1980s. The algorithm is very similar to the well-known simulated annealing algorithm described in [27] and [28], but its acceptance rules are different, and the threshold accepting leads to more stable results in a number of experiments, as described in [26] and [29]. Threshold accepting has been successfully applied to very different optimization tasks ranging from simple traveling-salesman problems through production and personal scheduling to distribution planning. The generic optimization approach works as follows:

Choose an initial configuration and threshold $T > 0$
Repeat
  Choose a new configuration
  Compute $\Delta E :=$ *quality*(*new configuration* − *old configuration*)
  IF $\Delta E > -T$ THEN *old configuration* := *new configuration*
  Lower T
Until T is low enough.

The only difference from the simulated annealing approach is the acceptance rule: In threshold accepting, the new solution is accepted when its quality is reduced by no more than T. In simulated annealing, the new solution is accepted with a probability of exp $(-T)$.

Elements required for the optimization

A number of elements are required for the budget optimization to be able to utilize this algorithm, as described in the following paragraphs.

Modeling of goals and constraints—Depending on the optimization mode (meet target service level or meet budget), two sets of goals and constraints are used, as shown in **Table 2**. The constraints are partly hard-coded (H), which means that they are automatically fulfilled by the implemented algorithm, and partly coded as weak restrictions (W), which means that they can be violated during the optimization process, but then a penalty of the amount of the violation is imposed.

A comparison between the two sets shows that C2–C6 are the same; only C1 and G1 are interchanged. This is a multicriteria goal which is treated as a weighted sum of the goal G1 and the weak constraints C1–C3. This approach entails the problem of having to scale various components of the objective function so that a system-wide optimum is found. The scaling issues are nontrivial, especially in the case of G1 and C1, where costs and

service level compete. How this has been solved is explained below in the section on the automatic scaling of goals and constraints.

Data import and export—The budget optimizer is designed as a separate module that is implemented as a plug-in to DIOS. This means that an interface between the DIOS core and the budget optimizer has been realized which transfers the required master data and returns the optimization results (StockYN, safety stock, ROROP) for each SKU from and to DIOS. This is an internal interface; it is not visible to the user. The user calls the budget optimizer from within DIOS, and the results are shown in DIOS. Even during the optimization process, a status window and progress bar show how the optimization is advancing, and a user can cancel the optimization at any time from within DIOS.

Mutations—All mutations are applied on single SKUs and change the replenishment parameters of that SKU. The three mutations are the following:

- Mutation 1: Change the StockNY status of a randomly chosen SKU.
- Mutation 2: Change the safety stock of a randomly chosen stocked SKU.
- Mutation 3: Change the ROROP of a randomly chosen stocked SKU.

Some mutations interfere with others. For instance, if the safety stock is reduced (Mutation 2) and it falls below the ROROP, the latter must be reduced as well to account for constraint C5. Similarly, if a SKU changes the StockYN status from *YN* to *YA* (Mutation 1), an initial setting for safety stock and ROROP is required. Experiments have shown that the solution is most stable if this choice is made so that a cost minimum has already been reached for that particular SKU. The alternative would have been to choose safety stock and ROROP randomly within the allowed ranges. However, this may easily lead to inferior choices, which are likely to be rejected by the metaheuristic. Consequently, the SKU will not be stocked in the end, although it should be stocked in the optimal case. These considerations are critical to the success of local search metaheuristics, such as the threshold accepting used here.

Optimization control—The heart of the optimization using the threshold-accepting algorithm is the cooling process. This is the control of the acceptance of mutations. A mutation can lead to a better or worse solution. In the beginning of an optimization run, a high percentage of mutations that lead to a worse result is accepted. This is required in order to avoid the situation in which the optimization process finds a local optimum from which it cannot escape to find the global optimum. This is often compared with finding the highest point in a landscape: If one is only allowed to go up, one may become stuck on a small hill, whereas finding the highest mountain requires some descending between the peaks. This is exactly the principle behind the acceptance of worse results. During the optimization, this acceptance is gradually reduced. For this cooling control and for the control about "how often and when which mutation should be called," the IBM TOP-C optimization library has been used. This is a subroutine library that contains these control elements. The user enters only the number of steps (after each step the value of the threshold T is adjusted) and the number of mutations per step. Because the required number of mutations per step turns out to be roughly proportional to the number of SKUs, the number is adjusted such that it automatically scales with the number of SKUs.

The result of the optimization is the setting of the StockYN status for each SKU as well as a safety stock and ROROP for all stocked SKUs such that the goal (G1) is reached and the constraints (C1–C6) are met. This information is fed back into DIOS through the internal interface.

Automatic scaling of goals and constraints

The multicriteria goal of the budget optimization comprises four components:

1. Total costs, the budget (currency); range: $MIN\_BG - MAX\_BG$.
2. System-wide target service level (percentage); range: $MIN\_SL\% - MAX\_SL\%$.
3. SKUs with changed stock status (integer); range: $0 - MAX\_SKU$.
4. Changes in safety stock and ROROP (integer); range: $0 - MAX\_CHANGES$.

Here MAX_BG is the maximum cost defined for the case in which all SKUs that can be stocked would have the maximum service level of 100 percent. MAX_SL is the maximum service level that can be reached. In general, this is not 100 percent, since SKUs that cannot be stocked will reduce MAX_SL. MAX_SKU is the maximum number of SKUs that can change the stock status, and $MAX\_CHANGES$ is the maximum change allowed for the safety stock and the ROROP. Further, a MIN_BG is defined. This is the budget that would exist if each SKU could be replenished in a cost-optimal way regardless of the service level reached. And finally, a MIN_SL is defined as the system-wide target service level reached at MIN_BG. Although it is possible that the system-wide target service level may temporarily drop below MIN_SL during the optimization, this is not an optimal solution because it will have a lower service level at higher

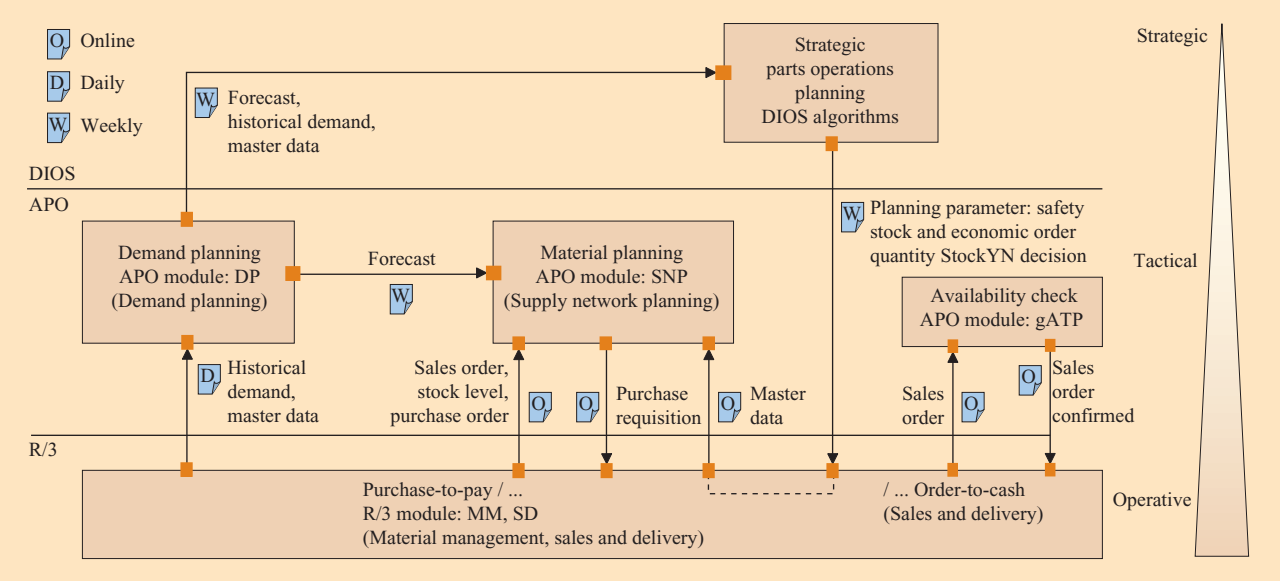

Figure 4

Integration of the IBM Inventory Budget Optimizer and DIOS in the SAP Advanced Planning and Optimization landscape.

costs, which does not make sense. The ranges can all be determined before the optimization starts.

All of the goal ranges above are different. They depend on the number of SKUs, the cost values, the currency, and the number of SKUs that cannot be stocked or must be stocked. However, the optimization is supposed to work independently of these ranges. For this reason, the components are all scaled so that they remain in comparable ranges. The most straightforward way of doing so would be to scale with the ranges. In that case, all components would range from 0 to 1. However, this does not work properly because the targets within the range cannot be reached equally easily. For example, if the target service level is just above the MIN_SL, it is easy to reach. If the target service level is close to MAX_SL, it is difficult to reach and leads to a large budget. Thus, the scaling of the components must be chosen in a nonlinear way. Several options were tested, and the following scaling was finally implemented, leading to very good results regardless of the target budget (TGT_BG) or the target service level (TGT_SL):

For the mode "Minimize budget, reach target service level,"

$Budget\ scale = \max\ [(MAX\_BG + MIN\_BG) \cdot 10^{-4}, MAX\_BG - MIN\_BG]$;

$Service\text{-}level\ scale = \max\ [(MAX\_SL + TGT\_SL) \cdot 10^{-4}, MAX\_SL - TGT\_SL]$.

For the mode "Maximize target service level, reach budget,"

$Budget\ scale = \max\ [(TGT\_BG + MIN\_BG) \cdot 10^{-4}, TGT\_BG - MIN\_BG]$;

$Service\text{-}level\ scale = \max\ [(MAX\_SL + MIN\_SL) \cdot 10^{-4}, MAX\_SL - MIN\_SL]$.

The max limitation is needed in order to avoid division by zero. The scaling for the other components is just set as the number of SKUs that can be stocked. The user can then still put an extra scaling in the range of 1 to 4 on top of the automatic scaling to fine-tune the stability of the optimization.

Solution stability

The optimization process using the threshold-accepting method is very fast, which is one of its major advantages. However, there is also a negative side: Solutions cannot automatically reproduce themselves. If the same initial data and parameters are taken but the optimization starts with another randomly generated number for the choice of a SKU, the final result will not be the same because of the stochastic nature of the optimization algorithm. The reason is not that the optimization result is inferior; on the contrary, the results are amazingly good in terms of how close they come to the optimum (see the results section). The reason is the existence of symmetries between SKUs. For example, given two SKUs A and B with very similar characteristics in terms of demand and value, in the optimal solution A is stocked but B is not. If the optimization run is repeated, B may be stocked and A may not. This type of behavior is correct in terms of the

optimum reached, yet it is unacceptable because it would cause too much stock movement and considerably reduce the acceptance of the optimization results.

Two different types of stability are required. The first states that the stock status should not change unless it significantly decreases costs. The second one requires that the safety stock and ROROP should change significantly only if the demand variation or other parameter changes justify it. These issues have been solved in an elegant way. The data provided to the budget optimizer contains the current StockYN status as well as the current safety stock and ROROP for each stocked SKU. Thus, it is possible to calculate the number of stock status changes and safety stock and ROROP changes. By putting a penalty on this, the symmetry mentioned above is broken, and if A is currently stocked but not B, this will remain so during the next run. Only in cases of significant improvement will the change be executed. It has been found that even a moderate penalty reduces the number of changes drastically.

There is a danger in this as well: A penalty that is too high causes a freezing of the solution, and it slowly drifts away from the true optimum. Such a penalty is actually misdirected. For instance, if one wishes to restrict the change of the stock status to two percent of the SKUs, this can be reached by increasing the penalty for StockYN changes sufficiently. However, if in the long run the natural fluctuation due to such elements as demand changes, new products, and old products requires a StockYN change at three percent of the SKUs, this increases the number of SKUs with a wrong stock status by one percent each time the optimization is executed. These issues have been tested and discussed in detail with the auto manufacturer, and a proper understanding and use of the penalty parameters has been achieved.

Integration of the budget optimizer into the SAP landscape

The DIOS solution exchanges data with an existing SAP R/3 and SAP APO system (**Figure 4**). Because there are a number of data elements that cannot be found in a standard R/3 or APO system (for example, the second reorder point and various cost items), these were incorporated into SAP as extension tables. There are two different use cases for DIOS: an automatic run (AUT), in which DIOS optimizes replenishment parameters on a weekly schedule, and a manual run (MAN), in which users can access DIOS at any given point in time to perform what-if analyses or to obtain more detailed optimization results than those DIOS transfers back to SAP.

Each (AUT) optimization run is divided into the following steps:

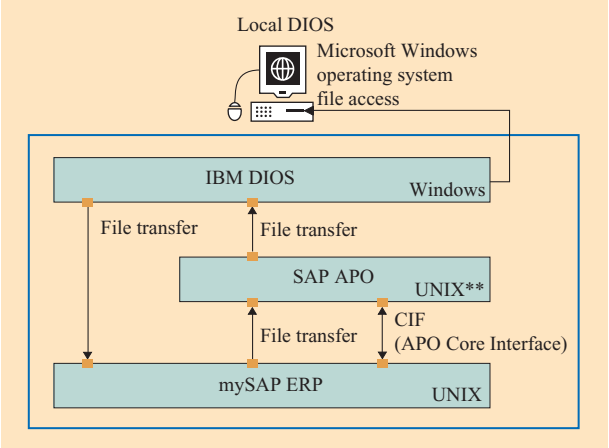

Figure 5

Data flow for the automatic run (AUT) use case.

A. Input data creation and export from SAP to DIOS.
   a. Creation of demand forecast within SAP APO DP.
   b. Upload of master data and forecasts from SAP to DIOS server.
   c. Upload of demand transaction data since the last upload (delta upload) from SAP to DIOS server.
   d. Upload of data in extension tables to DIOS server.
B. DIOS run.
   a. Data loading into DIOS from DIOS server.
   b. DIOS optimization run.
   c. Generation of optimization results.
C. Data import from DIOS to SAP.
   a. Export of the optimization results from DIOS.
   b. Import of the optimization results (delta download) into the SAP system.

For the (MAN) use case, steps A and C are not executed. **Figure 5** illustrates the data flow for the (AUT) use case. All file transfers are done using handshakes.

The data that is exported from SAP is kept on the DIOS Windows server so that authorized users can access the latest data at any time by using an installation of DIOS on their local machines (MAN use case). This allows planners to browse through the data DIOS is using and perform what-if analyses. However, for safety reasons a user is prohibited from exporting the results of such analyses and the associated optimization parameters to the SAP system. Instead, some SAP forms have been implemented that allow the alteration of many of these parameters and settings. In this way, user authentication

remains within SAP and need not be spread over several systems.

Results

Budget optimizer runtime

The runtime of the budget optimizer is relatively short. For about 70,000 SKUs it requires less than 20 minutes to finish the optimization on a T41 ThinkPad* running under Microsoft Windows XP**. Roughly half of the time is used for the preparation phase, which includes the calculation of the service-level relations and the calculation of the minimum and maximum budget. The rest is required for the optimization itself. This runtime is very short compared with the total planning process in the SAP Advanced Planning and Optimization system.

Optimization quality

The quality of the optimizer can be tested for two extreme cases: at the minimum budget and at the maximum service level. For these two situations, the budget and service level are known. The budget optimizer finds these solutions and deviates less than 0.1 percent from the optimum. For example, if the target service level is set to 99.999 percent, that target service level is indeed reached, and the corresponding budget is less than 0.1 percent away from the precalculated maximum budget. For target service levels between *MIN_SL* and *MAX_SL*, the real optimum solution is not known, of course. However, when the target service level is stepwise increased from *MIN_SL* to *MAX_SL*, it is found that the budget increases monotonically from MIN_BG to MAX_BG so that for lower values of the target service level, the budget increases only a little; for values of the target service level close to *MAX_SL*, it then increases rapidly to *MAX_BG*. This behavior is expected because of the high nonlinearity of the service-level relations.

Business results

The budget optimizer has been used for about two years, and the results are very satisfying. Here we discuss some of the major business results.

Of the SKUs sold through the RDC to dealers, 30 percent are stocked in the RDC. Weekly, one percent of those change their stock status. This leads to a continuous cleansing of the assortment. In the beginning, the customer was hesitant to follow the proposals concerning changed stock statuses, but it was gradually accepted as the positive influence on total costs and space occupation was realized. Having too many wrong SKUs in stock is an issue for most spare-parts warehouses. The continuous assortment adjustment solved this issue.

Service levels are measured continuously. It has been found that the measured service levels are very close to the target service levels: The differences are less than 0.2 percent. This confirms that the approach is correct, and in particular that the service-level relations, as described herein, give the correct availability results. Before the implementation of this solution, the service level was not met, yet stock was high and there were too many rush and back orders.

A stock savings potential of 30 percent can be reached. The customer has not yet fully realized this, since most spare parts are sold in small quantities, and therefore stock levels go down slowly, but it is expected that this will be the case in the next 12 months.

Further, there are a number of soft results. Acceptance of the solution continues to increase, because what-if analyses with the DIOS system answer many questions and thus remove many doubts. One example is given below. Planning spare-parts inventory remains a complex issue, and it is therefore important to work closely with the planners and respond to their questions and the issues they raise.

Here is one example of the strength and flexibility of what-if analyses with DIOS and the budget optimizer: The customer used to have simple rules for the StockYN decision. All SKUs that were sold three times or less in the last 12 months were not stocked, while all others were. These simple rules neglected the influence of the price of the SKU. Using the DIOS what-if analysis, it was shown that in order to reduce costs, many inexpensive SKUs with only one or two picks per year should be stocked. The obsolete risk for those SKUs is low because they are inexpensive, and the savings derive from avoiding back orders. On the other hand, expensive SKUs should be stocked only if the number of picks per year is much higher than three. After the results had been discussed in detail, the customer was convinced that the results of the budget optimizer could be trusted, and that the old stocking rule system should be replaced by the proposals of the new planning system.

Conclusion

The IBM spare-parts planning solution created for its automotive customer meets all of the requirements and thus fills the planning gap of SAP R/3 and APO. Further, it is much faster than originally expected, running in less than 20 minutes while one to two hours was promised as a maximum runtime. The specific requirements are typical for spare-parts planning and are certainly not restricted to this particular manufacturer. DIOS has good references across the industry, and the IBM Inventory Budget Optimizer is another important extension that can be of great value to a wide range of customers.

*Trademark, service mark, or registered trademark of International Business Machines Corporation.

**Trademark, service mark, or registered trademark of SAP AG, i2 Technologies US, Inc., Microsoft Inc., or The Open Group in the United States, other countries, or both.

References

1. R. Boedi and U. Schimpel, "Managing Risk Within Supply Chains: Using Adaptive Safety Stock Calculations for Improved Inventory Control," *Handbook of Integrated Risk Management for E-Business: Measuring, Modeling, and Managing Risk*, A. Labbi, Editor, J. Ross Publishing, Fort Lauderdale, FL, 2005, pp. 87–112.
2. B. Enslow, "IBM Puts a Pragmatic Face on Advanced Inventory Optimization," Aberdeen Group; see *http://www.aberdeen.com/c/report/research_briefs/RB_IBM_DIOS_BE_2915.pdf*.
3. i2 Service Budget Optimizer, i2 Technologies; see *http://i2.com/assets/pdf/PDS_service_budget_optimizer_v61_pds7242_072704.pdf*.
4. Inventory Chain Optimization (ICO), GAINSystems; see *http://www.gainsystems.com/*.
5. ToolsGroup, "What is Inventory Optimization?," white paper; may be accessed by filling out a free registration form at *http://www.toolsgroup.com/learning_more.html*.
6. ToolsGroup,"Recent Trends in Inventory Optimization," white paper; may be accessed by filling out a free registration form at *http://www.toolsgroup.com/learning_more.html*.
7. S. Minner, "Multiple-Supplier Inventory Models in Supply Chain Management: A Review," *Intl. J. Production Economics* **81/82**, No. 1, 265–279 (2003).
8. M. F. Neuts, "An Inventory Model with an Optional Time Lag," *J. Soc. Indust. & Appl. Math.* **12**, No. 1, 179–185 (1964).
9. H. E. Scarf, "The Optimality of (*S*, *s*) Policies in the Dynamic Inventory Problem," *Mathematical Methods in the Social Sciences*, K. J. Arrow, S. Karlin, and P. Suppes, Editors, Stanford University Press, Stanford, CA, 1960.
10. A. F. Veinott, "The Status of Mathematical Inventory Theory," *Manage. Sci.* **12**, No. 11, 745–777 (1966).
11. Y. Fukuda, "Optimal Policies for the Inventory Problem with Negotiable Leadtime," *Manage. Sci.* **10**, No. 4, 690–708 (1964).
12. A. S. Whittemore and S. C. Saunders, "Optimal Inventory Under Stochastic Demand with Two Supply Options," *SIAM J. Appl. Math.* **32**, No. 2, 293–305 (1977).
13. K. Moinzadeh and S. Nahmias, "A Continuous Review Model for an Inventory System with Two Supply Modes," *Manage. Sci.* **34**, No. 6, 761–773 (1988).
14. S. G. Johansen and A. Thorstenson, "An Inventory Model with Poisson Demands and Emergency Orders," *Intl. J. Production Economics* **56/57**, No. 1, 275–289 (1998).
15. D. Sculli and S. Y. Wu, "Stock Control with Two Suppliers and Normal Lead Times," *J. Oper. Res. Soc.* **32**, No. 11, 1003–1009 (1981).
16. R. V. Ramasesh, J. K. Ord, J. C. Hayya, and A. Pan, "Sole Versus Dual Sourcing in Stochastic Lead-Time (*s*, *Q*) Inventory Models," *Manage. Sci.* **37**, No. 4, 428–443 (1991).
17. D. J. Thomas and J. E. Tyworth, "Pooling Lead-Time Risk by Order Splitting: A Critical Review," *Transport. Res. Part E* **42**, 245–257 (2006); see *http://www.personal.psu.edu/faculty/d/j/djt11/papers/tre.pdf*.
18. J. C. Mitchell, "Multi-Item Inventory Systems with a Service Objective," *Oper. Res.* **36**, No. 5, 747–755 (1988).
19. U. W. Thonemann, A. O. Brown, and W. H. Hausman, "Easy Quantification of Improved Spare Parts Inventory Policies," *Manage. Sci.* **48**, No. 9, 1213–1225 (2002).
20. W. J. Hopp, M. L. Spearman, and R. Q. Zhang, "Easily Implementable Inventory Control Policies," *Oper. Res.* **45**, No. 3, 327–340 (1997).
21. M. Schwarz, "Die K-Kurven-Methode und Sicherheitsbestände," Master's Thesis, University of Hamburg, D-20146 Hamburg, Germany, 1999.
22. L. W. G. Strijbosch, R. M. J. Heuts, and E. H. M. van der Schoot, "A Combined Forecast–Inventory Control Procedure for Spare Parts," *J. Oper. Res. Soc.* **51**, No. 10, 1184–1192 (2000).
23. J. D. Croston, "Stock Levels for Slow-Moving Items," *Oper. Res. Quart.* **25**, No. 1, 123–130 (1974).
24. P. H. Zipkin, *Foundations of Inventory Management*, McGraw-Hill, Boston, MA, 2000.
25. H. Tempelmeier, *Inventory Management in Supply Networks—Problems, Models, Solutions*, Books on Demand GmbH, 22848 Norderstedt, Germany, 2006; ISBN 3833453737.
26. G. Dueck and T. Scheuer, "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing," *J. Comput. Phys.* **90**, No. 1, 161–175 (1990).
27. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science* **220**, No. 4598, 671–680 (1983).
28. R. W. Eglese, "Simulated Annealing: A Tool for Operational Research," *Euro. J. Oper. Res.* **46**, No. 3, 271–281 (1990).
29. G. Dueck, T. Scheuer, and H.-M. Wallmeier, "The C.H.I.P. Algorithm," *Technical Report 75.91.25*, IBM Germany, Heidelberg Scientific Center, D-69115 Heidelberg, Germany, 1991.

Received September 19, 2006; accepted for publication December 28, 2006; Internet publication May 23, 2007

Peter Korevaar *IBM Global Business Services, Gottlieb Daimler Strasse 12, D-68165 Mannheim, Germany (korevaar@de.ibm.com)*. Dr. Korevaar joined IBM in 1991 after receiving a Ph.D. degree in physics and astronomy from the University of Utrecht, The Netherlands, and completing a two-year postgraduate position at the University of Heidelberg, Germany. He began work on applying metaheuristic optimization techniques in logistics and developed the IBM Warehouse Site Planner, which optimizes two-echelon vertical supply chains (central warehouse to regional warehouse to customer). He began the development of the IBM Dynamic Inventory Optimization Solution (DIOS) in 1997. After two years, the DIOS development was transferred to the IBM Zurich Research Laboratory, Switzerland. In close cooperation with Dr. Boedi, the lead DIOS developer and coauthor of this paper, Dr. Korevaar worked on many customer engagements using DIOS and guided the ongoing development as a subject-matter expert on inventory optimization. The budget optimizer described in this paper was developed jointly by all of the authors. Dr. Korevaar now trains IBM consultants globally on DIOS and coaches project teams working on DIOS engagements worldwide.

Ulrich Schimpel *IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (uls@zurich.ibm.com)*. Mr. Schimpel joined IBM in 2004 after receiving an M.S. degree in information systems and management from the Friedrich-Alexander University, Germany. He is currently an external Ph.D. student at the University of Karlsruhe, Germany. He began working in the area of SCM, especially inventory management, and further worked on the development of IBM DIOS under the supervision of Dr. Richard Boedi. Mr. Schimpel also worked closely with his two coauthors in the budget optimizer customer engagement.

Richard Boedi *IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (rbo@zurich.ibm.com)*. Dr. Boedi joined IBM in 1998 after receiving his Ph.D. degree and Habilitation degree in mathematics from the University of Tübingen, Germany. He started work in the consulting group of IBM on continuing the development of the DIOS. He moved to IBM Research in 2000 and has been doing basic research and development for DIOS in close collaboration with Dr. Peter Korevaar. Dr. Boedi has worked on a large number of DIOS engagements and has given numerous internal and external education sessions about inventory management in general and DIOS in particular.