A High Performance Branch and Bound Algorithm for Solving Piecewise Linear Knapsack Problems

S. Kameshwaran, Y. Narahari
Electronic Enterprises Laboratory, Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India {kameshn@csa.iisc.ernet.in, hari@csa.iisc.ernet.in}

A class of *nonlinear knapsack problems* with *piecewise linear* cost structure is considered in this paper. Given multiple units of a set of items and a demand, the problem is to choose an optimal quantity for each item such that the demand is satisfied and the total cost of the chosen items is minimized. The cost of an item is a piecewise linear cost function defined over the quantity. This class of problems arises in numerous applications including electronic commerce (for example, selection of winning bids in electronic procurement auctions). In this paper, the piecewise linear knapsack problem is modeled as a mixed integer programming problem and a polynomial time algorithm is developed to solve its Lagrangian relaxation. Using the solution of the relaxed problem as a lower bound, a branch and bound algorithm is developed with appropriate search strategies. Computational experiments on a wide variety of randomly generated problems show that the proposed algorithm outperforms existing techniques over certain ranges of input parameters. The algorithm developed offers an efficient alternative and addresses an important current need for solving this class of problems.

Key words: programming; integer; branch and bound

1. Introduction

Knapsack problems are among the the most intensively studied combinatorial optimization problems (Pisinger and Toth, 1998; Martello and Toth, 1990; Kellerer et al., 2004). The knapsack problem considers a set of items, each having an associated cost and a weight. The problem (minimization version) is then to choose a subset of the given items such that the corresponding total cost is minimized while the total weight satisfies the specified demand. Well studied variants of the knapsack problems (Pisinger and Toth, 1998; Martello and Toth, 1990) include: *bounded/unbounded knapsack problems*, *subset-sum problems*, *change-making problems*, *multiple knapsack problems*, *multiple choice knapsack problems*, and *bin packing* *problems.* The applications of these problems vary from industrial applications and financial management to personal health-care.

1.1. Nonlinear Knapsack Problems

In *nonlinear knapsack problems* (NKPs) (also called *nonlinear resource allocation problems*), there are multiple units of each item and the cost of inclusion of an item varies nonlinearly with the number of units. A generic nonlinear knapsack problem can be formulated as the following nonlinear integer programming problem:

$$\text{(NKP)}: \qquad \min \sum_{j \in J} Q_j(q_j) \tag{1}$$

subject to

$$\sum_{j \in J} q_j \geq \underline{b}$$
$$\underline{a}_j x_j \leq q_j \leq \overline{a}_j x_j \qquad \forall j \in J$$
$$x_j \in \{0,1\},\ q_j \geq 0, \text{ integer} \quad \forall j \in J$$

Let N be the number of items, that is, $N = |J|$. For each item $j \in J$, there are multiple units $\overline{a}_j$ of that item . For each item j, the cost Q_j is a nonlinear function defined over the quantity range $[\underline{a}_j, \overline{a}^j]$. Given a demand $\underline{b}$, the problem is to choose a set of items, each with q_j units such that the sum of cost of all the items is minimized while the sum of units of all items satisfies the demand $\underline{b}$. If item j is selected ($x_j = 1$), then the quantity q_j satisfies $\underline{a}_j \leq q_j \leq \overline{a}_j$, else $q_j = 0$.

There are several variations to the above problem: the objective function is an arbitrary function $Q(q_1, \ldots, q_N)$, the weight associated with an item is w_j with the demand constraint as $\sum_{j \in J} w_j q_j \geq \underline{b}$, the demand constraint is more general like $\sum_{j \in J} g_j(q_j) \geq \underline{b}$, and the variables q_j are continuous. The problem considered in this paper is as in NKP above, with Q_j as a nonconvex piecewise linear cost function.

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, and in capacity planning in manufacturing, health-care, and computer networks (Bretthauer and Shetty, 2002). 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 (1988) provides an excellent collection of algorithms for demand constraint $\sum_{j \in J} q_j =$

$\underline{b}$. The paper by Bretthauer and Shetty (2002) reviews the algorithms and applications for NKP with generic demand constraint $\sum_{j \in J} g_j(q_j) \geq \underline{b}$.

The objective functions considered in most of the literature for convex knapsack problems are *separable* like in NKP. If the decision variables are continuous, the differentiability property of the convex functions, together with its optimality property and Karush-Kuhn-Tucker conditions are used in designing algorithms (Zipkin, 1980; Bitran and Hax, 1981). For problems with more general separable nonlinear convex constraint of the form $\sum_{j \in J} g_j(q_j) \geq \underline{b}$, *multiplier search methods* (Bretthauer and Shetty, 1995) and *variable pegging methods* (Kodialam and Luss, 1998; Bretthauer and Shetty, 2001) are the general techniques. The algorithms for problems with discrete variables are based on *Lagrangian relaxation* and *linear relaxation* of the original problem. For convex demand constraints, the solution methodologies include branch-and-bound (Bretthauer and Shetty, 1995), linearization, and dynamic programming (Mathur et al., 1986; Hochbaum, 1995).

Unlike in the case of convex knapsack problems, very little work has been done on nonconvex knapsack problems. Dynamic programming (DP) is the predominant solution technique for this problem. A pseudo-polynomial time DP algorithm was developed in Ibaraki and Katoh (1988). Approximation algorithms based on dynamic programming were proposed in Labbe et al. (1994) (the problem was called as capacitated plant allocation problem). Concave cost functions were considered in Moré and Vavasis (1991), 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 Bretthauer and Shetty (2002). The idea is to use the convex envelope (Horst and Tuy, 1990) of the cost function to obtain a lower bound, which can be used for pruning the search in the branch and bound algorithm.

1.2. The Piecewise Linear Knapsack Problem (PLKP)

The nonconvex piecewise linear knapsack problem (PLKP) considered in this paper is the same as NKP above, with the cost function Q_j as a piecewise linear function. The piecewise linear cost function Q_j defined over the quantity range $[\underline{a}_j, \overline{a}_j]$ is shown in Figure 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: $Q_j \equiv ((\underline{a}_j = \tilde{\delta}_j^0, \ldots, \overline{a}_j = \tilde{\delta}_j^{l_j}), (\beta_j^1, \ldots, \beta_j^{l_j}), (\tilde{n}_j^0, \ldots, \tilde{n}_j^{l_j}))$. For notational convenience, define $\delta_j^s \equiv \tilde{\delta}_j^s - \tilde{\delta}_j^{s-1}$ and n_j^s as the fixed cost associated with segment s. Note that, by this definition, $n_j^0 = \tilde{n}_j^0$. The function is assumed to be non-decreasing, but it need

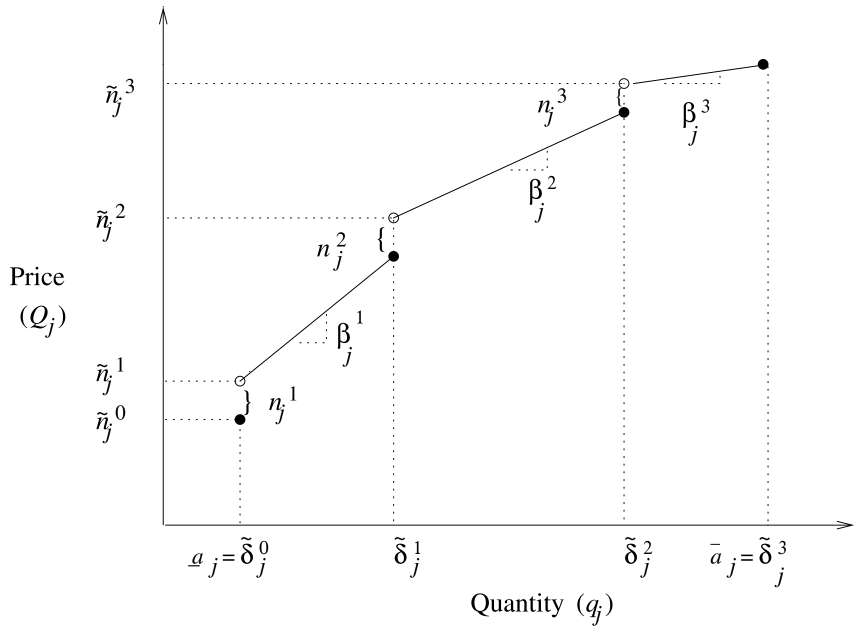

Figure 1: Piecewise Linear Cost Function Q_j

not be marginally decreasing as shown in the figure. The assumed cost structure is generic enough to include various special cases: concave, convex, continuous, and $\underline{a}_j = 0$. The PLKP with the generic cost structure was shown to be $\mathcal{NP}$-hard upon reduction from the knapsack problem in Kameshwaran (2004).

Table 1: Notation for the Nonconvex Piecewise Linear Knapsack Problem

$[\underline{a}_j, \overline{a}_j]$	$\underline{a}_j$ is the minimum quantity to be included and $\overline{a}_j$ is the upper limit
Q_j	Piecewise linear cost function for item j defined over $[\underline{a}_j, \overline{a}_j]$
l_j	Number of piecewise linear segments in Q_j
β_j^s	Slope of Q_j on $(\tilde{\delta}_j^{s-1}, \tilde{\delta}_j^s)$
δ_j^s	$\equiv \tilde{\delta}_j^s - \tilde{\delta}_j^{s-1}$
$\tilde{n}_j^s$	Price at $\tilde{\delta}_j^s$
n_j^s	Fixed cost associated with segment s of item j
Q_j	$\equiv ((\underline{a}_j = \tilde{\delta}_j^0, \ldots, \overline{a}_j = \tilde{\delta}_j^{l_j}), (\beta_j^1, \ldots, \beta_j^{l_j}), (\tilde{n}_j^0, \ldots, \tilde{n}_j^{l_j}))$

1.3. Motivation for PLKP

The discontinuous piecewise linear cost structure is common in telecommunication and electricity pricing, and in manufacturing and logistics. This cost structure for network flow problems with application is supply chain management was studied in Croxton (1999). Our motivation for considering this cost structure for knapsack problems is driven by its applications in electronic commerce, in particular, electronic procurement. For example, consider an organization interested in procuring multiple units of a certain good. Using its private electronic marketplace, the organization would invite bids from its suppliers, quoting the demand and the deadline. The suppliers, who wish to sell the goods, submit sealed bids, in which the price is given as a piecewise linear function of the quantity. The bid evaluation problem faced by the buyer is to choose a set of winning sellers and to determine the quantity to be bought from each winning seller such that the total cost is minimized while satisfying the demand and supply constraints. This is exactly the PLKP. The cost structure enables the suppliers to express their *volume discount* strategy or *economies of scale* and/or the production and logistics constraints. The volume discount strategy, which is *buy more and pay less* can be expressed with marginally decreasing cost functions. The discontinuities in the cost structure can capture the production and transportation constraints. Procurement auctions with piecewise linear cost curves are common in industry for long-term strategic sourcing (Davenport and Kalagnanam, 2001). A procurement scenario with piecewise linear cost function was considered in Kothari et al. (2003) and approximation algorithms based on dynamic programming were developed. The nonconvex piecewise linear knapsack problems also arise as subproblems in bid evaluation of complex procurement scenarios like multi-unit procurement of heterogeneous goods (Eso et al., 2001) and multiattribute procurement (Kameshwaran and Narahari, 2003).

1.4. Contributions and Outline of the Paper

In this paper, we consider piecewise linear knapsack problems and develop an efficient algorithm based on classical techniques such as branch and bound and Lagrangian relaxation. We are motivated by two strong reasons: (1) lack of efficient algorithms in the literature for solving these problems and (2) the vast application potential these problems have in practical situations.

This present paper supplements and complements a companion paper by the authors, Kameshwaran and Narahari (2004) which also develops algorithms for piecewise linear knapsack problems. The algorithms proposed there include:

- two exact algorithms based on dynamic programming: the first one has a worst case computational complexity of $O(NB)$ where N is the number of items and B is an upper bound on the optimal quantity; the second algorithm has a worst case complexity of $O(NC)$ where C is an upper bound on the optimal cost.
- a linear time heuristic algorithm based on LP relaxation (this algorithm is based on a linear time algorithm we develop for computing the convex envelope of a piecewise linear function).
- a 2-approximation algorithm with worst case complexity $O(N^2 \log^2 N)$.
- a fully polynomial time approximation scheme

In this present paper, our contributions are as follows.

- We consider a mixed integer linear programming (MILP) formulation of PLKP and develop an efficient $O(N^2)$ algorithm for optimally solving the Lagrangian relaxation of the MILP formulation. This solution provides a tight lower bound on the optimal value of the original PLKP problem.
- We set up a branch and bound (B&B) algorithm for solving the original PLKP problem optimally. The algorithm for Lagrangian relaxation above provides an efficient lower bounding technique for the B&B algorithm.
- We optimize the B&B algorithm for high performance through various strategies and carry out numerical experiments to show the efficacy of our proposed algorithm.

In our view, the algorithm developed offers an efficient alternative and addresses an important current need for solving this class of problems.

The paper is organized as follows. In Section 2, a mixed integer linear programming formulation is presented for PLKP. This formulation is taken from Kameshwaran (2004) and forms the target formulation that provides the basis for developing the algorithms later on. A polynomial time algorithm is proposed to solve the Lagrangian relaxation of the PLKP problem in Section 3. This is used as the lower bounding technique in Section 4 to develop a B&B algorithm. We discuss various strategies by which the B&B algorithm

Table 2: Acronyms

LP	Linear Programming
DP	Dynamic Programming
IP	Integer Program
MILP	Mixed Integer Linear Program
LR	Lagrangian Relaxation
NKP	Nonlinear Knapsack Problem
PLKP	Piecewise Linear Knapsack Problem
PCKM	Precedence Constrained Knapsack Model
B&B	Branch and Bound
CP	Candidate Problem
DFS	Depth First Search
BFS	Best First Search
BCF	Best Child First
NP	Non-Deterministic Polynomial Time

can be improved to yield high performance. We also describe computational experiments to examine the efficacy of the proposed algorithm. We conclude the paper in Section 5. The appendix provides the pseudo code and data structures for all the algorithms developed in the paper. The various acronyms used in paper are listed in Table 2, for easy reference.

2. An MILP Formulation for PLKP

2.1. Need for the MILP Formulation

The cost function Q_j of Figure 1 is nonlinear but due to the piecewise linear nature, the nonlinear knapsack problem can be modeled as a mixed integer linear programming (MILP) problem. There are three standard textbook models for modeling piecewise linear cost functions, which were formally studied in Croxton (1999). They are: *incremental*, *multiple choice*, and *convex combination* models. We now present another equivalent formulation proposed in Kameshwaran (2004). This is called the precedence constrained knapsack model (PCKM). We use the PCKM formulation for developing all the algorithms in this paper. In the PCKM formulation, each item j is split into $l_j + 1$ knapsack items, where l_j is the number of linear segments in the cost function Q_j. The formulation is equivalent to the earlier textbook models in terms of (1) the feasible solutions, (2) Lagrangian relaxation (with respect to the demand constraint), and (3) linear programming relaxation (Kameshwaran, 2004). However, due to the different inherent knapsack structure, it is extremely useful for developing different algorithms.

2.2. Precedence Constrained Knapsack Model (PCKM)

$$\text{(PCKM)}: \qquad \min \sum_j \left(n_j^0 d_j^0 + \sum_{s=1}^{l_j} \left(n_j^s d_j^s + \beta_j^s \delta_j^s x_j^s \right) \right) \tag{2}$$

subject to

$$d_j^1 \le d_j^0 \qquad \forall j \in J \tag{3}$$

$$x_j^s \le d_j^s \qquad \forall j \in J;\ 1 \le s \le l_j \tag{4}$$

$$x_j^s \ge d_j^{s+1} \qquad \forall j \in J;\ 1 \le s < l_j \tag{5}$$

$$\sum_{j \in J} \left(\underline{a}_j d_j^0 + \sum_{s=1}^{l_j} \delta_j^s x_j^s \right) \ge \underline{b} \tag{6}$$

$$d_j^s \in \{0,1\} \qquad \forall j \in J;\ 0 \le s \le l_j$$

$$x_j^s \in [0,1] \qquad \forall j \in J;\ 1 \le s \le l_j$$

The above formulation can be interpreted in the following way: each segment s of j (including the indivisible segment $s = 0$) is a knapsack item with weight δ_j^s and cost $\delta_j^s \beta_j^s$. The quantity q_j selected for item j is given by $q_j = \underline{a}_j d_j^0 + \sum_{s=1}^{l_j} \delta_j^s x_j^s$. However, these items have a precedence constraint (across each j) and therefore s can be selected only if $s-1$ has been already selected. Thus this is a generalization of the *precedence constrained knapsack* problem (more specifically *tree knapsack problems*) (Johnson and Niemi, 1983; Samphaiboon and Yamada, 2000). Precedence constrained knapsack problems are knapsack problems with some additional precedence constraints over a subset of items. The above formulation can be considered as a *generalization* due to the following reason: in the precedence constrained knapsack problem, all items have indivisible weight, but in this case the weights δ_j^s are divisible except for the $\underline{a}_j$. However, a partial weight can be accepted only if it is the last segment that is selected in j. Mathematically, $x_j^s > 0$ implies $x_j^{s-1} = 1$ and $x_j^s < 1$ implies $x_j^{s+1} = 0$. The above logical implications are handled by introducing binary decision variables d_j^s and are given by constraints (3) to (5). Also the fixed costs n_j^s are handled appropriately in the objective function. This formulation is similar to the *incremental model* (Manne and Markowitz, 1957), in which the continuous decision variable for a segment denotes directly the quantity chosen from that segment. If $\tilde{x}_j^s$ is the continuous decision variable in the incremental model for segment s of item j, then in terms of the PCKM formulation, $\tilde{x}_j^s = x_j^s \delta_j^s$. Thus, given a solution of the incremental model, one can find an equivalent solution to PCKM with the same objective value and vice versa. However, the PCKM problem reveals the hidden precedence constrained knapsack structure, which is otherwise not obvious with the incremental model.

3. Lagrangian Relaxation of PCKM Formulation

The Lagrangian problem for PCKM by dualizing the demand constraint is:

$$(P_\pi): \qquad \min \sum_j \left(n_j^0 d_j^0 + \sum_{s=1}^{l_j} \left(n_j^s d_j^s + \beta_j^s \delta_j^s x_j^s \right) \right) + \pi \left(\underline{b} - \sum_j \left(\underline{a}_j d_j^0 + \sum_{s=1}^{l_j} \delta_j^s x_j^s \right) \right) \tag{7}$$

subject to

$$d_j^1 \le d_j^0 \quad \forall j \in J \tag{8}$$

$$x_j^s \le d_j^s \quad \forall j \in J;\ 1 \le s \le l_j \tag{9}$$

$$x_j^s \ge d_j^{s+1} \quad \forall j \in J;\ 1 \le s < l_j \tag{10}$$

$$d_j^s \in \{0,1\} \quad \forall j \in J;\ 0 \le s \le l_j$$

$$x_j^s \in [0,1] \quad \forall j \in J;\ 1 \le s \le l_j$$

Let $Z(\cdot)$ denote the optimal objective value of the problem $(\cdot)$. The optimal objective value of the LR is obtained by solving the following Lagrangian dual problem:

$$(\text{LD}:\text{PCKM}): \qquad \max_{\pi \ge 0} Z(P_\pi) \tag{11}$$

LR with respect to the demand constraint has integrality property, that is, $Z(\text{LP}:\text{PCKM}) = Z(\text{LD}:\text{PCKM})$, where $Z(\text{LP}:\text{PCKM})$ is the optimal value of the linear relaxation of PCKM and $Z(\text{LD}:\text{PCKM})$ is the optimal value of the Lagrangian dual of PCKM. Let A be the coefficient matrix of the constraints (8) - (10). A has only elements from the set $\{0, 1, -1\}$. Further the transpose of A, A^T, has only two non-zero entries in each column and they sum to zero. Hence A^T and A are *totally unimodular* (Nemhauser and Wolsey, 1988), that is, the determinant of every square submatrix of A is 1, -1, or 0. In other words, x_j^s and d_j^s will always take integral values irrespective of the integrality restrictions. Thus the dual satisfies the *integrality property* (Geoffrion, 1974) and hence the lower bound generated by the Lagrangian dual is the same as the lower bound generated by the linear programming relaxation of PCKM.

The binary variables d_j^s for $s > 0$ are redundant as $d_j^s = x_j^s$ for the optimal solution. The above problem can be reformulated as:

$$(\mathrm{P}_\pi): \qquad \min \sum_j \sum_{s=0}^{l_j} (\hat{\beta}_j^s - \pi)\delta_j^s \hat{x}_j^s + \pi \underline{b} \tag{12}$$

subject to

$$\hat{x}_j^s \geq \hat{x}_j^{s+1} \quad \forall j \in J;\ 0 \leq s < l_j$$
$$\hat{x}_j^s \in \{0,1\} \quad \forall j \in J;\ 0 \leq s \leq l_j$$

where $\hat{\beta}_j^0 = n_j^0/\underline{a}_j$, $\hat{\beta}_j^s = (n_j^s + \beta_j^s\delta_j^s)/\delta_j^s$ for $s > 0$, and $\delta_j^0 = \underline{a}_j$. By assigning $d_j^0 = \hat{x}_j^0$ and $d_j^s = x_j^s = \hat{x}_j^0$, one can easily verify that the above two formulations are the same and yield the same optimal value.

3.1. Solving the Lagrangian Problem

The Lagrangian problem P_π splits into the following N subproblems, one for each item j:

$$\mathrm{P}_\pi^j: \qquad \min \sum_{s=0}^{l_j} (\hat{\beta}_j^s - \pi)\delta_j^s \hat{x}_j^s \tag{13}$$

subject to

$$\hat{x}_j^s \geq \hat{x}_j^{s+1} \quad 0 \leq s < l_j$$
$$\hat{x}_j^s \in \{0,1\}$$

The optimal value of the original Lagrangian problem is $Z(\mathrm{P}_\pi) = \sum_j Z(\mathrm{P}_\pi^j) + \pi\underline{b}$. For a given π, the subproblem for item j is solved as follows:

1. Let $\hat{r}_j = \arg\min_{0 \leq r \leq l_j} \sum_{s=0}^{r} (\hat{\beta}_j^s - \pi)\delta_j^s$.

2. If $\sum_{s=0}^{\hat{r}_j} (\hat{\beta}_j^s - \pi)\delta_j^s \leq 0$, assign $\hat{x}_j^s = 1$ for $s \leq \hat{r}_j$ and $\hat{x}_j^s = 0$ for $s > \hat{r}_j$.

3. If $\sum_{s=0}^{\hat{r}_j} (\hat{\beta}_j^s - \pi)\delta_j^s > 0$, assign $\hat{x}_j^s = 0$ for all s.

In case there are multiple solutions in Step 1, the maximum subscript is chosen to accept as much quantity as possible (note that this does not increase the cost). Solving the subproblem for item j involves choosing a minimum from $l_j + 1$ values, where each value can be computed in unit time. If $L = \sum_j (l_j + 1)$, the complexity of computing $Z(\mathrm{P}_\pi)$ is $O(L)$, which is $O(N)$.

3.2. An Efficient Algorithm for Solving the Lagrangian Dual

The Lagrangian dual problem that determines the lower bound is

$$(\mathrm{LD}:\mathrm{PCKM}): \qquad \max_{\pi \geq 0} Z(\mathrm{P}_\pi)$$

Subgradient optimization (Nemhauser and Wolsey, 1988; Fisher, 1981) techniques are generally used to solve the Lagrangian dual. But as a non-differentiable optimization technique, the convergence of the algorithm depends on the problem instance. In most cases, stopping criteria are used to terminate the algorithm at a near optimal solution. However, for the above Lagrangian dual, due to its structure, a polynomial time algorithm can be shown to exist. For a given π, let the total allocation for the Lagrangian problem P_π be $\hat{b}(\pi) = \sum_j \sum_{s=0}^{\hat{r}_j} \delta_j^s$. Note that this allocation is feasible to PCKM only if $\hat{b}(\pi) \geq \underline{b}$ and $\hat{b}(\pi)$ is an non-decreasing function of π. The proposed polynomial time algorithm uses the monotonic property of $\hat{b}(\pi)$. It starts with a minimum value of π and it is increased till the optimum value is reached. The technique of finding the next subsequent value for π and the optimality criterion are discussed next.

Consider just a single item j with three linear segments and cost structure as shown in Figure 2. As all β_j^s are positive (hence all $\hat{\beta}_j^s$), no segments will be selected for $\pi \leq 0$. Let at $\pi = \pi^1$, $r^1 \geq 0$ segments be chosen. As no segments are chosen for $\pi < \pi^1$, the $\mathrm{P}_{\pi^1}^j = 0$ with $\hat{x}_j^s = 1$ for $s \leq r^1$.

$$\begin{aligned} \mathrm{P}_{\pi^1}^j &= \textstyle\sum_{s=0}^{r^1} (\hat{\beta}_j^s - \pi^1)\delta_j^s = 0 \\ \Rightarrow \qquad \pi^1 &= \frac{\sum_{s=0}^{r^1} \hat{\beta}_j^s \delta_j^s}{\sum_{s=0}^{r^1} \delta_j^s} \end{aligned}$$

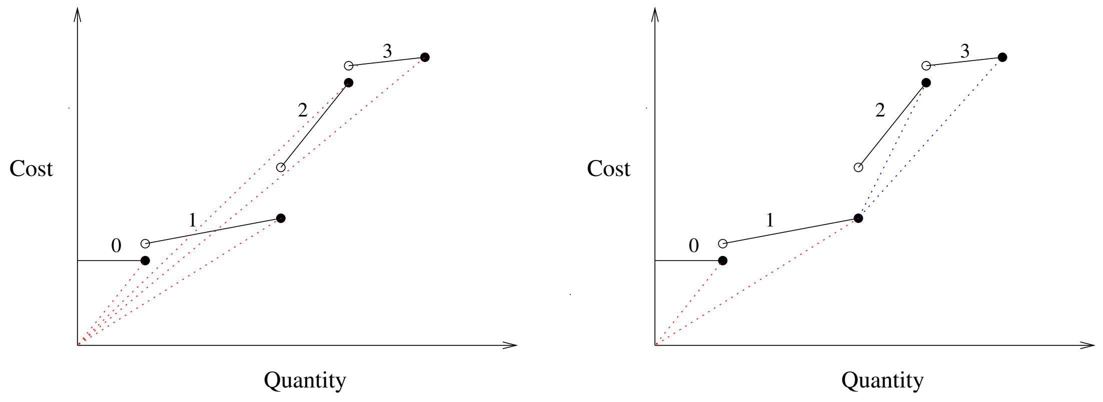

Figure 2: Determining the π

Define $\dot{\beta}_j^t = \frac{\sum_{s=0}^t \hat{\beta}_j^s \delta_j^s}{\sum_{s=0}^t \delta_j^s}$ for $0 \leq t \leq l_j$. Then $\pi^1 = \dot{\beta}_j^{r^1}$, where $r^1 = \arg\min_s\{\dot{\beta}_j^s\}$. The dotted lines in the left half of Figure 2 show the slopes $\dot{\beta}_j^s$. In the figure, $r^1 = 1$. Let at $\pi = \pi^2 > \pi^1$, r^2 segments be added to the solution. Note that for $\pi < \pi^2$ there were only r^1 segments. Hence the cost added by the segments $r^2 - r^1$ should be zero.

$$\sum_{s=r^1+1}^{r^2}(\hat{\beta}_j^s - \pi^2)\delta_j^s = 0$$

$$\Rightarrow \qquad \pi^2 = \frac{\sum_{s=r^1+1}^{r^2} \hat{\beta}_j^s \delta_j^s}{\sum_{s=r^1+1}^{r^2} \delta_j^s}$$

Redefine $\dot{\beta}_j^t = \frac{\sum_{s=r^1+1}^t \hat{\beta}_j^s \delta_j^s}{\sum_{s=r^1+1}^t \delta_j^s}$ for $r^1+1 \leq t \leq l_j$. Then $\pi^2 = \dot{\beta}_j^{r^2}$, where $r^2 = \arg\min_{r^1+1\leq s \leq l_j}\{\dot{\beta}_j^s\}$. These are shown in the right half of Figure 2, with $r^2 = 3$. The above technique can be extended to more than one item by choosing the minimum value of $\dot{\beta}_j^s$ across all items j and segments s. Let π^* be the optimal solution to the Lagrangian dual. The following proposition helps in deriving a polynomial time algorithm.

Proposition 1 *The following results are true:*

1. *The breakpoints of the piecewise linear $Z(\mathrm{P}_\pi)$ occur only at $\pi = \dot{\beta}_j^s$ for some j and s.*

2. *Optimal $\pi^* = \arg\min_{\pi \geq 0}\{\hat{b}(\pi) \geq \underline{b}\}$.*

Proof: (1) It is assumed here that $\dot{\beta}_j^s$ are suitably updated whenever new segments of item j are chosen. Let $\pi^1 = \dot{\beta}_j^s$ for some j and s and let r_j be the number of segments chosen for item j. The $\dot{\beta}_j^s$ are redefined wherever necessary. Pick a π^2 such that $\pi^1 < \pi^2 < \pi^3$ where $\pi^3 = \min_{j,s}\{\dot{\beta}_j^s\}$. For each item j, the r_j of π^1 will remain the same for π^2, as no new segment will be added to the solution. Thus the solution remains the same and $\hat{b}(\pi^2) = \hat{b}(\pi^1)$. However, the objective value changes.

$$\begin{aligned}
Z(\mathrm{P}_{\pi^2}) &= \sum_j \sum_{s=0}^{r_j} (\hat{\beta}_j^s - \pi^2)\delta_j^s + \pi^2 \underline{b} \\
&= \sum_j \sum_{s=0}^{r_j} (\hat{\beta}_j^s - \pi^2 + \pi^1 - \pi^1)\delta_j^s + (\pi^2 + \pi^1 - \pi^1)\underline{b} \\
&= Z(\mathrm{P}_{\pi^1}) + (\pi^1 - \pi^2)\sum_j \sum_{s=0}^{r_j} \delta_j^s + (\pi^2 - \pi^1)\underline{b} \\
&= Z(\mathrm{P}_{\pi^1}) + (\pi^2 - \pi^1)(\underline{b} - \hat{b}(\pi^2))
\end{aligned}$$

Based on the above relation,

- $Z(\mathrm{P}_{\pi^2})$ increases with slope $(\underline{b} - \hat{b}(\pi^2))$ if $\underline{b} > \hat{b}(\pi^2)$
- $Z(\mathrm{P}_{\pi^2})$ decreases with slope $(\underline{b} - \hat{b}(\pi^2))$ if $\underline{b} < \hat{b}(\pi^2)$
- $Z(\mathrm{P}_{\pi^2}) = Z(\pi^1)$ if $\underline{b} = \hat{b}(\pi^2)$

Thus the slope of $Z(P_\pi)$ does not change at a $\pi \neq \dot{\beta}_j^s$ for all j and s.
(3) $Z(\mathrm{P}_\pi)$ is concave and the maximum will occur at the π where $Z(\mathrm{P}_\pi)$ changes direction from increasing to decreasing. From the above discussion it is clear that the change of direction of $Z(\mathrm{P}_\pi)$ occurs when $\hat{b}(\pi)$ crosses $\underline{b}$. Hence $\pi^* = \arg\min_{\pi\geq 0}\{\hat{b}(\pi) \geq \underline{b}\}$. ■

Algorithm 1 `PCKM_LD`: *Algorithm to solve the Lagrangian dual of* PCKM.

```
1. (Initialize)
   $r_j = -1, \forall j;$
   $\dot{\beta}_j^t = \frac{\sum_{s=0}^t \hat{\beta}_j^s \delta_j^s}{\sum_{s=0}^t \delta_j^s}, \forall j, 0 \leq t \leq l_j;$
   $b = 0;$

2. while $(b < \underline{b})$ do:

    2.1. $m_j = \arg\min\{\dot{\beta}_j^s :\ r_j + 1 \leq s \leq l_j\}, \forall j$

    2.2. $\pi = \min_j\{m_j\};$

    2.3. do $\forall j$:

        (a) if $(\pi = m_j)$
            then $k \leftarrow r_j;$
                 $r_j \leftarrow \arg\max\{\dot{\beta}_j^s :\ \dot{\beta}_j^s = \pi, k < s \leq l_j\};$
                 $\dot{\beta}_j^t \leftarrow \frac{\sum_{s=k+1}^t \hat{\beta}_j^s \delta_j^s}{\sum_{s=k+1}^t \delta_j^s}, k < t \leq l_j$
                 $b \leftarrow b + \sum_{s=k+1}^{r_j} \delta_j^s;$
            fi;

        od;

   od;

3. $\pi^* = \pi;\ \hat{b}(\pi^*) = b;$
```

The updating of $\dot{\beta}_j^s$ need not be done till l_j in Step 2.3.(a), but can be stopped when the next r_j is found (that is when $\dot{\beta}_j^{(s+1)} > \dot{\beta}_j^s$). This essentially combines Steps 2.1 and 2.3.(a). Thus this can be done in $O(L)$. Implementing Step 2.2 takes $O(N)$. Hence the overall computational complexity is $O(N^2)$.

3.3. Primal Feasible Solution and Optimality Gap

A feasible solution to PCKM can be easily found from the optimal solution of the Lagrangian dual. Assign $d_j^0 = \hat{x}_j^0$ and $d_j^s = x_j^s = \hat{x}_j^s$. The allocation $\hat{b}(\pi^*) \geq \underline{b}$ and hence the above solution is feasible. Actually one can still make the solution better if $\tilde{b}(\pi^*) > \underline{b}$ by removing the excess allocated units.

$$\begin{aligned} Z(\text{LD} : \text{PCKM}) &\leq Z(\text{PCKM}) \\ \sum_j \sum_{s=0}^{\hat{r}_j} \hat{\beta}_j^s \delta_j^s + \pi^*(\underline{b} - \hat{b}(\pi^*)) &\leq Z(\text{PCKM}) \\ \sum_j \sum_{s=0}^{\hat{r}_j} \hat{\beta}_j^s \delta_j^s - Z(\text{PCKM}) &\leq \pi^*(\hat{b}(\pi^*) - \underline{b}) \end{aligned}$$

The worst optimality gap for the primal feasible solution derived from the Lagrangian dual is $\pi^*(\hat{b}(\pi^*) - \underline{b})$. Thus if $\hat{b}(\pi^*) = \underline{b}$, then the solution is optimal for PCKM.

3.4. Performance of the Heuristic

The proposed heuristic was tested on three different problem sets to study the computational time and the optimality gap. The **Type 1** problems had decreasing slopes for the linear segments in the cost function and the function parameters were highly correlated across the items. This resembles bids in procurement auctions where the cost of the item does not vary much across the suppliers. The **Type 2** problems had similar cost functions but the parameters were uncorrelated and the **Type 3** problems had uncorrelated and arbitrary piecewise linear functions. The number of segments was chosen randomly from 3 to 5 for **Type 1** problems and from 3 to 10 for **Type 2** and **Type 3** problems. To determine the optimality gap and the savings in computational time for the heuristic, the problem was solved to optimality using ILOG Concert Technology of CPLEX 9.1 Con (2001). The experiments were carried out on a Linux based PC equipped with a 3GHz Intel Xeon processor with 4GB RAM and the algorithms were coded in Java. Figures 3 and 4 show the average optimality gap

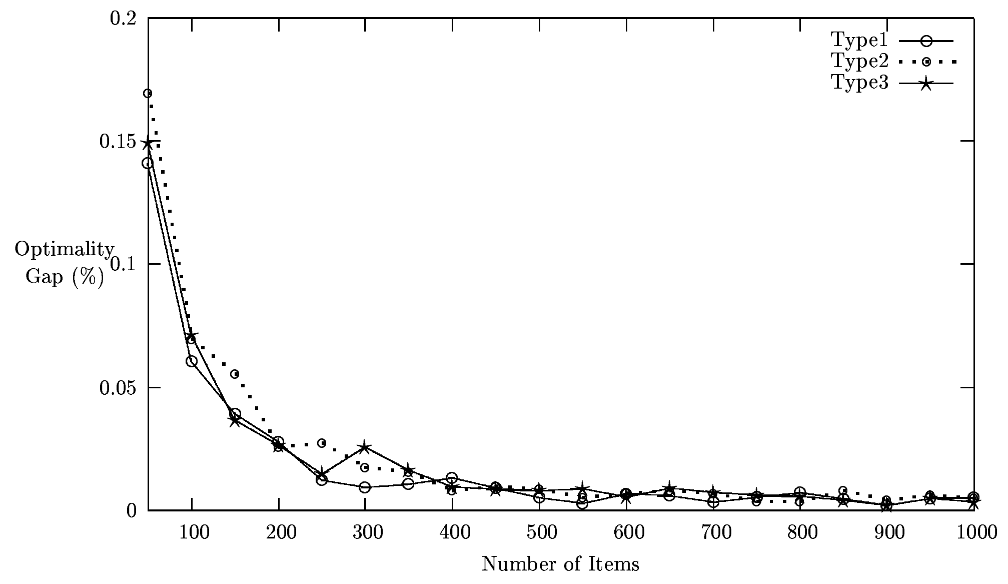

Figure 3: Optimality gap for the LR based heuristic

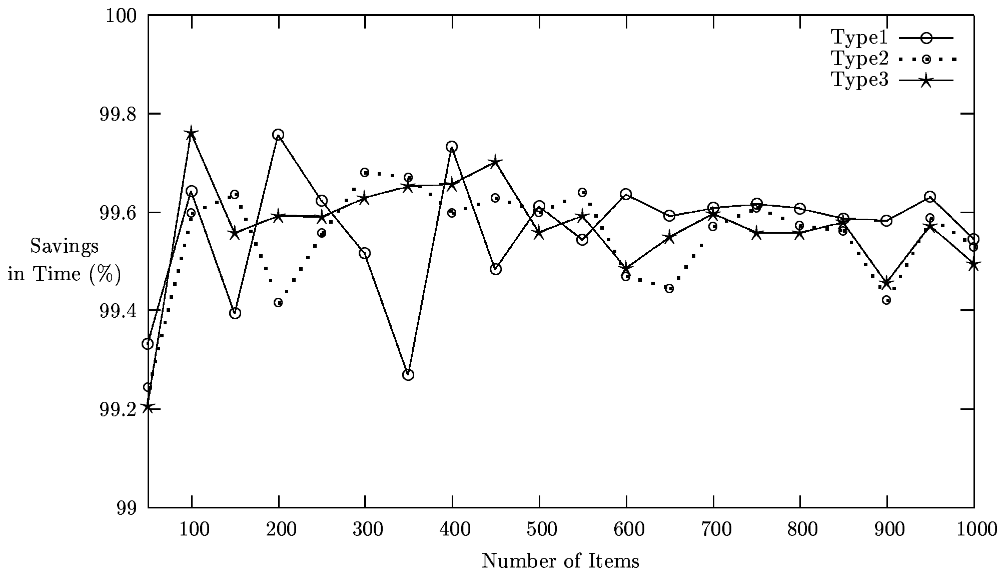

Figure 4: Savings in time for the LR based heuristic

and savings in time, computed over 20 randomly generated problems for each of the three types. There are no major variations in the optimality gap and the savings in time across the problem types. The average savings in time is around 99.5% with an average optimality gap of less than 0.02%. Thus the heuristic extremely efficient and can be effectively used in time constrained applications with negligible loss in optimality.

4. Branch and Bound Algorithm

In this section, an exact algorithm based on branch-and-bound (B&B) technique is developed, which uses the Lagrangian relaxation heuristic to generate lower bounds and incumbent solutions. As is well known, B&B is an exact intelligent enumerative technique that attempts to avoid enumerating a large portion of the feasible integer solutions. It is a widely used approach for solving discrete optimization, combinatorial optimization, and integer programming problems in general. For an excellent introduction, the reader is referred to Chandru and Rao (1999) and for a detailed exposure, to Chapter 15 of Murty (1985). The B&B approach first partitions the overall set of feasible solutions into two or more sets and as the algorithm proceeds the set is partitioned into many simpler and smaller sets, which are explored for the optimal solution. Each such set is represented algebraically by a *candidate problem* (CP). A typical iteration of B&B consists of:

- **Selection/Removal** of a CP from the list of CPs
- Determining the **lower bound** of the selected CP
- **Fathoming** or **pruning**, if possible, the selected CP
- Determining and updating the **incumbent** solution, if possible
- **Branching strategy**: If the CP is not fathomed, branching creates subproblems which are added to the list of CPs

The order of these steps depends on the strategies used in deploying the steps. The algorithm first starts with the original IP/MILP as the only CP in the list, that is, the entire feasible set of solutions is considered at this point. The above steps are repeated until the list of CPs is empty. Although the B&B technique is easy to understand, the implementation for a particular problem is a nontrivial task (Chandru and Rao, 1999) requiring:

- A lower bounding strategy with efficient procedures for solving these relaxations,
- Efficient data structures for handling the rather complicated book-keeping of the list of CPs,
- Clever strategies for selecting promising CPs, and
- Branching strategies that could effectively prune the enumeration tree.

B&B has been successfully applied to knapsack problems (Pisinger and Toth, 1998) and precedence constrained knapsack problems (Shaw and Cho, 1996). All these problems are $0-1$ integer problems, unlike the PLKP which is an MILP. The implementation of B&B for *piecewise linear knapsack problem* (PLKP) with various selection strategies is first discussed, followed by a comparison of different search strategies in respect of computational time.

4.1. Strategies for the Branch and Bound Algorithm

Lower Bounding Strategy

The lower bounding technique finds the lower bound objective value of the CP. The lower bounding technique should be selected such that it produces tight lower bounds in quick time. Motivated by the efficient algorithm developed in the previous section, Lagrangian relaxation is used as the lower bounding technique.

Fathoming and Pruning

If the lower bounding technique provides an optimal solution to the CP, then the CP is said to be fathomed, that is, no further probe into the CP is required. The best solution from the feasible set, represented by the CP, is found. A CP can also removed from further analysis using pruning. Suppose that a feasible solution to the original problem is known. If the lower bound of a CP is greater than the objective value of the known feasible solution, then the CP can be removed from further analysis as it cannot guarantee a better solution than what is already known.

Incumbent Solution

An *incumbent solution* is a feasible solution to the original problem. In the generic B&B, an incumbent solution is found when the lower bounding strategy yields a feasible solution to the original problem. In the case of PLKP, an incumbent solution can be found for each iteration by using the heuristic based on LR. This does not need much extra computation as it also finds a lower bound for the CP. The incumbent solution is found by suitably appending the fixed variables along with the solution generated by the heuristic. The best obtained incumbent solution obtained so far in the algorithm is updated and stored. This is used in pruning the CPs and when the algorithm terminates, the best incumbent solution is the optimal solution to the original problem.

Branching Strategy

If the CP is not fathomed or pruned then it is partitioned into two or more CPs and added to the list of CPs. The partitioning is achieved through branching on a variable, that is, by fixing a variable to some carefully selected feasible value. The branching strategy includes selection of the variable to be branched, the number of branches, and the criterion for branching. Recall that the Algorithm `PCKM_LD` terminates when the total demand accumulated so far exceeds the demand of the problem. The variable of the segment at which the accumulated demand exceeds the required demand is chosen as the branching variable. Two CPs are generated by fixing this variable as 1 and 0, respectively.

Selection Strategy

In each iterative step of B&B, a CP is selected and removed from the list of CPs for further analysis. The computational time of B&B depends crucially on the order in which the CPs are selected and examined. Different B&B algorithms are developed by deploying different selection strategies:

- **Depth First Search** (DFS): The CP that was added last to the list of CPs is chosen for exploration in this strategy. This is basically a last-in-first-out (LIFO) rule and the list of CPs can either be stored as a *stack* or by recursion. In this paper, it is implemented as a recursion that traverses the binary enumeration tree (since each CP is branched into exactly two CPs). Two strategies are possible here depending on which CP is first explored. In DFS10, the CP with variable fixed as 1 is explored first and then the CP with the corresponding variables fixed as 0. In DFS01 the traversal is just the opposite. These two traversals are equivalent to two DFSs, one in which left subtree is traversed first and the other in which the right subtree is traversed first.

Table 3: Average Number of Lower Bound Computations for 100 Items

	`dfs10`	`dfs01`	`bcf`	`bfs`
Type 1	1608	743	743	371
Type 2	216	113	113	56
Type 3	387	132	132	65

- **Best Child First** (BCF): In this strategy, when a CP is branched into two CPs, the one with the less lower bound is explored first. This requires that the lower bounds be computed for both the CPs before traversing.

- **Best First Search** (BFS): BFS is implemented by creating a priority queue that holds the list of CPs. At every iteration, the root of the priority queue, which is the CP with the least lower bound, is deleted from the queue and explored. If it is not fathomed or pruned, then two new CPs are generated and added to the queue. The priority queue is implemented using a binary heap. In the first two cases, which are recursions, only one instance of a CP is required to be maintained at every iteration. This is because the CP can be appropriately modified and re-modified whenever the algorithm enters and exits the recursion, respectively. However, for BFS, the information of the remaining CPs needs to be maintained explicitly, thus requiring more storage and processing.

The pseudo-code and data structures for the above algorithms are provided in the appendix.

4.2. Computational Experiments

Numerical experiments were conducted on several random instances of problems of the three data types. The time consuming step in the B&B approach is the lower bound computation. The experiments were carried out on a Linux based PC equipped with a 3GHz Intel Xeon processor with 4GB RAM and the algorithms were coded in Java.

Lower Bound Computations

Different search strategies were first compared based on the number of lower bound computations. Table 3 shows the average number of lower bound computations for a problem with 100 items. The average was taken over 100 problem instances. For all the three data types, DFS10 required the highest number of computations. The DFS01 and BCF required the same number of computations which shows that fixing the break segment $d_j^s = 0$ provides a better lower bound that fixing it as $d_j^s = 1$. The BFS, as expected, required the least number of lower bound computations. Among the three data types **Type 1** required more computations than the other two. Recall that **Type 1** data are highly correlated. The B&B approach works essentially by pruning solutions that are not promising to be optimal. With correlated data, the costs of the segments do not vary much and thus the lower bound obtained by fixing any of them will not be much different from fixing a different segment. Hence it requires more lower bound computations to prune the search space. **Type 3** data is not correlated and is not marginally decreasing. The number of computations was significantly less than that of **Type 1**. However, **Type 2** required less computations. This is uncorrelated but had marginally decreasing cost functions.

Average Running Time

The proposed B&B algorithm was then tested for running time, by comparing with the solution time by CPLEX and with that of another approach, namely the dynamic programming (DP) technique proposed in Kameshwaran (2004). The DP technique based on demand is a pseudo polynomial time algorithm, whose solution time depends on the given demand $\underline{b}$ and the number of linear segments L. CPLEX is a highly optimized generic solver, which uses branch-and-cut algorithms. The B&B algorithm with BFS strategy was compared with the above techniques to identify the problem parameters, for which our B&B performs better. The parameters considered were the number of items N, demand $\underline{b}$, and the problem type. The demand $\underline{b}$ is chosen as a fraction of the entire supply. The problem types show variation in the size of the problem in terms of the number of linear segments L (number of binary variables), for the same number of items. In addition, data values of the linear segments across different items vary significantly. Figures 5 to 7 show the average computational time taken over 20 problem instances for the different algorithms. The label *a-f* denotes the computational time for algorithm a for the problem with demand as a fraction f of the total supply. As expected, DP performed well for small size problems (in terms of L) and the computational time was dependent on the demand. The B&B and CPLEX approaches showed less variations with respect to $\underline{b}$. The number of items up to which B&B performed better than that of CPLEX was 50, 200, and 450 for **Type 1**, **Type 2**, and **Type 3** problems, respectively. The superior performance of CPLEX for very large sized problems has

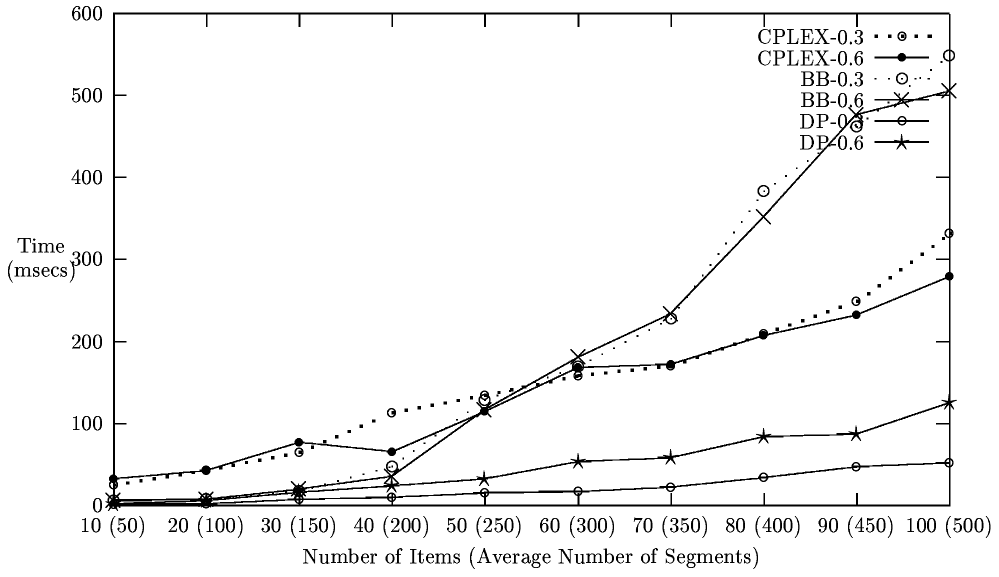

Figure 5: Average Running Time for 20 **Type 1** Problem Instances

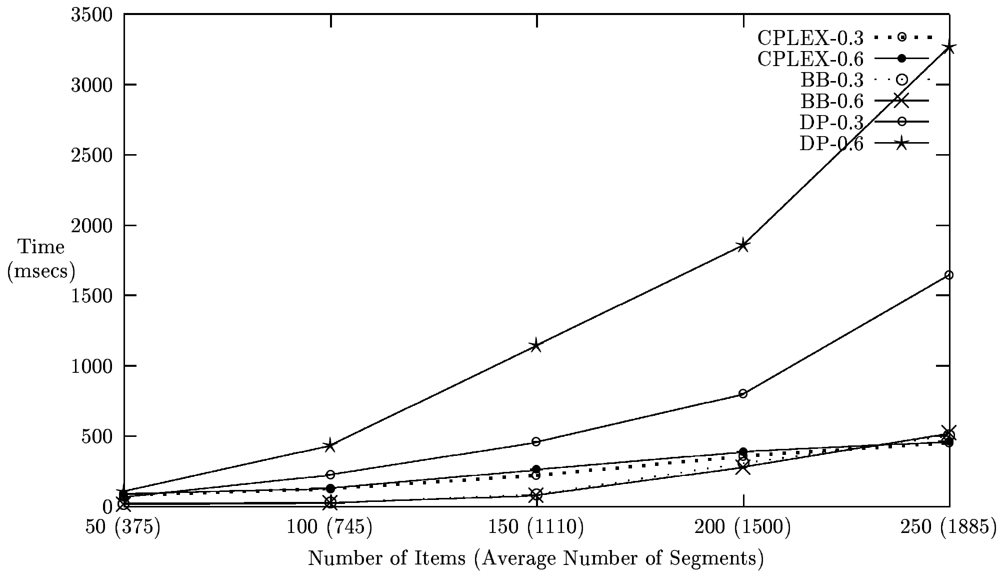

Figure 6: Average Running Time for 20 **Type 2** Problem Instances

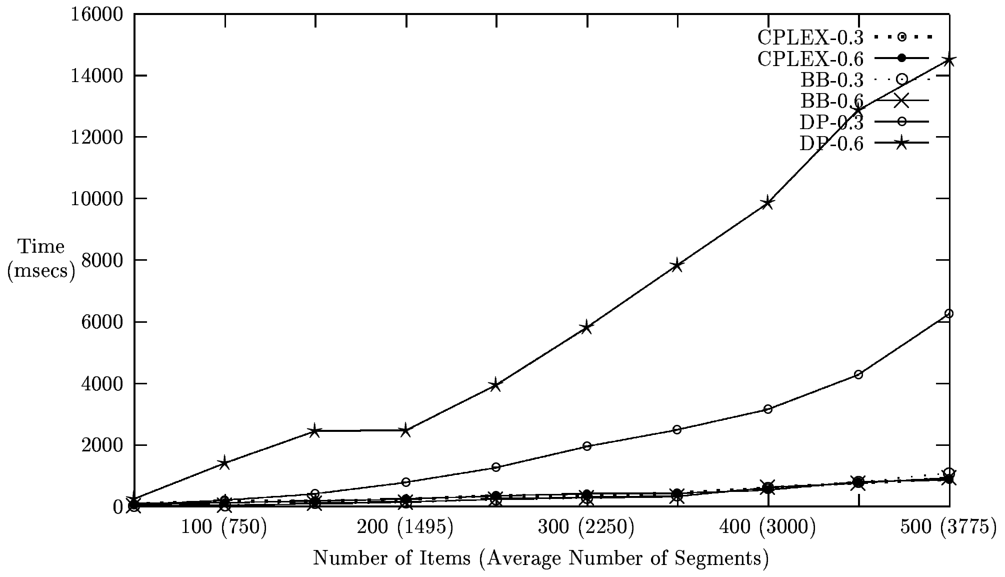

Figure 7: Average Running Time for 20 **Type 3** Problem Instances

to be viewed in the following contexts:

- CPLEX is a highly optimized and very well engineered generic solver, which uses branch-and-cut algorithms.
- Our B&B algorithm considers only the bounds generated by the Lagrangian relaxation technique. The inclusion of *cuts* and *prices*, like in branch-and-cut algorithms, will certainly improve the performance of the algorithm.

5. Conclusions

An important class of *nonlinear knapsack problems* with *piecewise linear* cost structure was considered in this paper. Given multiple units of a set of items and a demand, the problem is to choose an optimal quantity for each item such that the demand is satisfied and the total cost of the chosen items is minimized. The cost of an item is a piecewise linear cost function defined over the quantity. This class of problems arises in numerous applications including electronic commerce (for example, selection of winning bids in electronic procurement auctions).

The main contribution of this paper is an efficient branch-and-bound algorithm for piecewise linear knapsack problems. A $O(N^2)$ algorithm was first proposed to solve the Lagrangian relaxation of the PLKP. A heuristic was then developed to find the primal feasible solution to PLKP. Computational experiments performed on varying data sets showed that a primal feasible solution can be computed with greater than 99% savings in time, with an optimality gap of less than 0.02%. Branch-and-bound (B&B) algorithms were then developed, which used the above heuristic to obtain the lower bounds and the incumbent solutions. Different search strategies were used for the B&B algorithms and were compared with respect to running time and number of lower bound computations. The B&B with *best first search* strategy was compared with the dynamic programming technique and CPLEX solvers, for the computational time. The experiments identified the range of problem parameters for which B&B outperforms dynamic programming and CPLEX. The proposed algorithm considered only the bounds generated by the Lagrangian relaxation technique. The inclusion of *cuts* and *prices*, like in branch-and-cut algorithms to improve the performance of the algorithm is an important future work.

References

2001. *ILOG Concert Technology 1.1 User's Manual*. ILOG CPLEX.

Bitran, G. R., A. C. Hax. 1981. Disaggregation and resource allocation using convex knapsack problems with bounded variables. *Management Science* **27** 431–441.

Bretthauer, K. M., B. Shetty. 1995. The nonlinear resource allocation problem. *Operations Research* **43** 670–683.

Bretthauer, K. M., B. Shetty. 2001. A pegging algorithm for the nonlinear resource allocation problem. *Computers and Operations Research* **29** 505–527.

Bretthauer, K. M., B. Shetty. 2002. The nonlinear knapsack problem - algorithms and applications. *European Journal of Operations Research* **138** 459–472.

Chandru, V., M. R. Rao. 1999. Integer programming. M. J. Atallah, ed., *Handbook of Algorithms and Theory of Computing*. CRC Press, 32.1–32.45.

Cormen, T., C. Leiserson, R. Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge.

Croxton, K. L. 1999. Modeling and solving network flow problems with piecewise linear costs, with applications in supply chain management. PhD thesis, Operations Research Center, MIT, Cambridge, MA.

Davenport, A., J. Kalagnanam. 2001. Price negotiations for procurement of direct inputs. Research Report RC 22078, IBM Research, Yorktown Heights, NJ, USA.

Eso, M., S. Ghosh, J. Kalagnanam, L. Ladanyi. 2001. Bid evaluation in procurement auctions with piece-wise linear supply curves. Research Report RC 22219, IBM Research, Yorktown Heights, NJ, USA.

Fisher, M. L. 1981. The Lagrangian relaxation method for solving integer programming problems. *Management Science* **27** 1–18.

Geoffrion, A. M. 1974. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming Study* **2** 82–114.

Hochbaum, D. S. 1995. A nonlinear knapsack problem. *Operations Research Letters* **17** 103–110.

Horst, R., H. Tuy. 1990. *Global Optimization: Deterministic Approaches*. Springer, Berlin.

Ibaraki, T., N. Katoh. 1988. *Resource Allocation Problems: Algorithmic Approaches*. MIT Press, Cambridge, Massachusetts.

Johnson, D. S., K. A. Niemi. 1983. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research* **8** 1–14.

Kameshwaran, S. 2004. Algorithms for piecewise linear knapsack problems with applications in electronic commerce. PhD thesis, Department of Computer Science and Automation, Indian Institute of Science, Bangalore.

Kameshwaran, S., Y. Narahari. 2003. Trade determination in multiattribute exchanges. *Proceedings of IEEE Conference on E-Commerce CEC' 03*. IEEE Computer Society, 173–180.

Kameshwaran, S., Y. Narahari. 2004. Efficient algorithms for solving nonconvex piecewise linear knapsack problems. Working paper, Electronic Enterprises Laboratory, Department of Computer Science and Automation, Indian Institute of Science.

Kellerer, H., U. Pferschy, D. Pisinger. 2004. *Knapsack Problems*. Springer.

Kodialam, M. S., H. Luss. 1998. Algorithms for separable nonlinear resource allocation problems. *Operational Research* **46** 272–284.

Kothari, A., D. Parkes, S. Suri. 2003. Approximately strategy proof and tractable multi-unit auctions. *Proceedings of ACM Conference on Electronic Commerce (EC-03).*

Labbe, M., E. F. Schmeichel, S. L. Hakimi. 1994. Approximation algorithms for the capacitated plant allocation problem. *Operations Research Letters* **15** 115–126.

Manne, A. S., H. M. Markowitz. 1957. On the solution of discrete programming problems. *Econometrica* **25** 84–110.

Martello, S., P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementation.* John Wiley and Sons.

Mathur, K., H. M. Salkin, B. B. Mohanty. 1986. A note on a general non-linear knapsack problems. *Operations Research Letters* **5** 79–81.

Moré, J. J., S. A. Vavasis. 1991. On the solution of concave knapsack problems. *Mathematical Programming* **49** 397–411.

Murty, K. G. 1985. *Linear and Combinatorial Programming*. R. E. Krieger.

Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York.

Pisinger, D., P. Toth. 1998. Knapsack problems. D.-Z. Du, P. M. Pardalos, eds., *Handbook of Combinatorial Optimization*. Kluwer Academic Publications, 299–428.

Samphaiboon, N., T. Yamada. 2000. Heuristic and exact algorithms for the precedence constrained knapsack problem. *Journal of Optimization Theory and Applications* **105** 659–676.

Shaw, D., G. Cho. 1996. The critical-item, upper bounds, and a branch-and-bound algorithm for the tree knapsack problem. *Networks* **31** 205–216.

Zipkin, P. H. 1980. Simple ranking methods for allocation of one resource. *Management Science* **26** 34–43.

Appendix: Algorithms for B&B

A high level description of the B&B algorithms is presented in this section. The common variables and functions used in the B&B are shown in Tables 4 and 5, respectively. The variables are not just the primitive data types supported in standard programming languages but complex structures that can hold and manipulate information. The variable *cp*, for instance, represents a CP and holds enough information about a CP, like the free variables, variables with fixed values, demand and supply (demand and supply change because of fixing of certain variables), and other information like whether or not it is fathomed. Not all of this information is available at all times, for example, the *cp* can be known to be fathomed only after the lower bounding technique is applied on it. The functions shown in Table 5 can have several inputs and outputs. The implementation details are left unspecified for brevity. The B&B algorithms with different selection/search strategies are presented next.

Algorithm 2 `B&B_DFS01`: B&B *with* DFS01 *search strategy for* PLKP.

```
1. (Initialize)
   $cp \leftarrow$ PCKM; $bs = null$;
   $is = null$; $os = null$; $lb = \infty$;

2. dfs($cp$, $bs$, $is$, $os$, $lb$):

   2.1. if $cp$ infeasible return fi;

   2.2. lowerbound($cp$, $bs$, $is$, $lb$);

   2.3. update_inc_sol($is$, $os$);

   2.4. if $cp$ fathomed or pruned return fi;

   2.5. modify($cp$, $bs$, 0); dfs($cp$, $bs$, $is$, $os$, $lb$); remodify($cp$, $bs$, 0);

   2.6. modify($cp$, $bs$, 1); dfs($cp$, $bs$, $is$, $os$, $lb$); remodify($cp$, $bs$, 1);

   2.7. return;
```

The optimal solution is stored in *os*. The algorithm is a recursion that recursively explores and branches first on CP with branching variable fixed at 0 and then with that of CP fixed at 1. Note that only one instance of *cp* is maintained throughout and it is appropriately modified and re-modified to represent the CP currently in consideration. The algorithm

Table 4: Variables Used in B&B

Variables	**Description**
lb	Lower bound
cp	A complex structure that holds information necessary to define a CP: the free variables, demand, supply, lb, and other state information like infeasibility (supply less than demand) and whether it is fathomed or not.
bs	The branching segment identified by the item j and the segment s.
is, os	The incumbent solution to the PCKM containing the segments and the solution value.
pq	A priority queue of elements, where each element is a cp. Standard operations of priority queue like `deletemin`, `insert`, and `delete` are supported.

Table 5: Common Functions Used in B&B

Function	**Input**	**Output**	**Description**
`lowerbound`	cp	lb, bs, is	Determines the lower bound lb of the given cp and also finds the branching segment bs, which will be used to create child CPs of cp. The incumbent solution is is the optimal solution to cp if it is fathomed, otherwise it is constructed using the heuristics.
`modify`	cp, bs, 1/0	cp	Modifies the given cp by fixing the bs variable as 1/0 and creates the child cp with modified demand, supply, and variables.
`remodify`	cp, bs, 1/0	cp	It reverses the `modify` by restoring the original values and creating the parent cp from its child.
`update_inc_sol`	is, os	os	Updates and stores the best incumbent solution obtained in os

`dfs10` (B&B with DFS10 search strategy) is similar to the above but with Steps 2.5 and 2.6 interchanged so that the child CP fixed at 1 is explored first.

Algorithm 3 `B&B_BCF`: B&B *with* BCF *search strategy for* PLKP

```
1. (Initialize)
   $cp \leftarrow$ PCKM; $bs = null$;
   $is = null$; $os = null$; $lb = \infty$;

2. lowerbound($cp$, $bs$, $is$, $lb$); update_inc_sol($is$, $os$);

3. if $cp$ fathomed stop fi;

4. bcf($cp$, $bs$, $is$, $os$, $lb$):

   4.1. $child0 = child1 = true$;

   4.2. modify($cp$, $bs$, 0);

   4.3. if $cp$ infeasible
        then $child0 = false$;
        else lowerbound($cp$, $bs0$, $is$, $lb0$);
             update_inc_sol($is$, $os$);
        fi;

   4.4. if $cp$ fathomed or pruned then $child0 = false$ fi;

   4.5. remodify($cp$, $bs$, 0);

   4.6. modify($cp$, $bs$, 1);

   4.7. if $cp$ infeasible
        then $child1 = false$;
        else lowerbound($cp$, $bs1$, $is$, $lb1$);
             update_inc_sol($is$, $os$);
        fi;

   4.8. if $cp$ fathomed or pruned then $child1 = false$ fi;

   4.9. remodify($cp$, $bs$, 1);
```

```
4.10. if $lb0 < lb1$
      then if $child0 = true$
            then modify($cp$, $bs0$, 0);
                  bcf($cp$, $bs0$, $is$, $os$, $lb0$);
                  remodify($cp$, $bs0$, 0);
            fi;
            if $child1 = true$
            then modify($cp$, $bs1$, 1);
                  bcf($cp$, $bs1$, $is$, $os$, $lb1$);
                  remodify($cp$, $bs1$, 1);
            fi;
      else if $child1 = true$
            then modify($cp$, $bs1$, 1);
                  bcf($cp$, $bs1$, $is$, $os$, $lb1$);
                  remodify($cp$, $bs1$, 1);
            fi;
            if $child0 = true$
            then modify($cp$, $bs0$, 0);
                  bcf($cp$, $bs0$, $is$, $os$, $lb0$);
                  remodify($cp$, $bs0$, 0);
            fi;
      fi;
4.11. return;
```

The `B&B_BCF` is also a recursive algorithm like `B&B_DFS01`, but more work is done at each iteration to select the best child to explore. For example, the functions `modify` and `remodify` are called twice than that in DFS. Unlike the DFS, the lower bounds for the child CPs are first determined before branching.

Algorithm 4 `B&B_BFS`: B&B *with* BFS *search strategy for* PLKP

```
1. (Initialize)
   $cp \leftarrow$ PCKM; $bs = null$; $pq = null$;
   $is = null$; $os = null$; $lb = \infty$;
```

```
2. lowerbound($cp$, $bs$, $is$, $lb$); update_inc_sol($is$, $os$);

3. **if** $cp$ **fathomed stop fi**;

4. $pq$.insert($cp$, $lb$);

5. **while** $pq \neq \emptyset$ **do**:

    5.1. $(cp, bs) \leftarrow pq$.deletemin();

    5.2. **if** $cp$ **pruned stop fi**;

    5.3. $child0 = child1 = true$;

    5.4. modify($cp$, $bs$, 0);

    5.5. **if** $cp$ **infeasible**
         **then** $child0 = false$;
         **else** lowerbound($cp$, $bs0$, $is$, $lb0$);
              update_inc_sol($is$, $os$);
         **fi**;

    5.6. **if** $cp$ **fathomed** or **pruned then** $child0 = false$ **fi**;

    5.7. **if** $child0 = true$ **then** $pq$.insert($cp$, $bs0$) **fi**;

    5.8. remodify($cp$, $bs$, 0);

    5.9. modify($cp$, $bs$, 1);

   5.10. **if** $cp$ **infeasible**
         **then** $child1 = false$;
         **else** lowerbound($cp$, $bs1$, $is$, $lb1$);
              update_inc_sol($is$, $os$);
         **fi**;

   5.11. **if** $cp$ **fathomed** or **pruned then** $child1 = false$ **fi**;

   5.12. **if** $child1 = true$ **then** $pq$.insert($cp$, $bs1$) **fi**;

   5.13. remodify($cp$, $bs$, 1);

   **od**;
```

The above algorithm, unlike the previous two, is not a recursion. The list of CPs is stored in a priority queue. The `deletemin` operation of the priority queue returns the CP with the least lower bound. If the selected CP is not pruned, two child CPs are generated and added to the priority queue if they are not infeasible/fathomed/pruned. The priority queue is implemented using the binary heap data structure. Every insertion and deletion is of complexity $O(\log P)$, where P is the current size of the queue Cormen et al. (1990). The bookkeeping of the list of the CPs is space efficient in the previous two algorithms. The difference between the parent CP and the child CP is influenced by a single fixed variable and its value (0 or 1). Since the recursion traverses back and forth only across the parent CP and child CP, it is enough to remember the fixed variable and its value. The functions `modify` and `remodify` appropriately construct the current CP from its parent or child. But in the case of BFS, the algorithm jumps across CPs in no known order. Hence all the CPs in the list have to be in memory, which are stored in the priority queue.