한국경영과학회지
제40권 제3호
2015년 8월
http://dx.doi.org/10.7737/JKORMS.2015.40.3.023

Branch-and-Cut 알고리즘에서 Lot-Sizing 문제에 대한 Cutting Planes의 전산 성능 연구*

정 광 헌[†]
홍익대학교 경영대학

Computational Study of Cutting Planes for a Lot-Sizing Problem in Branch-and-Cut Algorithm

Kwanghun Chung
College of Business Administration, Hongik University

■ Abstract ■

In this paper, we evaluate the strength of three families of cutting planes for a lot-sizing problem. Lot-sizing problem is very basic MIP model for production planning and many strong valid inequalities have been developed for a variety of relaxations in the literature. To use three families of cutting planes in Branch-and-Cut framework, we develop separation algorithms for each cut and implement them in CPLEX. Then, we perform computational study to compare the effectiveness of three cuts for randomly generated instances of the lot-sizing problem.

Keywords : Mixed-Integer Programming (MIP), Branch-and-Cut, Lot-Sizing Problem, Separation

1. Introduction

Branch-and-Bound is a general method to solve mixed-integer programming (MIP) problems and widely used in MIP solvers such as CPLEX, GuRoBi, and Xpress. However, since LP relaxations are typically weak for real problems, the number of nodes in Branch-and-Bound tree may grow

논문접수일 : 2015년 07월 24일 논문게재확정일 : 2015년 07월 30일
* This work was supported by 2012 Hongik University Research Fund.
† 교신저자, khchung@hongik.ac.kr

exponentially and the computing time to the optimality becomes quite so long. In this case, valid inequalities can play a critical role to improve the performance by adding them to the original formulation and tightening the LP relaxation. The combined approach of branch-and-bound and cutting planes are called as Branch-and-Cut (BC) algorithm. While many cutting planes which are either general purpose or problem-dependent have been developed in the literature, the use of a specific cut is not so simple since we have to separate the most violated cut for a given fractional LP solution. This is called as a separation problem that affects the performance of the Branch-and-Cut algorithm.

The aim of this paper is to compare the effectiveness of different kinds of cuts in Branch-and-Cut algorithm. For that purpose, we consider a lot-sizing problem for which many strong valid inequalities have been developed. We study three families of cuts and develop separation algorithms for them. Then, we evaluate the strength of three cuts through the computational study.

1.1 Lot-Sizing Problem and MIP Formulation

In this paper, we consider a variant of the lot-sizing problem. A company wants to decide on a production plan for a set of items $K = 1, \cdots, \mathrm{K}$ over time periods $T = 1, \cdots, \mathrm{T}$. It needs to satisfy a demand for item $k \in K$ in period $t \in T$. In order to produce these items, a set of different machines $M = 1, \cdots, \mathrm{M}$ is used. The time required to produce one unit of item k on machine m is given by p_{km} for $k \in K$ and $m \in M$. Further, in order to produce any number of item k on machine m, the machine must be properly configured, which takes a fixed setup time f_{km} for $k \in K$ and $m \in M$. There is a limited amount of processing time r_{mt} available on each machine $m \in M$ in every time period $t \in T$. The variable production cost for item k in period t is given by g_{kt}. The variable cost for holding one unit of item k in inventory at the end of period t is given by h_{kt}. Finally, the fixed cost for the production of item k in period t is given by c_{kt}.

One of the possible MIP formulations for this problem is given below. We first define the decision variables as follows :

u_{kmt} = Number of items k produced in period t on mahcine m

v_{kmt} = 1 if item k is produced on machine m in period t, 0 otherwise

y_{kt} = Number of items k produced in period t

z_{kt} = 1 if item k is produced in period t, 0 otherwise

Using these variables, we establish the following model.

$$\min \sum_{k \in K} \sum_{t \in T} g_{kt} y_{kt} + \sum_{k \in K} \sum_{t \in T} c_{kt} z_{kt} + \sum_{k \in K} \sum_{t \in T} h_{kt} \left(\sum_{\tau=1}^{t} (y_{k\tau} - d_{k\tau}) \right)$$

$$s.t. \sum_{t=1}^{\tau} y_{kt} \geq \sum_{t=1}^{\tau} d_{kt}, \ \forall \tau \in T, \ \forall k \in K \tag{1}$$

$$(P) \quad y_{kt} \leq d_{tT}^{k} z_{kt}, \quad \forall k \in K, \ \forall t \in T \tag{2}$$

$$u_{kmt} \leq M_{kmt} v_{kmt}, \ \forall k \in K, \ \forall m \in M, \ \forall t \in T \tag{3}$$

$$\sum_{k \in K} f_{km} v_{kmt} + \sum_{k \in K} p_{km} u_{kmt} \leq r_{mt}, \ \forall m \in M, \ \forall t \in T \tag{4}$$

$$y_{kt} = \sum_{m \in M} u_{kmt}, \ \forall k \in K, \forall t \in T \tag{5}$$

$$v_{kmt} \in \{0, 1\}, \ z_{kt} \in \{0, 1\}, \ \forall k \in K, \tag{6}$$

$$\forall m \in M, \forall t \in T$$

$$u_{kmt} \geq 0, \ y_{kt} \geq 0 \ \forall k \in K, \ \forall t \in T \tag{7}$$

where $d^k_{rs} = \sum_{t=r}^{s} d_{kt}$.

Demand constraints (1) make sure that demand is satisfied without backlogging. Setup constraints (2) and (3) enforce binary variable $z_{mt}(v_{kmt})$ to be set to 1 in case item k is produced (on machine m) in period t. Capacity constraints (4) require that the processing time for production must not exceed the capacity of each machine m in every period t. Finally, constraints (5) say that item k can be produced on different machines.

The objective of this problem is to minimize the sum of variable production costs, setup costs, and inventory holding costs over the planning horizon. Regarding the value of M_{kmt} in constraints (3), we used the implicit upper bounds obtained from constraints (4), i.e.,

$$M_{kmt} = \lceil \frac{r_{mt}}{p_{km}} \rceil \ \text{ for } \ \forall m \in M, \forall t \in T.$$

1.2 Branch-and-Cut Algorithm

Branch-and-Bound algorithm is the basic and general method that has been used to solve MIPs. First, a relaxation of the MIP (P) which is an LP relaxation in general is solved to obtain the bounds. Denote by x^* the optimal solution of the LP relaxation in a node V. Further, let I be the set of integer variables. If x^*_j for all $j \in I$ is integral, then an optimal solution of (P) has been found and Branch-and-Bound algorithm terminates. Otherwise, we divide or partition the feasible region into two smaller partitions $P_1 = P \cap \{x \mid x_j \leq \lfloor x^*_j \rfloor\}$ and $P_2 = P \cap \{x \mid x_j \geq \lceil x^*_j \rceil\}$ where x_j for $j \in I$ is a currently fractional variable. Then, we add two subproblems P_1 and P_2 to the problem list L and solve each subproblem recursively until L is empty or some termination criteria are satisfied.

Cutting planes can be used inside of a Branch-and-Bound algorithm to strengthen the LP relaxations which help speed up the Branch-and-Bound search. This mixed use of strong cutting and branching is known as Branch-and-Cut algorithm, which we are considering to solve the lot-sizing problem (P) in this paper. We summarize the general Branch-and-Cut procedure in Algorithm 1 as shown below.

Algorithm 1 : Branch-and-Cut Algorithm

```
Input : $P$, $I$
Output : $x^{IP}$, $z^{IP}$

Initialization
  $L \leftarrow \{P\}$, $x^{IP} \leftarrow \emptyset$, $z^{IP} \leftarrow +\infty$
while $L \neq \emptyset$ do
  Check termination criteria
  Update list $L$
    if $z^W \geq z^{IP}$ for $W \in L$ then
      $L \leftarrow L \setminus \{V\}$
    end
  Node Selection
    Select $V \in L$ and let $L \leftarrow L \setminus \{V\}$
  while Cut Generation needs to be performed do
    Obtain $z^*$ and $x^*$ by solving LP over $V$
    Pruning
      by Infeasibility : $V = \emptyset$
      by Bounds : $z^* \geq z^{IP}$
      by Integrality
        if $z^* < z^{IP}$ then
          if $x^*_j \in \mathbb{Z}$ for all $j \in I$ then
            Update upper bound : $z^{IP} \leftarrow z^*$
            Update incumbent solution : $x^{IP} \leftarrow x^*$
          end
        end
  Call Primal Heuristics
  Call Cut Generation
    if $\exists$ a violated cut then
      add to the formulation
    end
  end
  Branching
end
```

When we apply a Branch-and-Cut algorithm to solve the MIP (P), we have to define and implement each subroutine in detail. For example, in order to add some cuts in branch-and-bound search, **Cut Generation** routine that we will discuss in Section 2 should be clearly clarified. To improve the speed of branch-and-bound search, we also describe one of the ideas to develop **Primal Heuristics** in Section 3.

Throughout the paper, some algorithmic decisions on other subroutines are made as follows;

- Termination criteria : Branch-and-Cut algorithm can be stopped by specifying a maximum number of nodes and maximum solution time. It is also terminated when the optimality gap is less than the given optimality criteria.
- Node selection : Among many node selection rules, we pick from the list L a node V which has the best bound.
- Branching : we choose as a branching variable the fractional integer variable with highest objective coefficient value.

The remaining of the paper is organized as follows. In Section 2, we consider three families of strong valid inequalities for the problem (P) and discuss how to separate the violated cut in the branch-and-bound search. Detailed separation algorithms for each cut are presented and some implementation details are given in Section 4. To improve the branch-and-bound search, we also develop primal heuristics that are described in Section 3. We perform the computational study for three cuts and present the numerical results in Section 4. Finally in Section 5, we conclude with remarks and direction of future research.

2 Cutting Planes

In this section, we study three families of cutting planes that can be applied to the lot-sizing problem (P). We also develop separation algorithms to add the original formulation in branch-and-bound processes. For each family of cuts, we now describe exact or heuristic algorithms to separate a violated cut from many valid inequalities.

2.1 (l, S) Inequalities

Consider first the set X^{LS} defined by the constraints (1) and (2) :

$$\begin{aligned} X^{LS} = \Big\{(y, z) \in R_+^{\mathrm{KT}} \times \{0, 1\}^{\mathrm{KT}} \mid & \\ \sum_{t=1}^{\tau} y_{kt} \geq \sum_{t=1}^{\tau} d_{kt},\ \forall\, \tau \in T,\ \ \forall\, k \in K & \\ y_{kt} \leq d_{t\mathrm{T}}^{k} z_{kt},\ \forall\, k \in K,\ \forall\, t \in T & \\ y_{kt} \geq 0, & \\ y_{k\mathrm{T}} = 0 \Big\}. & \end{aligned}$$

When we drop off the constraints (3)~(5) from the original model (P), we obtain X^{LS}, which is a traditional incapacitated lot-sizing problem. Clearly, X^{LS} is indeed a relaxation of (P). Barany et al. [4] introduced (l, S) inequalities and proved the following theorem.

Theorem 1 : *For given $l \in L := \{1, \cdots, \mathrm{T}-1\}$, $S \subseteq \{1, \cdots, l\}$ and $\bar{S} = L \setminus S$, (l, S) inequalities*

$$\sum_{t \in S} y_{kt} + \sum_{t \in \bar{S}} d_{tl}^{k} z_{kt} \geq d_{1l}^{k}$$

are facet-defining for the convex hull of X^{LS}. Further they also show that (l, S) inequalities provide complete description of the convex hull of X^{LS}.

Since this family of inequalities are exponentially many due to the selection of (l, S), we need to solve the separation problem that, for a given fractional solution (y^*, z^*), determines whether there exists a (l, S) inequality that is violated or not. In Algorithm 2 below, we describe the exact separation algorithm in detail. If the separation algorithm find out violated cuts, we add them to the formulation (P).

Algorithm 2 : Separation Algorithm for (l, S) Inequalities

```
Input : $(y^*, z^*)$
Output : $(l, S)$ cuts = $(\alpha_y, \beta_z, \delta)$

for $k = 1, \cdots, K$ do
  for $l = 1, \cdots, T-1$ do
    $\gamma_l^k \leftarrow \sum_{j=1}^{l} \min\{y_{kj}^*, d_{jl}^k z_{kj}^*\}$
    if $\gamma_l^k < d_{1l}^k$ then
      $L \leftarrow \{1, \cdots, l\}, S \leftarrow \{j \in L \mid y_{kj}^* > d_{jl}^k z_{kj}^*\}$
      if $j \in L$ then
        if $j \in S$ then
          $\alpha_j^k \leftarrow 1, \beta_j^k \leftarrow 0$
        else
          $\alpha_j^k \leftarrow 0, \beta_j^k \leftarrow d_{jl}^k$
        end
      else
        $\alpha_j^k \leftarrow 0, \beta_j^k \leftarrow 0$
      end
        $\delta^k \leftarrow d_{1l}^k$
    end
  end
end
```

It seems that implementation of Algorithm 2 requires $O(T^2)$ time. However, we can obtain the following proposition.

Proposition 1 : *Algorithm 2 can be implemented in* $O(T\, log T)$.

Proof : Note that $0 \le d_{j,j}^k z_{kj}^* \le d_{j,j+1}^k z_{kj}^* \le \cdots \le d_{j,T-1}^k z_{kj}^*$. Using bisection, for each $j \in L$, we can find an integer $l(j) \in \{j, \cdots, T-1\}$ such that $d_{j,l(j)-1}^k z_{kj}^* < y_{kj}^* \le d_{j,l(j)}^k z_{kj}^*$ in $O(\log T)$. □

2.2 Lifted Mixed Cover Inequalities

Since constraints (4) are 0-1 mixed integer knapsack inequalities, we next consider as a relaxation of (P) a generic 0-1 knapsack polyhedron S of the form :

$$S = \left\{(x, y) \in \{0, 1\}^m \times R^n \mid \sum_{j \in M} a_j x_j + \sum_{j \in N} b_j y_j \le d\right\}$$

where $a_j > 0 \, \forall j \in M$, $b_j > 0 \, \forall j \in N$, $d > 0$ and $y_j \le \nu_j \forall j \in N$. If there exists a subset $J \subseteq N$ satisfying $d - \sum_{j \in J} \nu_j b_j > 0$, then we rewrite the knapsack constraint as :

$$\sum_{j \in M} a_j x_j + \sum_{j \in N \setminus J} b_j y_j \le d - \sum_{j \in J} b_j \nu_j + \sum_{j \in J} b_j \overline{y_j} \quad (8)$$

by complementing the continuous variables y_j for $j \in J$, i.e. by introducing $\overline{y_j} := \nu_j - y_j$. Observe now that S can be relaxed into the set

$$S' = \left\{(x, s) \in \{0, 1\}^m \times R_+ \mid \sum_{j \in M} a_j x_j \le d' + s\right\} \quad (9)$$

by relaxing the variables y_j for $\forall j \in N \setminus J$ out of the problem, by substituting $d' = d - \sum_{j \in J} b_j \nu_j$, by defining a new variable $s := \sum_{j \in J} b_j \overline{y_j}$, and by relaxing the bound of s to be $[0, \infty)$.

Now, we observe that S' is a 0-1 knapsack polyhedron with a single continuous variable, which was studied by Marchand and Wolsey [9]. One of the facet-defining inequalities for the convex hull of S' can be obtained via superadditive lifting. Assume that $C \subseteq M$ is a strong cover for S' with the excess $\mu = \sum_{j \in C} a_j - d'$. Further, assume without loss of generality that $C = \{1, \cdots, q\}$ and $a_1 \geq a_2 \geq \cdots \geq a_q$. Then, Marchand and Wolsey [9] developed face-defining inequalities for the convex hull of S' using sequence-independent lifting as shown in the following theorem.

Theorem 2 : *Lifted cover inequality*

$$\sum_{j \in C} \min\{a_j, \mu\} x_j + \sum_{j \in M \setminus C} \phi_C(a_j) x_j \leq \sum_{j \in C} \min\{a_j, \mu\} - \mu + s \tag{10}$$

is facet-defining for the convex hull of S' where

$$\phi_C(a) = \begin{cases} i\mu & \text{if } A_i \leq a < A_{i+1} - \mu \\ & \text{for } i = 0, 1, \cdots, r-1 \\ i\mu + (a - A_i) & \text{if } A_i - \mu \leq a < A_i \\ & \text{for } i = 1, \cdots, r-1 \\ r\mu + (a - A_r) & \text{if } A_r - \mu \leq a, \end{cases}$$

$r = \max\{j \in C \mid a_j > \mu\}$, *and* $A_i = \sum_{j=1}^{i} a_j$ *for* $i = 1, \cdots, r$.

Since S' is a relaxation of S, we convert (10) into a valid inequality for S by substituting $s = \sum_{j \in J} b_j \overline{y_j}$ and $\overline{y_j} = \nu_j - y_j$ back, i.e.

$$\sum_{j \in C} \min\{a_j, \mu\} x_j + \sum_{j \in M \setminus C} \phi_C(a_j) x_j + \sum_{j \in J} b_j y_j \leq \sum_{j \in C} \min\{a_j, \mu\} - \mu + \sum_{j \in J} b_j \nu_j. \tag{11}$$

Therefore, we can say that (11) is a valid inequality for S.

Note that lifted mixed cover inequalities (LMCI) (11) are also exponentially many since we can choose in different ways the subset $J \subseteq K$ and a strong cover C in (P). We propose and describe a heuristic separation algorithm in Algorithm 3 below. In particular, we address the technical issues to find a subset $J \subseteq N$ in (8) and a strong cover $C \subseteq M$ in (9). In order to find the subset J for a given fractional solution (x^*, y^*), we need to minimize $\sum_{j \in N \setminus J} b_j y_j^*$, i.e.

$$\begin{aligned} \max \quad & \sum_{j \in J} b_j y_j^* \\ s.t. \quad & d - \sum_{j \in J} b_j \nu_j \geq 0. \end{aligned}$$

This idea was proposed and shown to be effective by Marchand and Wolsey [9]. In order to select J using the exact approach, we have to solve the knapsack problem :

$$(K1) \quad \begin{aligned} \max \quad & \sum_{j \in N} (b_j y_j^*) z_j \\ s.t. \quad & \sum_{j \in N} (b_j \nu_j) z_j \leq d \\ & z_j \in \{0, 1\} \ \forall j \in N. \end{aligned}$$

However, solving knapsack problem ($K1$) is very time-consuming and separation routines are called over and over in the Branch-and-Bound search. Further, since our ultimate goal is to find a violated LMCI, it suffices us to use a greedy heuristic to solve ($K1$) using the ratios $\frac{y_j^*}{\nu_j}$.

After J is determined by the greedy heuristic, we next find an initial cover C to get LMCI. For a given fractional solution (x^*, y^*), in order to find the most violated cover exactly, we have to solve the following knapsack problem :

$$\begin{aligned}\zeta = 1 - \min \quad & \sum_{j \in M} (1 - x_j^*) z_j \\ s.t. \quad & \sum_{j \in M} a_j z_j > d \\ & z_j \in \{0, 1\} \ \forall j \in M.\end{aligned}$$

This is also time-consuming, but here we can use the coefficient-independent cover generation approach which was introduced by Gu et al. [8]. Basic idea is to select first the variables with the highest LP values for the cover C. Let $C_1 = \{j \in M \mid x_j^* = 1\}$ and $\bar{d} = d - \sum_{j \in C_1} a_j$. Then, sort the variables in $M \setminus C_1$ in non-increasing order of LP values and add them to C until $\sum_{j \in C} a_j > b$. Let $C := C \cup C_1$. When the cover C obtained this way is not a strong cover, we delete the variables from C to convert it to a strong cover.

After J and C are determined in this way, we can easily obtain the LMCI (11) with the following Algorithm 3 as shown below. However, the inequality that we obtained may not be a violated cut since (10) is facet-defining for S', the relaxation of S, and we find J and C in the heuristic approach. Therefore, we need to check if the obtained inequality is violated or not at the final step. Note that Algorithm 3 is not exact but heuristic.

2.3 Split Cuts

Since the lot-sizing problem (P) is a 0-1 MIP, we can use a split cut that is one of the general cutting planes in MIP. Consider an MIP of the general form :

Algorithm 3 : Separation Heuristics for Lifted Mixed Cover Inequalities

```
Input : $(v^*, u^*)$, $u^U$
Output : Lifted Mixed Cover cuts = $(\alpha_v, \beta_u, \delta)$

Initialization
  $\nu \leftarrow n^U$, $\bar{u} \leftarrow \nu - u^*$
for $m = 1, \cdots, M$ do
    $A_m \leftarrow \sum_{k=1}^{K} f_{km}$
    for $t = 1, \cdots, T$ do
        Find $J \subseteq K$ maximizing $\sum_{j \in J} p_{jm} \times u_{jmt}^*$
            subject to $\sum_{j \in J} p_{jm} \times \nu_{jmt} < d_{mt}$
        for $k = 1, \cdots, K$ do
            if $k \in J$ then
                $\beta_{kmt} \leftarrow p_{km}$
            else
                $\beta_{kmt} \leftarrow 0$
            end
        end
        $d' \leftarrow d - \sum_{j \in J} p_{jm} \times \nu_j$, $\delta \leftarrow \sum_{j \in J} p_{jm} \times \nu_j$
        if $A_m > d'$ then
            Find a cover $C$ and calculate
                $\mu = \sum_{k \in C} f_{km} - d'$
            if $C \neq a\ strong\ cover$ then
                Convert $C$ into a strong cover and
                update $\mu$
            end
            Find a strong element $k$
            Sort $f_{km}$ in non-increasing order and
                $r \leftarrow \max\{k \in C \mid f_{km} > \mu\}$
            for $i = 1, \cdots, r$ do
                Compute $A_{im} \leftarrow \sum_{k=1}^{i} f_{km}$
            end
            for $k = 1, \cdots, K$ do
                if $k \in C$ then
                    $\alpha_{kmt} \leftarrow \min\{f_{km}, \mu\}$
                else
                    $\alpha_{kmt} \leftarrow \phi(f_{km})$
                end
            end
            $\delta \leftarrow \delta + \sum_{k \in C} \min\{f_{km}, \mu\} - \mu$
            if $\alpha_v{}^T v^* + \beta_u{}^T u^* - \delta > 0$ then
                $\alpha_v \leftarrow \alpha_v$ , $\beta_u \leftarrow \beta_u$ , $\delta \leftarrow \delta$
            end
        end
    end
end
```

$$P=\{(x,y)\in Z^m\times R^n \mid Ax+By\le d\}$$

where A is $k\times m$ matrix, B is $k\times n$ matrix, and $d\in R^k$ along with a disjunction $\pi'x\le\pi_0\vee\pi'x\ge\pi_0+1$ where $\pi\in Z^m$ and $\pi_0\in Z$. Since variables x in the set P are integer, it is trivial that $P\subseteq P_1\cup P_2$ where

$$P_1=\{(x,y)\in R^m\times R^n \mid Ax+By\le d,\ -\pi'x\ge-\pi_0\}$$

and

$$P_2=\{(x,y)\in R^m\times R^n \mid Ax+By\le d,\ \pi'x\ge\pi_0+1\}.$$

Hence, we have that $\mathrm{conv}(P)\subseteq\mathrm{conv}(P_1\cup P_2)$.

The convex hull of $P_1\cup P_2$ can be obtained by disjunctive programming techniques and valid inequalities for $\mathrm{conv}(P_1\cup P_2)$ are called split cuts. For a given fractional optimal solution (x^*,y^*) to the LP relaxation of P, we can perform the exact separation that determines if there exists a violated split cut $\alpha x+\beta y\ge\delta$ from the disjunction $\pi'x\le\pi_0\vee\pi'x\ge\pi_0+1$ by solving the cut generation linear program $(CGLP)$:

In order to make sure that $(CGLP)$ does not become unbounded, we need to add a normalization constraint. We can try

(i) $\delta\in\{-1,1\}$,

(ii) $\sum_{i=1}^{m}|\alpha_i|+\sum_{j=1}^{n}|\beta_j|\le 1$, or

(iii) $-1\le\alpha_i\le 1$,

for $\forall i=1,\cdots,m$ and $-1\le\beta_j\le 1$, $\forall j=1,\cdots,n$. Since we have seen the unbounded cases when (i) is used for (P), we impose (iii) in the computational study due to the ease of implementation. After solving $(CGLP)$ with the normalization constraint, if $z^*<0$, then the corresponding inequality (α,β,δ) is the violated cut for the current LP solution (x^*,y^*) and can be added to the formulation (P). Since the separation of split cuts is exactly performed by solving $(CGLP)$, we always obtain the split cut whenever $z^*<0$.

However, in order to obtain good split cuts, it is important to define the right disjunctions for the lot-sizing problem (P). Note that (P) has only binary variables v and z. Therefore, we can use the disjunction $x_j\le 0\vee x_j\ge 1$, i.e., $\pi=e_j$ and $\pi_0=0$ for $j\in I$. The split cuts derived from these disjunctions are called lift-and-project cuts; see Balas et al. [2]. When implementing the lift-and-project cuts, we have to decide how many cuts to generate at a given node. The time needed to generate a given number of cuts is usually longer with many rounds of a few cuts than fewer rounds

$$\begin{array}{llllllllll}
z^*= & \min & (x^*)'\alpha & +(y^*)'\beta & -\delta & & & & & \\
 & s.t. & \alpha & & & -A^Tu_1 & & +\pi v_1 & & =0\\
 & & & \beta & & -B^Tu_1 & & & & =0\\
 & & & & \delta & -d'u_1 & & +\pi_0 v_1 & & \le 0\\
(CGLP) & & \alpha & & & & -A^Tu_2 & & -\pi v_2 & =0\\
 & & & \beta & & & -B^Tu_2 & & & =0\\
 & & & & \delta & & -d'u_2 & & -(\pi_0+1)v_2 & \le 0\\
\end{array}$$

$$\alpha\in R^m,\ \beta\in R^n,\ \delta\in R,\ u_1\in R_+^k,\ u_2\in R_+^k,\ v_1\in R_+,\ \ v_2\in R_+$$

of many cuts because re-optimization is needed in every round of the cut generation. Hence, we define the parameter r as the percentage of the fractional variables. Then, the maximum number of cuts to be generated is set to

$$nCuts := \lceil |F| \times r \rceil$$

where F is the set of fractional variables in a node. Detailed steps for obtaining split cuts are summarized in Algorithm 4.

Algorithm 4 : Separation Algorithm for Split Cuts

```
Input : $x^* = (u^*, v^*, y^*, z^*)$, $I$, $r$
Output : split cuts = $(\alpha, \beta, \delta)$

Initialization
$F \leftarrow \{i \in I \mid x_i^* \notin Z\}$
$nCuts \leftarrow \lceil |F| \times r \rceil$
Sort $F$ in the non-increasing order of the fractional value
for $j = 1, \cdots, nCuts$ do
    Define a disjunction : $\pi \leftarrow e_j$, $\pi_0 \leftarrow 0$
    Call BuildCGLP
    Solve CGLP and let $(\alpha^*, \beta^*, \gamma^*)$ be an optimal solution
    if $Obj_{CGLP} < 0$ then
        $\alpha \leftarrow \alpha^*, \beta \leftarrow \beta^*, \delta \leftarrow \delta^*$
    end
end
```

In our computational experiments later, we test three cases : r = 10%, r = 50%, and r = 100%. The comparison of these cases are provided in Section 5.

3 Primal Heuristics

While tightening the formulation (P) by using the cutting planes discussed in Section 3 enables us to obtain the better lower bound, finding a good integer feasible solution is also important in a Branch-and-Bound algorithm. This is because it provides the upper bound on the objective value of (P) and ultimately allows to prune more nodes by the tighter bound.

Typically, there are two types of heuristics : construction heuristics that produce a feasible solution from the ground, and improvement heuristics that try to improve a given LP solution. Since the LP solutions associated with deeper nodes get closer to be integer solution, we can build a heuristic approach to convert an optimal solution to the LP relaxation of a node in the tree to a good feasible solution to (P). In this section, we consider two improvement heuristics : Dive-and-Fix heuristics that can be used for any general MIP problems and a Time-based Forward heuristics that we developed using characteristics of the lot-sizing problem (P).

3.1 LP-based Dive-and-Fix vs. RINS-based Dive-and-Fix

The basic idea of Dive-and-Fix (DF) heuristics is to take the LP solution at a node of the Branch-and-Bound tree and dive down the tree in search for finding a feasible solution. In general, since it iterates two steps to solve the LP and to fix the integer variables using the LP solution, this LP-based DF heuristics performs well with tight LP formulation. However, since it may not find good feasible solutions particularly at the early nodes of the tree, we also try a variant of Dive-and-Fix heuristics, which we call RINS-based DF heuristics.

While we fix the variables using the LP solution in LP-based DF, we fix the variables using the incumbent solution in RINS-based DF. Actually, the idea of RINS (Relaxation Induced Neighborhood Search) proposed by Danna et al. [6] is to explore the neighborhood between the LP solution and the IP solution by solving smaller MIPs again and again. However, solving small MIPs is computationally expensive in general, we just adopt the idea and take the combined approach that fix first the variables using the incumbent solution and solve the LP again. We provide two heuristics in Algorithm 5 as shown below.

Algorithm 5 : Dive-and-Fix Heuristics

```
Input : $x^* = (u^*, v^*, y^*, z^*)$, $I$, $x^{IP}$
Output : $x^H = (u^H, v^H, y^H, z^H)$

Initialization
$F \leftarrow \{i \in I \mid x_i^* \notin Z\}$
while $F \neq \varnothing$ do
  if $LP-based$ then
    Find the variable closest to integer
        $j \leftarrow \mathrm{argmin}_{j \in F}\{\min[x_j^*, 1 - x_j^*]\}$
    if $x_j^* < 0.5$ then
      fix $x_j = \lfloor x_j^* \rfloor = 0$
    else
      fix $x_j = \lceil x_j^* \rceil = 1$
    end
  else
    $RINS-based$ : Fix the variables $x_j = x_j^{IP}$ for all
        $j \in I$ with $x_j^* = x_j^{IP}$
  end
  Solve the resulting HeuristicLP : $x^{HLP} \leftarrow$ new
  HeuristicLP solution
  if $HeuristicLP\ is\ infeasible$ then
    STOP
  else
    $F \leftarrow \{i \in I \mid x_i^{HLP} \notin Z\}$
  end
end
$x^H \leftarrow x^{HLP}$
```

3.2 Time-Based Forward Heuristics

Since Dive-and-Fix is a problem-independent heuristics, it does not use the characteristics of the lot-sizing problem. We developed a Time-Based Forward (TBF) heuristics which fixes the integer variables as time period moves from the beginning to the end. At each time period, we fix the values of binary variables z first using the LP solution and then solve the LP again to find the feasible solution satisfying all the constraints. We next fix the values of v since the variables v depend on the decision of z. We continue to sequentially fix the binary variables z and v as the time moves to the end T. Details are presented in Algorithm 6.

Algorithm 6 : Time-based Forward Heuristics

```
Input : $x^* = (u^*, v^*, y^*, z^*)$, $I$
Output : $x^H = (u^H, v^H, y^H, z^H)$

Initialization
$F \leftarrow \{i \in I \mid x_i^* \notin Z\}$, $t \leftarrow 1$
while $F \neq \varnothing$ and $t \leq \mathrm{T}$ do
  for $i = 1, 2$ do
    if $i \equiv 1, 2$ then
      Fix first $z_{kt}^H\ \forall k \in K, \forall t \in T$
      if $z_{kt}^* \in Z$ then
        $z_{kt} = z_{kt}^*$
      else
        $z_{kt} = \lceil z_{kt}^* \rceil = 1$
      end
    else
      Fix next $v_{kmt}^H\ \forall k \in K, \forall m \in M, \forall t \in T$
      if $v_{kmt}^* \in Z$ then
        $v_{kmt} = v_{kmt}^*$
      else
        $v_{kmt} = \lceil v_{kmt}^* \rceil = 1$
      end
    end
  end
  Solve HeuristicLP : $x^{HLP} \leftarrow$ new HeuristicLP
  solution
  if $HeuristicLP\ is\ infeasible$ then
    STOP
  else
    $F \leftarrow \{i \in I \mid x_i^{HLP} \notin Z\}$
  end
  $t \leftarrow t + 1$
end
$x^H \leftarrow x^{HLP}$
```

One of the disadvantages of the TBF heuristics is that it may violate the capacity constraints at the end since early decision may prevent the generation of a feasible solution later.

4. Computational Study

We now perform a computational study to evaluate the strength of three families of cutting planes for the lot-sizing problem (*P*). Before we present the numerical results, we first describe how to solve (*P*) under what circumstances. We next describe how to implement each separation algorithm proposed before in CPLEX. Impacts of three cuts are compared and the combined use of each cut is also investigated in Section 4.3. Comparison of three heuristics is also presented at the end of this section.

4.1 Testing Environments

We implement a branch-and-cut algorithm using test instances of (*P*) are randomly generated. The problems are solved using CPLEX 12.1 on a Windows machine with Intel CORE i7 3.40 GHz processor and 8GB RAM. Basically, we perform two types of tests; one is for the decision of algorithmic parameters such as the number of cuts generated at a time and the cut rounds for each family of cuts, and the other is for the verification of our implementations through testing relatively large instances. For the first type of tests, we use 10 instances of size $(K, M, T) = (4, 2, 3)$. We specify the details for the second type of tests in Section 4.5.

4.2 Implementation Details

In order to implement Cut Generation and Primal Heuristics in Algorithm 1, we have to make some implementation decisions such as how many cuts are generated and how many rounds of cut generations are performed before branching. Here we describe how to choose those values by preliminary tests with 10 instances of size $(K, M, T) = (4, 2, 3)$.

- LS_CutGeneration : For (l, S) cuts, we run two types of cut rounds and four types of the options to select nodes where cuts are generated. <Table 1> shows that cut rounds does not affect the performance of the BC algorithm significantly, and the time needed for more rounds does not also increase much. Further, the more often we generate (l, S) cuts, the better the BC algorithm performs. However, many cuts are generated at the early nodes of the tree and fewer cuts are generated as the BC performs. Based on these observations, we decide to use one round of cuts generated at *Node Level* ≤ 5.

<Table 1> Where to Generate Cuts vs. How Many Cut Rounds for (l, S) cuts

	1 round			5 rounds		
	# Cuts	Nodes	Time(s)	# Cuts	Nodes	Time(s)
Root Node	0.80	45.70	3.22	0.80	45.70	3.46
Node Level ≤ 5	1.20	46.70	4.70	1.20	46.70	4.99
Node Level = 5*n*	1.00	46.30	3.49	1.00	46.30	3.64
Every Node	1.60	45.50	5.03	1.60	45.50	5.29

- LMC_CutGeneration : For LMC cuts, we similarly run two types of cut rounds and four types of the options to select nodes where cuts are generated. <Table 2> shows that cut rounds does not affect the performance of the BC algorithm, and the time for more rounds is also negligible. However, note here that more LMC cuts are generated at the deeper level of the tree. Since it is computationally inefficient to generate cuts at every node, we decide to use one round of cuts generated only if the node level is the multiple of 5.
- SPLIT_CutGeneration : In order to separate the split cuts in Section 3.3, we have to solve GCLP which is relatively larger than the original problem. As we already discussed before, more cuts in fewer rounds are better than less cuts in many rounds. Hence, we only compare one round with two rounds of the cut generation. Further, since the LP solver takes more time to solve the large size of LP, we decide to generate the split cuts only at the root node. Instead, we compare how the number of cuts generated affects to the improvement of the lower bound.

As we described before, we generate $\lceil |F| \times r \rceil$ number of lift-and-project cuts where F is the set of fractional variables and r is the percentage of fractional variables used as a disjunction to generate cuts. We test three values of r (10%, 50%, and 100%) and the results are shown in <Table 3>. We observe that more cuts clearly help improve the lower bound, but need to take extremely longer time. Therefore, we only generate one round of cuts with 50% of fractional variables.

- TBF_Heuristic : Without any tests, we see that the more often we call the primal heuristics, the more likely we find out the feasible solution, which leads the upper bound to converge to the optimal value quickly. However, the TBF heuristics needs to solve so many LPs over and over that it takes longer time just for the hope of finding better feasible solution. Therefore, we

〈Table 2〉 Where to Generate Cuts vs. How Many Cut Rounds for LMC Cuts

	1 round			5 rounds		
	# Cuts	Nodes	Time(s)	# Cuts	Nodes	Time(s)
Root Node	0.50	47.80	3.39	0.50	47.80	3.55
Node Level $\leq$ 5	0.80	48.40	3.63	0.80	48.40	3.64
Node Level = 5*n*	1.30	47.80	3.52	1.30	47.80	3.69
Every Node	2.10	48.10	3.87	2.10	48.10	3.65

〈Table 3〉 Number of Cuts Generated vs. Cut Rounds for Split Cuts

	1 round			2 rounds		
	GC(%)	Nodes	Time(s)	GC(%)	Nodes	Time(s)
10%	9.07	47.70	4.97	13.45	57.40	8.77
50%	24.99	53.90	49.84	30.22	63.40	234.14
100%	25.97	56.90	66.85	26.38	60.20	304.08

use the default setting that we call the TBF heuristics only where node levels are multiples of 5 and before cuts are added.

4.3 Performance of Three Families of Cuts

We evaluate the strength of three families of cuts discussed in Section 2. Performance measures that we used here are the number of cuts generated, the percentage of gap closed, the number of nodes in the tree, and the solution time. Let z^0 be the optimal value of the LP relaxation at the root node before the cuts are generated and z^{cut} be the optimal value of the LP relaxation at the root node after the cuts are generated and added to the formulation. Further, let z^{IP} be the objective value of the optimal integer solution that we have after BC terminates. We define the percentage of gap closed(GC) as

$$GC = \frac{z^{cut} - z^0}{z^{IP} - z^0} \times 100.$$

As the value of the gap closed is higher, the formulation gets tighter after the cuts are added. Similar to the tests before, we generate 10 instances with the size $(K, M, T) = (4, 2, 3)$ and solve each problem with three cuts respectively. Every cut is generated only at the root node and two rounds of cut generation is performed. Since it takes so much time to solve CGLP to generate split cuts, we use 10% of the total fractional variables as the disjunctions.

The results are presented in <Table 4>. Among three families of cuts, the performance of (l, S) cuts is shown to be the best. Although it need quite a short time to generate cuts, (l, S) cuts significantly close the gap after cuts are added and reduce the number of nodes in the tree. This is because constraints (1) and (2) play more important role to define the structure of the lot-sizing problem than others. LMC cuts are quickly generated, but they do not help improve the bounds significantly. Split cuts reasonably close the gap, but it relatively takes too much time to generate the cuts even for small size of the instances.

We used only one family of cuts so far, but we

〈Table 4〉 Performance of Three Families of Cuts

ID	(l, S) cuts				LMC cuts				Split Cuts			
	Cuts	GC (%)	Nodes	Time (sec)	Cuts	GC (%)	Nodes	Time (sec)	Cuts	GC (%)	Nodes	Time (sec)
1	1	17.69	63	4.72	0	0.00	61	4.05	2	17.69	183	32.70
2	2	10.23	25	3.11	1	0.12	25	2.84	4	8.47	27	11.78
3	1	100.00	9	0.75	0	0.00	11	0.73	2	0.00	7	1.70
4	0	0.00	19	1.13	0	0.00	19	1.05	2	0.00	13	2.05
5	2	52.86	13	1.78	1	0.00	22	2.44	3	54.59	13	3.94
6	0	0.00	31	2.17	1	0.01	31	2.20	2	1.87	28	5.48
7	1	38.78	95	6.30	0	0.00	105	6.28	2	38.78	99	8.48
8	0	0.00	182	9.33	1	0.02	182	9.47	2	0.00	184	15.64
9	0	0.00	1	0.11	0	0.00	1	0.14	2	0.00	1	1.45
10	1	11.68	19	2.81	1	0.00	21	2.91	2	13.11	19	4.48
Avg		23.12	45.70	3.22		0.02	47.80	3.21		13.45	57.47	8.77

also try the joint use of the cuts. Since the performance of (l, S) cuts are shown to dominate others, we always generate (l, S) cuts and add other cuts on top of them. We test three cases :

(i) both (l, S) cuts and LMC cuts are used,
(ii) both (l, S) cuts and split cuts are used, and
(iii) all cuts are used.

We observe in <Table 5> that we have the benefits when we use the cuts jointly. However, there is a trade-off between the gap closed and the solution time. As we addressed before, it takes long time to generate a split cut since we have to solve a large size of CGLP. Those results are also applied here since the solution time increases too much when split cuts are used. Therefore, we conclude that we generate (l, S) cuts and LMC cuts in one round of the cut generation when we test the general instances in Section 4.5.

4.4 Performance of Three Heuristics

We compare the effectiveness of three heuristics described in Section 3. Since Dive-and-Fix is used to find the feasible solutions for general MIPs without using the problem-specific structures, we can easily guess that Time-based Forward heuristics performs better than DF heuristics. <Table 6> shows this by computational results. We test 10 instances of size $(K, M, T) = (4, 2, 3)$ with three heuristics and put the average of the values down in each row. Clearly, TBF produces an improved feasible solution more often, but it also requires time to solve the LPs iteratively. We have not seen any issues to apply TBF to small size of problems, but we have to set the maximum time that we allow to run TBF for large size of instances in Section 4.5.

〈Table 6〉 Performance of Three Heuristics

	# of improved solution	Nodes	Time(s)
LP DF	0.50	46.10	11.91
RINS DF	0.60	45.70	5.87
TBF	2.00	43.60	6.72

4.5 Numerical Results

The main goal of this paper is to see the effectiveness of different families of cuts in Branch-and-Bound framework. Since we make the implementation decisions through the preliminary tests before, we now verify our implementations by testing a few general instances. We solve test problems 1-10 of different sizes $(K, M, T) = (3, 3, 3), (4, 4, 4)$, and $(5, 5, 5)$. We set the maximum time to 5,000 seconds as the stopping criteria and use both (l, S) cuts and LMC cuts jointly. All the parameters for the cut generation are used as we decide before. <Table 7> summarized the lower bounds, the upper bounds, and the optimality gap that we obtained.

Our algorithms solve most of the problems to the optimality, but for a few instances, BC algo-

〈Table 5〉 Joint Impacts of Three Families of Cuts

(l, S)+LMC				(l, S)+Split				All Cuts			
Cuts	GC (%)	Nodes	Time (sec)	Cuts	GC (%)	Nodes	Time (sec)	Cuts	GC (%)	Nodes	Time (sec)
1.50	23.14	43.50	8.99	8.80	26.32	67.70	72.04	9.40	26.33	68.30	75.60

〈Table 7〉 Results for Large Instances

ID	(K, M, T) = (3, 3, 3)			(K, M, T) = (4, 4, 4)			(K, M, T) = (5, 5, 5)		
	LoBnd	UpBnd	OptGap	LoBnd	UpBnd	OptGap	LoBnd	UpBnd	OptGap
1	3792.00	3792.00	0.00	7041.00	7041.00	0.00	9875.00	9983.00	1.09
2	3429.00	3429.00	0.00	9372.00	9372.00	0.00	18486.00	18486.00	0.00
3	2945.00	2945.00	0.00	7846.00	8724.00	11.19	13941.93	14017.73	0.54
4	2992.00	2992.00	0.00	6962.00	6962.00	0.00	14938.00	14938.00	0.00
5	3236.00	3236.00	0.00	7997.00	7997.00	0.00	15788.00	15788.00	0.00
6	3909.00	3909.00	0.00	8097.81	8111.00	0.16	19564.00	25394.00	29.80
7	3869.00	3869.00	0.00	7546.20	7546.20	0.00	18311.00	24775.00	35.30
8	2584.00	2584.00	0.00	7041.00	7041.00	0.00	19467.74	19570.82	0.53
9	1725.00	1725.00	0.00	5461.00	5461.00	0.00	13313.00	13313.00	0.00
10	3430.00	3430.00	0.00	6899.00	6899.00	0.00	17844.00	17844.00	0.00

rithm terminates as the maximum solution time exceeds before finding the optimal IP solution. In particular, instance 7 of size $(K, M, T) = (5, 5, 5)$ has some numerical issues in solving the heuristic LPs and spends so long time in finding the improved solution even the lower bounds does not improve any more. For those instances, we have seen that the best integer feasible solution obtained by BC is estimated to be within the optimality gap. This results is useful when optimal solutions are fairly difficult to find in practice.

5. Concluding Remarks

In this paper, we considered three families of cuts for a lot-sizing problem in order to evaluate the strength of cuts in Branch-and-Cut algorithm. We developed separation algorithms for three cuts and provided implementation details that are applied in CPLEX. Through the computational study, we obtained that (l, S) cuts are quickly generated and significantly helpful to improve the gap closed. While LMC cuts take short time to be generated, they are not significant to improve the bound. Split cuts may be helpful to close the gap, but it takes much longer computing time to generate the cuts.

For the direction of future research, the first approach that we can try is to use the more general cuts in MIP. Among general cutting planes in MIP, we use lifted knapsack cover cuts and split cuts in Section 2, but we can also add Gomory Mixed-Integer Cuts (GMIC); see Gomory [7]. According to the study of Balas et al. [3] and Bixby and Rothberg [5], GMIC turns out to be the most effective cutting plane in practice. In order to generate a GMIC, we first solve LP relaxation and find a basic variable with a fractional value from the simplex tableau. For each basic variables, we generate GMICs and add them to the formulation if violated.

To generate a lift-and-project cut in Section 2, we always formulate and solve the Cut Generation LP (*CGLP*). Since (*CGLP*) become fairly large size LP to solve just for generating one cut, it is computationally inefficient as shown in Section 4. However, Balas and Perregaard [1] give an idea to obtain lift-and-project cuts without solving

CGLP. They show that there is a precise correspondence between the basic feasible solution of CGLP and the basic solutions of the LP relaxation. Using this correspondence, we do not need to formulate and solve the CGLP explicitly, but compute deep lift-and-project cuts by pivoting directly in the LP relaxation.

References

[1] Balas, E. and M. Perregaard, "A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed-integer gomory cuts for 0-1 programming," *Mathematical Programming*, Vol.94(2003), pp.221-245.

[2] Balas, E., S. Ceria, and G. Cornuéjols, "A lift-and-project cutting plane algorithm for mixed 0-1 Programs," *Mathematical Programming*, Vol.58(1993), pp.295-324.

[3] Balas, E., S. Ceria, G. Cornuéjols, and R.N. Natraj, "Gomory cuts revisited," *Operations Research Letters*, Vol.19(1996), pp.1-9.

[4] Barany, I., T.J. Van Roy, and L.A. Wolsey, "Uncapacitated lot-sizing : The convex hull of solutions," *Mathematical Programming*, Vol.22(1984), pp.32-43.

[5] Bixby, R. and E. Rothberg, "Progress in computational mixed integer programming-a look back from the other side of the tipping point," *Annals of Operations Research*, Vol. 149(2007), pp.37-41.

[6] Danna, E., E. Rothberg, and C. Le Pape, "Exploring relaxation induced neighborhoods to improve MIP Solutions," *Mathematical Programming*, Vol.102(2005), pp.71-90.

[7] Gomory, R.E., "An algorithm for the mixed integer program," *Technical report RM-2597*, The RAND Corporation, 1960.

[8] Gu, Z., G.L. Nemhauser, and M.W.P. Savelsbergh, "Lifted cover inequalities for 0-1 integer programs : Computation," *INFORMS Journal on Computing*, Vol.10(1998), pp.427-437.

[9] Marchand, H. and L.A. Wolsey, "Aggregation and mixed integer rounding to solve MIPs," *Operations Research*, Vol.49(1998), pp.363-371.

[10] Marchand, H. and L.A. Wolsey, "The 0-1 knapsack problem with a single continuous variable," *Mathematical Programming*, Vol.85 (1999), pp.15-33.