Abstract

This paper considers the minimization version of a class of nonconvex knapsack problems with piecewise linear cost structure. The items to be included in the knapsack have a divisible quantity and a cost function. An item can be included partially in the given quantity range and the cost is a nonconvex piecewise linear function of quantity. Given a demand, the optimization problem is to choose an optimal quantity for each item such that the demand is satisfied and the total cost is minimized. This problem and its close variants are encountered in manufacturing planning, supply chain design, volume discount procurement auctions, and many other contemporary applications. Two separate mixed integer linear programming formulations of this problem are proposed and are compared with existing formulations. Motivated by different scenarios in which the problem is useful, the following algorithms are developed: (1) a fast polynomial time, near-optimal heuristic using convex envelopes; (2) exact pseudo-polynomial time dynamic programming algorithms; (3) a 2-approximation algorithm; and (4) a fully polynomial time approximation scheme. A comprehensive test suite is developed to generate representative problem instances with different characteristics. Extensive computational experiments show that the proposed formulations and algorithms are faster than the existing techniques.

Keywords: Piecewise linear knapsack problem; Precedence constrained knapsack problem; Multiple choice knapsack problem; Linear relaxation; Dynamic programming; Convex envelope; Approximation algorithm; Fully polynomial time approximation scheme

1. Introduction

A generic nonlinear knapsack problem (also called as nonlinear resource allocation problem) has a demand $B$ and a set $J$ of $N$ items. Let the index $j$ denote an item in set $J$. Each item $j \in J$ has a nonlinear cost function $Q_j(b)$ defined over the quantity $b \in [a_j, \bar{a}_j]$. The optimization problem is to choose an integer quantity $q_j \in \{0\} \cup [a_j, \bar{a}_j]$ for each item $j$ to meet the demand $B$ such that the total cost $\sum_{j \in J} Q_j(q_j)$ of the accumulated demand is minimized. The problem can be formulated as the following nonlinear integer programming problem:

\[
\begin{align*}
\text{(NKP)}: \quad \min & \quad \sum_{j \in J} Q_j(q_j) \\
\text{subject to} & \quad \sum_{j \in J} q_j \geq B, \\
& \quad a_j x_j \leq q_j \leq \bar{a}_j x_j, \quad \forall j \in J, \\
& \quad x_j \in \{0, 1\}, \quad q_j \geq 0, \quad \text{integer} \quad \forall j \in J.
\end{align*}
\]

The constraint $q_j \in \{0\} \cup [a_j, \bar{a}_j]$ is implemented as a linear inequality using auxiliary binary decision variable $x_j$. There are several variations to the above problem: The objective function is an arbitrary function $Q(q_1, \ldots, q_J)$; each item has a weight $w_j$ with the demand constraint as $\sum_{j \in J} w_j q_j \geq B$; a generic demand constraint $\sum_{i \in I} g_i(q_i) \geq B$, for suitably defined functions $g_i$ ($i \in J$); and the variables $\{q_i\}$ are continuous. The problem considered in this paper is as in NKP above, with $\{Q_i\}$ as nonconvex piecewise linear functions.
1.1. Relevant work

The NKP and its variants are encountered either directly, or as a subproblem, in a variety of applications, including production planning, financial modeling, stratified sampling, capacity planning in manufacturing, health-care, and computer networks. Different available algorithms for the problem were developed with varying assumptions on the cost function $Q_j$ and the demand constraint. The book by Ibaraki and Katoh provides an excellent collection of algorithms for demand constraint $\sum_{j \in J} q_j = B$. The paper by Brethauer and Shetty reviews the algorithms and applications for NKP with generic demand constraint $\sum_{j \in J} g_j(q_j) \geq B$.

For convex knapsack problems, the $\{Q_j\}$ are convex. If the decision variables are continuous, the differentiability and optimality property of the convex functions can be used in designing algorithms. For problems with separable nonlinear convex constraint of the form $\sum_{j \in J} h_j(q_j) \geq B$, multiplier search methods and variable pegging methods are the general techniques. The algorithms for problems with discrete variables are based on Lagrangian and linear relaxations. For convex demand constraints, the solution methodologies include branch-and-bound, linearization, and dynamic programming.

Unlike convex knapsack problems, very little work is available on nonconvex knapsack problems. Dynamic programming (DP) is the predominantly used solution technique for the nonconvex versions. A pseudo-polynomial time DP algorithm was developed in [17]. Approximation algorithms based on DP were proposed in [25] (the problem was called as capacitated plant allocation problem). Concave cost functions were considered in [29], but the solution technique used local minimizers for obtaining a local optimal solution. Use of branch-and-bound algorithm was proposed as a promising technique in [5].

1.2. Contributions and outline

In this paper, our attention is focused on nonconvex piecewise linear knapsack problems (PLKP), where $Q_j$, $\forall j \in J$, is nonconvex and piecewise linear. The demand constraint is the same as in NKP. First, we introduce the cost function and the associated knapsack problem in Section 2 and briefly describe several application scenarios in which this problem provides a natural model. We study the PLKP problem from several different perspectives, motivated by a number of different scenarios in which it is used. Two new mixed integer linear programming formulations are proposed for the problem in Section 3. These are: (1) precedence constrained knapsack model and (2) multiple choice knapsack model. Based on these formulations, we then set out to design several algorithms for solving this optimization problem.

First, motivated by applications that demand a good quality solution in a relatively short time, we develop, in Section 4, a fast linear programming based heuristic based on convex envelopes. Next we design exact algorithms based on dynamic programming in Section 5. In both the above cases, we present the results of extensive experimentation to show the efficiency and superiority of our algorithms. Finally, in Section 6, we develop a 2-approximation algorithm and a fully polynomial time approximation scheme for the $\mathcal{NP}$-hard PLKP. Section 7 concludes the paper.

2. Nonconvex piecewise linear knapsack problem (PLKP)

The nonconvex and piecewise linear cost function $Q_j$ defined over the quantity range $[\bar{a}_j, \bar{a}_j]$ is shown in Fig. 1. Table 1 provides the notation. The cost function $Q_j$ can be represented by tuples of break points, slopes, and costs at break points: $Q_j \equiv (\{\bar{a}_j = \delta^0_j, \ldots, \bar{a}_j = \delta^r_j\}, \{\beta^1_j, \ldots, \beta^r_j\}, (n^0_j, \ldots, n^r_j))$. For notational convenience, define $\delta^0_j = \delta^r_j = 1$ and $n^0_j$ as the fixed cost associated with segment $s$. Note that, by this definition, $n^0_j = n^r_j$. The function is assumed to be strictly increasing, but not necessarily marginally decreasing as shown in the figure. The assumed cost structure is generic enough to include various special cases: concave, convex, continuous, and $\bar{a}_j = 0$. The PLKP with the generic cost structure was shown to be $\mathcal{NP}$-hard upon reduction from the knapsack problem in [20].

2.1. Motivation for the PLKP problem

The discontinuous piecewise linear cost structure for network flow problems with application in supply chain management was studied in [6]. Our motivation for considering this cost structure for knapsack problems is driven by its applications in winner determination of volume discount procurement auctions. Consider an industrial procurement of $B$ units of some good. The suppliers submit volume discount bids, which are cost functions as shown in Fig. 1. The winner determination problem faced by the buyer is...
to choose a set of winning suppliers and to determine the quantity to be bought from each winning supplier such that the total cost is minimized while satisfying the demand and supply constraints. The resulting problem is exactly the PLKP. The cost structure enables the suppliers to express volume discount strategy, economies of scale, and the production and logistics constraints. Procurement auctions with piecewise linear cost curves are common in industry for long-term strategic sourcing \[8\]. Approximation algorithms based on dynamic programming were developed in \[23\]. PLKP also arise as subproblems in complex procurement scenarios like multi-unit procurement of heterogeneous goods \[10\] and multiattribute procurement \[21\].

Another potential application of the PLKP is the capacitated plant allocation problem. Many industrial organizations operate several plants with different production cost structures. An important operational decision is to determine the production level of each plant in order to minimize the cost of the requested total production. The cost of production (and possibly transportation) is a nonlinear function of quantity produced and can generally be represented by a nonconvex piecewise linear cost function. A similar version of the problem was considered in \[25\], with a generic non-decreasing cost structure, but with \( q_j = 0 \), \( \forall j \in J \).

### 2.2. Related knapsack problems

The tree knapsack problem and multiple choice knapsack problem are similar and closely related to the PLKP. Consider each item \( j \) of PLKP to be a class consisting of knapsack items corresponding to the linear segments of \( Q_j \). The piecewise linear nature of \( Q_j \) enables one to view the PLKP in two different ways. First, consider each linear segment \( s \) in \( Q_j \) as a knapsack item with weight \( \delta^s_j \) and cost \( n^s_j + \beta^s_j \delta^s_j \).

These items have a precedence constraint and therefore \( s \) can be included in the knapsack only if \( s - 1 \) has been already included. This is a precedence constrained knapsack \[32\], more specifically a tree knapsack problem \[19\]. PLKP differs from the above with the \( \{ \delta^s_j \} \) being divisible.

The second interpretation of PLKP is to consider the item \( j \) as a class consisting of \( l_j \) mutually exclusive knapsack items, out of which, at most one item can be selected. In this case, the item \( s \) in class \( j \) has weight \( \delta^s_j \) and cost \( n^s_j \). This is similar to the multiple choice knapsack problem \[24,30\]. PLKP is a generalization of this problem as it can accept partial allocation of the items in a given range. The above two knapsack structures are crucially exploited in this paper to come up with elegant mathematical formulations and efficient new algorithms.

### 2.3. A test suite

A test suite was developed to generate various problem instances. The intention is to study in a comprehensive way the performance of the mathematical programming formulations and algorithms for different problem instances using computational experiments. The following parameters were considered in the generation of the problem instance: continuity, marginally decreasing cost, similarity of the cost functions, \( q_j = 0 \) or >0, and \( n^s_j = 0 \) or >0. Each of the above parameters has two possible values. A continuous function has all the jump costs \( n^s_j = 0 \), whereas a discontinuous function has non-zero values. A marginally decreasing function will have \( \beta^s_j > \beta^{s+1}_j \), whereas an arbitrary cost function need not have any order over \( \beta^s_j \). For similar cost functions, the functions are closely related (CR) with values for parameters \( l_j, \beta^s_j, n^s_j \), and \( \delta^s_j \) chosen randomly in a close range. To make the cost functions unrelated across the items (UR), these parameters are chosen randomly from a wide range as shown in Table 2.

### 3. Mixed integer linear programming formulations for the PLKP

The \( \{ Q_j \} \) functions are nonlinear but due to their piecewise linear nature, the nonlinear PLKP can be modeled as a mixed integer linear programming (MILP) problem.

### Table 1

**Notation for the PLKP**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>([a_j, \bar{a}_j])</td>
<td>Minimum quantity to be included and upper limit</td>
</tr>
<tr>
<td>(Q_j)</td>
<td>Piecewise linear cost function for item ( j ) defined over ([a_j, \bar{a}_j])</td>
</tr>
<tr>
<td>(l_j)</td>
<td>Number of piecewise linear segments in ( Q_j )</td>
</tr>
<tr>
<td>(\beta^s_j)</td>
<td>Slope of ( Q_j ) on ( (\delta^s_j, \delta^{s+1}_j) )</td>
</tr>
<tr>
<td>(\delta^s_j)</td>
<td>( \delta^s_j - \delta^{s-1}_j )</td>
</tr>
<tr>
<td>(n^s_j)</td>
<td>Fixed cost associated with segment ( s ) of item ( j )</td>
</tr>
<tr>
<td>(\bar{Q}_j(\delta^s_j) + n^s_j)</td>
<td>Piecewise linear cost function for item ( j )</td>
</tr>
</tbody>
</table>

### Table 2

**The set of values for parameters of \( Q_j \)**

<table>
<thead>
<tr>
<th>( Q_j )</th>
<th>CR</th>
<th>UR</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n^s_j )</td>
<td>{0.8, 0.9, 1}</td>
<td>{0.2, 0.3, . . . , 1}</td>
</tr>
<tr>
<td>( \delta^s_j )</td>
<td>{30, 40, 50}</td>
<td>{20, 30, . . . , 100}</td>
</tr>
<tr>
<td>( \beta^s_j )</td>
<td>{0.8, 0.85, . . . , 1}</td>
<td>{0.4, 0.45, . . . , 1}</td>
</tr>
</tbody>
</table>
There are three standard textbook models for modeling piecewise linear cost functions, which were formally studied in [6,7]. They are incremental (IM), multiple choice (MCM), and convex combination (CCM) models. In this paper, two more equivalent formulations are proposed, with two different knapsack structures: precedence constrained knapsack model (PCKM) and multiple choice knapsack model (MCKM). The decision variables and the constraints for the above formulations are given in Table 3.

In all the formulations, a binary variable and a continuous variable are associated with each of the linear segments in the cost function. In the IM, the continuous variable \( q'_j \) (\( 1 \leq s \leq l \)) denotes the quantity chosen from segment \( s \). Note that a continuous variable is not used for the indivisible segment 0. Hence if \( q'_j > 0 \) then \( q'_j = \delta_j s \) for \( s' < s \). The binary variables \( d'_j \) (\( 0 \leq s \leq l \)) are used to handle the above logical implications. The quantity chosen for an item incrementally adds up along the linear segments and hence the name. In the MCM, the binary variable \( v'_j = 1 \) indicates that the total quantity chosen lies in the segment \( s \). Hence, if \( q'_j > 0 \), then the total quantity \( q_j = q'_j \). At most one of the binary variables (and hence the continuous variables) can be non-zero and hence the name multiple choice model. The CCM is similar to the MCM, with the quantity in segment \( s \) chosen as a convex combination of the end points of the segment. The above three textbook models are the same with respect to the (1) set of feasible solutions, (2) Lagrangian relaxation (with respect to the demand constraint), and (3) linear programming relaxation [7].

The proposed new formulations PCKM and MCKM are similar to the IM and the MCM, respectively. The continuous variables in the proposed models are normalized to vary between 0 and 1. These two models, however, reveal the hidden knapsack structures discussed in Section 2.2. The proposed formulations are useful in developing novel algorithms, exploiting their knapsack structures. The heuristic algorithm based on linear programming (LP) relaxation (Section 4) and the 2-approximation algorithm (Section 6) are developed using the PCKM formulation while the dynamic programming based algorithms (Section 5) and the fully polynomial time approximation scheme (Section 6) are developed using the MCKM formulation.

### 3.1. Computational experiments

The MILP formulations were modeled and solved using CPLEX 10.0 for different problem instances generated by the test suite. The intention is to evaluate the formulations in terms of solution time using a commercial package. This will help practitioners choose the most appropriate formulation for their requirements. The experiments were carried out on a Windows XP based PC equipped with a 3 GHz Intel P4 processor with 760 MB RAM. The experiments and the test suite were coded in Java. The average solution time for different problem types with \( N = 250 \) is shown in Table 4. The time is averaged over 100 instances of each problem type. The running time varies across the formulations for each problem type and across problem types for each formulation. With respect to the formulations, the proposed MCKM formulation took the least time across all the problem types, except for a few cases, where it took slightly higher computational time than that of CCM. Though MCKM is similar to MCM, the solution time of MCKM was significantly less than that of MCM. However, no such significant difference was noted between IM and PCKM. The running times of the formulations depend significantly on the algorithms used and hence it should be noted that these observations are with respect to CPLEX 10.0.

As expected, the problem type also influenced the solution time. For MCKM, the solution time varied from 75 ms to as long as 7998 ms, depending on the problem type. Henceforth in the analysis we will refer to only the MCKM formulation. In the 5-tuple truth value representation of problem types, let X denote either of the truth values T or F, but fixed for each parameter. For example, XTXTFT refer to problem types with the X for each parameter have been fixed to either T or F independent of the X parameter. Thus XTXXFT can represent 4 problem types. To know the influence of, say, continuity on the solution time, one has to compare problem types XTXXX with XFFFF.

### Table 3

**MILP formulations for PLKP**

<table>
<thead>
<tr>
<th>Model</th>
<th>Variables</th>
<th>Constraints</th>
<th>Quantity ( q_j )</th>
<th>Cost ( Q(q_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>( d'_j \in {0, 1} )</td>
<td>( d'<em>{j+1} \leq \delta'</em>{j+1} \leq q'<em>j \leq \delta'</em>{j+1} )</td>
<td>( q'<em>j + \sum</em>{j=0}^{l} \delta'_{j} )</td>
<td>( \sum_{j=0}^{l} (\delta'_j q'_j + \beta'_j q'_j) )</td>
</tr>
<tr>
<td>MCM</td>
<td>( v'_j \in {0, 1} )</td>
<td>( \sum_{j=0}^{l} \delta'<em>{j+1}/\delta'</em>{j} \leq q'<em>j \leq \delta'</em>{j+1}/\delta'_{j} )</td>
<td>( q'<em>j + \sum</em>{j=0}^{l} \delta'_{j} )</td>
<td>( \sum_{j=0}^{l} (\delta'_j q'_j + \beta'_j q'_j) )</td>
</tr>
<tr>
<td>CCM</td>
<td>( v'_j \in {0, 1} )</td>
<td>( \sum_{j=0}^{l} \delta'<em>{j+1}/\delta'</em>{j} \leq q'<em>j \leq \delta'</em>{j+1}/\delta'_{j} )</td>
<td>( q'<em>j + \sum</em>{j=0}^{l} \delta'_{j} )</td>
<td>( \sum_{j=0}^{l} (\delta'_j q'_j + \beta'_j q'_j) )</td>
</tr>
<tr>
<td>PCKM</td>
<td>( d'_j \in {0, 1} )</td>
<td>( d'<em>{j+1} \leq \delta'</em>{j+1} \leq \delta'_{j} )</td>
<td>( q'<em>j + \sum</em>{j=0}^{l} \delta'_{j} )</td>
<td>( \sum_{j=0}^{l} (\delta'_j q'_j + \beta'_j q'_j) )</td>
</tr>
<tr>
<td>MCKM</td>
<td>( v'_j \in {0, 1} )</td>
<td>( \sum_{j=0}^{l} \delta'<em>{j+1}/\delta'</em>{j} \leq q'<em>j \leq \delta'</em>{j+1}/\delta'_{j} )</td>
<td>( q'<em>j + \sum</em>{j=0}^{l} \delta'_{j} )</td>
<td>( \sum_{j=0}^{l} (\delta'_j q'_j + \beta'_j q'_j) )</td>
</tr>
</tbody>
</table>
4.1. Convex envelopes and LP relaxation

It was shown in [6] that the LP relaxation of the incremental model approximates the cost function $Q_j$ with its convex envelope. Since the LP relaxations of PCKM and the incremental model are equivalent ($q'_j = \delta_i^j x_i'$), the LP relaxation of PCKM approximates the function $Q_j$ with its convex envelope. Thus the LP relaxation is equivalent to solving the PCKM with the convex envelope of $Q_j$ as its cost function.

Definition 1 (Convex Envelope [16]). Let $M \subset \mathbb{R}^n$ be convex and compact and let $f : M \to \mathbb{R}$ be lower semicontinuous on $M$. A function $g : M \to \mathbb{R}$ is called the convex envelope of $f$ on $M$ if it satisfies:

1. $g(x)$ is convex on $M$,
2. $g(x) \leq f(x)$ for all $x \in M$,
3. there is no function $h : M \to \mathbb{R}$ satisfying (1), (2), and $g(\bar{x}) < h(\bar{x})$ for some point $\bar{x} \in M$.

In other words, the convex envelope is the best underestimating convex function that approximates the original function. Let $\hat{Q}_j$ be the convex envelope of the piecewise linear cost function $Q_j$. Let PCKM$_{LP}$ denote the LP relaxation of PCKM and let $\{q'_j\}$ be the optimal quantity chosen in PCKM$_{LP}$. Then the result that relates LP relaxation and convex envelopes [6] states that $Z(\text{PCKM}_{LP}) = \sum \hat{Q}_j(q'_j)$. Thus solving the LP relaxation is equivalent to solving the convex envelope problem. Based on this, PCKM$_{LP}$ will be solved using convex envelopes. For mathematical convenience, it is assumed here that the piecewise linear cost function is defined over $[0, a_j]$ instead of $[a_j, a_j]$. The slope for the indivisible segment $j$ is $\beta_j = \frac{n_j}{2}$ and hence $Q_j(q_j) = \frac{n_j}{2} q_j$ for $q_j \in [0, a_j]$.

The convex envelope $\hat{Q}_j$ is also piecewise linear and hence the algorithm iteratively determines the break points and the slopes from that of $Q_j$. The process is illustrated in Fig. 2. Let the slopes of $\hat{Q}_j$ be denoted by $\{\gamma_j\}$. The first break point is obviously 0, and from (0, 0) the slopes to $Q_j$ at all other break points of $Q_j$ are evaluated. The minimum of the slopes $\gamma_j$ determines the next break point. Proceeding likewise, the slopes from the current break point ($\delta_j$, $Q_j$) to $Q_j$ at the remaining break points are determined.

Algorithm 1. Convex_ENV: Algorithm to find the convex envelope of a piecewise linear cost function $Q_j$.

1. (Initialize) $c = 0$; $b = 0$; $u = 0$; $r = 1$;
2. while ($u \leq l_j$) do:
   2.1. $x'_j = \min_{u \leq i \leq l_j} \hat{y}^i = \frac{n_j + \beta_j - c}{\beta_j - b}$; $\hat{s} = \arg \min_{u \leq i \leq l_j} \hat{y}^i$
   2.2. $S_j = \{u, \ldots, \hat{s} \}$; $\gamma'_j = \sum_{i=0}^{\hat{s}} \delta'_j$
   2.3. $c = \hat{n}_j' + \beta'_j \hat{s}$; $b = \delta'_j$
   2.4. $u = \hat{s} + 1$; $r \leftarrow r + 1$
   od;
3. $t_j = r$;
4.2. LP relaxation based heuristic

The constraints of PCKM imply that \( d_j^r \geq x_j^r \) for all \( j \in J \) and \( 0 < s \leq l_j \). The objective is of minimization type and all coefficients and variables in the objective are non-negative. Hence in the optimal solution, \( d_j^r \) would take the minimal possible value: \( d_j^r = x_j^r \). Thus the binary variables \( d_j^r \) are redundant and can be removed from the formulation. The LP relaxation of PCKM is:

\[
\begin{equation}
\text{PCKM}_{\text{LP}}: \quad \min \sum_{j \in J} \sum_{s=0}^{l_j} (\bar{a}_j^s \delta_j^s x_j^s)
\end{equation}
\]

subject to \( x_j^r - \delta_j^s x_j^s \geq \bar{b}_j \) \( \quad \forall j \in J; \ 0 \leq s < l_j \)

\[
\sum_{j \in J} \sum_{s=0}^{l_j} \delta_j^s x_j^s \leq \bar{b}_j
\]

\( \forall j \in J; \ 1 \leq r \leq t_j \).

Proposition 1. The PCKM with the convex envelope \( \overline{Q}_j \) as the cost function is a continuous knapsack problem.

Proof. The formulation of the problem is:

\[
\begin{equation}
\text{PCKM}_{\text{CE}}: \quad \min \sum_{j \in J} \sum_{r=1}^{t_j} (\bar{a}_j^r x_j^r)
\end{equation}
\]

subject to \( x_j^r \geq \bar{b}_j \) \( \quad \forall j \in J; \ 1 \leq r \leq t_j \)

\[
\sum_{j \in J} \sum_{r=1}^{t_j} \delta_j^r x_j^r \leq \bar{b}_j
\]

\( \forall j \in J; \ 1 \leq r \leq t_j \).

Note that \( x_j^r < \bar{b}_j \) since it is a convex function. Since all \( x_j^r > 0 \) and \( x_j^r \in [0, 1] \), segment \( r \) will be obviously chosen before \( r + 1 \) in the optimal solution. Hence the constraints \( x_j^r < \bar{b}_j \) are redundant. The resulting problem is a continuous knapsack problem with each segment \( r \) of \( j \) as an item with weight \( \bar{a}_j^r \) and unit cost \( x_j^r \).

The optimal values for \( \{ \bar{x}_j^r \} \) of PCKM_{LP} can be obtained from the optimal \( \{ \bar{x}_j^r \} \) of PCKM_{CE} as follows:

\[
\bar{x}_j^r = \bar{x}_j^r \quad \forall s \in S_j, \ 1 \leq r \leq t_j, \ \forall j \in J
\]

In Step 2.1, the segment with maximum index \( s \) is chosen in the event of a tie. The time complexity of the algorithm can be seen to be \( O(t_j^2) \). The following properties of the function \( Q_j \) can be easily verified:

- \( \overline{Q}_j \) is continuous and piecewise linear, and \( Q_j(q_j) \leq \overline{Q}_j(q_j) \) over its domain \([0, \bar{a}_j]\).
- If \( s' = \max\{s \in S_j^r\} \) for some \( 1 \leq r \leq t_j \), then \( \overline{Q}_j(\delta_j^r) = \overline{Q}_j(\delta_j^s) \).
- \( \gamma_j^s = \sum_{r \in S_j^r} \delta_j^s \) for \( 1 \leq r \leq t_j \) and \( \sum_{r=1}^{t_j} |S_j^r| = l_j + 1 \).
- \( \gamma_j^s = \sum_{r \in S_j^r} \delta_j^s \) and \( \gamma_j^s < \gamma_j^{s+1} \) for \( 1 \leq r < t_j \) and hence convex (as it is continuous and piecewise linear).

With reference to Fig. 2, \( l_j = 3, t_j = 2, S_j^1 = \{0, 1\} \), and \( S_j^2 = \{2, 3\} \).

**Theorem 1.** Algorithm 1 constructs the convex envelope \( \overline{Q}_j \) of the piecewise linear function \( Q_j \).

**Proof.** It can be easily verified from the construction of the algorithm that \( Q_j \) is convex over \([0, \bar{a}_j]\) and \( Q_j(q_j) \leq \overline{Q}_j(q_j) \). Thus the conditions 1 and 2 of Definition 1 are satisfied. For proving condition 3, let there exist a function \( \overline{Q}_j \), which satisfies conditions 1 and 2, and \( \overline{Q}_j(w) > \overline{Q}_j(w) \) for some \( w \in [0, \bar{a}_j] \). Let \( r \) be the segment of \( \overline{Q}_j \) which includes \( w \). If \( s = \min\{s \in S_j^r\} \) and \( s' = \max\{s \in S_j^r\} \), then \( w \in [\delta_j^{s-1}, \delta_j^{s'}] \) (refer Fig. 3). By construction of the algorithm, at the end points of the segment \( r \), \( Q_j(\cdot) = \overline{Q}_j(\cdot) \). Since \( \overline{Q}_j \) is an underestimating convex function,

\[
\overline{Q}_j(q_j) \leq \overline{Q}_j(q_j) \quad \text{for} \quad q_j = \delta_j^{s-1}, \delta_j^{s'},
\]

\[
\Rightarrow k \overline{Q}_j(\delta_j^{s-1}) + (1-k) \overline{Q}_j(\delta_j^{s'}) < k \overline{Q}_j(\delta_j^{s-1}) + (1-k) \overline{Q}_j(\delta_j^{s'}) \quad \forall k \in [0, 1],
\]

\[
\Rightarrow \overline{Q}_j(k \delta_j^{s-1} + (1-k) \delta_j^{s'}) < \overline{Q}_j(k \delta_j^{r-1} + (1-k) \delta_j^{r'}) \quad \forall k \in [0, 1],
\]

\[
\Rightarrow \overline{Q}_j(w) \leq \overline{Q}_j(w),
\]

which contradicts the assumption of \( \overline{Q}_j \). Thus condition 3 is also satisfied and \( \overline{Q}_j \) is the convex envelope of \( Q_j \). □
Algorithm 2. LP\_PCKM\_Heu: Heuristic based on LP relaxation of PCKM.

1. Construct the convex envelope $\tilde{Q}_j$ of $Q_j$, $\forall j \in J$.
2. Solve PCKM\_CE using the algorithm for continuous knapsack problem.
3. Determine the optimal solution $\mathbf{x}$ to PCKM\_LP using (8).
4. Construct feasible solution $(\mathbf{d}, \mathbf{x})$ to PCKM from $\mathbf{x}$ using the rounding heuristic.

Let $L = \sum_{j \in J} l_j$. The time complexity of step 1 is $O(L^2)$, that of the remaining steps is $O(L)$, and hence the total time complexity is $O(L^3)$.

4.3. Computational experiments

Computational experiments were performed for the problem types generated using the test suite. The objective was to study the reduction in computational time through use of the proposed convex envelope based algorithm (as against solving the linear relaxation by the simplex algorithm) and the optimality gap of the solution provided by the heuristic. 100 instances of each of the problem types were generated using the test suite for different values of $N$. The MCKM formulation was used to model the PLKP and CPLEX was used solve PLKP and its LP relaxation (using dual simplex algorithm).

The structure of the problem had virtually no effect on the solution time of the LP based heuristics. The average time in milliseconds is shown in Table 5 for the LP based heuristic using CPLEX and the proposed convex envelope based algorithm LP\_PCKM\_Heu. As these values did not change significantly across problem types, the time for problem type TFTFT is only shown in the table. The proposed algorithm is orders of magnitude faster than the simplex algorithm implemented by CPLEX.

The quality of the solution given by the heuristic was measured using the optimality gap, computed against the optimal solution provided by the MCKM formulation. It is obvious that the the optimality gap is influenced by the problem structure, unlike the solution time. The results are shown in Table 6. The optimality gap was greater for problem types with $n_j > 0$ than for problem types with $n_j = 0$ for $q_j = 0$ (by comparing problems XXXFT and XXXFF). With respect to the similarity parameter (XXXX versus FXXXX), UR functions had greater gap

<table>
<thead>
<tr>
<th>No.</th>
<th>Problem Type</th>
<th>Number of Items $N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TTTTT</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>TTTFT</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>TTTFF</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>TTTFT</td>
<td>600</td>
</tr>
<tr>
<td>5</td>
<td>TTTFF</td>
<td>800</td>
</tr>
<tr>
<td>6</td>
<td>TTTFF</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>TTTTT</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>TTTFT</td>
<td>200</td>
</tr>
<tr>
<td>9</td>
<td>TTTFF</td>
<td>400</td>
</tr>
<tr>
<td>10</td>
<td>TTTFT</td>
<td>600</td>
</tr>
<tr>
<td>11</td>
<td>TTTFF</td>
<td>800</td>
</tr>
<tr>
<td>12</td>
<td>TTTFF</td>
<td>1000</td>
</tr>
<tr>
<td>13</td>
<td>TTTTT</td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>TTTFT</td>
<td>200</td>
</tr>
<tr>
<td>15</td>
<td>TTTFF</td>
<td>400</td>
</tr>
<tr>
<td>16</td>
<td>TTTFT</td>
<td>600</td>
</tr>
<tr>
<td>17</td>
<td>TTTFF</td>
<td>800</td>
</tr>
<tr>
<td>18</td>
<td>TTTFF</td>
<td>1000</td>
</tr>
<tr>
<td>19</td>
<td>TTTTT</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>TTTFT</td>
<td>200</td>
</tr>
<tr>
<td>21</td>
<td>TTTFF</td>
<td>400</td>
</tr>
<tr>
<td>22</td>
<td>TTTFT</td>
<td>600</td>
</tr>
<tr>
<td>23</td>
<td>TTTFF</td>
<td>800</td>
</tr>
<tr>
<td>24</td>
<td>TTTTT</td>
<td>100</td>
</tr>
</tbody>
</table>
than the CR functions. However, there were no significant differences in gap with respect to continuity (XTXXX versus XFXXXX) and marginally decreasing (XXXTXX versus XFXXXX).

5. Exact algorithms based on dynamic programming

Dynamic programming (DP) is a traditionally used exact solution technique to the knapsack problem \cite{1} and has been successfully applied to its variants. With the knapsack problems having two parameters, cost and demand, DP formulations based on each of the parameters are quite common. The idea is to make one of these parameters as an independent variable and determine the other recursively.

In \cite{25}, a nonlinear knapsack problem was considered (called as capacitated plant location problem) with non-decreasing cost functions. The problem is more generic compared to the PLKP, as the cost functions are not necessarily piecewise linear. A DP algorithm based on cost was proposed to solve the problem to optimality. Based on the above algorithm in \cite{25}, one can develop two naive DP algorithms for PLKP based on cost and demand, respectively. These are pseudo-polynomial time algorithms with time complexity \(O(N\overline{C}^2)\) for the DP based on cost and \(O(N\overline{B}^2)\) for DP based on demand, where \(\overline{C}\) is an upper bound on the optimal cost and the \(\overline{B}\) is an upper bound on the accumulated demand of the optimal solution. The size of the DP table for the DP based on cost is \(NC\) and that of DP based on demand is \(N\overline{B}\).

The \(\overline{C}\) and \(\overline{B}\) can be obtained in polynomial time. To determine \(\overline{C}\), one can select items in the order of the increasing unit costs \(Q(\overline{a})/\overline{a}_s\), till the demand \(B\) is satisfied. This simple heuristic runs in \(O(N\log N)\) time. For \(\overline{B}\), note that this simple algorithm runs in \(O(\overline{B})\) time.

We now present the DP algorithm for MKP. Without loss of generality, it is assumed that \(B, \{\hat{\delta}_j\}\), and \(\{Q_i\}\) (at the break points) are integers. Let \(V_j(c)\) denote the maximum demand that can be satisfied for a cost \(c\) with MKP classes \(1, \ldots, j\). The \(V_j(c)\) can then be formulated as:

\[
Z(MKP) = \min \{c : V_N(c) \geq B\}. \tag{9}
\]

The \(V_j(c)\) can be recursively defined as:

\[
V_j(c) = \max \left\{\max_{s} \left( V_{j-1}(c - \hat{\delta}_s) + \hat{\delta}_s, V_{j-1}(c) \right) \right\}, \tag{10}
\]

where \(c = 0, \ldots, C\) with \(C\) an upper bound on the optimal cost of MKP. It is easy to verify that the recursion indeed leads to the optimal solution. At stage \(j\), there are two possibilities: either include or not to include one of the MKP items from class \(j\). The boundary conditions for the recursion are \(V_j(0) = 0\) and \(V_j(c) = -\infty\) if one cannot accumulate any demand for cost \(c\) with classes \(0, \ldots, j\).

The heuristic to determine \(\overline{C}\) for MKP is similar to the heuristic used in the case of the PLKP. The total time complexity to evaluate the DP table is \(O(L\overline{C})\) and the size of the DP table is \(NC\). It is worth noting that the above optimal value is an upper bound on that of PLKP. Using the above formulation, we solve the PLKP as follows. Any item \(i\) could have fractional allocation in PLKP. So, we solve the MKP \(N\) additional times, each time without an item \(i\). Let MKP\(-i\) denote the MKP without \(i\). With a slight abuse of notation, we use \(V_j^{-i}(c)\) to denote the accumulated demand for the MKP\(-i\). It clearly follows that for all \(i \in J\):

\[
V_j^{-i}(c) = V_j(c), \quad \forall j < i, \quad \forall c. \tag{11}
\]

Hence MKP\(-i\) requires additional \(N - i\) evaluations of \(V\). The total space to store all the \(V\) values of all the MKPs...
is $N^2\overline{C}$ and the time complexity is $O(NL\overline{C})$. If $i$ has the fractional allocation $b \in [\hat{a}_i, \bar{a}_i]$ in the optimal solution, then the optimal cost of PLKP is $Q_i(b) + c$, where $V_N^i(c) = B - b$. Let $Z'$ denote this optimal cost such that $i$ has fractional allocation.

$$Z' = \min_{b \in [\hat{a}_i, \bar{a}_i]} \{Q_i(b) + \min_{c: V_N^i(c) = B - b}\}.$$  \hspace{1cm} (12)

The minimum is taken over all possible allocations $b$ from $i$ in $[\hat{a}_i, \bar{a}_i]$, with contribution $Q_i(b)$. The demand $B - b$ is exactly satisfied from MKP if possible. $V$ determines the maximum demand for a given cost and hence many different costs can be achieved for the same demand $B - b$. The minimum of such costs is chosen as the contribution from MKP. For the ease of establishing the time complexity, we use the following technique to determine the $Z'$.

$$Z' = \min_{c \leq \overline{C}} \{c + Q_i(b) : b = (B - V_N^i(c)) \in [\hat{a}_i, \bar{a}_i]\}. \hspace{1cm} (13)$$

The above is the same as that of (12), but the minimum is taken over all possible contributions from MKP $V_N^i$ in terms of cost $c$, instead of quantity $b$. Clearly, it takes $O(\overline{C})$ steps to determine $Z'$ using the MKP. The optimal solution to PLKP can be easily determined by investigating all $Z'$ and MKP (solutions with $B' \geq B$).

$$Z(PLKP) = \min_{i \in J} \left\{ \min_{c \leq \overline{C}} Z_i, \min_{c \leq \overline{C}} \{c : V_N(c) \geq B\} \right\}. \hspace{1cm} (14)$$

The first term takes $O(N\overline{C})$ steps and the second term takes $O(L\overline{C})$ steps. Hence, the overall time complexity including the running time for MKP is $O(NL\overline{C})$. On the same lines as the above algorithm that cleverly exploits the multiple choice knapsack structure, a DP algorithm based on demand can be developed with time complexity $O(NL\overline{B})$.

### 5.1. Computational experiments

Computational experiments were performed to compare the solution time of the proposed MKP based DP algorithms with the naive formulations. Both the cost based and demand based DP were considered. Extensive experimentation using the test suite is not required as the structure of the problem instance is irrelevant to the DP algorithms (which depend only on $N$, $L$, $\overline{C}$, and $\overline{B}$). Two types of problems were considered with the following characteristics: $\delta_j \in \{20, 30, 40, 50\}$ (DP1) and $\delta_j \in \{100, 200, \ldots, 500\}$ (DP2). The values of other parameters for both problem types were: $\sigma_j \in \{2, 3, 4, 5\}$, $\beta_j \in \{1, 2, \ldots, 5\}$, and $n_j \in \{20, 40, 50\}$. The two types differ only by the values of $\{\delta_j\}$, which results in varying values for $\overline{B}$ and $\overline{C}$. The experiments were conducted for small values of $N$ and results are shown in Table 7. The computational time is averaged over 100 instances of each problem type. The upper bounds $\overline{B}$ and $\overline{C}$ shown in the table are average values. The approximate average values of $L$ for $N = 5, 10, 15, 20, \text{ and } 25$ were 17, 35, 52, 70, and 87, respectively.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>$N$</th>
<th>$\overline{C}$</th>
<th>$\overline{B}$</th>
<th>$\overline{Demand}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP1</td>
<td>5</td>
<td>603.5</td>
<td>6.91</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1105.5</td>
<td>47.8</td>
<td>6.42</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1606.4</td>
<td>113.95</td>
<td>22.35</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>2168.0</td>
<td>219.35</td>
<td>53.58</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>2723.5</td>
<td>354.96</td>
<td>105.93</td>
</tr>
<tr>
<td>DP2</td>
<td>5</td>
<td>3694.0</td>
<td>323.24</td>
<td>7.08</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>6909.3</td>
<td>2038.31</td>
<td>46.93</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>10407.4</td>
<td>5440.15</td>
<td>159.69</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>13191.5</td>
<td>9583.26</td>
<td>359.89</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>16333.5</td>
<td>15298.69</td>
<td>651.19</td>
</tr>
</tbody>
</table>

There is significant reduction in the computational time of MKP based DP algorithms based on cost. For the DP algorithms based on demand, the reduction in running time is higher for problems with larger $\overline{B}$.

### 6. Approximation algorithms

The PLKP is an $N \cdot \overline{C}$-hard problem and therefore it is of immediate interest to investigate the possibility of approximation algorithms. An approximation algorithm is necessarily polynomial, and is evaluated by the worst or average case possible relative error over all possible instances of the problem. In this section, a 2-approximation algorithm is proposed for PLKP, which will be subsequently used to design a fully polynomial time approximation scheme.

#### 6.1. A 2-approximation algorithm

For a minimization problem, an $\varepsilon$-approximation ($\varepsilon \geq 1$) algorithm is a polynomial time algorithm that yields a solution with a value that is at most $\varepsilon$ times the optimum solution value. A 2-approximation algorithm is proposed here for the PLKP. Approximation algorithms for maximization version of the knapsack problems are well studied [31,18,26,27]. However, these algorithms do not give the same approximation ratio for the minimization version and often, it is not even possible to characterize the ratio (though the exact algorithms for maximization problems can solve the minimization problems). On the contrary, very few papers [12,25,13] consider the minimization problem. The general idea for such an algorithm is outlined in [12]. It is a multi-run greedy algorithm that identifies a critical item at each run and the final solution guarantees a 2-approximation ratio. Following the same approach, we design our algorithm by using the LP relaxation to identify a segment of an item, which we call as the break segment. The break segment can be either a part of or not a part of the optimal solution and if it is in the optimal solution it can either be partially or fully allocated. Considering all these possibilities, we construct a 2-approximation algorithm...
solution. If the 2-approximation is not guaranteed, the segment is removed and the LP relaxation is again applied. The algorithm stops when a 2-approximation is guaranteed or when it is impossible to remove any more segments. In the following, we use the decision variables \( d, x, \) and \( z \) to denote the constructed feasible solution of PCKM. The following result is straightforward.

**Proposition 3.** The feasible solution to PCKM, constructed using the LP rounding heuristic, has at most one \( x_i < 1 \). If all \( x_i \) are either 1 or 0, then either the solution is optimal or there is exactly one \( 0 < x_j < 1 \) in the LP solution that was rounded to obtain \( d_j = 1 \).

The above proposition identifies a unique segment of a feasible PCKM solution that was constructed from the LP solution. The variable corresponding to this segment \( x_j \) or \( d_j \), referred to as the break segment. Let \( I \) denote a set of segments \( \{(j, s)\} \) that preserve the precedence constraints and \( 1_{PCKM}(I, A, F, j, s, b) \) denote the LP relaxation procedure on the set \( I \). The other arguments in the procedure are outputs: \( A \) is the set of segments with LP solution \( x_j = 1 \), \( F \) is the set of segments with fractional values \( 0 < x_j < 1 \), \( j \) and \( i \) define the break segment \((j, i)\), and \( b = x_j d_j \) if \( j \neq 0 \) else \( b = d_j \). Let the quantity \( b \) accepted from the break segment be called as the break quantity. Note that this quantity is the accepted quantity in the LP solution. Let \( co(\text{Set}) \) be the cost of segments in set \( \text{Set} \) evaluated using the values of PCKM variables \( x \) and \( wt(\text{Set}) \) denote the demand accumulated from the segments in \( \text{Set} \).

\[
\begin{align*}
\text{co}(\text{Set}) &= \sum_{(j, s) \in \text{Set}} b_j\delta_j x_j, \quad (15) \\
\text{wt}(\text{Set}) &= \sum_{(j, s) \in \text{Set}} \delta_j. \quad (16)
\end{align*}
\]

Following is our 2-approximation algorithm for PLKP using the PCKM formulation.

**Algorithm 3.** 2-APPROX: A 2-approximation algorithm for PLKP.

1. (Initialize) \( I = \{(j, s)\}: 0 \leq s \leq n \}; \text{R} = \emptyset; \ z = \infty; \ I' = \emptyset; \)
2. while \( wt(I) \geq B \) do:
   1.1 \( 1_{PCKM}(I, A, F, j, s, b) \);
   1.2 \( z \leftarrow \min(z, \text{co}(A) + \text{co}(F)) \);
   1.3 if \( F = \emptyset \) then stop fi;
   1.4 if \( \text{co}(F) \leq \text{co}(A) \) then stop fi;
   1.5 \( I \leftarrow I \setminus \{j, s\}; \text{R} \leftarrow \text{R} \cup \{\exists s \geq (j, s)\}; \)
   1.6 if \( \delta \neq 0 \):
      1.6.1. \( \delta \leftarrow b - 1; I' \leftarrow I \cup \{j, s\} \setminus \{\exists s \geq (j, s)\}; \)
      1.6.2. \( c' = \sum_{\exists s \leq (j, s)} \delta \beta_j; \ c'' = \sum_{\exists s \leq (j, s)} \delta \beta_j; \)
      1.6.3. while \( wt(I') \geq (B - b') \) do:
         (a) \( 1_{PCKM}(I, A', F, j', s', b') \);

The above algorithm terminates at Steps 2.3 or 2.4, the theorem is proved. If \( \text{co}(F) > \text{co}(A) \), then there are four possibilities:

1. The break segment \((j, i)\), with quantity greater than or equal to \( b \) is in the optimal solution.
2. The break segment \((j, i)\), with quantity greater than or equal to \( b \) is not in the optimal solution.
3. The break segment \((j, i)\), with quantity greater than \( b \), is in the optimal solution.
4. The break segment \((j, i)\), with quantity less than \( b \), is not in the optimal solution.

The above cover all the possibilities and at least one of them is true. If (1) is true, then \( \text{co}(F) \leq Z(\text{PCKM}) \) and hence...
z \leq (\text{co}(A) + \text{co}(F)) \leq 2 \times Z(\text{PCKM}). \quad (18)

If (2) or (4) is true, then the break segment can be removed from \( I \) as it will not alter the optimal solution. The algorithm can be stopped if (1) is known to be true, but since it is not known which of the possibilities is true, the algorithm is continued further. Thus if (1) is true, the 2-approximation is guaranteed. If not, we remove the break item from \( I \) (Step 2.5). Due to the precedence constraints, the removal of break segment leads to the removal of the subsequent segments in \( j \) from \( I \). Ignoring Step 2.6, \( z \) is a 2-approximation value if either of the possibilities (1), (2), and (4) is true. This is because, at every run of \( 1_p, p \in \text{I}(\text{km}) \), at least one of the segments is removed from \( I \) and the set \( R \) had the removed items. The algorithm terminates at 2.3 or 2.4 guaranteeing a 2-approximation solution, otherwise it terminates when \( w(I) < B \). At this stage, at least one of the segments from \( R \) would be in the optimal solution, as otherwise the problem is infeasible. Since the possibility of the segments in the optimal solution is taken careful of, a 2-approximation is guaranteed. The possibility (3) is true only if the break segment is not the indivisible segment. This is taken care of in Step 2.6. If the break segment is in the optimal solution with quantity less than \( b \), then all segments preceding \( s \) in \( j \) belong to the optimal solution. A new PLKP is created with segments \( I' \) and demand \( b' \). The \( I' \), however, has the break segment with quantity \( b - 1 \), as the optimal quantity is less than \( b \). The Step 2.6.3 is similar to Step 2, except that the problem considered at step 2.6.3 assumes that the break item is in the optimal solution with quantity less than \( b \). Thus the solution obtained in step 2.6.3 guarantees a 2-approximation for \( I' \). The algorithm, hence, yields a 2-approximation solution for all the above four possibilities.

6.2. Computational complexity of the 2-approximation algorithm

The time complexity of Step 2.1 is \( O(L^2) \) and that of Step 2.6.3 is \( O(L^4) \). Step 2 has to repeated \( O(L) \) times and hence the total time complexity is \( O(L^5) \). The algorithm could be implemented in a more efficient way by implementing Step 2.6 independently of Step 2. The improved efficiency is achieved by avoiding repeated invocation of the LP relaxation routine. Note that in every run of Step 2, the segments in set \( A \) are preserved, that is, \( A \) is a non-decreasing set only including new segments in the consecutive runs. Thus, in every run, only the new segments to be added to \( A \) are to be identified. This can be implemented in the following way. First evaluate the convex segments of all the cost functions in \( O(L^2) \) time. Next, create a bin-ary heap of the convex segments in \( O(L) \) time [11]. The PCKM_CE can be solved by choosing the convex segments in increasing slopes till the demand is satisfied. This is equivalent to removing the root (minimum element) of the binary heap, till the demand is satisfied. The break segment and the break quantity can be found from the last convex segment removed. Note that \( F \) contains the linear segments corresponding to this last convex segment and \( A \) contains all the segments removed, except the last segment. According to Step 2.5, segments that follow the break segment should be removed from \( I \). This is equivalent to removing the remaining convex segments of \( j \) from the heap. However, linear segments preceding \( s \) in \( F \) should be added in \( I \). For this, we find the convex envelopes just for these segments and add to the heap. \( I \) is now the segments in \( A \), plus the segments in the binary heap. Now the new demand is \( B - w(A) \). We remove the root elements from the heap till this new demand is satisfied. Thus Step 2 (without 2.6) can be implemented by maintaining a single binary heap and a set \( A \). The time complexity of deletion of the root in the heap is \( O(\log L) \). In the worst case, \( L \) segments need to be removed and hence the complexity is \( O(\log L) \). At every run, the convex envelope needs to be evaluated for the segments \( s < s \), which are in \( F \). In the worst case, it has to be done for all the segments and hence the time complexity is \( O(L^2) \). The new convex segments then need to be inserted into the heap. Each insertion has \( O(\log L) \) complexity and hence the total complexity is \( O(L^2) \). Thus the total time complexity of Step 2, excluding 2.6, is \( O(L^3) \). Step 2.6 is similar to the above, except that the demand and certain segments are changed. Hence, Step 2.6 also takes \( O(L^2) \) time. However, it may be required to do this for every segment and hence the total complexity is \( O(L^3) \). Thus we have the total time complexity of the 2-approximation algorithm as \( O(L^3) \).

6.3. A fully polynomial time approximation scheme

An algorithm is called as a fully polynomial time approximation scheme (FPTAS) if for a given error parameter \( \epsilon \), it provides a solution with value \( z \) such that \( z \leq (1 + \epsilon)z^* \) for a problem instance with optimal objective value \( z^* \), in a running time that is polynomial in the size of the problem \((N, L, P, C) \) and \( \epsilon \). The DP based on cost is the essential building block for the FPTAS [15]. Recall that the running time of DP based on cost is \( O(NL^2C) \) where \( C \) is some upper bound on the objective value. The idea of the FPTAS is to exploit this DP algorithm and the 2-approximation algorithm. The costs \( c_j' = \delta_j' \beta_j' \) are scaled, thus reducing the running time of the algorithm to depend on the new scaled value. However, the optimal solution for the scaled problem need not be optimal to the original. With a judicious selection of the scaling factor, the optimal solution of the scaled problem can be made arbitrarily close to that of the original problem. Following is the FPTAS for the PLKP. Let \( C_2 \) be the objective value of the 2-approximation algorithm and \( \epsilon \) the required approximation ratio.

Algorithm 4. DP-FPTAS: FPTAS for PLKP

1. \( k = \frac{c_j^*}{2} \)
2. if \( k > 1 \) then \( c_j' = \left[ \frac{c_j'}{k} \right] \); otherwise \( c_j' = c_j' \); \( \overline{C} = C_2; \)
3. $C_\varepsilon \leftarrow \text{DP\_Cost}(\hat{c}, \overline{C})$

The algorithm first determines the scaling factor $k$ based on the desired error parameter $\varepsilon$ (if $k \leq 1$, scaling is not required). The $\text{DP\_Cost}(\hat{c}, \overline{C})$ is the DP algorithm developed in Section 5, applied to the scaled problem with cost $\hat{c}$ and upper bound $\overline{C}$. Let $\text{Cost}(S, \mathbf{c}, \overline{C})$ denote the total cost of items in set $S$ evaluated with cost $\mathbf{c}$.

**Proposition 4.** The value $\overline{C}$ is an upper bound to the scaled problem.

**Proof.** If $k \leq 1$ then the proposition is obvious. For $k > 1$, let the proposition be false. Then, $\frac{C_2}{k} < C_\varepsilon$, where $C_\varepsilon$ is the optimal objective value of the scaled problem. Let $A_2$ be the set of items with quantities for the 2-approximate solution for the unscaled problem. Then, $\text{co}(A_2, \mathbf{c}) = C_2$.

$$\text{co}(A_2, \mathbf{c}) \leq \left[ \frac{C_2}{k} \right] < C_\varepsilon. \quad (19)$$

The first inequality follows from the fact that $|x_1| + \ldots + |x_n| \leq |x_1 + \ldots + x_n|$ for any real numbers $x_1, \ldots, x_n$. Note that the items in $A_2$ are feasible to PLKP (irrespective of the cost structure) and therefore $\text{co}(A_2, \mathbf{c}) \geq C_\varepsilon$. This contradicts the above relation and hence $C_\varepsilon \leq \overline{C}$. □

The above proposition is necessary for the $\text{DP\_FPTAS}$ algorithm, since without that, one cannot guarantee an optimal solution to the scaled problem.

**Theorem 3.** Algorithm $\text{DP\_FPTAS}$ is an FPTAS for the PLKP.

**Proof.** Let $\varepsilon$ be the given error parameter and $C_\varepsilon$ the optimal objective value of the scaled problem obtained by $\text{DP\_FPTAS}$. Then according to the definition of FPTAS, one has to prove $C_\varepsilon \leq (1 + \varepsilon)C^*$ and the running time of $\text{DP\_FPTAS}$ is polynomial in $N, L$, and $\frac{1}{\varepsilon}$. First consider the case where $k > 1$. Let $A^*$ be the set of items with optimal quantities to the PLKP and $A$, the set of items with optimal quantities to the scaled problem. Then, $\text{co}(A^*, \mathbf{c}) = C^*$ and $\text{co}(A, \mathbf{c}) = C_\varepsilon$. As $\hat{c}_j = \left\lfloor \frac{c^*_j}{k} \right\rfloor$, $c^*_j \geq k\hat{c}_j \geq c^*_j - k$.

$$\text{co}(A^*, \mathbf{c}) \geq k \times \text{co}(A^*, \mathbf{c}) \geq k \times \text{co}(A, \mathbf{c}) \geq k \times \text{co}(A, \mathbf{c}) - k \geq \text{co}(A, \mathbf{c}) - k. \quad (20)$$

The first inequality is obvious from the definition and the second inequality follows from the inequality $x - 1 \leq |x|$. By the definition of $A^*$, $A$, and $\hat{c}$,

$$\text{co}(A, \mathbf{c}) \geq k \times \text{co}(A^*, \mathbf{c}) \geq k \times \text{co}(A, \mathbf{c}) - k \geq \text{co}(A, \mathbf{c}) - k. \quad (21)$$

The first inequality follows from the fact that items of $A^*$ are optimal to the unscaled problem; the second inequality follows from $c^*_j \geq k\hat{c}_j$; the third inequality is due to the optimality of items of $A$, to the scaled problem; and the fourth inequality is because $k\hat{c}_j \geq c^*_j - k$. Thus,

$$\text{co}(A^*, \mathbf{c}) \geq \text{co}(A, \mathbf{c}) - k \geq \text{co}(A, \mathbf{c}) - Lk \Rightarrow \text{co}(A, \mathbf{c}) \leq C^* + Lk \Rightarrow \text{co}(A, \mathbf{c}) \leq C^* + \frac{\epsilon C^*}{2} \leq C^* + \frac{\epsilon C^*}{2} \Rightarrow C_\varepsilon \leq (1 + \epsilon)C^*.$$

The fourth inequality follows from the 2-approximation bound $C_2$. The time taken by the algorithm is the time taken to determine the 2-approximate solution $C_2$ ($O(L^3)$) plus the time taken by the DP ($O(NL\overline{C})$).

$$NL\overline{C} = NL \left\lfloor \frac{C_2}{k} \right\rfloor \leq NL. \quad C_2 = \frac{2NL^2}{k} \quad (22)$$

The total time complexity is $O(L^3)$ (as $N \leq L$) and hence the algorithm $\text{DP\_FPTAS}$ is an FPTAS for PLKP. □

7. Conclusions

This paper considered an important class of nonlinear knapsack problems with nonconvex and piecewise cost linear functions. The cost structure considered was generic enough to include various special cases like continuous, linear, concave, convex, marginally decreasing, discontinuous fixed costs, etc. The problem is NP-hard and was studied from two perspectives: (1) the practical purpose of solving the problem in real-world applications such as e-commerce with less computational effort and (2) theoretical viewpoint of studying the approximability of the hard problem.

Two mixed integer programming formulations were proposed and compared with the standard textbook formulations in terms of solution time by a commercial optimization package. The computational experiments conducted over a set of 24 representative problem types show that the proposed MCKM formulation had the least computational time for all the problem types.

A fast polynomial time heuristic was proposed to solve the linear relaxation of the problem using convex envelopes. Computational experiments showed that the proposed heuristic was order of magnitude faster than compared to solving using the traditional simplex algorithm. Pseudo-polynomial time exact algorithms based on dynamic programming were developed. These formulations are faster than the existing naive formulations. A 2-approximation algorithm and a fully polynomial time approximation scheme were also developed.

There are some interesting research problems with respect to solution techniques. An intelligent branch and bound technique that exploits the fast convex envelopes based linear relaxation can possibly solve the problem at a much faster rate than other enumerative search techniques. There is also scope for improving the computational and memory requirements of the dynamic programming algorithms. The information from the LP
relaxation can possibly leveraged to fix some of the variables in the dynamic programming to their optimal values. Another research direction is to develop the concept of core problems for PLKP, similar to that of the knapsack problem [30], where only a subset of items are considered having a high probability of being in the optimal solution.

References