LEHIGH UNIVERSITY

418 FINAL PROJECT

# A review of mixed integer knapsack problems

*Author:*
Chenxin Ma, Xi He

*Supervisor:*
Prof. Ted Ralphs

Fall 2014

## Contents

# 1 Introduction

Consider a positive integer $n$, letting $b \in \mathbb{Q}$, $a \in \mathbb{Q}^n$, $l \in \{\mathbb{Q} \cup \{-\infty\}\}^n$, $u \in \{\mathbb{Q} \cup \{+\infty\}\}^n$ and $I \subset [n] := \{1, ..., n\}$. The Mixed Integer Knapsack Set is defined as

$$K = \{x \in \mathbb{R}^n : a^T x \leq b, l \leq x \leq u, x_i \in \mathbb{Z}, \forall i \in I\}. \tag{1}$$

Furthermore, if we have $c \in \mathbb{Q}^n$ and assume $l_i$ is finite for each $i \in [n]$, then the Mixed Integer Knapsack Problem (MIKP) can be described as:

$$\max\{c^T x : x \in K\}. \tag{2}$$

We assume that for each $k = 1, 2, \ldots, n$ either $a_k \neq 0$ or $c_k \neq 0$, otherwise, we could remove variable $x_k$ without affecting the problem.

In this report, we present a new branch-and-bound algorithm for MIKP. The methodology that we propose is a linear-programming-based algorithm which exploits dominance conditions. We further make use of lexicographic-domination conditions to eliminate problems with symmetry. One interesting aspect of this approach is that it differs from traditional linear-programming based algorithms by allowing feasible solutions to be pruned during the branching phase.

It might be very difficult to solve MIKP, even by using very effective mixed integer programming solvers such as CPLEX [1]. However, the proposed algorithm is shown to be very effective in solving instances of MIKP, much more effectice than CPLEX in fact, both in the amount of time taken to solve problems as by the size of the branch and bound tree explored to find the optimal solution.

In the following content, we will state procedures aiming to solve MIKP

1. An easy way of identifying unbounded solutions.
2. A way of pre-processing instances of MIKP.
3. The issue of quickly solving the LP-relaxation of MIKP.
4. A simple branch-and-bound algorithm for MIKP.
5. A enhancement of the branch-and-bound algorithm by introducing domination-criteria.

And finally, we will analyze the computational results of our algorithm and compare it with the general mixed-integer-programing solver CPLEX.

# 2 Algorithms for solving MIKP

## 2.1 Infeasible, unbounded, and trivial instances of MIKP

We aim to use a simple procedure to verify that our MIKP instances are either infeasible, unbounded or trivial, where 'trivial' stands for that our instances are very easy to solve.

Actually, it is very easy to detect the infeasibility of a problem.

**Lemma 1.** *If there is any variable $x_i$ with $i \in [n]$ such that $a_i > 0$ and $l_i = \infty$, or such that $a_i < 0$ and $u_i = \infty$, then the problem is feasible.*

We define the concept of 'efficiency' which can tell us how valuable it is relative to the amount of capacity it uses up in the knapsack constraints.

| | |
|---|---|
| potentiator | $(a_k \leq 0, c_k > 0, u_k = +\infty)$ or $(a_k \geq 0, c_k < 0, l_k = -\infty)$ |
| accumulator | $(a_k < 0, c_k = 0, u_k = +\infty)$ or $(a_k > 0, c_k = 0, l_k = -\infty)$ |
| incrementor | $(a_k > 0, c_k > 0, u_k = +\infty)$ or $(a_k < 0, c_k < 0, l_k = -\infty)$ |
| decrementor | $(a_k > 0, c_k \geq 0, l_k = -\infty)$ or $(a_k < 0, c_k \leq 0, u_k = -\infty)$ |

Table 1: Status of a MIKP

**Definition 1.** Consider $k \in [n]$ and define

$$e_k = \begin{cases} c_k/a_k & \text{if } a_k \neq 0, \\ +\infty & \text{if } a_k = 0 \text{ and } c_k > 0, \\ -\infty & \text{if } a_k = 0 \text{ and } c_k < 0. \end{cases} \tag{3}$$

We say that $e_k$ is the efficiency of variable $x_k$.

Furthermore, we also define some status that use them to claim the situation of our problem. Actually, we say that

These four definition are very useful, actually, we can obtain the following lemma

**Lemma 2.** *If MIKB is feasible and admits a potentiator, then MIKP is unbounded.*

**Lemma 3.** *If MIKB is feasible, and admits an incrementor $x_i$ and a decrementor $x_j$ such that $e_i > e_j$, then MIKP is unbounded.*

**Proposition 1.** *MIKP is unbounded if and only one of the following conditions hold,*

- *MIKP is feasible and admits a potentiator $x_j$.*
- *MIKP is feasible and admits an incrementor $x_i$ and a decrementor $x_j$ such that $e_i > e_j$.*

Note that even if MIKP is bounded, it may still admit an accumulator. We further define the 'triviality' of MIKP.

**Definition 2.** Consider an instance of MIKP which is feasible and not unbounded. If MIKP has an accumulator, we say that MIKP is trivial.

Actually, a trivial MIKP can be easily solved by considering the coefficients of the problem.

**Proposition 2.** *Assume that MIKP is feasible and not unbounded. In addition, let $j$ correspond to an accumulator of MIKP. For each $k \in [n]$ such that $k \neq j$ define:*

- $U_k = \begin{cases} \lfloor u_k \rfloor & \textit{if } k \in I, \\ u_k & \textit{otherwise.} \end{cases}$
- $L_k = \begin{cases} \lceil l_k \rceil & \textit{if } k \in I, \\ l_k & \textit{otherwise.} \end{cases}$
- $x_k = \begin{cases} U_k & \textit{if } (c_k > 0) \textit{ or } (c_k = 0 \textit{ and } u_k < \infty), \\ L_k & \textit{if } (c_k < 0) \textit{ or } (c_k = 0 \textit{ and } l_k > -\infty), \\ 0 & \textit{if } c_k = 0 \textit{ and } x_k \textit{ is free.} \end{cases}$

*In addition, with respect to $k = j$, we claim that*

$$x_j = \begin{cases} \max\{\lceil -\frac{\sum_{k\neq j} a_k x_k - b}{a_j} \rceil, l_k\} & \text{if } a_j < 0, \\ \min\{\lfloor -\frac{\sum_{k\neq j} a_k x_k - b}{a_j} \rfloor, u_k\} & \text{if } a_j > 0. \end{cases} \tag{4}$$

*Then, we derive that $x$ is well-defined and corresponds to an optimal solution of MIKP.*

Actually, we can build an algorithm to detect infeasibility, unbounded and find trivial solutions.

**Algorithm 1** Detecting infeasibility unbounded and finding trivial solutions

**Input:** $c, a, b, e, l, u$
**Output:** $status$

```
e⁺ ← −∞; e⁻ ← +∞
for i = 1 to n do
    if a_i > 0 and l_i = −∞ or a_i < 0 and u_i = ∞ then
        status ← infeaible
    end if
    if x_i is a potentiator then
        status ← potentiator
        return
    else if x_i is an incrementor and e_i > e⁺ then
        e⁺ ← e_i
    else if x_i is an decrementor and e_i < e⁻ then
        e⁻ ← e_i
    end if
end for
if e⁺ > e⁻ then
    status ← incrementor/decrementor pair
    return
else if e⁻ = 0 then
    status ← accumulator
    A solution x is given.
end if
return
```

## 2.2 Preprocessing an instance of MIKP

In this section we are concerned with reducing an instance of MIKP to another, equivalent instance of MIKP which is easier to solve. A series of procedures for pre-processing an instance of MIKP are now presented. For a thorough introduction to preprocessing see [2].

**Test if MIKP is infeasible, trival or unbounded.** Using Algorithm [1] to test if MIKP is infeasible, trivial or unbounded. If MIKP is feasible, not trivial and not unbounded, which means it has no potentiators and no accumulators. In addition if variable $x_i$ is an incrementor, and $x_j$ a decrementor, then $e_i e_j$.

**Strength bound.** We give a strength bound for our problem. First, we define

- $U_k = \begin{cases} +\infty & \text{if } a_k \leq 0, \\ (b - \sum_{i\neq k, a_i>0} a_i l_i - \sum_{i\neq k, a_i<0} a_i u_i)/a_k & \text{otherwise.} \end{cases}$

- $L_k = \begin{cases} -\infty & \text{if } a_k \geq 0 \\ (b - \sum_{i \neq k, a_i > 0} a_i u_i - \sum_{i \neq k, a_i < 0} a_i l_i)/a_k & \text{otherwise.} \end{cases}$

Then, we redefine the stronger bounds of our problem

$$\begin{cases} u_k = \min\{u_k, U_k\}, l_k = \max\{l_k, L_k\} & \text{if } k \notin I, \\ u_k = \min\{\lfloor u_k \rfloor, \lfloor U_k \rfloor\}, l_k = \max\{\lceil l_k \rceil, \lceil L_k \rceil\} & \text{if } k \notin I. \end{cases} \tag{5}$$

**Fix values of variable.** For a given variable $x_k$, we claim

$$x_k = \begin{cases} u_k & \text{if } a_k \leq 0 \text{ and } c_k \geq 0, \\ l_k & \text{if } a_k \geq 0 \text{ and } c_k \leq 0. \end{cases} \tag{6}$$

After fixing variables as described above, we can substitute out the values in MIKP and obtain a smaller problem with a new right-hand side. In the smaller problem, each variable $x_k$ satisfies either $(a_k > 0, c_k > 0)$ or $a_k < 0, c_k < 0$.

**Complement variables.** For simplicity, we introduce new variable to make sure the lower-bound is always non-negative. Consider a variable $x_k$, and then we set

$$x_k := \begin{cases} x_k - l_k & \text{if } -\infty < l_k < 0, \\ u_k - x_k & \text{if } l_k = -\infty. \end{cases} \tag{7}$$

**Sort data.** Sort the variables in order of decreasing efficiency. Break first ties if variables are of integer type or not. Break second ties by value of $a_k$.

**Aggregate variables.** For any given two variables $x_i$ and $x_j$, $i, j \in I$, if $a_i = a_j$, $c_i = c_j$. We aggregate these two variables into a single variable $x_k$ such that $a_k = a_i, c_k = c_i, l_k = l_i + l_j, u_k = u_i + u_j$ and $k \in I$. This procedure will be very helpful later in spending up the branch and bound algorithm.

After the several steps, our finally propose is to reformulate the original MIKP to the following PP-MIKP

$$\begin{aligned} \max \quad & \sum_{k \in P \cup N} c_k x_k \\ s.t. \quad & \sum_{k \in P \cup N} a_k x_k \leq b \\ & l_k \leq x_k \leq u_k, \forall k \in P \cup N \\ & x_k \in \mathbb{Z}, \forall k \in I. \end{aligned} \tag{8}$$

where $P = \{k : c_k > 0 \text{ and } a_k > 0\}$ and $N = \{k : c_k < 0 \text{ and } a_k < 0\}$. The PP-MIKP satisfies the following conditions:

- PP-MIKP is feasible.
- PP-MIKP is not unbounded, and is not trivial.
- The variable indices are sorted by efficiency.
- All variables $x_k$ are such that ($a_k > 0$ and $c_k > 0$) or ($a_k < 0$ and $c_k < 0$).
- For each $k \in P \cup N$, we have $l_k 0$.
- For each $k \in P \cup N$ , all finite bounds are tight; that is, there exists a feasible solution to MIKP which achieves the bound.
- There are no two identical variables.

### 2.3 Solving the LP relaxation of PP-MIKP

In this section, we discuss how to solve the linear programming relaxation of problem (8). That is, the problem LP-PP-MIKP.

$$\begin{aligned} \max \quad & \sum_{k \in P \cup N} c_k x_k \\ s.t. \quad & \sum_{k \in P \cup N} a_k x_k \leq b \\ & l_k \leq x_k \leq u_k, \forall k \in P \cup N \end{aligned} \tag{9}$$

Note that (9) is nothing more than a linear programming problem. As thus, any Simplex-based linear programming software package would do to solve it. Goycoolea presented a Simplex-liked algorithm, which extends in a simple way Dantzigs algorithm for solving the linear programming relaxation of bounded, positive coefficient knapsack problems.

**Definition 3.** We say that $x^* \in \mathbb{R}^{|P|+|N|}$ is tight for (8) if,

$$\sum_{k \in P \cup N} a_k x_k^* = b. \tag{10}$$

**Definition 4.** $x^* \in \mathbb{R}^{|P|+|N|}$ is $k-efficient$ for (9) if $l_k \leq x_k^* \leq u_k$ and

- $i \in P$ and $i > k$ implies $x_i^* = l_i$.
- $i \in P$ and $i < k$ implies $x_i^* = u_i$.
- $i \in N$ and $i > k$ implies $x_i^* = u_i$.
- $i \in N$ and $i < k$ implies $x_i^* = l_i$.

It can be easy to show that if there exists a tight feasible solution for (9), then there exists a tight optimal solution for (9). If $x^*$ is tight and k-efficient for (8), then $x^*$ is an optimal solution of (9).

The Phase I Algorithm takes as input an instance of (9), and does one of two things: (a) It proves that the instance is infeasible, or (b) it generates $x$, a k-efficient solution of the instance, having non-negative slack. The algorithm begins by defining $k = \max\{j \in P \cup U : j \in N \text{ and } u_j = +\infty\}$, assuming that if the latter set is empty, then $k = -1$. If $k = -1$, it generates the solution $x$, where

$$x_j := \begin{cases} l_j & \text{if } j \in P, \\ u_j & \text{if } j \in N. \end{cases} \tag{11}$$

If $k > 1$ it generates the solution $x$, for $j \in (P \cup N) \setminus \{k\}$,

$$x_j := \begin{cases} u_j & \text{if } j \in P \text{ and } j < k, \\ l_j & \text{if } j \in P \text{ and } j > k, \\ u_j & \text{if } j \in N \text{ and } j > k, \\ l_j & \text{if } j \in N \text{ and } j < k, \end{cases} \tag{12}$$

and

$$x_k = \max\Big\{ -\frac{1}{a_k}\Big(\sum_{j \neq k} a_j x_j - b, l_k\Big)\Big\}.$$

When $k = -1$, then the algorithm may generate an infeasible solution. In this case it is easy to see that the problem itself is infeasible. Also note that when the algorithm generates a feasible solution, this solution will be efficient. Thus, if the solution is tight it will be optimal.

## References

[1] ILOG CPLEX. High-performance software for mathematical programming and optimization, 2005.

[2] Martin WP Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.