

A cross entropy algorithm for the Knapsack problem with setups

M. Caserta*, E. Quiñonez Rico, A. Márquez Uribe

Instituto Tecnológico de Monterrey, Calle del Puente, 222, Col. Ejidos de Huipulco, Del. Tlalpan, México DF, 14380, México

Available online 2 May 2006

Abstract

In this article we propose a new metaheuristic-based algorithm for the Integer Knapsack Problem with Setups. This problem is a generalization of the standard Integer Knapsack Problem, complicated by the presence of setup costs in the objective function as well as in the constraints. We propose a cross entropy based algorithm, where the metaheuristic scheme allows to relax the original problem to a series of well chosen standard Knapsack problems, solved through a dynamic programming algorithm. To increase the computational effectiveness of the proposed algorithm, we use a turnpike theorem, which sensibly reduces the number of iterations of the dynamic algorithm. Finally, to testify the robustness of the proposed scheme, we present extensive computational results. First, we illustrate the step-by-step behavior of the algorithm on a smaller, yet difficult, problem. Subsequently, to test the solution quality of the algorithm, we compare the results obtained on very large scale instances with the output of a branch and bound scheme. We conclude that the proposed algorithm is effective in terms of solution quality as well as computational time.

© 2006 Elsevier Ltd. All rights reserved.

Keywords: Combinatorial optimization; Metaheuristics; Knapsack Problem; Scheduling; Dynamic programming; Cross entropy

1. Introduction

The Integer Knapsack Problem with Setup is described as an integer Knapsack Problem with additional fixed costs of setup discounted both in the objective function and in the constraints. In the literature, it raises in two different contexts. First, it may be considered as a natural generalization of the pure integer knapsack problem. Secondly, it is viewed as a particular case of a well studied problem, the Multi-Item Capacitated Lot Sizing Problem with Setup Times, as presented in Miller et al. [1]. In both cases, the study of the Integer Knapsack Problem with Setups presented in this paper gives some insight on its importance as combinatorial problem. Moreover, it provides further insight of the difficulty of each one of these problems. In addition, very recently, a new application of the integer knapsack problem with setups has been proposed in Jacobson et al. [2], where one wants to optimize a baggage screening performance measure subject to a finite amount of resources.

Due to its vast industrial applicability, researchers have devoted special attention to the Multi-item Capacitated Lot sizing problem MCL with setup times (see Dzielinski and Gomory [3], Pochet and Wolsey [4], Eppen and Martin [5], and Constantino [6]), where one wants to find the optimal schedule of a set of items over a time horizon, with items competing for a shared capacity. The problem is complicated by the existence of setup costs as well as inventory

* Corresponding author. Tel.: +52 55 5483 2189; fax: +52 55 5483 2163.

E-mail addresses: marco.caserta@itesm.mx (M. Caserta), quinonez@itesm.mx (E. Quiñonez Rico), alberto.marquez@itesm.mx (A. Márquez Uribe).

costs. For this reason, a trade off exists between producing an item at every period, and, consequently, paying less inventory but more setup, or producing more sparingly, hence paying less setup but more inventory. Since the multi-item capacitated lot sizing problem with setups is very difficult to solve to optimality, many researchers have tried to tackle the problem by working on relaxations of the same. A good description of some well studied relaxations of MCL is provided by Miller et al. [1].

In the literature we can find two broad approaches to MCL, both based upon the definition of a relaxation of the original problem. On the one hand, some authors tackle the problem by relaxing the time dimension, which is, they define the single period relaxation, where only one period at a time is considered. On the other hand, other researchers propose a different kind of relaxation, based upon the conversion of the original, multi-item problem, to a single-item, multi-period lot sizing problem with setups.

Miller et al. [7] exploit the first kind of relaxation, based upon the reduction of the model to a single-period problem. They call this relaxation PI, for preceding inventory, since the model takes into account initial inventory levels for each item, inherited from the preceding period. The solution approach is based upon the study of the polyhedral structure of the model and the definition of cover inequalities for PI, which induce facets of the convex hull.

In a different study, Miller et al. [8] present a special case of PI, the multi-item single-period capacitated lot sizing problem with setups, called PIC, where setup times and demands are constant for all items. For this special case, the authors provide a polynomial time algorithm. They prove that the inequalities identified in Miller et al. [7] suffice to solve PIC by linear programming in polynomial time.

With respect to the second type of relaxation, the single item relaxation, some indications are provided in Pochet [9], Pochet and Wolsey [4] and Miller et al. [10]. They provide guidelines about how to derive valid inequalities for the single-item problem with the aim of applying the same inequalities to the multi-item problem.

Quite differently, Trigeiro et al. [11] tackle the problem by relaxing the capacitated lot sizing to an uncapacitated lot sizing problem, where the lagrangian relaxation of the capacity constraints is exploited. The original problem is decomposed into a set of uncapacitated single item lot sizing problem. The lagrangian dual costs are then updated by subgradient optimization and the single item problem is solved via dynamic programming.

In this paper we propose a different approach to MCL, somehow related to the single-period relaxation. However, rather than solving the single-period problem through a study of the polyhedral structure of the model, we reduce the single-period problem to a series of Integer Knapsack Problems with Setups. We effectively solve each Knapsack Problem by using a metaheuristic-based scheme, which allows to simplify the Knapsack Problem with Setups to a set of standard Knapsack Problems, easily solved via dynamic programming.

The paper has the following structure: Section 2 presents the formulation of the Integer Knapsack Problem with Setups, along with an explanation of how MCL can be reduced to a series of Knapsack Problems. Section 3 offers a brief introduction to the Cross-Entropy (CE) method and describes the proposed CE scheme for the Knapsack Problem. Section 4 presents the overall algorithm, while Section 5 offers an indication of the effectiveness of the algorithm by comparing the proposed algorithm with two well-known algorithms for the knapsack problem. Furthermore, we present computational results, both on small problems and on very large scale instances of the Knapsack Problem. Finally, Section 6 summarizes results and findings of the paper.

2. Integer Knapsack problem with setups

The Integer Knapsack Problem with Setups is concerned with the choice of the set of production items of maximum value, where a unitary value is associated to each item. Furthermore, each item takes some unitary amount of resources. In addition, a fixed amount of resources is consumed when the production line of an item is setup and, consequently, a fixed cost must be discounted from the overall production schedule value. Indeed, in the context of programming several items on a single machine with a limited amount of available time, switching among items takes some time to setup the machine, hence reducing the time available for actual production. Moreover, a fixed cost must be discounted from the total production value, since no item is produced while setting up the machine.

Let $y_j \in \mathbb{Z}_+$ indicate the number of items of type j chosen, with $j = 1, \dots, n$. Let us set $x_j \in \mathbb{B}$ to 1 if at least one unit of item j is chosen and to 0 otherwise. A standard formulation of the Integer Knapsack Problem with Setups as a

linear model is

$$\begin{aligned} \max \quad & \sum_{j=1}^n (c_j y_j - f_j x_j) \\ \text{IKPS: s.t.} \quad & \sum_{j=1}^n (a_j y_j + m_j x_j) \leq b, \\ & y_j - M x_j \leq 0 \quad \text{for } j = 1, 2, \dots, n \\ & \mathbf{y} \in \mathbb{Z}_+^n \quad \text{and} \quad \mathbf{x} \in \mathbb{B}^n. \end{aligned}$$

The number n stands for the number of items, c_j and a_j are the unitary value and unitary resource consumption of item i , respectively. Parameters f_j and m_j represent the setup cost and setup time that must be paid before any production of item i can occur. Finally, $M > 0$ is a large number. The objective of the problem is to select the appropriate number of each item, having under consideration setup times. The first constraint ensures that the production plan does not use more resources than available, including resources used by setup times. The second set of constraints ensures that appropriate setup time and cost are paid whenever production of an item occurs. Throughout this paper it is assumed that $c_j > 0$, $a_j > 0$, $f_j \geq 0$, and $m_j \geq 0$ for all $j = 1, 2, \dots, n$. Additionally, without loss of generality, it will be assumed that $b - m_j - a_j > 0$ for all $j = 1, \dots, n$.

Model IKPS is encountered as a subproblem in a number of well-known problems, such as capacitated lot-sizing problems or cutting stock problems. Furthermore, IKPS could be seen as a special case of a more general knapsack problem, known as multiple-class integer knapsack problem (MCKP) when each class holds a single item. Some interesting details about the relation among different kinds of knapsack problems is provided in Perrot and Vanderbeck [12] as well as in Kellerer et al. [13].

Some approaches to IKPS presented in the literature are those of Sural et al. [14], and Perrot and Vanderbeck [12]. Sural et al. [14] studied a special case of IKPS, taking $f_j = 0$ and $a_j = 1$ for all items j . They proposed a primal heuristic used within a branch-and-bound framework. Perrot and Vanderbeck [12] proposed a dynamic programming scheme for IKPS that is a generalization of the standard dynamic programming for the integer Knapsack problem. The straightforward application of the scheme leads to an algorithm that runs in $\mathcal{O}(nb^2)$ time.

A quite different approach to solve an IKPS problem is pursued by transforming it into a more versatile knapsack formulation, known as MCKP, for which efficient algorithms have been recently proposed. An interesting approach is provided by Pisinger [15], where a minimum algorithm for the multiple choice knapsack problem is proposed. The author presents an efficient algorithm that first uses a partitioning algorithm to solve the linear programming relaxation of MCKP. The solution provided by the partitioning algorithm is employed as starting feasible solution by a dynamic programming algorithm applied to the integer knapsack problem. The algorithm always works on a subset of classes, called *core*, that is expanded while the problem is solved. Furthermore, in order to reduce the number of variables in the problem, some preprocessing and variable fixing techniques are employed. It is worth noting that any IKPS could be reduced to MCKP by first creating a class for each item and then transforming the integer problem into its binary counterpart. To transform IKPS into MCKP, let us create a class j , corresponding to item j with ub_j elements, where

$$ub_j = \left\lfloor \frac{b - m_j}{a_j} \right\rfloor$$

provides the maximum number of articles of item j that could be selected respecting the knapsack constraint. Consequently, the class for item j would be made up by $ub_j + 1$ elements, namely $\{0, 1, \dots, ub_j\}$, each one associated to a binary variable y_{ij} and indicating the selection of $0, 1, \dots, ub_j$ elements of item j , respectively. Obviously, we need to enforce $\sum_{i=0}^{ub_j} y_{ij} = 1$. Setup costs and setup times would be implicitly considered in the definition of the new profits and resource usage for each element within the class. For example, for a given class j ($j = 1, \dots, n$), the unit profit value could be defined as:

$$c_{ij} = \begin{cases} 0 & \text{for } i = 0, \\ i \times c_j - f_j & \text{for } i = 1, \dots, ub_j. \end{cases}$$

A similar transformation is carried out with each a_{ij} . Finally, the MCKP formulation would be:

$$\begin{aligned}
 \text{(MCKP):} \quad & \max \quad \sum_{j=1}^n \sum_{i=0}^{ub_j} c_{ij} y_{ij} \\
 & \text{s.t.} \quad \sum_{j=1}^n \sum_{i=0}^{ub_j} a_{ij} y_{ij} \leq b \\
 & \quad \sum_{i=1}^{ub_j} y_{ij} = 1 \quad \text{for } j = 1, 2, \dots, n, \\
 & \quad \mathbf{y} \in \mathbb{B}^{\bar{n}}
 \end{aligned}$$

where $\bar{n} = \sum_{j=1}^n ub_j + n$ indicates the total number of binary variables. The problem has n classes (one for each item in IKPS) and \bar{n} binary variables. A comparison of the proposed algorithm with the MCKP presented by Pisinger [15], both in terms of computational complexity and performances is presented in Section 5.

An alternative transformation of IKPS to MCKP is possible if setup costs and times are treated explicitly, as in Perrot and Vanderbeck [12]. Once again, they suggest the creation of a class for each item in IKPS. However, this time, within each class, n_j elements are defined, where n_j is equal to $\lfloor \log_2 ub_j \rfloor + 1$. In this case, a 0–1 variable for each bit needed to represent the binary counterpart of each possible integer solution is created. Obviously, each feasible integer solution of IKPS is uniquely represented by a feasible binary solution of MCKP. Within each class j , unit profit value and unit resource usage are defined as follows:

$$\begin{aligned}
 c_{ij} &= 2^{i-1} c_j \quad \text{for } i = 1, \dots, n_j \quad \text{and} \quad j = 1, \dots, n, \\
 a_{ij} &= 2^{i-1} a_j \quad \text{for } i = 1, \dots, n_j \quad \text{and} \quad j = 1, \dots, n.
 \end{aligned}$$

The main advantage of this transformation over the previous one is that we do not need to ensure anymore that only one element within each class is chosen and, consequently, the family of constraints $\sum_{i=0}^{ub_j} y_{ij} = 1$ can be eliminated from the formulation. However, the main disadvantage of this representation is that bounds on the original variables y_j must be treated explicitly. Perrot and Vanderbeck propose a dynamic programming algorithm for this problem that runs in $\mathcal{O}(nb^2 \log b)$ time in the case with tight bound on variables and $\mathcal{O}(nb \log b)$ time when the bounds are not tight, which is, the explicit upper bound for item j is greater than ub_j .

Section 5 presents a comparisons of the proposed metaheuristic algorithm with these two multiple-class models both from the computational complexity and computational results on very large instances of the IKPS.

Our interest in the IKPS problem arises from the study of a well-known production planning problem, the general lot-sizing problem, known as MCL. Let us study the maximization version of MCL. We claim that IKPS is a special case of MCL. This problem assumes that each item has a unitary value and a unitary cost of inventory varying over each period of the planning horizon. Additionally, any time the production of an item is started, setup costs and times must be discounted from the objective function and from the resource availability constraint, respectively. Let $y_{jt} \in \mathbb{Z}_+$ be the number of articles of item j produced in period t ; let s_{jt} be the number of articles of item j held as inventory in period t ; furthermore, $x_{jt} \in \mathbb{B}$ is set to 1 if at least an article of item j is produced and to 0 otherwise. The resulting linear model is:

$$\begin{aligned}
 \text{MCL:} \quad & \max \quad \sum_{j=1}^n \sum_{t=1}^T (c_{jt} y_{jt} - f_{jt} x_{jt} - h_{jt} s_{jt}) \\
 & \text{s.t.} \quad \begin{cases} y_{jt} + s_{j,t-1} - s_{jt} \leq d_{jt} & \text{for } j = 1, 2, \dots, n, \\ y_{jt} - Mx_{jt} \leq 0 & \text{for } t = 1, 2, \dots, T. \end{cases} \\
 & \quad \sum_{j=1}^n (a_{jt} y_{jt} + m_{jt} x_{jt}) \leq T_t \quad \text{for } t = 1, 2, \dots, T. \\
 & \quad s_{j0} = s_{jT} = 0 \quad \text{for } j = 1, 2, \dots, n. \\
 & \quad \mathbf{y} \in \mathbb{Z}_+^n, \quad \mathbf{x} \in \mathbb{B}^n \quad \text{and} \quad \mathbf{s} \in \mathbb{Z}_+^n.
 \end{aligned}$$

The symbols n and T stand for the number of items and the number of periods, respectively, c_{jt} indicates the value of item j in period t , h_{jt} is the inventory holding cost of item j during period t , and f_{jt} and m_{jt} are the fixed costs and the setup times associated to item j during period t , respectively. Clearly, if we consider only one period, this is, $T = 1$, and we dualize the demand constraints, MCL is reduced to IKSP.

3. Cross entropy framework

The main idea of CE is related to the design of an effective learning mechanism throughout the search. It associates an estimation problem to the original combinatorial optimization problem, called the associated stochastic problem, characterized by a density function Φ . The stochastic problem is solved by identifying the optimal importance sampling density Φ^* , which is the one that minimizes the Kullback–Leibler distance with respect to the original density Φ . This distance is also called the CE between Φ and Φ^* . The minimization of the CE leads to the definition of “optimal” updating rules for the density functions and, consequently, to the generation of improved feasible vectors. The method terminates when convergence to a point in the feasible region is achieved.

The most important features of CE have been thoroughly exposed in de Boer et al. [16]. For this reason, in this paper we only present those features of CE that are relevant to the problem hereby discussed.

Let us consider the general 0–1 integer maximization problem

$$(P): z^* = \max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}),$$

where $\mathcal{X} \subset \mathbb{B}^n$ represents the feasible region. The CE method associates a stochastic estimation problem to (P) aimed at estimating

$$\mathbb{P}(S(\mathbf{x}) \geq z).$$

With this aim, let us define a family of density functions Φ on \mathcal{X} , parameterized by a vector \mathbf{p} . For example, let us assume that we can define a family of Bernoulli density functions:

$$\Phi(\mathbf{x}, \mathbf{u}) = \prod_{i=1}^n (u_i)^{x_i} (1 - u_i)^{1-x_i}. \tag{1}$$

Consequently, the associated stochastic estimation problem is

$$(EP): \mathbb{P}(S(\mathbf{x}) \geq z) = \sum_{\mathbf{x} \in \mathcal{X}} I_{\{S(\mathbf{x}) \geq z\}} \Phi(\mathbf{x}, \mathbf{u}),$$

where $I_{\{S(\mathbf{x}) \geq z\}}$ is the indicator function, whose value is 1 if $S(\mathbf{x}) \geq z$ and 0 otherwise.

We want to estimate $l = \mathbb{P}(S(\mathbf{x}) \geq z)$ for $z = z^*$. It is possible to estimate l via Importance Sampling (IS). Let us take a random sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from a different density function $\Phi(\mathbf{x}, \mathbf{p})$. In this case, we can use the likelihood ratio estimator

$$\hat{l} = \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq z\}} \frac{\Phi(\mathbf{x}, \mathbf{u})}{\Phi(\mathbf{x}, \mathbf{p})},$$

where $\Phi(\mathbf{x}, \mathbf{p})$ constitutes a change of measure and is chosen such that the CE between $\Phi(\mathbf{x}, \mathbf{p})$ and $\Phi(\mathbf{x}, \mathbf{u})$ is minimal. This corresponds to solving the problem

$$(SP): \max_{\mathbf{p}} \hat{D}(\mathbf{p}) = \max_{\mathbf{p}} \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq z\}} \ln \Phi(\mathbf{x}, \mathbf{p}),$$

where $\Phi(\mathbf{x}, \mathbf{p})$ is given by (1). To find the maximum of (SP), we set

$$\frac{\partial \hat{D}(\mathbf{p})}{\partial \mathbf{p}} = 0. \tag{2}$$

Finally, from (2), we get the optimal updating rule

$$\hat{p}_i = \frac{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq z\}} x_i}{\sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq z\}}}, \quad i = 1, \dots, n. \tag{3}$$

Rule (3) is iteratively used with the aim of generating a sequence of increasing threshold values z^0, z^1, \dots , converging either to the global optimum z^* or to a value closer to it. At each iteration t , the new value of z^t is used to generate a

better vector \mathbf{p}^t . The new vector \mathbf{p}^t is, in turn, used to draw a new sample population drawn under a different distribution function, which will lead to better values of z . The process stops when either we reach the global optimum value z^* or the vector \mathbf{p} converges to a vector in \mathcal{X} .

The main algorithm is made up by three phases: (1) draw a random sample population $\sim \text{Ber}(\mathbf{p})$; (2) project the infeasible vectors into the feasible region \mathcal{X} ; and (3) update density parameters via rule 3. Steps (1)–(3) are repeated until a stopping criterion is met. More details about the algorithm are provided in Section 4.

4. Algorithm for Knapsack problem with setup

In this section we present a CE-based algorithm for IKPS. Let us define the lower bound of a variable $y_j, j \in N$, as:

$$lb_j = \begin{cases} \left\lfloor \frac{f_j}{c_j} \right\rfloor + 1 & \text{if } b - a_j \left(\left\lfloor \frac{f_j}{c_j} \right\rfloor + 1 \right) \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

We assume that any item whose $lb_j = -1$ can be eliminated from the problem. This implies that an item is produced only if its fixed costs can be covered. Furthermore, given a binary vector $\mathbf{x} \in \mathbb{B}^n$, let us define $J^{\mathbf{x}} = \{j \in N : x_j = 1\}$ as the set of variables set to 1 in \mathbf{x} .

The proposed algorithm is based upon the CE framework presented in Section 3. The central idea of the algorithm is the design of an “intelligent” guessing mechanism. Problem IKPS is especially hard due to the presence of fixed costs and setup times. However, if we knew which items should be in the knapsack, the problem could be reduced to an integer knapsack problem and solved via a dynamic programming algorithm. A naive approach could be to randomly pick some of the items of the original problems, discount their fixed costs and setups, and solve the associated integer knapsack problem. However, since the number of possible combination of items is extremely large, the random approach is unpractical.

Based upon the meritorious CE Framework, we developed a scheme aimed at intelligently guess which items should be in the knapsack. At each iteration t , we produce a population of N binary vectors based upon a Bernoulli distribution $\Phi_t(\mathbf{x})$ characterized by a parameter vector \mathbf{p}_t :

$$\Phi_t(\mathbf{x}) = \prod_{j=1}^n p_{t,j}^{x_j} (1 - p_{t,j})^{1-x_j}, \quad x_j \in \{0, 1\}, \quad j = 1, \dots, n.$$

We define an *associated knapsack problem* AKP for each binary vector \mathbf{X}_i , which is:

$$\begin{aligned} \text{max} \quad & z = \sum_{j \in J^{\mathbf{X}_i}} c_j y_j, \\ \text{AKP: s.t.} \quad & \sum_{j \in J^{\mathbf{X}_i}} a_j y_j \leq \bar{b}, \\ & \mathbf{y} \in \mathbb{Z}_+^{|J^{\mathbf{X}_i}|}, \end{aligned}$$

where $\bar{b} = b - (\sum_{j \in J^{\mathbf{X}_i}} a_j lb_j + m_j)$. AKP is a classical integer knapsack problem derived from IKPS by assuming that at least lb_j articles are chosen for each item $j \in J^{\mathbf{X}_i}$. However, since we enforce that an item enters into the knapsack only if it covers its fixed costs, there is no guarantee that, for a given vector \mathbf{X}_i , the resulting \bar{b} be positive. For this reason, every time an infeasible vector \mathbf{X}_i is generated, we apply a heuristic scheme aimed at producing a new vector \mathbf{X}_i^F , whose AKP is feasible. The simple heuristic scheme employed to modify an infeasible Bernoulli vector \mathbf{X}_i is the following: we iteratively set to zero a component $x_j \in J^{\mathbf{X}_i}$ until a feasible AKP is created. At each iteration, the component x_j to be set to zero is chosen as:

$$j = \underset{k \in J^{\mathbf{X}_i}}{\text{argmin}} \left\{ c_k \frac{b_k - m_k}{a_k} - f_k \right\}.$$

In the following, let us suppose that the random vector \mathbf{X}_i be feasible, since it is always possible to apply the aforementioned scheme to any infeasible vector to reach a feasible solution. Given a random vector \mathbf{X}_i , we need to evaluate

how good is the guess, in terms of which items should be in the knapsack, by solving the AKP. The optimal solution of AKP is the resulting $S(\mathbf{X}_i)$ (see Section 3), which is the best solution for the knapsack problem with setup when only items in $J^{\mathbf{X}_i}$ are considered. Problem AKP can be solved by using the dynamic recursion:

$$z(d) = \begin{cases} 0, & d = 0, \dots, \bar{d} - 1, \\ \max \left\{ z(d - 1), \max_{\substack{j \in J^{\mathbf{X}_i} \\ a_j \leq d}} \{c_j + z(d - a_j)\} \right\}, & d = \bar{d}, \dots, \bar{b}, \end{cases} \quad (4)$$

where $\bar{d} = \min_{j \in J^{\mathbf{X}_i}} \{a_j\}$. It is easy to see that the objective function value of the optimal solution of the original knapsack problem is $z^* = z^{\text{AKP}} + \bar{z}$, where $z^{\text{AKP}} = z(\bar{b})$ and $\bar{z} = \sum_{j \in J^{\mathbf{X}_i}} c_j l b_j - f_j$.

The steps of the proposed algorithm are illustrated by Algorithm 1 below along with measures of complexity. The overall complexity of the algorithm is $\mathcal{O}(n' \bar{b} N \log N)$, where $n' \ll n$ is the number of items considered in AKP, $\bar{b} < b$ is the right-hand side value of the knapsack constraint after discounting setup times for those items in AKP and N is the cross-entropy population size (no more than 100 for every problem instance tested).

Algorithm 1: CE-Knapsack with Setup

- 1: generate initial Bernoulli probabilities $\mathbf{p}^0 \in (0, 1)$. Set $t := 0 - \mathcal{O}(n)$
 - 2: draw a sample population $\mathbf{X}_1, \dots, \mathbf{X}_N \sim \text{Ber}(\mathbf{p}^t) - \mathcal{O}(nN)$
 - 3: **for** each random vector \mathbf{X}_i **do**
 - 4: define (AKP) associated knapsack problem
 - 5: restore feasibility if needed $-\mathcal{O}(|J^{\mathbf{X}_i}|)$ projection scheme
 - 6: solve (AKP) $-\mathcal{O}(|J^{\mathbf{X}_i}| \bar{b})$. use dynamic recursion (4)
 - 7: **end for**
 - 8: sort sample population in ascending order w.r.t. the objective function value $-\mathcal{O}(N \log N)$.
 - 9: select best ρN points of sample population $-\mathcal{O}(\rho N)$ quantile
 - 10: compute \mathbf{p}^{t+1} as $(\mathcal{O}(\rho N))$: use updating rule (3)
 - $p_j^{t+1} = \frac{\sum_{i=\lceil \rho N \rceil}^N x_{ij}}{\lceil \rho N \rceil}, \quad j = 1, \dots, n$
 - 11: **if** $\mathbf{p}^{t+1} \in \mathbb{B}^n$ **then**
 - 12: STOP.
 - 13: **else**
 - 14: $t \leftarrow t + 1$ and go back to step 2
 - 15: **end if**
-

5. Computational results

In this section we present the results of the algorithm on some instances of the knapsack problem with setup. The algorithm has been implemented in C++, compiled with the GNU g++ compiler, and run on an Pentium 4 1.2 GHz Linux Workstation with 256 MB of RAM memory.

This section is organized as follows: we first describe the behavior of the algorithm on a small, yet hard, knapsack problem. We provide an in-depth analysis of the variation of the probability vector \mathbf{p} at each iteration. Next, we illustrate the effectiveness of the algorithm on random generated large scale problems, indicating the computational time of the algorithm and the distance of the heuristic solution with respect to the global optimal solution, whenever possible.

Table 1 presents the parameters of an illustrative knapsack problem with setups, as presented in problem IKPS. This problem has seven items, each one with a profit value c_j , a fixed cost f_j , a resource requirement a_j and a setup requirement m_j . The overall resource availability is $b = 119, 567$.

The global optimum for the example problem is $y_1 = 33, y_3 = 1, y_5 = 1, y_2 = y_4 = y_6 = y_7 = 0, x_1 = x_3 = x_5 = 1, x_2 = x_4 = x_6 = x_7 = 0$, with an objective function value of 19, 767. Table 2 presents the results of the proposed

Table 1
Parameters of the example problem

Vector	Item						
	1	2	3	4	5	6	7
c	592	381	273	55	48	37	23
f	59	38	27	5	4	3	2
a	3524	2356	1767	589	528	451	304
m	593	381	271	58	45	32	21

Table 2
Results of the cross entropy scheme on the example problem

Iteration	p_1	p_2	p_3	p_4	p_5	p_6	p_7
0	0.50	0.50	0.50	0.50	0.50	0.50	0.50
1	1.00	0.30	0.35	0.45	0.55	0.30	0.40
2	1.00	0.15	0.55	0.15	0.65	0.15	0.35
3	1.00	0.00	1.00	0.00	0.70	0.10	0.15
4	1.00	0.00	1.00	0.00	0.95	0.00	0.05
5	1.00	0.00	1.00	0.00	1.00	0.00	0.00

algorithm on the example instance. The parameters of the algorithm are $N = 100$ (sample population size) and $\rho = 0.2$ (quantile size).

The algorithm converges after 5 iterations to $\mathbf{p}^* = [1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0]$. This implies that the final solution will have $y_1 \geq 1$, $y_3 \geq 1$, and $y_5 \geq 1$. We now need to solve the AKP with respect to \mathbf{p}^* . The AKS is:

$$\begin{aligned}
 \text{(AKP):} \quad & \max \quad z = c_1 y_1 + c_3 y_3 + c_5 y_5 \\
 & \text{s.t.} \quad a_1 y_1 + a_3 y_3 + a_5 y_5 \leq \bar{b}, \\
 & \quad \mathbf{y} \in \mathbb{Z}_+^3,
 \end{aligned}$$

where $\bar{b} = b - a_1 - a_3 - a_5 - w_1 - w_3 - w_5 = 112, 839$. Problem AKP is based upon the hypothesis that at least one article of each item y_j for which $p_j = 1$ will be in the knapsack.¹ Consequently, the final solution is computed as $y_j^* = y_j^{\text{AKP}} + 1$, where y_j^{AKP} is the optimal solution to AKP. Similarly, the global objective function value is $z^* = z^{\text{AKP}} + \bar{z}$, where z^{AKP} is the optimal objective function value for AKP and $\bar{z} = c_1 - f_1 + c_3 - f_3 + c_5 - f_5$.

Problem AKP can be easily solved using a dynamic algorithm for the general integer knapsack problem in a fashion similar to recursion (4). However, we exploit a turnpike theorem from Nemhauser and Wolsey [17].

Let us assume, as is the case in the illustrative example, that all elements are sorted such that $c_j/a_j \geq c_{j+1}/a_{j+1}$, $j = 1, \dots, n$. Let us define \mathbf{y}^d as the optimal solution to the AKP(d), with $d = \min_{j=1, \dots, n} \{a_j\}, \dots, \bar{b}$, and

$$p(d) = \min\{j \in J^{\mathbf{P}} : y_j^d > 0\},$$

which is, $p(d)$ gives the index of the first item in the optimal solution of AKP(d). Furthermore, let us define $\bar{a} = \max_{j \in J^{\mathbf{P}} \setminus \{1\}} \{a_j\}$.

An important result for the general knapsack problem is:

Proposition 1. *If $p(d - \bar{a}) = p(d - \bar{a} + 1) = \dots = p(d - 1) = 1$, then $z(b) = c_1 + z(b - a_1)$ for all $b \geq d$.*

¹ Note that $lb_1 = lb_3 = lb_5 = 1$.

Table 3
Global optimal solution of the example problem

Solution	Item						
	1	2	3	4	5	6	7
y	33	0	1	0	1	0	0
x	1	0	1	0	1	0	0

Table 4
Results of the Algorithm on random generated instances

No. Integer variables	b	Time Algorithm ^a	Time B&B ^a
50	250	0.1	0.1
100	250	0.1	1
200	250	0.1	2
500	1000	0.2	15
1000	3000	0.9	48
10,000	20,000	12.5	96,432

^aCPU time in seconds on Pentium 4 1.2 GHz.

In the example, we have $\bar{a} = \max\{1767, 528\} = 1767$. We also observed that $p(d) = 1$, for $d \in [3, 524, 5, 291]$, which is, $p(d) = 1$ for at least \bar{a} consecutive solutions. Consequently, we can straightforwardly compute

$$y_1 = \left\lfloor \frac{\bar{b}}{a_1} \right\rfloor = 32.$$

If we iteratively applied the turnpike scheme, we would just need to solve a smaller knapsack problem, derived from AKP and obtained by eliminating y_1 from the problem and discounting the resource used so far. Consequently, we would get:

$$(AKP') : \begin{aligned} \max \quad & z = c_3 y_3 + c_5 y_5, \\ \text{s.t.} \quad & a_3 y_3 + a_5 y_5 \leq \bar{\bar{b}}, \\ & \mathbf{y} \in \mathbb{Z}_+^2, \end{aligned}$$

where $\bar{\bar{b}} = \bar{b} - a_1 y_1 = 71$. Obviously, AKP' is not feasible since $\min\{a_3, a_5\} > \bar{\bar{b}}$.

The optimal solution of AKP is $y_1 = 32, y_3 = 0$, and $y_5 = 0$. Consequently, the final solution to the original problem is the one given in Table 3. The objective function value is $z^* = z^{AKP} + \bar{z} = 18,944 + 823 = 19,767$, which corresponds to the global optimum. The computational time of the proposed algorithm is less than 0.1 CPU seconds.

Table 4, illustrates the performance of the algorithm on large scale instances of problem IKPS. We randomly generated some very large scale instances and we also designed a simple branch and bound code to solve these problems. We then compared the global optimal solution provided by the branch and bound algorithm with the heuristic solution obtained by the proposed algorithm. The branch and bound algorithm uses the Cbc module of the COIN-OR Library [18] and has been implemented in C++ on a Pentium 4 1.2 GHz Linux Workstation with 256 MB of RAM memory. We compiled the code using the GNU g++ compiler.

We tested the algorithm on random generated uncorrelated instances as well as on strongly correlated instances. All the uncorrelated instances were solved to optimality in less than 5 s with problem size of up to 10,000 variables and $b = 20,000$. We then designed a random generation scheme to generate strongly correlated values, as presented in Pisinger [15],² that makes use of the following rules:

- a_j randomly generated in $[5, 50]$.
- $c_j = a_j + 10$.
- f_j randomly generated in $[2c_j, 5c_j]$.

² According to Martello and Toth [19], strongly correlated instances seem to be harder to solve.

Table 5
Comparison of the proposed Algorithm with Pisinger’s Algorithm

k	n	r	Time PSG	Time CQM
5	10	3	NA	< 0.1
5	15	10	0.1	< 0.1
10	10	10	0.1	< 0.1
10	15	10	0.1	< 0.1
100	100	50	0.1	< 0.1
500	500	50	NA	< 0.1
500	1000	100	NA	< 0.1
500	1000	500	1	0.5
1000	1000	200	NA	0.5
1000	1000	500	1	0.5
10,000	10,000	1000	NA	9

- w_j randomly generated in $[2a_j, 5a_j]$.
- $b = \sum_{j=1}^n r_j a_j$, with r_j randomly generated in $[100, 150]$.

Table 4 presents the performances of the algorithm on those instances. The first column presents the number of integer variables in the problem (note that the total number of variables is twice as much, since, for each integer variable, a binary variable must be defined); the second column gives the right-hand side value of the knapsack constraint. Finally, the third and fourth columns compare the computational time of the proposed algorithm with the one of the COIN-OR solver. For each class of problems we randomly generated five different instances and tested the algorithm on each one of them. For this reason, the computational times are expressed as average values.

In all the instances generated the proposed scheme always finds the global optimal solution, which is, the same solution found by the branch and bound scheme. Obviously, a meaningful comparison of the computational times of the two algorithms cannot be performed, since one might argue that a specialized branch and bound scheme should be used. However, the sole intention of the authors was to test the quality of the solution provided by the metaheuristic scheme. As previously mentioned, the proposed scheme finds the global optimal solution 100% of the times, regardless of the instance size.

Finally, let us compare the performances of the proposed scheme with two efficient algorithms for MCKP. The first algorithm analyzed is the one proposed by Pisinger [15]. It is worth noting that the author proposes an algorithm for the multi-class knapsack problem. However, as we illustrated in Section 2, is always possible to transform IKPS into MCKP. We offer a comparison of the computation complexity as well as some empirical results obtained running the Pisinger’s code on random generated instances of IKPS.

Let us consider first that, given an IKPS problem with n items, where each item can be chosen up to ub_j times, the transformation of IKPS into MCKP leads to a problem with $\bar{n} = \sum_{j=1}^n ub_j + n$ binary variables. Let us indicate with n_j the average number of elements in a class, which is, $n_j = \bar{n}/n$ (Note that, in the worst case, $\bar{n} = nb + n = n(b + 1)$.) Pisinger’s [15] algorithm runs in $\mathcal{O}(n^2 n_j \log n_j)$ in the worst case, which is, when all the classes must be added to the core problem in order to reach the optimal solution. The author points out that, for strongly correlated instances, this is the case most of the time, as confirmed by computational experiments presented in [15]. The computational complexity of the proposed scheme, $\mathcal{O}(n' \bar{b} N \log N)$, compares favorably with the complexity of the algorithm in Pisinger [15]. Obviously, the drawback of the proposed algorithm is that, due to its heuristic nature, it does not guarantee optimality of the solution found. It is also worth noting that the random generation scheme used for our experiments produces strongly correlated multiple-class instances. The instance for the MCKP problem is constructed by cumulating strongly correlated knapsack instances, which is, $a_{ij} = \sum_{k=1}^j 2 \times a_{ik} / (k - 1)$. Such instance will have no dominated items, affecting the effectiveness of reduction schemes (see discussion in Pisinger [15]).

In order to further test the effectiveness of the proposed scheme, we converted very large instances of IKPS into MCKP and run the ANSI-C code of Pisinger’s algorithm, obtained by the author, on these instances. Table 5 compares computational time and solution quality of both algorithm, CQM (Caserta, Quiñonez and Márquez) versus PSG (Pisinger).

We could compare small instances of IKPS (up to $n = 1000$) with our algorithm and computational time was comparable (less than 2 s for both algorithms).³ However, we could not solve larger instances of IKPS with the code provided by Pisinger [15] (see details in Table 5). In Table 5, the first column indicates the number of classes in MCKP, the second column indicates the number of items within each class (constant for all classes), the third column is used within the random generation scheme to set the maximum value of w_{ij} in the knapsack constraint (see Pisinger [15]). Finally, columns 4 and 5 compare the computational time of Pisinger's code (PSG) with the proposed metaheuristic scheme (CQM). NA indicates that it was not possible to run the code with those parameters. Note that all the instances are *strongly correlated*, which is $t = 3$ in the code. As mentioned, comparison could be conducted for small problems, while we were not able to realize a comparison when the number of classes and of items within each class was larger. The test was performed on a Linux Workstation Pentium 4 1.2 GHz with 256 MB of RAM memory. The code used to test (PSG) was provided by the author. It was implemented in ANSI C and we compiled with the GNU C compiler. (CQM) was implemented in C++ and compiled with the GNU C++ compiler. In the table, time is measured in CPU seconds.

By observing Table 5, it is possible to conclude that our algorithm compares favorably in terms of computational time and complexity with the algorithm of Pisinger [15]. Obviously, it is worthwhile to highlight that the proposed algorithm is based upon a heuristic approach while Perrot and Vanderbeck [12] and Pisinger [15] propose exact approaches.

Finally, let us briefly compare the computational complexity of the algorithm in Perrot and Vanderbeck [12] with the proposed scheme. As mentioned in Section 2, the overall complexity of the IKPS algorithm in Perrot and Vanderbeck [12], after converting it to a binary MCKP and generating $\bar{n} = \sum_{j=1}^n \lfloor \log_2 ub_j \rfloor + 1$ binary variables (in the worst case, $\bar{n} = nb$), is $\mathcal{O}(nb^2 \log b)$ if the problem has tight bound on variables and $\mathcal{O}(nb \log b)$ when the bound are not tight, which is, the explicit upper bound for item j is greater than ub_j . We can observe that the proposed algorithm has lower complexity when considering explicit upper bounds on variables and a comparable complexity when no explicit upper bounds are present.

6. Conclusion

We have presented a new metaheuristic scheme for the Integer Knapsack Problem with Setups, where one wants to find out which items should be produced on a single machine, respecting resource availability constraints and taking into account setup times and costs. The problem acquires special interest when considered in the context of a more general scheduling problem, the multi-item, multi-period capacitated lot sizing problem. We showed that the Integer Knapsack Problem with Setups can be seen as a special case of the capacitated lot sizing problem, when only a period of the planning horizon is considered.

The backbone of the proposed algorithm is a Cross Entropy based scheme that defines an “intelligent” guessing mechanism aimed at choosing which items should be in the knapsack. Once the set of meritorious items is identified via Cross Entropy, a dynamic recursion is used to find out how many units of each item should be produced. The proposed algorithm has been thoroughly tested and the results testify its robustness as well as effectiveness in solving large instances of the Knapsack Problem with Setups.

Further research is envisioned in this direction. A natural avenue of research related to this work goes in the direction of extending the scheme to the general lot sizing problem, where a set of single-period problems must be linked together through the computation of inventory levels. Another line of study is offered by the development of a new, more effective, dynamic scheme for the knapsack problem with setups that exploits turnpike theorems and sensibly reduces the computational time.

References

- [1] Miller AJ, Nemhauser GL, Savelsberg MW. Solving multi-item capacitated lot-sizing problems with setup times by branch and cut. Core dp 2000/39. Belgium: Université Catholique de Louvain, Louvain-la-Neuve; August 2000.
- [2] Jacobson SH, McLay LA, Kobza JE, Bowman JM. Modeling and analyzing multiple station baggage screening security system performance. Naval Research Logistics 2005;52(1):30–45.
- [3] Dzielinski MAH, Gomory RE. Optimal programming of lot-sizing inventory and labor allocation. Management Science 1965;11:874–90.

³ We run all the tests on the same machine, which is Pentium 4 1.2 GHz with 256 MB of RAM memory on a Linux Workstation.

- [4] Pochet Y, Wolsey LA. Polyhedra for lot sizing with Wagner-within costs. *Mathematical Programming* 1994;67:297–323.
- [5] Eppen GD, Martin RK. Solving multi-item lot-sizing problems using variable redefinition. *Operation Research* 1987;35:832–48.
- [6] Constantino M. A cutting plane approach to capacitated lot-sizing with start-up costs. *Mathematical Programming* 1996;75:353–76.
- [7] Miller AJ, Nemhauser GL, Savelsbergh MWP. On the polyhedral structure of a multi-item production planning model with setup times. *Mathematical Programming Series B* 2003;94:375–405.
- [8] Miller AJ, Nemhauser GL, Savelsbergh MWP. A Multi-item production planning model with setup times: algorithms, reformulations, and polyhedral characterizations for a special case. *Core dp 2001/6*, Belgium: Université Catholique de Louvain, Louvain-la-Neuve; 2001.
- [9] Pochet Y. Valid inequalities and separation for capacitated economic lot sizing. *Operations Research Letters* 1988;7:109–16.
- [10] Miller AJ, Nemhauser GL, Savelsbergh MWP. On the capacitated lot sizing and continuous 0–1 Knapsack polyhedra. *European Journal of Operations Research* 2000;125:298–315.
- [11] Trigeiro WW, Thomas JL, McCain JO. Capacitated lot sizing with setups. *Management Science* 1989;35:141–61.
- [12] Perrot N, Vanderbeck F. Knapsack problems with setups. Technical Report U-04.03, Mathématiques Appliquées Bordeaux (MAB), Université Bordeaux 1, 2004.
- [13] Kellerer H, Pferschy U, Pisinger D. Knapsack problems. Springer operations research. Berlin: Springer; 2004.
- [14] Sural H, Van Wassenhove LN, Potts CN. The bounded Knapsack problem with setups. Technical Report 97-71-TM, INSEAD, 1997.
- [15] Pisinger D. A minimal algorithm for the multiple-choice Knapsack problem. *European Journal of Operational Research* 1995;83:394–410.
- [16] de Boer P, Kroese DP, Mannor S, Rubinstein RY. A tutorial on the cross-entropy method. *Annals of Operations Research* 2005;134(1):19–67.
- [17] Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. Wiley-interscience series in discrete mathematics and optimization. New York: Wiley; 1999.
- [18] Lougee-Heimer R. The common optimization interface for operations research. *IBM Journal of Research and Development* 2003;47(1):57–66.
- [19] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York: Wiley; 1990.