A shortest-path-based approach for the stochastic knapsack problem with non-decreasing expected overfilling costs

Troels Martin Range[a,b,*], Dawid Kozlowski[c], Niels Chr. Petersen[c]

[a] *Hospital of South West Jutland and Institute of Regional Health Research: Centre of South West Jutland, University of Southern Denmark, Finsensgade 35, DK-6700 Esbjerg, Denmark*
[b] *Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Alfred Getz veg 3, NO-7491 Trondheim, Norway*
[c] *Department of Business and Economics, and COHERE, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark*

Abstract

The knapsack problem (KP) is concerned with the selection of a subset of multiple items with known positive values and weights such that the total value of selected items is maximized and their total weight does not exceed capacity. Item values, item weights, and capacity are known in the deterministic case. We consider the stochastic KP (SKP) with stochastic item weights. For this variant of the SKP we combine the chance constrained KP (CCKP) and the SKP with simple recourse (SRKP). The chance constraint allows for a violation of capacity, but the probability of a violation beyond an imposed limit is constrained. The violation of the capacity constraint is also included in the objective function in terms of a penalty function as in the SRKP. Penalty is an increasing function of the expected number of units of violation with proportionality as a special case. We formulate the SKP as a network problem and demonstrate that it can be solved by a label-setting dynamic programming approach for the shortest path problem with resource constraints (SPPRC). We develop a dominance criterion for an elimination of states in the dynamic programming approach using only the deterministic value of items along with mean and variance of the stochastic weight of items corresponding to the associated paths in the underlying network. It is shown that a lower bound for the impact of potential extensions of paths is available as an additional means to limit the number of states provided the penalty cost of expected overtime is convex. Our findings are documented in terms of a computational study.

Keywords: Stochastic Knapsack Problem, Dynamic Programming, Shortest Path Problem with Resource Constraints

2010 MSC: 90C15, 90C27, 90C35

*Corresponding author
Email address: Troels.Martin.Range@rsyd.dk (Troels Martin Range)

1. Introduction

Suppose that we are given a set of items where each item has an associated positive value as well as a positive weight. The knapsack problem (KP) is then the problem of selecting a subset of these items such that the aggregate value of the selected items is maximized, but at the same time such that the aggregate weight does not exceed a predefined capacity. The KP has many variants and related problems. We refer to Martello & Toth (1990) and Kellerer et al. (2004) for a comprehensive treatment of the variants. If all items and parameters of the KP are known with certainty then the KP is deterministic, whereas if either the items themselves or their parameters are uncertain then the KP is referred to as a stochastic KP (SKP).

We investigate a variant of the KP where the values of the items are deterministic and the weights of the items are uncertain but with known means and variances of the weights. As a consequence of the uncertain weights we cannot guarantee that the predefined weight capacity is satisfied. In case it cannot be satisfied we say that the knapsack is overfilled and the excess weight is the overfill of the knapsack. There are at least two approaches to ensure that the knapsack does not get too overfilled. The first is to penalize the expected overfilling of the knapsack. This approach is often referred to as the stochastic knapsack problem with simple recourse (SRKP). The second approach is to utilize the distribution to ensure that the weight capacity is satisfied with a pre-specified probability. This is known as the chance constrained knapsack problem (CCKP). We focus on a stochastic knapsack problem where we penalize the expected overfilling of the knapsack and, in addition, ensure that the overfilling is no larger than a pre-specified limit with a given probability. Thus, we introduce a combination of the SRKP and the CCKP. In addition we extend the problem such that the items belong to disjoint classes in which at most one element can be selected.

The main contribution of the paper is the development of a shortest-path-based dynamic programming algorithm for solving this problem. The algorithm is based on the so-called shortest path problem with resource constraints (SPPRC) for which a label setting algorithm is applied. We develop an easily checked dominance criterion to enable reduction in the number of states of the dynamic programming algorithm and a state bounding procedure for the case where the cost of overfilling is a convex function. We test the approach on a large number of randomly generated instances and show that the state bounding procedure significantly increases the performance of the algorithm.

The KP often arises as a part of other problem types. The generalized assignment problem (GAP) is one particular problem for which the the KP is an integral part. The GAP has a number of items and a number of heterogeneous servers. Each server has a limited amount of time available to process items. The cost of processing a specific item on a specific server is given. Likewise, the amount of time the specific server requires to process the specific item is given. The GAP is then the problem of assigning each item to exactly one server such that the total cost is minimized and no server uses more time than

the predefined limit. This problem can be decomposed into a set partitioning problem and a number of KPs by the Dantzig-Wolfe decomposition, see Barnhart et al. (1998).

Our motivation for investigating this particular variant of the SKP comes from a decomposition of a real-life hospital scheduling problem where patients should be allocated to surgeries on future dates. On each future date within a planning horizon a set of surgeons are available. Each surgeon has an amount of available time for surgery on that date. The duration of a surgery of a specific patient conducted by a specific surgeon is stochastic with known mean and variance. A subset of patients to undergo surgery then has to be selected on that specific date for that specific surgeon. However, the surgeon is not interested in having overtime and the expected overtime is penalized by a non-decreasing function. For each date, surgeon, and patient combination a value of that patient having surgery by that surgeon on that day is given. Indeed, this corresponds to the variant of the stochastic knapsack problem we are investigating.

In the hospital scheduling case the stochastic knapsack problem only solves a part of the scheduling problem and is made an integral part of a larger framework. If the surgeons could choose freely which patients to operate on each day then we would have a set of independent stochastic knapsack problems where the surgery duration corresponds to the weight of the knapsack, the overtime corresponds to the overfill of the knapsack, and the value of having a patient undergoing surgery on a specific date with a specific surgeon is given. Obviously, the surgeons cannot choose freely, as each patient should be treated only once. Thus, the knapsack problems are dependent. Indeed, the problem becomes a GAP where each patient should be assigned a to specific surgeon on a specific date but also such that the patient is assigned exactly once throughout the planning horizon. This is the variant of the knapsack problem analyzed in this paper. A further description of the application to the surgery allocation problem is provided by Range et al. (2016).

The remainder of the paper unfolds as follows: In Section 2 we briefly describe related literature. In Section 3 we give the details on the model, and in Section 4 we describe the network-based solution approach. This is followed by a computational study in Section 5. Finally, concluding remarks and potential future developments are discussed in Section 6. Proofs of the propositions are given in the appendix.

2. Related literature

The deterministic KP has been widely studied in the literature. Kellerer et al. (2004) give a comprehensive introduction to the problem as well as a range of variants and generalizations. The focus in this introduction is on the deterministic variant of KP.

For the deterministic KP Gilmore & Gomory (1966) introduce a dynamic programming procedure and Toth (1980) further develops the dynamic programming approach for the problem. Frieze (1976) describes how to solve several variants of the minimizing integer KP by applying shortest path algorithms.

In order to solve the KP as a shortest path problem the author constructs an acyclic network with a node for each possible value of the sum of weights in the interval from zero to the weight capacity. In the case of the weights being integer the network has a node for each integer from zero to the weight capacity. An arc connects two nodes if an item has the weight corresponding to the difference between the node numbers. In this case the arc has a cost corresponding to the value of the item. Application of Dijkstra's shortest path algorithm yields a solution to the minimizing integer KP. Minoux & Ribeiro (1984) show that the equality constrained knapsack problem can be modeled as a constrained shortest path problem. More recently Figueira et al. (2010) apply labeling algorithms in networks to solve the multi-objective integer KP where weights are implicitly assumed to be integer.

While there is consensus in the literature on what the deterministic KP is, the stochastic variant of the problem comes in different flavors. This is indeed natural since the stochasticity can arise in several different parts of the problem. However, the most profound distinction is whether the knapsack problem is static or dynamic. The static variant assumes that all items are known – though their parameters are stochastic – before solving the problem. Hence the stochasticity lies in either the values, the weights, or the capacity. By contrast, the dynamic variant typically assumes that items arrive sequentially and that the parameters become known at arrival. For each arriving item it has to be decided (prior to the arrival of the next item) whether or not it should be included in the knapsack. Our focus is on the static variant of the stochastic knapsack. However, the reader is referred to Kleywegt & Papastavrou (1998), Kleywegt & Papastavrou (2001), Dean et al. (2008), and İlhan et al. (2011) for studies on the dynamic variant.

The static variant of the stochastic KP can be further subdivided based on where the stochasticity arises. In the first case, the values of the items are stochastic while the weights and the capacity are deterministic. This problem is sometimes referred to as the target achieving knapsack problem as a specific threshold value should be attained. Henig (1990) develops a dynamic programming approach for this problem, while Morton & Wood (1998) investigate both dynamic programming and integer programming procedures for a similar problem.

In the second case, only the capacity is uncertain. When there is no distributional information on the capacity Disser et al. (2017) construct approximation methods yielding bounds on the objective regardless of the capacity. It is shown that it is always possible to pack the knapsack in such a way that the aggregate value is no worse than a factor two of the optimal value for the case where the capacity is known. In the case where distributional information about the capacity is known Witchakul et al. (2008) investigate the problem and devise a heuristic approach to identify a solution. Merzifonluoğlu et al. (2012) apply a branch-and-bound algorithm to solve a model where stochastic capacity is a special case. Özaltin et al. (2010) solve a bilevel problem where a leader makes a decision affecting the capacity. However, the capacity is not only affected by the leader's decision but also by a random variable. The follower must next solve a knapsack problem with stochastic capacity in order to determine his or her

best decision.

The third case is when the weights of the items are stochastic while the values of the items as well as the capacity of the knapsack is deterministic. This is the case we are investigating in this paper. Two variants are prevalent: (1) the chance constrained knapsack problem (CCKP) and (2) the simple recourse knapsack problem (SRKP). The first is based on the notion that the knapsack constraint should be satisfied with a prespecified probability. This gives rise to so-called chance-constrained programming approach introduced by Charnes & Cooper (1959). Klopfenstein & Nace (2008) apply robust optimization techniques to approximate the solution for the CCKP whereas Kosuch & Lisser (2010) apply a branch-and-bound algorithm. The CCKP is used for scheduling problems where a violation of resource capacity is only allowed with a prespecificed probability, see for instance Hans et al. (2008) and Pinedo (2009). The latter variant penalizes the expected violation of the capacity constraint. Merzifonluoğlu et al. (2012) provide an extensive discussion of the SRKP when the item weights are normally distributed. Chen & Ross (2015) construct a heuristic search algorithm for the SRKP. The penalty per unit expected overfill is in these papers constant by assumption, i.e., the penalty function is linear in expected overfill. The penalty function is in the current paper non-decreasing in expected overfill with the linear function as a special case.

We suggest a solution approach based on a shortest path formulation of the stochastic knapsack problem. To be more specific, the formulation is based on the so-called shortest path problem with resource constraints (SPPRC) which is a generalization of the shortest path problem with time windows. The shortest path problem with time windows was discussed by Desrochers & Soumis (1988), who gave a permanent label setting (dynamic programming) algorithm based on the observation that time is strictly increasing and can be used to make extensions where the extended state would never be eliminated. In effect, they divide time into buckets no larger than the minimum amount of time required for an extension and then conduct the extension in increasing order of the bucket indices.

While time is an important part of many shortest path problems, it can be generalized into so-called resource constraints. Desaulniers et al. (1998) introduced the notion of resource extension functions (REFs) which are functions taking a resource state and computing the consequence on the resource state when it is extended along an arc. The authors show that if the REFs are non-decreasing and separable then a standard dominance criterion can be applied for elimination of states. Irnich (2008) further elaborates on the characteristics and properties of REFs. A general description of SPPRCs is given by Irnich & Desaulniers (2005).

The development of SPPRC algorithms has in many cases been an integral part in the development of solution procedures for more complex related problems. Desrochers et al. (1992) embedded SPPRC in a column generation approach to solve the Vehicle Routing Problem with Time Windows. This led to the so-called Elementary SPPRC, where paths are prohibited from having cycles even though the

underlying network contains cycles, and dynamic programming algorithms have been developed, see for instance Feillet et al. (2004), Chabrier (2006), and Righini & Salani (2006) and Lozano & Medaglia (2013). Gauvin et al. (2014) solve a vehicle routing problem with stochastic demands by column generation. They solve a shortest path problem related to the one we apply, where the cumulative expected demand and cumulative variance are discretized to create a state-space graph. We do not apply a discretization in our solution approach. Shortest path problems are also used in the context of staff scheduling or rostering where the underlying graph is acyclic as described by for example Eveborn & Rönnqvist (2004), Lusby et al. (2012), and Smet et al. (2016).

3. The extended stochastic knapsack model

Suppose that we are given the set of items, $\mathcal{J} = \{1, \ldots, J\}$, and with each item a value or a revenue, r_j, as well as a random weight, X_j, with expected weight, $\mu_j = \mathbb{E}[X_j] > 0$, and variance, $\sigma_j^2 = \mathbb{V}[X_j] \geq 0$.[1] Certainty regarding item weight prevails if the variance is zero, and the problem becomes deterministic if the variance is zero for all items. A capacity limit, T, on the total weight of the selected items is given. The expected amount with which the total weight violates the capacity limit is penalized by a non-decreasing function, $f : \mathbb{R}_+ \to \mathbb{R}$, having $f(0) = 0$. Furthermore, a maximum overfill, S, is allowed such that the probability that the weight $T + S$ is surpassed does not exceed a predefined level.

For each item $j \in \mathcal{J}$ we let $x_j \in \{0, 1\}$ be a binary variable indicating whether or not the item is included in the solution. We will denote a given solution $\overline{\mathbf{x}} = (\overline{x}_1, \ldots, \overline{x}_J)$. It is assumed that the stochastic variables, X_j, are independent, and as a consequence, we can rely on the central limit theorem to obtain an approximation of the distribution. The weight of the knapsack corresponds to the realization of the stochastic variables of the items included (i.e., $\overline{X} = \overline{x}_1 X_1 + \ldots, \overline{x}_J X_J$), and we can measure the mean fill of the knapsack for a solution, $\overline{\mathbf{x}}$, as

$$\overline{\mu} = \sum_{j \in \mathcal{J}} \mu_j \overline{x}_j$$

and the standard deviation as

$$\overline{\sigma} = \sqrt{\sum_{j \in \mathcal{J}} \sigma_j^2 \overline{x}_j} \quad .$$

We have by the central limit theorem that $\overline{X} = \sum_{j \in \mathcal{J}} \overline{x}_j X_j$ is approximately normally distributed with mean $\overline{\mu}$ and standard deviation $\overline{\sigma}$.

The remaining parts of this section describe the components of the SKP we solve. First, in Section 3.1, we discuss how to approximate the expected overfill of the knapsack. While some overfill is allowed, and we penalize the expected overfill, there is a limit to the magnitude of the overfill. We discuss how

[1] We use the both terms value and revenue for the positive contribution of an item to the objective.

this limit is enforced in Section 3.2. One can think of the penalization as a soft constraint while enforcing a limit on the overfill is a hard constraint. Next, in Section 3.3, we briefly describe how the items can be partitioned into groups where at most one element can be selected from each group. Finally, we state the full model in Section 3.4.

3.1. Approximation of the expected overfill

We are interested in identifying the expected overfilling of the knapsack as the cost, f, depends on the expected overfilling. We denote $O = \max\{0; X - T\} = (X - T)^+$ as the stochastic variable measuring the overfill. The following proposition states an approximation of the expected overfill $\mathbb{E}[O]$:

Proposition 1. *Let $X_1, \ldots, X_n$ be a set of independent stochastic variables with means μ_i and variances σ_i^2 for $i = 1, \ldots, n$. Let $X = X_1 + \ldots + X_n$ and $O = (X - T)^+$ for a constant $T \geq 0$. Denote $\mu_X = \mu_1 + \ldots + \mu_n$ and $\sigma_X^2 = \sigma_1^2 + \ldots + \sigma_n^2$. Then*

$$\mathbb{E}[O] \approx \sigma_X \left(\phi(k) - k(1 - \Phi(k))\right)$$

where $\phi(\cdot)$ is the probability density function and $\Phi(\cdot)$ is the cumulative distribution function for the standard normal distribution and $k = (T - \mu_X)/\sigma_X$.

Kleywegt et al. (2002) observe without proof and Kosuch & Lisser (2010) provide a proof of a similar result in the case where the X variables are normally distributed. We will denote the approximation of the expected overfill based on μ and σ when the knapsack has capacity T as

$$h_T(\mu, \sigma) = \sigma \left[\phi\left(\frac{T - \mu}{\sigma}\right) - \frac{T - \mu}{\sigma}\left(1 - \Phi\left(\frac{T - \mu}{\sigma}\right)\right)\right] \tag{1}$$

which corresponds to the approximation given in Proposition 1. An illustration of this approximation along with $\max\{0, \mu - T\}$ is given in Figure 1. In the figure we have fixed σ at either of the four levels 10, 20, 50, and 100. Below we will formally argue that $h_T(\mu, \sigma)$ is increasing in both μ and σ and $\max\{0; \mu - T\}$ serves as a lower bound for $h_T(\mu, \sigma)$ when we decrease σ. The first order derivatives of $h_T(\mu, \sigma)$ are stated in Proposition 2.

Proposition 2. *Let $\mu \geq 0$ and $T \geq 0$, and let $\phi(\cdot)$ be the probability density function and $\Phi(\cdot)$ be the cumulative distribution function for the standard normal distribution, and let $h_T(\mu, \sigma)$ be defined as in* (1)*. Then*

$$\frac{\partial h_T(\mu, \sigma)}{\partial \mu} = 1 - \Phi\left(\frac{T - \mu}{\sigma}\right)$$

and

$$\frac{\partial h_T(\mu, \sigma)}{\partial \sigma} = \phi\left(\frac{T - \mu}{\sigma}\right) \quad .$$

An important consequence of Proposition 2 is that $h_T(\mu, \sigma)$ is strictly increasing, as $\phi(k) > 0$ and $\Phi(k) < 1$ for all k. Intuitively, this is not surprising, as we would expect the overfill to increase when we fill up the

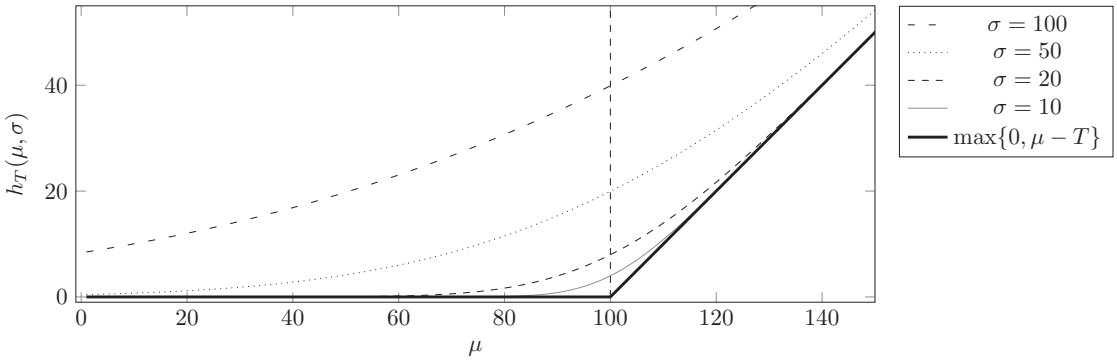

Figure 1: Illustration of $h_T(\mu, \sigma)$ with $\sigma = 10, 20, 50, 100$ and $\max\{0, \mu - T\}$ for $T = 100$.

knapsack more as well as when the uncertainty increases. We will exploit this when setting up the solution approach for the problem. We can also obtain a deterministic lower bound for the function $h_T(\mu, \sigma)$ (i.e., where the standard deviation approaches zero and therefore the mean becomes the deterministic value of the weight). This bound is given in Proposition 3.

Proposition 3. *Let $h_T(\mu, \sigma)$ be defined as in (1) and $\mu \geq 0$ be an arbitrary mean value. Then*

$$h_T(\mu, \sigma) \quad \geq \quad \max\{0; \mu - T\}$$

and $h_T(\mu, \sigma) \searrow \max\{0; \mu - T\}$ monotonically for $\sigma \searrow 0$.

Proposition 3 fits the intuition that if we have no uncertainty about the weight of the items, then the expected overfill will be zero if the fill, μ, is below T, while it will be $\mu - T$ if μ is above T. We will use the lower bound given in Proposition 3 to accelerate the solution process.

3.2. The maximum overfill

From a managerial perspective it is of interest not to violate the limit $T + S$. In our variant of the SKP we use T as the capacity of the knapsack. This capacity may be violated due to the stochasticity of the weights of the items. Some violation may be accepted, but the violation should not be too large. The amount $S \geq 0$ indicates the acceptable limit of violation of the knapsack capacity. However, as the item weights are stochastic we can only guarantee this with a certain probability.

The total weight of the knapsack is the sum of the weights of the individual items in it, and this sum is approximately normally distributed. Observe the following constraint:

$$\sum_{j \in \mathcal{J}} \mu_j x_j + \beta \sqrt{\sum_{j \in \mathcal{J}} \sigma_j^2 x_j} \leq T + S \tag{2}$$

where $\beta \geq 0$ corresponds to the number of standard deviations we are interested in having as slack from the mean and up to $T + S$. As the sum of stochastic variables is approximately normally distributed, this

translates to the probability of having a weight no larger than $T+S$. The assumption that $\beta \geq 0$ reflects the fact that we would like the fill of the knapsack to be less than or equal to $T+S$ with a probability of at least 50% (corresponding to the case where $\beta = 0$). When β is increased, then so is the corresponding probability of the fill being no larger than $T+S$. Indeed, this corresponds to a chance constraint stating that the probability of surpassing $T+S$ for the sum of the weights of the individual items should not be larger than a predefined value.

3.3. Partitioning of the set of items and relation to the integer stochastic knapsack problem

In some applications the items of $\mathcal{J}$ are partitioned into K groups of items within which maximally one item can be chosen. Let $\mathcal{K} = \{1, \ldots, K\}$ be the index set of the groups and let $\mathcal{J}_k \subseteq \mathcal{J}$ with $\mathcal{J} = \cup_{k\in\mathcal{K}}\mathcal{J}_k$ and $\mathcal{J}_k \cap \mathcal{J}_h = \emptyset$ for $h, k \in \mathcal{K}$ and $h \neq k$. Then maximally one element from each set, $\mathcal{J}_k$, can be chosen which corresponds to

$$\sum_{j\in\mathcal{J}_k} x_j \leq 1, \quad k \in \mathcal{K} \quad . \tag{3}$$

If $|\mathcal{J}_k| = 1$ for all $k \in \mathcal{K}$, then the problem is a binary SKP. An important special case of this is where the number of items we can select of the same type is not restricted to being binary, but rather integer with some upper bound (i.e., $0 \leq x_j \leq u_j$ and $x_j \in \mathbb{Z}$). In this case we construct u_j new items, x_{aj}, corresponding to having $a = 1, \ldots, u_j$ items, with $r_{aj} = ar_j$, $\mu_{aj} = a\mu_j$, and $\sigma^2_{aj} = a\sigma^2_j$. These new items are then collected into one group, k, such that maximally one of them can be used. Consequently, we can have an integer SKP.

3.4. The model

Using Proposition 1 we can approximate the expected overfilling $\mathbb{E}[O] \approx h_T(\mu_X, \sigma_X)$ and thereby set up the objective function

$$\sum_{j\in\mathcal{J}} r_j x_j - f(h_T(\mu_X, \sigma_X)) \tag{4}$$

which is to be maximized. We can now state the SKP as

$$\max \quad \sum_{j\in\mathcal{J}} r_j x_j - f\left(h_T\left(\sum_{j\in\mathcal{J}} \mu_j x_j, \sqrt{\sum_{j\in\mathcal{J}} \sigma^2_j x_j}\right)\right) \tag{5}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{J}_k} x_j \leq 1, \qquad k \in \mathcal{K} \tag{6}$$

$$\sum_{j\in\mathcal{J}} \mu_j x_j + \beta\sqrt{\sum_{j\in\mathcal{J}} \sigma^2_j x_j} \leq T+S \tag{7}$$

$$x_j \in \{0,1\}, \qquad j \in \mathcal{J} \quad . \tag{8}$$

The objective (5) maximizes the profit, i.e., the difference between revenue and expected cost. The first set of constraints, (6), ensures that maximally one element from each partition, $\mathcal{J}_k$ with $k \in \mathcal{K}$, is chosen,

and it is identical to constraint (3). Given that we can calculate μ_X and σ_X, then (7) ensures that the mean fill will be at least β standard deviations below the strict upper bound of $T + S$, which is indeed identical to (2). Finally, the domain of the x_j variable is given in (8). The model (5)-(8) is inherently non-linear. Fortunately, the structure of the model lends itself to a network-based approach, which we will discuss in Section 4.

Note that the model handles the potential violation of the capacity in two different but related ways. On one hand it penalizes the expected violation of the capacity constraint in the objective. If the penalty function is linear then the penalization corresponds to that of the SRKP. On the other hand, constraint (7) is an approximation of a chance constraint as in CCKP. If the penalty function is removed from the objective, then the problem becomes an approximation of the CCKP, whereas if (7) is removed then the problem becomes an SRKP.

One can ask, why combine these two approaches into a single problem? The reason is that the two approaches handle different situations. The penalization of the expected overfill is a soft constraint where we try to strike a balance between the revenue of the items and the costs of violating the capacity. If the revenue is very high for the items compared to the cost function, then we may get a large violation. We compensate for this by making the cost function increasing and convex. However, sometimes it is not acceptable with a too large violation and, as a consequence, a hard constraint limiting the violation of the capacity has to be applied. Due to the stochastic nature of the item weights the hard constraint is a chance constraint.

4. A shortest path based approach

The problem given in Section 3 can be formulated as a shortest path problem with resource constraints (SPPRC) on a special directed graph. In this section we will first set up the graph and then we will describe how to formulate the elements of the SKP by applying resource extension functions. This leads to a dynamic programming algorithm on the graph, where dominance is necessary only on the cost function, the mean, and the variance but where it is not necessary to apply dominance on the expected overfill. We close this section by discussing a bound that can be used to eliminate potential states from the dynamic program.

As we are going to use a shortest path algorithm we will minimize the negative profit function instead of maximizing the profit function directly. Thus, the objective we minimize is

$$f(h_T(\mu_X, \sigma_X)) - \sum_{j \in \mathcal{J}} r_j x_j \tag{9}$$

which corresponds to (4) multiplied by -1, and the resulting objective value of this minimization should also be multiplied by -1 to obtain the profit.

The graph can be constructed as follows. First denote $\mathcal{V} = \{o, d\} \cup \mathcal{J}$ as the set of nodes or vertices of the graph, where $o = 0$ corresponds to the origin and $d = J + 1$ to the destination, and the remaining nodes correspond to each of the elements of $\mathcal{J}$. The origin, o, and destination, d, have no associated revenue, mean, or variance and we put $r_o = r_d = 0$, $\mu_o = \mu_d = 0$, and $\sigma_o^2 = \sigma_d^2 = 0$.

The set of arcs is $\mathcal{A} \subseteq \{(i, j) \in \mathcal{V} \times \mathcal{V} | i < j\}$. The resulting graph will be acyclic, as all arcs have a tail-node index that is lower than the head node index. With each arc $(i, j) \in \mathcal{A}$ we associate the simple accumulation of revenue $r_{ij} = r_j$, mean $\mu_{ij} = \mu_j$, and variance $\sigma_{ij}^2 = \sigma_j^2$, corresponding to the values in the head node. All arcs (i, j) with $i, j \in \mathcal{J}_k$ with $k \in \mathcal{K}$ are excluded from the set of arcs, $\mathcal{A}$, which makes it impossible to use two items from the same subset, $\mathcal{J}_k$. Furthermore, we let $x_{ij} \in \{0, 1\}$ indicate whether or not arc (i, j) is used in a solution.

An example of a $\mathcal{J} = \{1, \ldots, 5\}$ item graph is illustrated in Figure 2. $\mathcal{J}$ is partitioned into the two subsets $\mathcal{J}_1 = \{1, 2\}$ and $\mathcal{J}_2 = \{3, 4, 5\}$, where $\mathcal{K} = \{1, 2\}$. These sets are illustrated by the dashed boxes. Arcs from o and to d are faded in order to ease the exposition.

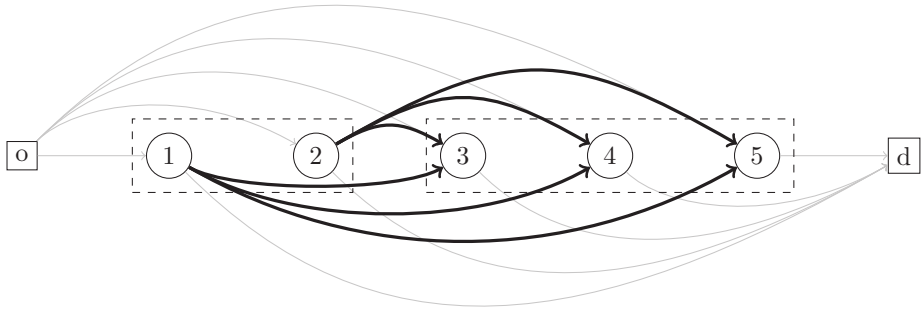

Figure 2: Example of graph with $\mathcal{J} = \{1, \ldots, 5\}$, $\mathcal{K} = \{1, 2\}$, $\mathcal{J}_1 = \{1, 2\}$, and $\mathcal{J}_2 = \{3, 4, 5\}$

A path $\mathcal{P} = (o, v_1, \ldots, v_p, d)$ from the origin, o, to the destination, d, in the graph is a connected sequence of arcs $(v_q, v_{q+1}) \in \mathcal{A}$ for $q = 0, \ldots, p$, where $v_0 = o$ and $v_{p+1} = d$. We write that an arc $(i, j) \in \mathcal{P}$ if the path traverses the arc and, likewise, that $i \in \mathcal{P}$ if the path visits node i. A path, $\mathcal{P}$, translates into a solution for the KP by $x_{ij} = 1$ if and only if $(i, j) \in \mathcal{P}$ and

$$x_j = \sum_{(i,j) \in \mathcal{A}} x_{ij} \quad . \tag{10}$$

Thus, if we can identify a path through this graph satisfying (7), then we have a knapsack solution. Observe that due to the exclusion of arcs between nodes from the sets $\mathcal{J}_k$, a path will automatically satisfy the constraints (6).

We solve the KP by repeated extension of paths to successor nodes, where the successor nodes of

$i \in \mathcal{V}$ are defined as $Succ(i) = \{j \in \mathcal{V} | (i,j) \in \mathcal{A}\}$. That is, we initialize the problem with a singleton path, $\mathcal{P} = (o)$, and extend it to all possible successor nodes to obtain the paths (o, v_1), where $v_1 \in \mathcal{J}$. Then each of these is extended to its successor nodes to obtain paths (o, v_1, v_2). This is repeated until all paths have been extended to all their possible successors. However, the number of potential paths increases exponentially with the size of K and can be bounded by $O(2^K)$, and as a consequence caution needs to be taken to avoid this. Indeed, not all paths are valid due to the maximum fill bound (2) and not all paths are good with respect to the objective.

With a partial path $\mathcal{P} = (o, v_1, \ldots, v_p)$ we will associate a state, $\mathcal{S}(\mathcal{P})$, of the path. The state corresponds to the accumulated revenue, mean, and variance, i.e. $\mathcal{S}(\mathcal{P}) = (r(\mathcal{P}), \mu(\mathcal{P}), \sigma^2(\mathcal{P}))$, where

$$r(\mathcal{P}) = \sum_{(i,j)\in\mathcal{P}} r_{ij}, \quad \mu(\mathcal{P}) = \sum_{(i,j)\in\mathcal{P}} \mu_{ij}, \quad \sigma^2(\mathcal{P}) = \sum_{(i,j)\in\mathcal{P}} \sigma^2_{ij}$$

and, based on this, the cost, which is slightly more complicated, is calculated as

$$C(\mathcal{P}) = f\left(h_T\left(\mu(\mathcal{P}), \sqrt{\sigma^2(\mathcal{P})}\right)\right) - r(\mathcal{P}) \quad .$$

Note that the cost, $C(\mathcal{P})$, depends on the revenue, $r(\mathcal{P})$, the mean, $\mu(\mathcal{P})$, and the variance, $\sigma^2(\mathcal{P})$, of the path and it is therefore not separable from these. As it is not separable and it is the cost we are minimizing, the proof of the validity of the standard componentwise dominance criterion given by Desaulniers et al. (1998) does not apply directly to our situation. Even though the proof given by Desaulniers et al. (1998) cannot be used in our case, we provide an alternative proof for the specific problem we address (see proof of Proposition 4 in the appendix) which shows that the componentwise dominance criterion is valid in our case.

While we can calculate the mean, variance, and cost of a given path directly, it is practical to have simple ways of calculating these values when we extend a path to a new node. We assume that the path $\mathcal{P} = (o)$ has $C(\mathcal{P}) = \mu(\mathcal{P}) = \sigma^2(\mathcal{P}) = 0$, as it corresponds to an empty knapsack solution. If we extend the path $\mathcal{P}$ with the node v_{p+1} to obtain the path $Q = (o, v_1, \ldots, v_p, v_{p+1})$, then we have to identify the state $\mathcal{S}(\mathcal{Q})$. The extended value of the revenue, the mean, and the variance can be calculated by the following recursions:

$$r(\mathcal{Q}) = r(\mathcal{P}) + r_{v_p v_{p+1}} \tag{11}$$

and

$$\mu(\mathcal{Q}) = \mu(\mathcal{P}) + \mu_{v_p v_{p+1}} \tag{12}$$

and

$$\sigma^2(\mathcal{Q}) = \sigma^2(\mathcal{P}) + \sigma^2_{v_p v_{p+1}} \tag{13}$$

whereas the cost is calculated by the recursion

$$C(\mathcal{Q}) = C(\mathcal{P}) - r_{v_p v_{p+1}} + f\left(h_T\left(\mu(\mathcal{Q}), \sqrt{\sigma^2(\mathcal{Q})}\right)\right) - f\left(h_T\left(\mu(\mathcal{P}), \sqrt{\sigma^2(\mathcal{P})}\right)\right) \tag{14}$$

i.e., as the cost of path $\mathcal{P}$ from which the gained revenue has been subtracted and to which the increase in cost of expected overfill has been added.

An extension is feasible if it satisfies (6) and (7). Constraint (6) is trivially satisfied due to the construction of the graph. However, constraint (7) has to be checked. We observe that $\mu(\mathcal{Q}) > \mu(\mathcal{P})$ and $\sigma^2(\mathcal{Q}) \geq \sigma^2(\mathcal{P})$ due to $\mu_{v_p v_{p+1}} > 0$ and $\sigma^2_{v_p v_{p+1}} \geq 0$, respectively, and as a consequence

$$\mu(\mathcal{P}) + \beta\sqrt{\sigma^2(\mathcal{P})} < \mu(\mathcal{Q}) + \beta\sqrt{\sigma^2(\mathcal{Q})}$$

for any non-negative value β, and the extension of a path will result in increasing value on the left-hand side of (7). Hence, if $\mu(\mathcal{Q}) + \beta\sqrt{\sigma^2(\mathcal{Q})} > T + S$, then constraint (7) is violated for path $\mathcal{Q}$ and, more importantly, for any extension of $\mathcal{Q}$. Therefore, the path $\mathcal{Q}$ and all its extensions will be infeasible. We therefore say that path $\mathcal{P}$ is *not* extendable to the node v_{p+1}.

A reduction of the number of states is important to limit the combinatorial explosion of the algorithm. It is of particular interest to be able to compare paths in order to eliminate those paths for which better paths exist. A simple dominance relation is given in Proposition 4.

Proposition 4. *Let $\mathcal{P}_1, \mathcal{P}_2$ be two paths from o to node i. If*

1. $r(\mathcal{P}_1) \geq r(\mathcal{P}_2)$,
2. $\mu(\mathcal{P}_1) \leq \mu(\mathcal{P}_2)$, *and*
3. $\sigma^2(\mathcal{P}_1) \leq \sigma^2(\mathcal{P}_2)$,

then for any feasible extension of path $\mathcal{P}_2$ a corresponding feasible extension of path $\mathcal{P}_1$ exists yielding no larger variance, mean, or cost.

Proposition 4 states that path $\mathcal{P}_2$ can never be better than the path $\mathcal{P}_1$. This makes intuitive sense, as $\mathcal{P}_1$ gets a higher revenue with less, but more certain, filling of the knapsack than $\mathcal{P}_2$. We will say that the state $\mathcal{S}(\mathcal{P}_1)$ (weakly) dominates state $\mathcal{S}(\mathcal{P}_2)$ and we will write $\mathcal{S}(\mathcal{P}_1) \preceq \mathcal{S}(\mathcal{P}_2)$. Clearly, in the case that $\mathcal{S}(\mathcal{P}_1) \preceq \mathcal{S}(\mathcal{P}_2)$ we can always obtain a solution by extending $\mathcal{P}_1$, which is at least as good as extending $\mathcal{P}_2$, and we will therefore remove $\mathcal{P}_2$ from consideration.

It is interesting to note that the dominance criterion in Proposition 4 does not depend on the shape of the overfill cost function, f, as long as it is non-decreasing. As a consequence, we can embed the solution of this type of KP in an SPPRC framework.

The extension of paths and their states gives rise to an algorithm where we consider each node in the graph $(\mathcal{V}, \mathcal{A})$ once and extend all undominated paths from that node to all possible successor nodes. The pseudo code for the dynamic programming is given in Algorithm 1. The algorithm is generic in the sense that it applies in general to shortest path problems on directed acyclic graphs. However, it does not apply to the case where cycles are present.

Algorithm 1: Full extension label setting dynamic programming algorithm

```
1  Λ_i = ∅;
2  Λ_o = {S((o))};
3  foreach i ∈ V in topological order do
4      foreach S(P) ∈ Λ_i do
5          foreach j ∈ Succ(i) do
6              if extendable(S(P) , j ) then
7                  S(Q) = extend( S(P), j );
8                  if not dominated( Λ_j, S(Q) ) then
9                      eliminate_dominated( Λ_j, S(Q) );
10                     insert( Λ_j, S(Q) );
11                 end
12             end
13         end
14     end
15 end
```

We denote by Λ_i all the undominated states present in node $i \in \mathcal{V}$. These sets are initialized to be empty, except for the origin in which the initial state of the path, $\mathcal{P} = (o)$, is inserted. Then each of the nodes in $\mathcal{V}$ is processed in topological order. Due to the topological order it will never be possible to extend states to a node from which we have already extended states. Consequently, when extending states from Λ_i, we know that the states included in Λ_i constitute the final set of undominated states for the subset of nodes with indices less than or equal to node i. All the undominated states in node i are extended to successor nodes, j, if possible, and the dominance check is commenced to identify whether or not any state previously extended to node j dominates the newly generated state. The check of whether or not state $\mathcal{S}(\mathcal{P})$ is extendable to node j corresponds to checking constraint (7), while the extended state $\mathcal{S}(\mathcal{Q})$, when extending $\mathcal{S}(\mathcal{P})$ to node j, corresponds to calculating (11)-(13) as well as the cost (14). If the state is undominated, then it is used to eliminate other previously generated states after which it is inserted into the set, Λ_j, of undominated states of node j.

Algorithm 2: Incremental extension label setting dynamic programming algorithm

$\Lambda_i = \emptyset$;
$\Lambda_o = \{\mathcal{S}((o))\}$;
$\Delta_i = \emptyset$
$L = L^{init}$
while $\cup_{i \in \mathcal{V}} (\Lambda_i \setminus \Delta_i) \neq \emptyset$ **do**
  foreach $i \in \mathcal{V}$ *in topological order* **do**
    $\Theta_i = \Lambda_i \setminus \Delta_i$
    while $\ell \leq L$ **do**
      $\mathcal{S}(\mathcal{P}) = \underset{\mathcal{S}(\mathcal{P}') \in \Theta_i}{\arg\min} \{C(\mathcal{P}')\}$
      foreach $j \in Succ(i)$ **do**
        if *extendable*$(\mathcal{S}(\mathcal{P})\ ,\ j\ )$ **then**
          $\mathcal{S}(\mathcal{Q}) = \text{extend}(\ \mathcal{S}(\mathcal{P}), j\ )$;
          if *not dominated*$(\ \Lambda_j, \mathcal{S}(\mathcal{Q})\ )$ **then**
            eliminate_dominated$(\ \Lambda_j, \mathcal{S}(\mathcal{Q})\ )$;
            insert$(\ \Lambda_j, \mathcal{S}(\mathcal{Q})\ )$;
          end
        end
      end
      $\Delta_i = \Delta_i \cup \{\mathcal{S}(\mathcal{P})\}$
      $\Theta_i = \Theta_i \setminus \{\mathcal{S}(\mathcal{P})\}$
      $\ell = \ell + 1$
    end
  end
  $L = 2L$
end

In some cases we can exploit having an upper bound value of the optimal objective value – see Section 4.1. However, Algorithm 1 applies a breadth-first strategy for the extension, and as a consequence the upper bound will decrease slowly throughout the process.

To remedy this we apply a limited extension strategy inspired by Burke & Curtois (2014), who apply dynamic programming for identifying individual schedules in nurse rostering. The intuition is for each node only to extend the L best non-extended states. In our context, the best are those having the lowest objective value. Thus, we divide the execution of the algorithm into a number of stages where – in each stage – we extend (at most) L states from each node in topological order. If any nodes have unextended states after a stage, then we start a new stage, potentially with a larger value of L. By doing this we emphasize a greedy approach in the first stages of the algorithm such that we can obtain a reasonable

upper bound solution value, and then, in the later stages, we explore the remaining states.

In Algorithm 2 we describe dynamic programming with limited extension. This is an adaption of Algorithm 1, where we keep track of which states have been extended and split the solution process into stages, where in each stage at most $L|\mathcal{V}|$ states are extended. In addition to Algorithm 1, the set Δ_i stores the states that have already been extended, while the set Θ_i corresponds to the set of non-extended states.

We initialize the algorithm with $L = L^{init}$, where $L^{init} \geq 1$ is the maximum number of states we wish to extend from each node in the first stage. After each stage we double the number of states, L, we allow to be extended. In line 9 the states are selected in increasing order of cost, such that we greedily select the most promising states in the beginning of the algorithm.[2] If we put $L^{init} = \infty$, then Algorithm 2 will be equivalent to Algorithm 1.

The computational complexity depends on the potential number of states that have to be extended from each node. In the case where the revenue, the mean, and the variance are all integer and we have a strict upper bound in the form of constraint (2), it is possible to (pseudo-) polynomially bound the number of states that are undominated for a given node, see, for instance Desrochers & Soumis (1988). However, in our case we do not necessarily have all integer values for the revenue, mean, or variance, and we do not require (2) to be present. As a consequence, the algorithms are in principle bounded only by the number of potential paths in the graph, which in general is not polynomially bounded. Thus, we are interested in further reducing the number of potential states that do not lead to an optimal solution. We do this in the following section.

4.1. State bounding for convex expected overfill cost functions

Until now we have assumed that f was increasing. If, however, f is also convex, then we can accelerate the solution process by bounding the objective of possible extensions. This bounding relies on a combination of a deterministic knapsack lower bound and the lower bound on the expected overfill given by Proposition 3.

We let UB be the value of the best known solution at any point of the algorithm. Initially, this is set to 0, as it is the trivial solution including no items. Given a path, $\mathcal{P}$, with cost $C(\mathcal{P})$ we let $LB(\mathcal{P})$ be a lower bound on the additional cost of any possible extension of path $\mathcal{P}$ to the destination node, d. With this bound we can eliminate the state $\mathcal{S}(\mathcal{P})$ if

$$C(\mathcal{P}) + LB(\mathcal{P}) \geq UB \quad . \tag{15}$$

However, the crux is to identify the lower bound, $LB(\mathcal{P})$. Assume that we have the path $\mathcal{P} = (o, v_1, \ldots, v_p)$

[2]It should be noted that we have chosen "most promising" to correspond to the cost, but other alternatives could be used, for example, the cost per unit of the mean, $c(\mathcal{P})/\mu(\mathcal{P})$.

and some feasible extension $\mathcal{Q} = (w_1, \ldots, w_q, d)$ of path $\mathcal{P}$. Then we can construct the full feasible path $(\mathcal{P}, \mathcal{Q}) = (0, v_1, \ldots, v_p, w_1, \ldots, w_q, d)$ as the concatenation of $\mathcal{Q}$ to $\mathcal{P}$. By the cost recursion (14) we must have

$$C(\mathcal{P}, \mathcal{Q}) - C(\mathcal{P}) = -\sum_{j \in \mathcal{Q}} r_j + f\left(h_T\left(\mu(\mathcal{P}, \mathcal{Q}), \sqrt{\sigma^2(\mathcal{P}, \mathcal{Q})}\right)\right) - f\left(h_T\left(\mu(\mathcal{P}), \sqrt{\sigma^2(\mathcal{P})}\right)\right) \tag{16}$$

and we have to identify the lower bound in the cost of any extension, $\mathcal{Q}$, such that $C(\mathcal{P}, \mathcal{Q}) - C(\mathcal{P}) \geq LB(\mathcal{P})$. We do this by observing that the last term of the right-hand side of (16) is constant for any extension, and then construct convex lower bounding functions for the two remaining terms.

Before setting up the lower bound we sort the nodes such that the sequence of the $\mathcal{J}_k$ sets reflects a decreasing potential of improving the objective value. Within each set, $\mathcal{J}_k$, we denote the maximum mean value

$$M_k = \max\{\mu_j \,|j \in \mathcal{J}_k\}, \quad k \in \mathcal{K} \quad .$$

Define for each $k \in \mathcal{K}$ the value L_k as the minimum ratio of the negative item revenue, $-r_j$ and the expected item weight, μ_j, for items, j, in the set $\mathcal{J}_k$:

$$L_k = \min\left\{ \frac{-r_j}{\mu_j} \middle| \, j \in \mathcal{J}_k \right\}, \quad k \in \mathcal{K} \quad .$$

The value $M_k L_k$ represents a lower bound on the value that we can attain from selecting an element, $j \in \mathcal{J}_k$. That is, $M_k L_k \leq -r_j$ for all $j \in \mathcal{J}_k$. We will assume that set $\mathcal{K}$ is sorted in increasing order of $M_k L_k$ (i.e., for $h, k \in \mathcal{K}$ with $h < k$ then $M_h L_h \leq M_k L_k$).

Now we turn to identifying a lower bound for $\sum_{j \in \mathcal{Q}} -r_j$ for any extension, $\mathcal{Q}$, of path $\mathcal{P}$. This bound is closely related to a lower bound obtained by solving a fractional KP. The lower bound we are searching for is for all possible extensions of a given path, and consequently we are only interested in the subsets, $\mathcal{J}_k$, to which we can extend this path. Hence, for the path $\mathcal{P} = (o, v_1, \ldots, v_p)$ we denote the set

$$\mathcal{E}(\mathcal{P}) = \{k \in \mathcal{K} | k > \max\{h \in \mathcal{K} | v_p \in \mathcal{J}_h\}\} \subseteq \mathcal{K}$$

as the index set of subsets to which path $\mathcal{P}$ can be extended. This corresponds to all the indices of subsets with a larger index than the last subset added.

Suppose that $m \in \mathbb{R}_+$ is a non-negative amount corresponding to the expected weight of additional items that can be put into the knapsack. Then we can include items from the sets $\mathcal{J}_h$ with $h \in \mathcal{E}(\mathcal{P})$ and h no larger than

$$k(\mathcal{P}, m) = \max\left\{ k \in \mathcal{E}(\mathcal{P}) \middle| \sum_{i \in \mathcal{E}(\mathcal{P}), i \leq k} M_i \leq m \right\}$$

without surpassing the value, m. The next element, $k(\mathcal{P}, m) + 1$, will either not be added or only be added at a fractional value. Due to the fact that $L_k M_k \leq -r_j$ for all $j \in \mathcal{J}_k$ we have the following lower

bound on the value of any extension with expected weight, m, of additional items put in the knapsack:

$$z(\mathcal{P},m) = \sum_{\substack{i\in\mathcal{E}(\mathcal{P})\\ i\leq k(\mathcal{P},m)}} M_iL_i + L_{k(\mathcal{P},m)+1}\left(m - \sum_{\substack{i\in\mathcal{E}(\mathcal{P})\\ i\leq k(\mathcal{P},m)}} M_i\right) \quad . \tag{17}$$

Note that if $\mathcal{J}_k$ contains only one element for each k, then this lower bound corresponds to the LP-relaxed value of a binary knapsack problem with elements from $\mathcal{E}(\mathcal{P})$ and capacity m. As the elements in $\mathcal{K}$ are sorted in increasing order of M_kL_k, we have that the slope (but not the function itself) of the function $z(\mathcal{P},m)$ is non-decreasing in m and it is therefore (weakly) convex. Thus, we have a decreasing convex function as a lower bound on the cost when increasing the mean weight of additionally added items.

We now turn our attention to the bounding of $f(h_T(\mu(\mathcal{P},\mathcal{Q}),\sqrt{\sigma^2(\mathcal{P},\mathcal{Q})}))$ in (16). We observe from Proposition 3 that f is increasing and that

$$\begin{aligned} f\left(h_T\left(\mu(\mathcal{P},\mathcal{Q}),\sqrt{\sigma^2(\mathcal{P},\mathcal{Q})}\right)\right) &\geq f\left(\max\left\{0;\mu(\mathcal{P},\mathcal{Q})-T\right\}\right)\\ &= f\left(\max\left\{0;\mu(\mathcal{P})+\sum_{j\in\mathcal{Q}}\mu_j - T\right\}\right)\\ &= f\left(\max\left\{0;\mu(\mathcal{P})+m-T\right\}\right) \end{aligned}$$

where $m=\sum_{j\in\mathcal{Q}}\mu_j$. Then note that $\mu(\mathcal{P})$ and T are constants and that $\max\{\mu(\mathcal{P})+m-T\}$ is a convex function in m. As a consequence we have that $f\left(\max\left\{0;\mu(\mathcal{P})+m-T\right\}\right)$ is a convex function in m.

We have now expressed the lower bound $z(\mathcal{P},m)$ for $\sum_{j\in\mathcal{Q}}-r_j$ when $\mathcal{Q}$ is allowed to use items having an aggregate expected weight of at most m, and likewise we have the lower bound $f\left(\max\left\{0;\mu(\mathcal{P})+m-T\right\}\right)$ for $f(h_T(\mu(\mathcal{P},\mathcal{Q}),\sqrt{\sigma^2(\mathcal{P},\mathcal{Q})}))$ when $\mathcal{Q}$ is allowed to use items having an aggregate expected weight of m. Both of these lower bounds are convex functions and so will the sum of these be. Consequently, minimizing the sum of the lower bounds will yield a unique minimum (if one exists) when we minimize over m. The aggregate expected weight, m, of additional items is at least 0. It is, however, bounded from above by (2), where we let $\beta=0$ (i.e., $m\leq T+S-\mu(\mathcal{P})$). Thus, the domain we are minimizing over is $0\leq m\leq T+S-\mu(\mathcal{P})$. We denote the domain $\mathcal{M}(\mathcal{P})=[0;T+S-\mu(\mathcal{P})]$, and state the lower bound on the cost of any extension as

$$LB(\mathcal{P}) = \min_{m\in\mathcal{M}(\mathcal{P})}\left\{z(\mathcal{P},m)+f\left(\max\left\{0;\mu(\mathcal{P})+m-T\right\}\right)\right\} - f\left(h_T\left(\mu(\mathcal{P}),\sqrt{\sigma^2(\mathcal{P})}\right)\right) \quad . \tag{18}$$

Note that the convexity of f is only used to guarantee that a local optimum is a global optimum in (18) when minimizing. By combining (15) with (18), we obtain the following proposition.

Proposition 5. *Let $\mathcal{P}$ be a feasible path from o to j and let UB be an upper bound on the solution value* (9) *when minimizing* (9) *subject to the constraints* (6)-(8)*. If*

$$\min_{m\in\mathcal{M}(\mathcal{P})}\left\{z(\mathcal{P},m)+f\left(\max\left\{0;\mu(\mathcal{P})+m-T\right\}\right)\right\} \geq UB+\sum_{i\in\mathcal{P}} r_i \tag{19}$$

then no extension of $\mathcal{P}$ will attain a lower objective value than UB.

The constraint (15) applying the bound is in our case used on time of creation of the state and on time of extension of the state. This corresponds to line 7 and line 6, respectively, in Algorithm 1 and line 12 and line 11, respectively, in Algorithm 2. In practice, we calculate the left-hand side of (19) when creating the state and then we check and use this cached value whenever doing the actual check.

4.1.1. Numerical example

In this section we will provide a small numerical example of the lower bound for a specific convex cost function. Suppose that $f(x) = \frac{1}{2}x^2$, which is non-decreasing and convex in the interval of non-negative numbers. Let $T = 100$ and $S = 20$. Let $\mathcal{P}$ be a path that has resulted in a mean of $\mu(\mathcal{P}) = 94$. The interval of the additional weight of items add is then $\mathcal{M}(\mathcal{P}) = [0; 100 + 20 - 94] = [0; 26]$. For simplicity we assume that the items are not mutually excluding, meaning that the sets, $\mathcal{J}_k$, are all singleton sets, and we put $\mathcal{J}_k = \{k\}$. Suppose that we are given four items, which have not yet been used, with mean and revenue given in Table 1. The elements are sorted in increasing order of L_k and in this case there

$j(=k)$	r_j	μ_j	M_k	L_k
a_1	10	5	5	-2
a_2	8	5	5	$-1\frac{3}{5}$
a_3	15	10	10	$-1\frac{1}{2}$
a_4	10	10	10	-1

Table 1: Data for example of a lower bound

is a one-to-one correspondence between the elements of $\mathcal{J}$ and the elements of the set $\mathcal{K}$. Furthermore, these items constitute the set $\mathcal{E}(\mathcal{P}) = \{a_1, \ldots, a_4\}$.

Now we need to identify $m \in \mathcal{M}(\mathcal{P})$, which minimizes $z(\mathcal{P}, m) + f\left(\max\{0; \mu(\mathcal{P}) + m - T\}\right)$ on the left-hand side of (19). We note for $m \leq T - \mu(\mathcal{P}) = 100 - 94 = 6$ that $\max\{0; \mu(\mathcal{P}) + m - T\} = 0$ and as the slope is negative for $z(\mathcal{P}, m)$ the optimal value of m is trivially 6 in the interval $[0; 6]$. For $m > 6$ it is more interesting. We observe that $z(\mathcal{P}, m)$ changes slope, given by L_k, in the points 5, 6, 10, and 20. In this example we denote $g(m) = z(\mathcal{P}, m) + f\left(\max\{0; \mu(\mathcal{P}) + m - T\}\right)$, which can be stated explicitly on the domain $[0; 26]$ as

$$g(m) = \begin{cases} & -2m, & & 0 \leq m < 5 \\ & -1\frac{3}{5}(m-5) & -10, & 5 \leq m < 6 \\ \frac{1}{2}(m-6)^2 & -1\frac{3}{5}(m-5) & -10, & 6 \leq m < 10 \\ \frac{1}{2}(m-6)^2 & -1\frac{1}{2}(m-10) & -18, & 10 \leq m < 20 \\ \frac{1}{2}(m-6)^2 & -1(m-20) & -33, & 20 \leq m \leq 26 \end{cases}.$$

We have illustrated part of the function $g(m)$ in Figure 3. The optimal value is found for $m = 7\frac{3}{5}$ yielding a value of $-12\frac{22}{25}$.

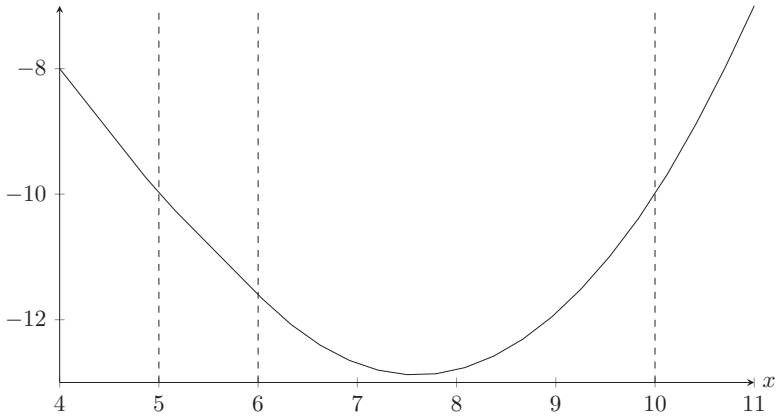

Figure 3: Illustration of the function $g(m)$.

4.1.2. A note on complexity

Finding the minimum of this function amounts to identifying the minimum of each of the individual functions. If for any of the individual functions we obtain a minimum outside the domain for which it contributes to $g(m)$, then we can, due to convexity, disregard all the individual functions in the opposite direction. That is, if the minimum falls below the boundary, then we can disregard all functions with larger boundaries than the current, and likewise if the minimum falls above the boundary, then we can disregard all functions with smaller boundaries than the current function. Thus, in the worst case we have to find the minimum of $O(\log J)$ functions to identify the minimum of $g(m)$. As we can explicitly calculate the minimum of the quadratic function in $O(1)$ time complexity, we have that the complexity of calculating this bound will be $O(\log J)$.

5. Computational Study

To illustrate the most important properties of the shortest-path-based approach for solving the SKP we conduct a computational study. First, in Section 5.1 we describe the instances we have generated to conduct the computational study. And second, we describe and discuss the computational results in Section 5.2.

5.1. Test instances

In order to test the algorithms we have generated a number of random instances. These instances are based on generating lists of items with randomly generated revenue, mean, and variance, and we combine each of these with scenarios determining parameters such as maximal fill size and expected overfill penalty function.

The instances with sets of items are generated as follows: First we have three categories, small, medium, and large, based on the number of items being 100, 250, and 500 items, respectively. Within each of these categories a further subdivision based on the variance is made. This subdivision is small, medium, and large variance. Finally, we have two types of groupings of items – one where all items can be used in conjunction and one where the items are in groups of five being mutually exclusive. Hence, we have 18 distinct categories of instances. Within each category 20 random instances are generated, where the revenue, r_j, for each item is drawn from the uniform distribution between 0 and 100, and the expected weight, μ_j, for each item is drawn from the uniform distribution between 10 and 50. When the variance is small we draw σ_j^2 from the uniform distribution between 0 and 10, whereas for medium and large variances we draw from the uniform distributions between 10 and 25, and 25 and 50, respectively. In all we have 360 distinct item sets. A summary of the setup of the item sets is given in Table 2.

Parameter	Type	Value
J	Small	100
	Medium	250
	Large	500
r_j	-	$U(0, 100)$
μ_j	-	$U(10, 50)$
σ^2	Small	$U(0, 10)$
	Medium	$U(10, 25)$
	Large	$U(25, 50)$
$\mathcal{K}$	No groups	$\lvert\mathcal{K}\rvert = \lvert\mathcal{J}\rvert$
	Groups of 5	$\lvert\mathcal{J}_k\rvert = 5$, $k \in \mathcal{K}$, $\mathcal{K} = \{1, \ldots, \lvert\mathcal{J}\rvert/5\}$

Table 2: Setup of the item sets.

For each item set we can test a number of scenarios based on other parameters of the KPs. These parameters are chosen in the following way. The limit of the total fill, T, is tested for small, medium, and large values, being 100, 250, and 500, respectively. Recall that we may violate this limit by additional S units. We test this for S being 0, corresponding to the case where a violation of the T units is prohibited, $T/5$ (i.e., a proportion of T), and infinite, stating that no upper limit on the violation exists. β states the number of standard deviations with which we should be within the limit of $T+S$. We test the cases where β is equal to 0, 1, and 2. Finally we can test two different types of expected overfill penalty functions. In the first case we assume the function to be a (specific) linear increasing function $f(x) = 5x$, whereas in the second case we assume the function to be a (specific) quadratic function, $f(x) = \frac{1}{2}x^2$, of the expected overfill. In all we have 54 distinct scenarios, and a summary of the scenarios is given in Table 3. If we

Parameter	Type	Value
T	Small	100
	Medium	250
	Large	500
S	T is the limit	0
	Proportion	$T/5$
	No upper limit	∞
β	bounded by mean	0
	bounded by mean plus 1 std. dev.	1
	bounded by mean plus 2 std. dev.	2
$f(x)$	Linear	$5x$
	Quadratic	$\frac{1}{2}x^2$

Table 3: Different scenarios used for each random instance.

combine the generated sets of items with the scenarios we have 19,440 test cases in all.

5.2. Computational results

We have conducted a series of tests based on the test cases described in Section 5.1. The algorithms were implemented in C++ using the TDM-GCC 5.1.0 64bit g++ compiler with the option -O3 turned on. The experiments were conducted on a system equipped with a Xeon CPU EX-2630 v4 @2.20GHz processor having 10 physical cores and 32Gb RAM. We executed nine instances concurrently.[3] The computer had a Windows 7 operating system. Each execution of a test instance was given a maximum of 300 seconds. If the time limit was reached, then the test was prematurely terminated and the best upper bound was reported.

We compare four different approaches based on the shortest path formulation of the problem. These are:

FNb: Using full (F) extension of each node with no bounding (Nb). This corresponds to having $L^{init} = \infty$ and $LB(\mathcal{P}) = -\infty$.

INb: Using incremental (I) extension of each node with no bounding (Nb). This corresponds to having $L^{init} = 1$ and $LB(\mathcal{P}) = -\infty$.

[3]The concurrent execution gives an overhead compared to solving the problems using a single thread. However, we choose to do this to accelerate the overall testing process.

		FNb			INb			FB			IB		
$\|J\|$	T	S	U	D	S	U	D	S	U	D	S	U	D
100	100	70.32	29.68	0.00	71.20	28.80	0.00	100.00	0.00	0.00	100.00	0.00	0.00
100	250	43.29	27.03	29.68	30.60	40.60	28.80	100.00	0.00	0.00	100.00	0.00	0.00
100	500	11.62	31.67	56.71	3.33	27.27	69.40	94.03	5.97	0.00	100.00	0.00	0.00
250	100	65.28	34.72	0.00	62.13	37.87	0.00	100.00	0.00	0.00	100.00	0.00	0.00
250	250	0.00	65.28	34.72	0.00	62.13	37.87	83.47	16.53	0.00	99.81	0.19	0.00
250	500	0.00	0.00	100.00	0.00	0.00	100.00	10.37	73.10	16.53	89.30	10.51	0.19
500	100	40.74	59.26	0.00	22.87	77.13	0.00	99.03	0.97	0.00	99.86	0.14	0.00
500	250	0.00	40.74	59.26	0.00	22.87	77.13	48.19	50.84	0.97	93.80	6.06	0.14
500	500	0.00	0.00	100.00	0.00	0.00	100.00	0.00	48.19	51.81	60.70	33.10	6.20

Table 4: Percentage of instances solved within the time limit. For each combination of $|J|$ and T 2160 instances-scenario combinations are possible. For each algorithm, FNb, INb, FB, and IB, column S gives the percentage solved, column U gives the percentage of tested but unsolved instances, and column D indicates the percentage of disregarded instances. An instance-scenario combination is disregarded if the same algorithm has failed to solve a corresponding instance-scenario with the only difference that it has a smaller T value.

FB: Using full (F) extension of each node with bounding (B). This corresponds to having $L^{init} = \infty$ and using (19) for bounding.

IB: Using incremental (I) extension of each node with bounding (B). This corresponds to having $L^{init} = 1$ and using (19) for bounding.

All of these are tested using Algorithm 2, where we note that when $L^{init} = \infty$, then Algorithm 2 is equivalent to Algorithm 1.

Table 4 shows for each of the variants of the algorithms the percentage of instances solved within the time limit of 300 seconds. Each row in the table corresponds to a combination of the number of items $|J|$ and the limit, T, as these are the primary indicators of how difficult an instance is to solve. For each combination of number of items $|J|$ and fill limit, T, we have 2160 instance-scenario combinations. The first two columns indicate $|J|$ and T. The remaining columns are divided into four sections – one for each approach tested. For each approach three columns are given. The first column (S) indicates how large a percentage of the 2160 instances the algorithm solved to optimality. The second column (U) indicates the percentage of the 2160 instances that were tested but unsolved due to the time limit. The last column (D) indicates the percentage of instances that were disregarded in the testing. We have disregarded instances for which the algorithm has reached the time limit for a lower value of T, as they would (most likely) reach the time limit too.

It is not surprising that when we increase either $|J|$ or T, then any of the algorithms will time out on more instances (except in the case where all instances are solved to optimality). In the case where we use no bound (FNb and INb) we observe that the full extension approach (FNb) solves more instances than

the incremental method (INb). This is due to a slight increase in the number of dominance checks, which is necessary when we use the incremental approach compared to the full approach. We see quite the opposite picture when comparing the full extension approach using bounding (FB) with the incremental extension approach using bounding (IB). In this case IB solves significantly more instances than FB. The reason is that IB obtains a tighter upper bound much faster than FB and can, as a consequence, eliminate states earlier in the process. Furthermore, IB is indeed the only one of the tested approaches that is able to solve any of the instances with $|J| = 500$ and $T = 500$ within the time limit. We observe that using the bound significantly increases the number of instances that can be solved by the algorithms.

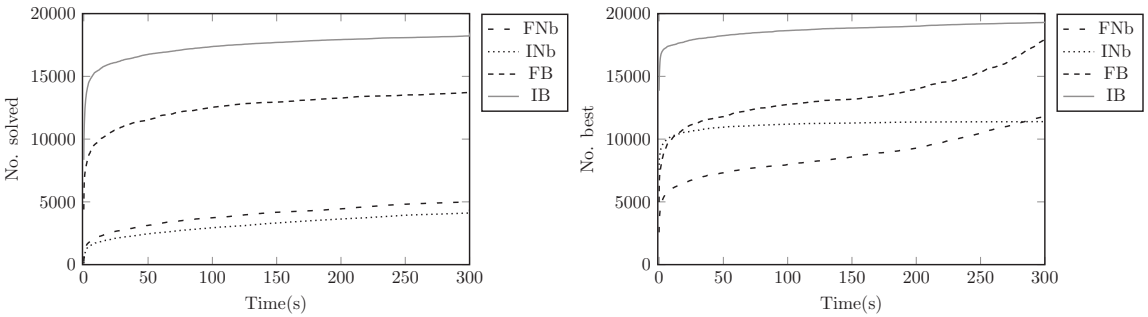

Figure 4: The horizontal axis is time in seconds, while the vertical axis is the accumulated number of instances. Left: The cumulative number of instances solved within time. Right: The cumulative number of instances, where the best solution is found within time.

Figure 4 shows to the left the cumulative number of instances each method is able to solve within a given time and to the right the cumulative number of instances for which each method reaches the best solution no later than a given time. We observe that the IB method solves more instances within two-and-a-half seconds than any of the other methods do in 300 seconds.[4] Similarly, we see that FB solves significantly more instances (7367 instances to be precise) within the first second than the two approaches – FNb and INb – not using the bound within 300 seconds. Hence, using the bound-based elimination from Proposition 5 is superior to not using this elimination.

The cumulative number of instances that obtain the best solution within a given time can be seen in the right panel of Figure 4. That is, after this time no better solution is found. In many cases the best solution is found very early. However, the behavior of the full extension methods (FNb and FB) is different from that of the incremental methods (INb and IB). While the full extension methods find new best solutions throughout the 300 seconds, the incremental methods find most of the best solutions very quickly. Moreover, these solutions are often optimal. Consequently, the incremental approach yields a

[4]The IB method has solved 13,900 instances within 2.5 sec., while the FNb, the INb, and the FB have solved 5014 instances, 4258 instances, and 13725 instances, respectively, within 300 sec.

reasonable heuristic for the SKP.

In general, we observe that for the approaches not using the bound (FNb and INb) it is easier to solve the grouped instances. This is due to the number of potential paths in the graph becoming significantly smaller when some of the items are mutually exclusive. However, we see the opposite effect for the approaches using the bound (FB and IB). This is due to a weakening of the bounding when we group the items, and therefore we cannot eliminate as many states based on the bound.

The IB approach is superior to the other approaches and we therefore turn to this approach and investigate which cases are difficult for this approach to solve. In order to do this we limit our attention to the set of instances having $|\mathcal{J}| = 500$ and $T = 500$, as we anticipate that the effects will be most pronounced in these instances.

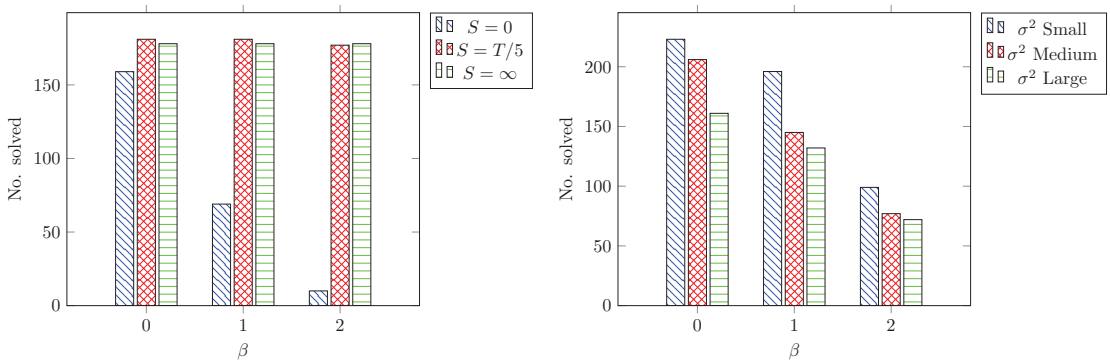

Figure 5: The number of instances solved for different values of $\beta \in \{0, 1, 2\}$ by IB when $|J| = 500$ and $T = 500$. Left: The number of instances solved for combinations of β and S; $S = 0$,$S = T/5$, and $S = \infty$. Right: The number of instances solved for combinations of β and variance of the instances; Small ($\sigma^2 \in [0, 10]$), Medium ($\sigma^2 \in [10, 25]$), and Large ($\sigma^2 \in [25, 50]$).

In Figure 5 we illustrate how many instances are solved within the time limit when we vary β, S, and the variance of the item size. To the left we observe that when $S = 0$, the number of instances solved decreases drastically when we increase β. The reason for this is that the number of states eliminated due to the lower bound from Proposition 5 decreases significantly. The lower bound (18) does not incorporate the maximal overfill constraint (2), while the upper bound satisfies this constraint. As a consequence, the gap between these two bounds becomes too large for constraint (19) to be satisfied. We do not see the same effect for $S = T/5$ or for $S = \infty$, as (2) is less constraining when identifying the upper bound.

To the right in Figure 5 we see that it is easier to solve instances with small variance than instances with large variance. Again, the lower bound calculation in (18) becomes weaker when the variance increases, as we have applied Proposition 3 in order to obtain a deterministic lower bound for the penalty of the expected overfill, and we know that the expected overfill increases with the variance, as illustrated in Figure 3 and proved in Proposition 2.

To further illustrate the effects we show in Figure 6 the number of instances the IB algorithm is able

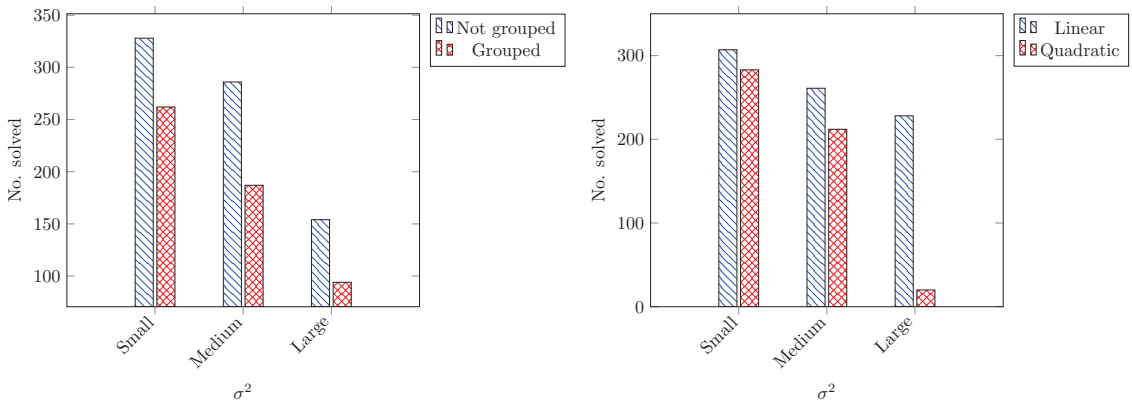

Figure 6: The number of instances solved by IB when $|J| = 500$ and $T = 500$ for different variances; Small ($\sigma^2 \in [0, 10]$), Medium ($\sigma^2 \in [10, 25]$), and Large ($\sigma^2 \in [25, 50]$). Left: The number of instances solved based on grouping of items; Not grouped ($|\mathcal{K}| = |\mathcal{J}|$) and Grouped ($|\mathcal{J}_k| = 5$ for all $k \in \mathcal{K}$ and $\mathcal{K} = \{1, \ldots, |\mathcal{J}/5|\}$). Right: The number of instances solved when the penalty function is linear, $f(x) = 5x$, and quadratic, $f(x) = \frac{1}{2}x^2$.

to solve for different levels of variance and for different groupings and penalty functions. As observed above, more instances can be solved when the variance is small compared to when it is large. It is also observed that it is easier to solve the non-grouped instances compared to the grouped instances for the IB algorithm. And finally, it is easier to solve problems when the penalty function is linear than when it is quadratic. In fact, when the instances are grouped, the variance is larger, and we use the quadratic penalty function, then none of the instances of size $\mathcal{J} = 500$ and $T = 500$ are solved to optimality by the IB algorithm. Again, this is due to the gap between the upper and lower bounds becoming too large for constraint (19) to be satisfied, and therefore fewer states are eliminated early in the algorithm.

To summarize the computational study, we observe that combining incremental extensions of states with the bounding procedure yields by far the best results. It is observed that the incremental bounding needs to be supplemented by the use of a bounding procedure to outperform the full extension approach. Furthermore, we observe that the bounding is effective in many cases, but is challenged in the cases where variance is large. Finally, but not surprisingly, when the gap between the upper bound solution and the lower bound used becomes large, then the IB approach will have difficulty solving the instances. Thus, to improve the method the incorporation of constraint (2) could be investigated, but we have left this for future research.

6. Concluding remarks and future research

In this paper we have illustrated how a stochastic knapsack problem (SKP) can be modeled as a shortest path problem with resource constraints (SPPRC). We have demonstrated that applying incremental extension combined with a state bounding procedure yields the best results of the approaches

tested. Furthermore, it is shown that the incremental extension approach provides a reasonable heuristic for the problem.

In relation to our motivation for solving the problem, we have applied this approach to a generalized assignment problem based formulation for the dynamic surgery allocation problem of Range et al. (2016). In this context the number of potential surgeries within the T time units is not larger than 100, and our variant of the SKP works very well as the pricing problem of a column generation procedure.

We have limited ourselves to the case where we have sets of mutually excluding items as well as convex cost of overfilling and bounds on the potential overfilling. The method is not limited to this, however. Below we give examples of potential extensions of the model, which can easily be added.

As the problem is formulated as an SPPRC, it is possible to include more resource constraints. For instance, if a bound on the maximal number of items is present, then it can be formulated by using a resource extension function and resource windows on each of the node in the item graph. Likewise, it is possible to include a resource extension function and corresponding resource windows guaranteeing that at least a specific number of items are selected.

In the case where we have an integer knapsack problem, each of the item types can be formulated as a group, where visiting a node in the group corresponds to selecting a specific number of that item type. In this case, it is possible to have individual revenues – which are not necessarily linear in the number of items – on the number of items selected of each type. This could be the case when it is desirable to have multiple items of the same type in the solution.

Fundamentally, knapsack problems do not incorporate sequencing of the items selected to be in the knapsack. However, the shortest-path-based approach described in this paper can be generalized to include sequencing. This would require that we reformulate the acyclic graph to a potentially cyclic graph where we would only be allowed to visit each group maximally once. As a consequence, the problem becomes an elementary shortest path problem with resource constraints. This problem is, however, strongly $\mathcal{NP}$-hard, and one should expect that the size of the instances that we would be able to solve is significantly smaller than the size of the instances discussed in the present paper.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, *46*, 316–329. doi:10.1287/opre.46.3.316.

Burke, E. K., & Curtois, T. (2014). New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, *237*, 71 – 81. doi:10.1016/j.ejor.2014.01.039.

Chabrier, A. (2006). Vehicle routing problem with elementary shortest path based column generation.

Computers & Operations Research, *33*, 2972 – 2990. doi:10.1016/j.cor.2005.02.029. Part Special Issue: Constraint Programming.

Charnes, A., & Cooper, W. W. (1959). Chance-constrained programming. *Management Science*, *6*, 73–79. doi:10.1287/mnsc.6.1.73.

Chen, K., & Ross, S. M. (2015). Static stochastic knapsack problems. *Prob. Eng. Inf. Sci.*, *29*, 527–546. doi:10.1017/s0269964815000170.

Dean, B. C., Goemans, M. X., & Vondrák, J. (2008). Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of OR*, *33*, 945 – 964. doi:10.1287/moor.1080.0330.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M. M., Soumis, F., & Villneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. G. Crainic, & G. Laporte (Eds.), *Fleet Management and Logistics* (pp. 57–93). Kluwer. doi:10.1007/978-1-4615-5755-5_3.

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, *40*, 342–354. doi:10.1287/opre.40.2.342.

Desrochers, M., & Soumis, F. (1988). A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, *26*, 191–212. doi:10.1080/03155986.1988.11732063.

Disser, Y., Klimm, M., Megow, N., & Stiller, S. (2017). Packing a knapsack of unknown capacity. *SIAM Journal on Discrete Mathematics*, *31*, 1477–1497. doi:10.1137/16M1070049.

Eveborn, P., & Rönnqvist, M. (2004). Scheduler – a system for staff planning. *Annals of Operations Research*, *128*, 21–45. doi:10.1023/B:ANOR.0000019097.93634.07.

Feillet, D., Dejax, P., Gendreau, M., & Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, *44*, 216–229. doi:10.1002/net.20033.

Figueira, J. R., Tavares, G., & Wiecek, M. M. (2010). Labeling algorithms for multiple objective integer knapsack problems. *Computers & Operations Research*, *37*, 700 – 711. doi:10.1016/j.cor.2009.06.026.

Frieze, A. M. (1976). Shortest path algorithms for knapsack type problems. *Mathematical Programming*, *11*, 150–157. doi:10.1007/bf01580382.

Gauvin, C., Desaulniers, G., & Gendreau, M. (2014). A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. *Computers & Operations Research*, *50*, 141 – 153. doi:10.1016/j.cor.2014.03.028.

Gilmore, P. C., & Gomory, R. E. (1966). The theory and computation of knapsack functions. *Operations Research*, *14*, 1045–1074. doi:10.1287/opre.14.6.1045.

Hans, E., Wullink, G., van Houdenhoven, M., & Kazemier, G. (2008). Robust surgery loading. *European Journal of Operational Research*, *185*, 1038 – 1050. doi:10.1016/j.ejor.2006.08.022.

Henig, M. I. (1990). Risk criteria in a stochastic knapsack problem. *Operations Research*, *38*, 820–825. doi:10.1287/opre.38.5.820.

İlhan, T., Iravani, S. M. R., & Daskin, M. S. (2011). The adaptive knapsack problem with stochastic rewards. *Operations Research*, *59*, 242 – 248. doi:10.1287/opre.1100.0857.

Irnich, S. (2008). Resource extension functions: properties, inversion, and generalization to segments. *OR Spectrum*, *30*, 113–148. doi:10.1007/s00291-007-0083-6.

Irnich, S., & Desaulniers, G. (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, & M. Solomon (Eds.), *Column Generation* (pp. 33–65). Springer US. doi:10.1007/0-387-25486-2_2.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-24777-7.

Kleywegt, A. J., & Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations Research*, *46*, 17–35. doi:10.1287/opre.46.1.17.

Kleywegt, A. J., & Papastavrou, J. D. (2001). The dynamic and stochastic knapsack problem with random sized items. *Operations Research*, *49*, 26–41. doi:10.1287/opre.49.1.26.11185.

Kleywegt, A. J., Shapiro, A., & de Mello, T. H. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, *12*, 479–502. doi:10.1137/S1052623499363220.

Klopfenstein, O., & Nace, D. (2008). A robust approach to the chance-constrained knapsack problem. *Operations Research Letters*, *36*, 628 – 632. doi:10.1016/j.orl.2008.03.006.

Kosuch, S., & Lisser, A. (2010). Upper bounds for the 0-1 stochastic knapsack problem and a b&b algorithm. *Annals of Operations Research*, *176*, 77–93. doi:10.1007/s10479-009-0577-5.

Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*, 378 – 384. doi:10.1016/j.cor.2012.07.008.

Lusby, R., Dohn, A., Range, T. M., & Larsen, J. (2012). A column generation-based heuristic for rostering with work patterns. *Journal of the Operational Research Society*, *63*, 261–277. doi:10.1057/jors.2011.27.

Martello, S., & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc.

Merzifonluoğlu, Y., Geunes, J., & Romeijn, H. (2012). The static stochastic knapsack problem with normally distributed item sizes. *Mathematical Programming*, *134*, 459–489. doi:10.1007/s10107-011-0443-5.

Minoux, M., & Ribeiro, C. (1984). A transformation of hard (equality constrained) knapsack problems into constrained shortest path problems. *Operations Research Letters*, *3*, 211–214. doi:10.1016/0167-6377(84)90028-2.

Morton, D. P., & Wood, R. K. (1998). On a stochastic knapsack problem and generalizations. In D. L. Woodruff (Ed.), *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search: Interfaces in Computer Science and Operations Research* (pp. 149–168). Boston, MA: Springer US. doi:10.1007/978-1-4757-2807-1_5.

Özaltin, O. Y., Prokopyev, O. A., & Schaefer, A. J. (2010). The bilevel knapsack problem with stochastic right-hand sides. *Operations Research Letters*, *38*, 328–333. doi:10.1016/j.orl.2010.04.005.

Pinedo, M. L. (2009). Planning and scheduling in health care. *Planning and Scheduling in Manufacturing and Services*, (pp. 291–316). doi:10.1007/978-1-4419-0910-7_12.

Range, T. M., Kozlowski, D., & Petersen, N. C. (2016). Dynamic job assignment: A column generation approach with an application to surgery allocation. Discussion Papers on Business and Economics, University of Southern Denmark.

Righini, G., & Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, *3*, 255 – 273. doi:10.1016/j.disopt.2006.05.007. Graphs and Combinatorial Optimization.

Smet, P., Ernst, A. T., & Berghe, G. V. (2016). Heuristic decomposition approaches for an integrated task scheduling and personnel rostering problem. *Computers & Operations Research*, *76*, 60 – 72. doi:10.1016/j.cor.2016.05.016.

Toth, P. (1980). Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, *25*, 29–45. doi:10.1007/BF02243880.

Witchakul, S., Ayudhaya, P. S. N., & Charnsethikul, P. (2008). A stochastic knapsack problem with continuous random capacity. *Journal of Mathematics and Statistics*, *4*, 269–276. doi:10.3844/jmssp.2008.269.276.

Appendix A. Proofs

Proof of Proposition 1. Assume that $X_1, \ldots, X_n$ are independent stochastic variables with means μ_i and variances σ_i^2 for $i = 1, \ldots, n$, and that $T \geq 0$ is a constant. Denote $X = X_1 + \ldots + X_n$, $\mu_X = \mu_1 + \ldots + \mu_n$ and $\sigma_X^2 = \sigma_1^2 + \ldots + \sigma_n^2$. By the central limit theorem we have that X is approximately normally distributed (i.e., $X \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right)$).

We can now calculate the expected value of the stochastic variable $O = (X - T)^+$

$$\mathbb{E}[O] \approx \int_T^\infty (x - T) f_X(x) dx \quad . \tag{A.1}$$

Let

$$T = \mu_X + k\sigma_X \quad . \tag{A.2}$$

Then we have

$$\mathbb{E}[O] \approx \int_{\mu_X + k\sigma_X}^\infty (x - \mu_X - k\sigma_X) f_X(x) dx \tag{A.3}$$

$$= \int_{\mu_X + k\sigma_X}^\infty (x - \mu_X - k\sigma_X) \frac{1}{\sigma_X \sqrt{2\pi}} e^{-\frac{(x-\mu_X)^2}{2\sigma_X^2}} dx \quad . \tag{A.4}$$

After substituting $u = (x - \mu_X)\sigma_X$ and simplifying we get

$$\mathbb{E}[O] \approx \sigma_X \int_k^\infty (u - k) \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du, \tag{A.5}$$

which is a special function of the unit normal distribution. Specifically, let

$$G_u(k) = \int_k^\infty (u - k) \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du \quad . \tag{A.6}$$

Using the special property of the unit normal distribution that

$$\int_k^\infty u \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du = \frac{1}{\sqrt{2\pi}} e^{-\frac{k^2}{2}} \quad . \tag{A.7}$$

(A.6) can be expressed as

$$G_u(k) = \frac{1}{\sqrt{2\pi}} e^{-\frac{k^2}{2}} - k \int_k^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du \tag{A.8}$$

$$= \phi(k) - k(1 - \Phi(k)) \tag{A.9}$$

where ϕ and Φ denote probability density function and cumulative distribution function of a unit normal random variable, respectively. As a result the expected overfill can be represented as a simple function of unit normal distribution

$$\mathbb{E}[O] \approx \sigma_X \left[\phi(k) - k(1 - \Phi(k))\right] \tag{A.10}$$

where $k = (T - \mu_X)/\sigma_X$. □

Proof of Proposition 2. Let

$$\begin{aligned} h_T(\mu,\sigma) &= \sigma\left[\phi\left(\tfrac{T-\mu}{\sigma}\right) - \tfrac{T-\mu}{\sigma}\left(1-\Phi\left(\tfrac{T-\mu}{\sigma}\right)\right)\right] \\ &= \sigma\phi\left(\tfrac{T-\mu}{\sigma}\right) + (T-\mu)\Phi\left(\tfrac{T-\mu}{\sigma}\right) - (T-\mu) \end{aligned}$$

for fixed T. First we identify the first order derivative of $h_T(\mu,\sigma)$ with respect to μ

$$\begin{aligned} \frac{\partial h_T(\mu,\sigma)}{\partial \mu} &= \sigma\frac{\partial}{\partial\mu}\left(\phi\left(\tfrac{T-\mu}{\sigma}\right)\right) - \frac{\partial}{\partial\mu}(T-\mu) + \frac{\partial}{\partial\mu}\left((T-\mu)\Phi\left(\tfrac{T-\mu}{\sigma}\right)\right) \\ &= \sigma\tfrac{T-\mu}{\sigma^2}\phi\left(\tfrac{T-\mu}{\sigma}\right) + 1 - \tfrac{T}{\sigma}\phi\left(\tfrac{T-\mu}{\sigma}\right) - \Phi\left(\tfrac{T-\mu}{\sigma}\right) + \tfrac{\mu}{\sigma}\phi\left(\tfrac{T-\mu}{\sigma}\right) \\ &= 1-\Phi\left(\tfrac{T-\mu}{\sigma}\right) \quad . \end{aligned}$$

We can calculate the first order derivative of $h_T(\mu,\sigma)$ with respect to σ as

$$\begin{aligned} \frac{\partial h_T(\mu,\sigma)}{\partial \sigma} &= \frac{\partial}{\partial\sigma}\left(\sigma\phi\left(\tfrac{T-\mu}{\sigma}\right)\right) + (T-\mu)\frac{\partial}{\partial\sigma}\left(\Phi\left(\tfrac{T-\mu}{\sigma}\right)\right) \\ &= \frac{\partial}{\partial\sigma}\left(\frac{\sigma}{\sqrt{2\pi}}e^{-\left(\frac{T-\mu}{\sigma}\right)^2/2}\right) + (T-\mu)\frac{\partial}{\partial\sigma}\left(\int_{-\infty}^{\frac{T-\mu}{\sigma}}\frac{1}{\sqrt{2\pi}}e^{-\frac{u^2}{2}}du\right) \\ &= \frac{1}{\sqrt{2\pi}}\left(\frac{T-\mu}{\sigma}\right)^2 e^{-\left(\frac{T-\mu}{\sigma}\right)^2/2} + \frac{1}{\sqrt{2\pi}}e^{-\left(\frac{T-\mu}{\sigma}\right)^2/2} - \frac{1}{\sqrt{2\pi}}\left(\frac{T-\mu}{\sigma}\right)^2 e^{-\left(\frac{T-\mu}{\sigma}\right)^2/2} \\ &= \tfrac{1}{\sqrt{2\pi}}e^{-\left(\frac{T-\mu}{\sigma}\right)^2/2} \\ &= \phi\left(\tfrac{T-\mu}{\sigma}\right) \quad . \end{aligned}$$

□

Proof of Proposition 3. From Proposition 2 we know that for $\sigma > 0$ the function of expected overfill is strictly increasing, and consequently strictly decreasing when we decrease σ. We now show that when we decrease σ towards zero the expected overtime will converge to $\max\{0;\mu-T\}$.

Suppose that we decrease σ towards zero. Then we observe three cases:

$T-\mu=0$: Only the term $\sigma\phi\left(\tfrac{T-\mu}{\sigma}\right)$ matters, as the remaining terms are zero. We have that $\phi(0) = 1/\sqrt{2\pi}$ and as a result $\sigma\phi(0)\to 0$ as $\sigma\to 0$.

$T-\mu>0$: In this case we have that for $\sigma\searrow 0$ then $\frac{T-\mu}{\sigma}\nearrow\infty$ and for some number $k\nearrow\infty$ we have that $\phi(k)\to 0$ and $\Phi(k)\to 1$. Thus $h_T(\mu,\sigma)\to 0$ for $\sigma\searrow 0$.

$T-\mu<0$: When $T-\mu$ is negative we have that $\frac{T-\mu}{\sigma}\searrow-\infty$ when $\sigma\searrow 0$ and for some number $k\searrow 0$ we have that $\phi(k)\to 0$ and $\Phi(k)\to 0$, and consequently $h_T(\mu,\sigma)\to -(T-\mu)=\mu-T$.

Thus, the function $h_T(\mu,\sigma)$ converges monotonically from above towards $\max\{0;\mu-T\}$. □

Proof of Proposition 4. The proof is in two parts, where the first part proves that any feasible extension of $\mathcal{P}_2$ is also a feasible extension of $\mathcal{P}_1$ while the second part is to prove that for any extension $\mathcal{Q}$ of $\mathcal{P}_2$ the same extension of $\mathcal{P}_1$ will result in no larger costs (i.e., that $C(\mathcal{P}_1,\mathcal{Q}) \le C(\mathcal{P}_2,\mathcal{Q})$).

To show that any extension of $\mathcal{P}_2$ is also a feasible extension of $\mathcal{P}_1$ amounts to checking (7) because (6) and (8) are automatically satisfied due to the network structure. Suppose that $\mathcal{Q}$ is a feasible extension of $\mathcal{P}_2$, then we have that $\mu(\mathcal{P}_2,\mathcal{Q})+\beta\sqrt{\sigma^2(\mathcal{P}_2,\mathcal{Q})} \leq T+S$. But as both $\mu(\cdot)$ and $\sigma^2(\cdot)$ are additive we have that $\mu(\mathcal{P}_2,\mathcal{Q})=\mu(\mathcal{P}_2)+\mu(\mathcal{Q})$ and $\sigma^2(\mathcal{P}_2,\mathcal{Q})=\sigma^2(\mathcal{P}_2)+\sigma^2(\mathcal{Q})$. Consequently, we have

$$\begin{aligned} T+S &\geq \mu(\mathcal{P}_2,\mathcal{Q})+\beta\sqrt{\sigma^2(\mathcal{P}_2,\mathcal{Q})} \\ &= \mu(\mathcal{P}_2)+\mu(\mathcal{Q})+\beta\sqrt{\sigma^2(\mathcal{P}_2)+\sigma^2(\mathcal{Q})} \\ &\geq \mu(\mathcal{P}_1)+\mu(\mathcal{Q})+\beta\sqrt{\sigma^2(\mathcal{P}_1)+\sigma^2(\mathcal{Q})} \\ &= \mu(\mathcal{P}_1,\mathcal{Q})+\beta\sqrt{\sigma^2(\mathcal{P}_1,\mathcal{Q})} \end{aligned}$$

showing that $\mathcal{P}_1$ satisfies (7) and therefore any feasible extension of $\mathcal{P}_2$ will also be a feasible extension of $\mathcal{P}_1$.

Now we turn to show that the feasible extension $\mathcal{Q}$ of $\mathcal{P}_2$ will yield no larger cost when applied to $\mathcal{P}_1$ than when it is applied to $\mathcal{P}_2$. First observe that when (1)-(3) of Proposition 4 are satisfied then $-r(\mathcal{P}_1) \leq -r(\mathcal{P}_2)$ and

$$f\left(h_T\left(\mu(\mathcal{P}_1),\sqrt{\sigma^2(\mathcal{P}_1)}\right)\right) \leq f\left(h_T\left(\mu(\mathcal{P}_2),\sqrt{\sigma^2(\mathcal{P}_2)}\right)\right)$$

due to Proposition 2. Therefore we know that $C(\mathcal{P}_1) \leq C(\mathcal{P}_2)$. Secondly, due to the additivity of the revenue we will obtain the same difference in revenue for the extended paths (i.e., that $r(\mathcal{P}_1,\mathcal{Q}) - r(\mathcal{P}_2,\mathcal{Q}) = r(\mathcal{P}_1) - r(\mathcal{P}_2)$) meaning that the extension will result in the same increase in revenue for both $\mathcal{P}_1$ and $\mathcal{P}_2$. Again, due to the additivity of the mean and the variance we have that

$$\mu(\mathcal{P}_1,\mathcal{Q}) = \mu(\mathcal{P}_1)+\mu(\mathcal{Q}) \leq \mu(\mathcal{P}_2)+\mu(\mathcal{Q}) = \mu(\mathcal{P}_2,\mathcal{Q})$$

and

$$\sqrt{\sigma^2(\mathcal{P}_1,\mathcal{Q})} = \sqrt{\sigma^2(\mathcal{P}_1)+\sigma^2(\mathcal{Q})} \leq \sqrt{\sigma^2(\mathcal{P}_2)+\sigma^2(\mathcal{Q})} = \sqrt{\sigma^2(\mathcal{P}_2,\mathcal{Q})} \quad .$$

As both $h_T(\cdot,\cdot)$ and $f(\cdot)$ are non-decreasing we have that

$$f\left(h_T\left(\mu(\mathcal{P}_1,\mathcal{Q}),\sqrt{\sigma^2(\mathcal{P}_1,\mathcal{Q})}\right)\right) \leq f\left(h_T\left(\mu(\mathcal{P}_2,\mathcal{Q}),\sqrt{\sigma^2(\mathcal{P}_2,\mathcal{Q})}\right)\right)$$

and therefore $C(\mathcal{P}_1,\mathcal{Q}) \leq C(\mathcal{P}_2,\mathcal{Q})$, which concludes the proof.

□