Octanary Polyhedral Branch and Bound for Integer Programs

James P. Bailey

Department of Engineering Systems and Design,
Singapore University of Technology and Design,
Singapore
E-mail: james_bailey@sutd.edu.sg

Todd Easton

Department of Industrial and Manufacturing Systems Engineering,
Kansas State University,
Manhattan, KS, USA
E-mail: teaston@ksu.edu

Fabio Vitor

Department of Mathematics,
University of Nebraska at Omaha,
Omaha, NE, USA
E-mail: fabioftv@ksu.edu

Abstract: This paper introduces the octanary branching algorithm (OBA), a polyhedral branching technique to solve integer programs. Unlike the traditional branch and bound algorithm, each of OBA's branching nodes generates eight children instead of two. Four of them are created by equality constraints, while the other four use inequalities. This branching strategy allows a dimension reduction of the linear relaxation space of the four equality children, which should enable OBA to find quality integer solutions sooner than the branch and bound algorithm. Computational experiments showed that the branch and bound algorithm required over one billion nodes to identify a solution that is at least as good as the solution found by OBA after only half a million nodes. Consequently, OBA should replace the branch and bound algorithm during the first portion of the branching tree, be used to identify a warm start solution, or be implemented as a diving strategy.

Keywords: Branch and Bound; Hyperplane Branching; Branching Polyhedra; Random Diving; Integer Programming.

Reference to this paper should be made as follows: Bailey, J.P., Easton, T. and Vitor, F. (201X) 'Octanary polyhedral branch and bound for integer programs', *International Journal of Operational Research*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: James Bailey is a Postdoctoral Researcher in the Engineering Systems Design Pillar at the Singapore University of Technology and Design. He obtained his Ph.D. in Algorithms, Combinatorics, and Optimization from the Georgia Institute of Technology, his M.S. in Industrial Engineering and B.S. degrees in both Mathematics and Industrial Engineering from Kansas

State University. His main research areas are machine learning, combinatorial optimization, social choice, and game theory.

Todd Easton received a B.S. in Mathematics with a minor in Statistics from Brigham Young University, a M.S. in Operations Research from Stanford University, and a Ph.D. in Industrial Engineering from Georgia Institute of Technology. He worked as a Postdoctoral Fellow at Georgia Institute of Technology, and then joined the department of Industrial and Manufacturing Systems Engineering at Kansas State University. He is currently an Associate Professor and also a University Distinguished Teaching Scholar. His research interests are in combinatorial optimization and teaching techniques.

Fabio Vitor is an Assistant Professor in the department of Mathematics at the University of Nebraska at Omaha. He received a Ph.D. in Industrial Engineering and a M.S. in Operations Research from Kansas State University, and a B.S. in Industrial Engineering from Maua Institute of Technology (Brazil). He also worked for Monsanto, Kalmar (Cargotec Corporation), and Volkswagen. His research interests include the development of algorithms to more quickly solve continuous and discrete optimization problems such as linear, nonlinear, and integer programs.

1 Introduction

Integer programming is one of the most important classes of optimization problems. For decades, integer programs have been used to model numerous complex systems in the public and private sectors. Quickly finding an optimal solution to these mathematical models is vital to decision makers in order to provide better products and services to meet consumer needs. Because of the importance of integer programs, several techniques have been developed throughout the years to more quickly solve these models. This paper presents the octanary branching algorithm, a new technique that can help decrease the time to solve integer programming problems.

Formally, define an integer program (IP) as:

$$\begin{aligned} \text{maximize } & z = c^T x \\ \text{subject to } & Ax \leq b \\ & x \in \mathbb{Z}^n_+, \end{aligned}$$

where n and $m \in \mathbb{Z}_+$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. The feasible region of an IP is denoted as $P = \{x \in \mathbb{Z}^n_+ | Ax \leq b\}$ and its optimal solution is z^{*IP} along with x^{*IP}. Moreover, a bounded integer program is an IP with the additional constraints $x \leq u$ where $u \in \mathbb{R}^n$. For every IP, the corresponding linear relaxation (LR) problem is defined by:

$$\begin{aligned} \text{maximize } & z = c^T x \\ \text{subject to } & Ax \leq b \\ & x \in \mathbb{R}^n_+. \end{aligned}$$

The feasible region of the linear relaxation problem is a polyhedron denoted as $P^{LR} = \{x \in \mathbb{R}^n_+ | Ax \leq b\}$, and its optimal solution is z^{*LR} along with x^{*LR}.

Integer programming has been used to model and manage a wide array of real world problems. These applications are frequently identified in several industries. For instance,

IPs have optimized financial planning operations (Hamilton and Moses, 1973; Krokhmal et al., 2002; Pendharkar and Rodger, 2006; Singh et al., 2012), scheduling and allocation of resources (Subramanian et al., 1994; Toffolo et al., 2016; Vuthipadadon and Olafsson, 2007; Yin et al., 2017), facility location problems (Kose and Karabay, 2016; Noor-E-Alam et al., 2014), and production planning (Abraham and Rao, 2008; Ben-Arieh et al., 2009; Kashkoush et al., 2012).

Integer programs also play an important role when it comes to supply chain (Duong and Bui, 2018; Limpianchob, 2017; Salam et al., 2015), transportation (Albashabsheh and Heier Stamm, 2019; Delli and Sinha, 2019; Gifford et al., 2018; Sinha et al., 2016), and health care (Heier Stamm et al., 2017; Lee et al., 2003; Muggy and Heier Stamm, 2017; Stahl et al., 2005). Even complex systems such as power generation (Antunes et al., 2004; Carrion and Arroyo, 2006; Zhan and Zheng, 2018) have benefited from integer programming. Other usage of IPs also include compressed sensing matrices (Bailey et al., 2012) and their application in metrology (Ma, 2010), high resolution single pixel cameras (Ma, 2009), and magnetic resonance imaging (Lustig et al., 2007, 2008).

Unfortunately, IPs are NP-hard (Karp, 1972) and an exponential amount of time may be required to optimally solve this class of mathematical models. Integer programs are frequently solved by the branch and bound algorithm (Land and Doig, 1960), but other critical techniques have also been developed. For example, cutting planes (Balas and Zemel, 1978; Chvátal, 1973; Gomory, 1969; Hickman and Easton, 2015a,b; Vitor, 2015; Vitor and Easton, 2016, 2019), column generation (Ford and Fulkerson, 1958; Gilmore and Gomory, 1961, 1963; Lübbecke and Desrosiers, 2005; Nemhauser, 2012), and decomposition or partitioning algorithms (Benders, 1962; Conejo et al., 2006; Dantzig and Wolfe, 1960, 1961; Ergünes et al., 2017) are common methods to speed up the solution time of IPs.

The branch and bound algorithm has changed little since it was first created. Some improvements include various branching strategies that indicate which node should be evaluated next and strategies that use variables with certain properties to branch (see Section 2 for additional details). However, of the existing strategies, there is no indication on which is best as the performance of a strategy varies greatly between problem classes and even instances in the same class.

The motivation of this paper is derived from the attempt to find quality integer solutions quickly in a branching tree. The objective of this research is to create a new branching algorithm where the branching variables assume integer values that are relatively close to the linear relaxation solution of the corresponding parent's node.

This paper's primary contribution is the octanary branching algorithm, a new method that can help improve the solution time of IPs. This scheme's structure generates eight children per parent instead of two from the branch and bound algorithm. The dimension of the feasible linear relaxation space of four of the eight children nodes generated by the octanary branching algorithm is strictly less than the dimension of their parent's feasible linear relaxation space. Computational experiments demonstrate that the octanary branching algorithm identifies quality integer solutions earlier than the branch and bound algorithm. Consequently, the octanary branching algorithm should be implemented at the beginning of the branching tree, as a warm start solution, or as a diving scheme.

The remainder of the paper is organized as follows. Section 2 describes different existing branching techniques so the reader can understand how the proposed method advances the knowledge in integer programming. Section 3 presents the theoretical and algorithmic foundation of the octanary branching algorithm. Section 4 describes the implementation and

results of a computational study developed to test the effectiveness of the new algorithm. Section 5 concludes the paper and presents potential topics for future research.

2 Branching Techniques

Solving integer programming problems is a critical research topic in operations research. Numerous approaches have been discovered since the late 1950s. This section provides a small sample of the research dedicated to branching techniques. This includes the well-known branch and bound algorithm, different search strategies, selection of branching variables, and other advanced topics such as hyperplane and polyhedra branching. For further reference on the presented topics and other research on branching methods not discussed in this paper, the work of Nemhauser and Wolsey (1999), Linderoth and Savelsbergh (1999), Achterberg et al. (2005), Conforti et al. (2014), and Morrison et al. (2016) are suggested.

2.1 Branch and Bound Algorithm

The primary method to solve IPs is the branch and bound algorithm (BB), created by Land and Doig (1960). This technique is guaranteed to find an optimal solution to IPs, if one exists, in finite time. The branch and bound algorithm generates a potentially exponential tree structure of bounds and constraints where each node of the branching tree corresponds to a linear relaxation problem.

Given an IP, BB begins by storing the IP's linear relaxation in the root node T_1. An unfathomed node T_p is selected and its linear relaxation problem is solved to obtain z^{*T_p} and x^{*T_p}. If $x^{*T_p} \notin \mathbb{Z}_+^n$, then BB finds an $i \in \{1, ..., n\}$ such that $x_i^{*T_p} \notin \mathbb{Z}_+$. Two new nodes from T_p are generated and these nodes are referred to as T_p's children. Both children begin with T_p's linear relaxation problem. The first less than or equal to child, T_p^L, adds the constraint $x_i \leq \lfloor x_i^{*T_p} \rfloor$. The second greater than or equal to child, T_p^G, includes the constraint $x_i \geq \lfloor x_i^{*T_p} \rfloor + 1$. This type of branching is referred in this paper as standard branching.

The branch and bound algorithm stores the best known integer solution, z^{best} and x^{best}. If $z^{*LR} > z^{best}$ and $x^{*LR} \in \mathbb{Z}_+^n$, then z^{best} and x^{best} are replaced with z^{*LR} and x^{*LR}, respectively. The branch and bound algorithm continues until all nodes are fathomed. A node is fathomed when its linear relaxation problem is infeasible, $x^{*LR} \in \mathbb{Z}_+^n$, z^{*LR} is inferior to the objective function value of the best known integer solution, or two children nodes are created. When all nodes are fathomed, BB terminates and reports an optimal solution if one exists. If there is no solution, then BB reports that the IP is infeasible. One can see that the motivation for this paper is to improve upon the simplistic branching constraints of $x_i \leq \lfloor x_i^{*T_p} \rfloor$ and $x_i \geq \lfloor x_i^{*T_p} \rfloor + 1$.

The branching step of BB is indeterminate. During any iteration, there are many unfathomed nodes that need to be evaluated. Selecting which node to evaluate can have significant implications on the efficiency of the algorithm. Memory capacity becomes a large issue when selecting a search strategy as the size of the tree can grow exponentially. However, it is possible that the solution time will greatly increase when attempting to use a strategy that decreases memory usage. Consequently, various search strategies used alongside BB have been explored in order to decrease both the time to solve IPs and also

the amount of memory usage. Generally, search strategies fall into the categories of depth first, breadth first, or best bound.

2.2 Search Strategies

Depth first search is a method where BB evaluates a child node and keeps moving downward through the tree until all ancestors are fathomed. Once a fathomed node is found, then BB backtracks and repeats the process. Depth first strategies are typically given a direction to explore first, such as left or right. Depth first search is a memory efficient method of exploring the branching tree. Let the depth of any node be denoted as d with the root node having a depth of $d = 1$. When evaluating a node at depth d, there are at most $d + 1$ unevaluated nodes remaining in the tree. Because the number of unevaluated nodes is linear with the current depth of the node being evaluated, the amount of memory needed to store the tree is small. In addition, depth first strategies tend to locate integer solutions or infeasible nodes quickly resulting in many nodes being fathomed.

There is no guarantee, however, on the effectiveness of using a depth first strategy. It is possible to spend a great deal of time diving to find an integer solution that fathoms very few nodes in the tree. If an optimal solution does not appear as a descendant of a node, then substantial effort is wasted fathoming that portion of the tree. These issues are typically magnified when dealing with large problems.

Breadth first search is a strategy that evaluates all nodes at depth d prior to evaluating any nodes at depth $d + 1$. This alleviates the concern of whether or not the correct node was selected to explore that is associated with depth first search. The primary downfall of breadth first search is its inability to identify infeasible nodes or integer solutions. When evaluating a node at depth d, there are at most 2^d unevaluated nodes remaining in the tree. With an exponential relationship between the depth and number of nodes, memory issues rapidly become a limiting factor when attempting to solve IPs with breadth first search.

Many state-of-the-art commercial and open source mathematical programming solvers frequently use some form of best bound strategy. Best bound uses the objective function value as a means of determining which node of the tree to explore first. In this case, the node with the best z^{*LR} is evaluated first. The thought behind this strategy is that a better objective function value is the best candidate for enumeration. The logic follows that an optimal integer solution would be the child of a node with a good objective function value. Finding a good integer solution quickly can decrease the size of the current tree, which prevents the need to evaluate many new nodes and leads to an improved solution time.

In an attempt to find a good integer solution quickly, a common practice is to use a hybrid of depth first and best bound search known as random diving (IBM ILOG CPLEX Optimization Studio CPLEX User's Manual, 2016; Walker, 1960). In this strategy, BB uses the best child search for a set number of iterations. At preset intervals, the algorithm switches to depth first search until the path is fathomed. This method makes an attempt to quickly discover a superior integer solution.

There is no indication as to which is the best search strategy as the performance varies greatly between problem classes and even instances in the same class. While one strategy may be effective for a certain class of problems, there is no guarantee that it will work well for all problems.

2.3 Branching Variable Selection

Prior to branching, BB must select a variable such that $x_i^{*T_p} \notin \mathbb{Z}$. Empirical evidence shows that the choice of $x_i^{*T_p}$ is vital to the amount of effort spent solving an IP (Nemhauser and Wolsey, 1999). Often there exists a set of variables that when fixed at integer values, force all other variable to be integers. Because there is no robust method of determining such variables, often the priority of branching is user defined. Two common ways of selecting priorities are through degradation and penalties (Achterberg et al., 2005; Linderoth and Savelsbergh, 1999; Morrison et al., 2016).

Degradation attempts to estimate the change obtained in z^{*T_p} by forcing $x_i^{*T_p}$ to be integral. Suppose that $x_i^{*T_p}$ is noninteger. Define $f_i^{*T_p}$ such that $x_i^{*T_p} = \lfloor x_i^{*T_p} \rfloor + f_i^{*T_p}$. By branching on $x_i^{*T_p}$, one would expect that the objective function value would decrease by $D_i^{-*T_p} = p_i^{-*T_p} f_i^{*T_p}$ for the left child and by $D_i^{+*T_p} = p_i^{+*T_p}(1 - f_i^{*T_p})$ for the right child. In such a case, the coefficients $p_i^{-*T_p}$ and $p_i^{+*T_p}$ can be specified or estimated in several different ways.

Penalties involve more taxing calculations to determine the coefficients $p_i^{-*T_p}$ and $p_i^{+*T_p}$ in order to generate a lower bound on the change in z^{*T_p}. Penalties have been extensively used in early commercial and open source mathematical programming solvers. However, empirical methods have shown that the cost of finding the penalties exceed the benefit of the information given (Nemhauser and Wolsey, 1999).

Given $D_i^{-*T_p}$ and $D_i^{+*T_p}$, it is common to select $x_i^{*T_p}$ such that the minimum of $D_i^{-*T_p}$ and $D_i^{+*T_p}$ is maximized. The rationale is that maximizing the minimum decrease in the objective function value will result in obtaining the optimal z^* value more quickly. Another common approach is to select $x_i^{*T_p}$ such that the maximum of $D_i^{-*T_p}$ and $D_i^{+*T_p}$ is maximized. The idea in doing this is that the branch with the lower objective function value will quickly be fathomed by dominance.

2.4 Nonsimplistic Branching Techniques

Research has also investigated other advanced nonsimplistic branching strategies. Common methods include the so-called hyperplane branching and branching polyhedra. Both techniques typically involve branching on more than one single variable at each node of the branching tree.

Hyperplane branching attempts to increase the efficiency of the enumeration tree by branching on multiple variables. This technique follows a similar structure as BB. That is, one child includes the constraint $\sum_{i=1}^{n} \alpha_i x_i \leq \beta$ and the other child has the constraint $\sum_{i=1}^{n} \alpha_i x_i \geq \beta + 1$ where $\alpha \in \mathbb{Z}^n$ and $\beta \in \mathbb{Z}$. Typically, hyperplane branching requires α to have at least two nonzero coefficients, but standard branching is a type of hyperplane branching with exactly one nonzero coefficient.

While hyperplane branching has demonstrated potential good results (Jörnsten and Värbrand, 1991; Mehrotra and Li, 2011; Ryan and Foster, 1981), the method is limited by several factors. Unlike BB, which only stores the index of the branching variable and its lower or upper bound, hyperplane branching stores the coefficient of each variable in each node. Consequently, hyperplane branching linearly increases the size of the simplex basis by the depth of the node. This typically results in an increase in the time to solve each linear relaxation problem and escalates the memory requirements at each node.

One common type of hyperplane branching is referred to as branching on sets. Given $S \subseteq N$, two children are created. One child adds the hyperplane $\sum_{i \in S} x_i = 0$ while the other includes $\sum_{i \in S} x_i \geq 1$. A variation of this process is proposed by Easton et al. (2003) where the idea is extended to numerous children for the root node. In this case, binary integer programs are considered and some of the children have constraints $\sum_{i \in S} x_i = |S|$, $\sum_{i \in S} x_i = |S| - 1$, $\sum_{i \in S} x_i = |S| - 2$, and $\sum_{i \in S} x_i \leq |S| - 3$.

An alternate branching scheme is referred to as branching polyhedra. In this case, each child solves an optimization problem subject to a polyhedron intersected with the parent's linear relaxation space. There is no restriction that requires the same family of branching polyhedra for each parent. For simplicity and generality of code, this paper assumes that any polyhedral branching step creates k children by intersecting the parent node's feasible linear relaxation space with the same family of polyhedra.

Easton and Lee (2012) showed that branching on polyhedra of any bounded IP terminates in finite time as long as the following three conditions hold: (1) any noninteger linear relaxation solution at the parent's node must not be a feasible solution to the linear relaxation problem of any of its k children; (2) every IP's integer feasible solution must be in one of the k branching polyhedra; (3) every extreme point for each of the branching polyhedra must be integer. Furthermore, Easton and Lee developed the quaternary hyperplane branching algorithm, which branches on four polyhedra. This method utilizes hyperplane branching constraints and internal cutting planes to generate an efficient quaternary branching scheme.

Observe that the research developed for this paper is a branching polyhedra technique, and is an advancement to the work of Easton and Lee. While the quaternary hyperplane branching algorithm from Easton and Lee generates four children per parent's node, the octanary branching algorithm creates eight new nodes. Four of these nodes are created by equality constraints while the other four by inequalities. The following section describes the theoretical and algorithmic foundations of this new advancement.

3 Octanary Branching Algorithm

This section describes the octanary branching algorithm (OBA) and provides an example to demonstrate its implementation. Theoretical results are presented to show that OBA correctly solves any bounded IP in finite time. Additionally, some theoretical benefits of OBA are provided. Preliminary results to this paper are also found in one of the author's thesis (Bailey, 2012).

Given an IP, OBA assigns the IP's linear relaxation to T_1. An unfathomed node, T_p, is selected and solved to obtain z^{T_p} and x^{T_p}. The primary difference between OBA and BB is the branching step of the algorithm. Denote the node being evaluated as the parent node T_p. If $x^{*T_p} \notin \mathbb{Z}_+^n$ and $z^{*T_p} > z^{best}$, then OBA finds an i and $j \in \{1, ..., n\}$ such that $x_i^{*T_p}$, $x_j^{*T_p} \notin \mathbb{Z}_+$, and $i \neq j$. If no such j exists, then OBA selects any $j \neq i \in \{1, ..., n\}$. If the IP has only one variable, then $i = j = 1$.

Eight new nodes are generated and each node becomes a child of T_p. Each child's feasible space takes T_p's feasible region, denoted as P_p, and intersects one of the eight polyhedra. The eight polyhedra are denoted as $P_p^{ab_k}$ where a or b specify less than or equal to, or greater than or equal to constraints for x_i and x_j, and k specifies if the child has equality or inequality constraints. That is, each parent has eight child nodes, which are linear relaxation problems of the form:

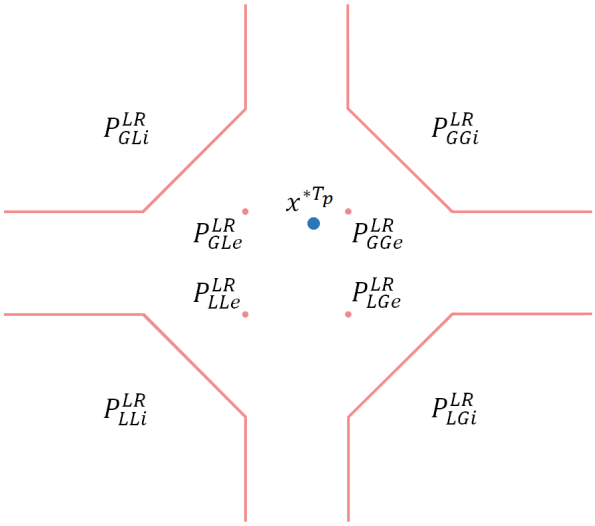
$$\begin{aligned} &\text{maximize } z = c^T x \\ &\text{subject to} \quad P_p \cap P_p^{ab_k}. \end{aligned}$$

The eight branching polyhedra are formally defined in equations (1)-(8) where $\beta_i = \lfloor x_i^{*T_p} \rfloor$ and $\beta_j = \lfloor x_j^{*T_p} \rfloor$.

$$P_{LL_e}^{LR} = \{x \in \mathbb{R}_+^n | x_i = \beta_i, x_j = \beta_j\} \tag{1}$$
$$P_{GL_e}^{LR} = \{x \in \mathbb{R}_+^n | x_i = \beta_i + 1, x_j = \beta_j\} \tag{2}$$
$$P_{LG_e}^{LR} = \{x \in \mathbb{R}_+^n | x_i = \beta_i, x_j = \beta_j + 1\} \tag{3}$$
$$P_{GG_e}^{LR} = \{x \in \mathbb{R}_+^n | x_i = \beta_i + 1, x_j = \beta_j + 1\} \tag{4}$$
$$P_{LL_i}^{LR} = \{x \in \mathbb{R}_+^n | x_i \leq \beta_i, x_j \leq \beta_j, x_i + x_j \leq \beta_i + \beta_j - 1\} \tag{5}$$
$$P_{GL_i}^{LR} = \{x \in \mathbb{R}_+^n | x_i \geq \beta_i + 1, x_j \leq \beta_j, -x_i + x_j \leq -\beta_i + \beta_j - 2\} \tag{6}$$
$$P_{LG_i}^{LR} = \{x \in \mathbb{R}_+^n | x_i \leq \beta_i, x_j \geq \beta_j + 1, x_i - x_j \leq \beta_i - \beta_j - 2\} \tag{7}$$
$$P_{GG_i}^{LR} = \{x \in \mathbb{R}_+^n | x_i \geq \beta_i + 1, x_j \geq \beta_j + 1, -x_i - x_j \leq -\beta_i - \beta_j - 3\} \tag{8}$$

OBA continues solving each node's linear relaxation problem until all nodes are fathomed. OBA follows the same fathoming rules as BB. However, a branching node is fathomed once all eight children are created, instead of two. When all nodes are fathomed, OBA terminates and reports an optimal solution if one exists. If there is no solution, then OBA reports that the IP is infeasible. Algorithm 1 formally presents OBA.

Figure 1: OBA's branching structure



The motivating factor behind this polyhedra branching structure is that the solution to the four children closest to the parent's linear relaxation solution have two variables set to integer values. The outer four children, which only create bounds for the variables, are pushed further from the parent's optimal solution and are likely to have worse objective function values. Figure 1 presents a graphic depiction of this polyhedra branching structure in $\mathbb{R}^2$ without the intersection of the parent node's feasible region.

Algorithm 1 Octanary Branching Algorithm (OBA)

Let $T \leftarrow \{T_1\}$ be the enumeration tree where T_1 is the IP's linear relaxation;
$z^{best} \leftarrow -\infty$ and $q \leftarrow 1$;
while there exists an unfathomed node in T **do**
  Let T_p be any unfathomed node in T;
  Solve T_p;
  if T_p is infeasible **then**
    $T_p \leftarrow$ fathomed;
  end if
  if $x^{*T_p} \in \mathbb{Z}^n$ **then**
    $T_p \leftarrow$ fathomed;
    if $z^{*T_p} > z^{best}$ **then**
      $z^{best} \leftarrow z^{*T_p}$;
      $x^{best} \leftarrow x^{*T_p}$;
    end if
  end if
  if $z^{*T_p} \leq z^{best}$ **then**
    $T_p \leftarrow$ fathomed;
  end if
  if T_p is unfathomed **then**
    Select a distinct i and $j \in \{1, ..., n\}$ such that $x_i^{*T_p}$ and $x_j^{*T_p} \notin \mathbb{Z}$;
    if there does not exist such a j **then**
      if $\{1, ..., n\} \setminus \{i\} \neq \emptyset$ **then**
        Let $j \in \{1, ..., n\} \setminus \{i\}$;
      end if
    else
      $j \leftarrow i$;
    end if
    Create eight children of T_p, nodes T_{q+1} to T_{q+8}, such that each of these linear relaxation problems are maximize $z = c^T x$ subject to $P_p \cap P_p^{ab_k}$ for all a, b, and k;
    $T_p \leftarrow$ fathomed;
    $q \leftarrow q + 8$;
  end if
end while
if $z^{best} = -\infty$ **then**
  return IP is infeasible;
else
  return z^{best} and x^{best}
end if

Prior to providing the main theoretical contributions of OBA, Example 1 demonstrates its implementation. The example has no priority for branching variables, and depth first left search is chosen to evaluate the tree.

Example 1: *Consider the following IP:*

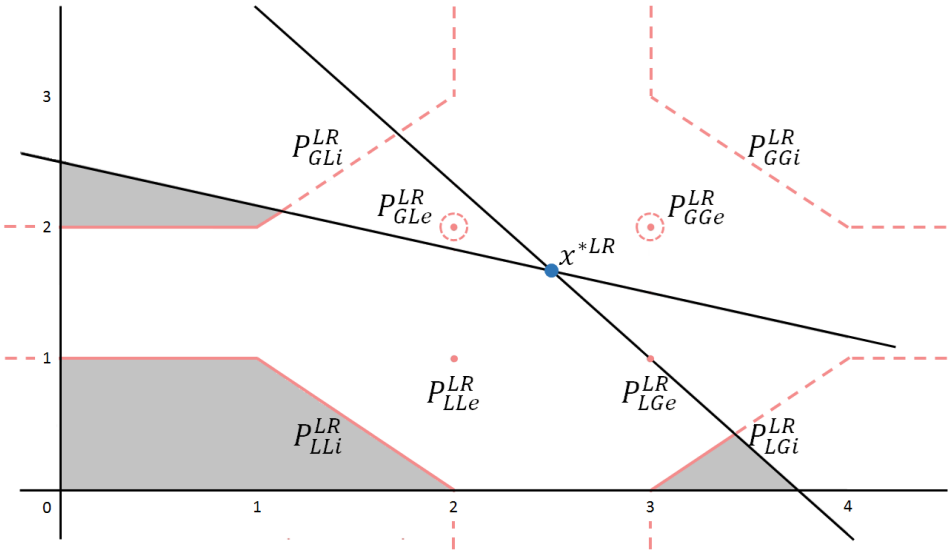
$$\begin{aligned} \text{maximize } z = & \ 5x_1 + 4x_2 \\ \text{subject to} \quad & \ 2x_1 + 6x_2 \leq 15 \\ & \ 4x_1 + 3x_2 \leq 15 \\ & \ x_1 \ , \quad x_2 \in \mathbb{Z}_+. \end{aligned}$$

The first iteration of OBA finds the solution to the root node, T_1, which is $z^{*T_1} = \frac{115}{6}$ and $x^{*T_1} = \left(\frac{5}{2}, \frac{5}{3}\right)$ with $\beta_1 = \left\lfloor \frac{5}{2} \right\rfloor = 2$ and $\beta_2 = \left\lfloor \frac{5}{3} \right\rfloor = 1$. The resulting polyhedra branching occurs with the creation of eight child nodes from T_1:

$T_2 = T_1^{LL_e}$, which is T_1 with the constraints $x_1 = 2$ and $x_2 = 1$;
$T_3 = T_1^{GL_e}$, which is T_1 with the constraints $x_1 = 3$ and $x_2 = 1$;
$T_4 = T_1^{LG_e}$, which is T_1 with the constraints $x_1 = 2$ and $x_2 = 2$;
$T_5 = T_1^{GG_e}$, which is T_1 with the constraints $x_1 = 3$ and $x_2 = 2$;
$T_6 = T_1^{LL_i}$, which is T_1 with the constraints $x_1 \leq 2$, $x_2 \leq 1$ and $x_1 + x_2 \leq 2$;
$T_7 = T_1^{GL_i}$, which is T_1 with the constraints $x_1 \geq 3$, $x_2 \leq 1$, and $-x_1 + x_2 \leq -3$;
$T_8 = T_1^{LG_i}$, which is T_1 with the constraints $x_1 \leq 2$, $x_2 \geq 2$, and $x_1 - x_2 \leq -1$;
$T_9 = T_1^{GG_i}$, which is T_1 with the constraints $x_1 \geq 3$, $x_2 \geq 2$ and $-x_1 - x_2 \leq -6$.

The partitioning scheme for T_1 is presented in Figure 2. The figure depicts all eight $P_p^{ab_k}$ structures intersected with P_p. This results in three infeasible spaces. For clarity, the three infeasible spaces are dashed in the figure. Even though these spaces are labeled, they do not contain any feasible solutions.

Figure 2: OBA's branching structure for Example 1



Since depth first left search is used, the first node evaluated is T_2. Evaluating this node yields $z^{*T_2} = 14$ and $x^{*T_2} = (2, 1)$. Because $z^{*T_2} = 14 > -\infty$ and $x^{*T_2} \in \mathbb{Z}^2_+$, T_2 is fathomed, z^{best} is set to $z^{*T_2} = 14$, and x^{best} is set to $x^{*T_2} = (2, 1)$. Evaluating T_3 results

in $z^{*T_3} = 19$ and $x^{*T_3} = (3, 1)$. Since $z^{*T_3} > z^{best}$ and $x^{*T_3} \in \mathbb{Z}^2_+$, T_3 is fathomed, z^{best} is set to 19, and x^{best} becomes $(3, 1)$.

Nodes T_4 and T_5 are infeasible. Solving T_6 results in $z^{*T_6} = 10$ and $x^{*T_6} = (2, 0)$, and this node is fathomed. Node T_7 has $z^{*T_7} = \frac{132}{7} \leq 19$, and T_7 is fathomed. The solution to T_8 yields $z^{*T_8} = \frac{113}{8} \leq 19$, and T_8 is also fathomed. Finally, node T_9 is fathomed because it is infeasible. Since all nodes have been fathomed, OBA terminates and reports an optimal solution $z^* = 19$ and $x^* = (3, 1)$. The full enumeration tree for this problem is presented in Figure 3.

The following two lemmas help show that OBA solves a bounded IP in finite time. The first lemma proves that the branching polyhedra of the children generated by OBA have integer extreme points.

Lemma 1: *For any β_i and $\beta_j \in \mathbb{Z}_+$, the extreme points of $P^{LR}_{LL_e}$, $P^{LR}_{GL_e}$, $P^{LR}_{LG_e}$, $P^{LR}_{GG_e}$, $P^{LR}_{LL_i}$, $P^{LR}_{GL_i}$, $P^{LR}_{LG_i}$, and $P^{LR}_{GG_i}$ are integer if $i \neq j$.*

Proof: For any β_i and $\beta_j \in \mathbb{Z}_+$, consider OBA's eight branching polyhedra $P^{LR}_{LL_e}$, $P^{LR}_{GL_e}$, $P^{LR}_{LG_e}$, $P^{LR}_{GG_e}$, $P^{LR}_{LL_i}$, $P^{LR}_{GL_i}$, $P^{LR}_{LG_i}$, and $P^{LR}_{GG_i}$. Exactly two equations create $P^{LR}_{ab_e}$ and three equations create $P^{LR}_{ab_i}$ where a and b specify less than or equal to or greater than or equal to constraints. The structure of all eight embedded matrices, when changed to equality constraints, take the form:

$$\begin{bmatrix} 1\,0 \\ 0\,1 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} 1\,0\,1\,0\,0 \\ 0\,1\,0\,1\,0 \\ 1\,1\,0\,0\,1 \end{bmatrix}$$

for $P^{LR}_{ab_e}$ and $P^{LR}_{ab_i}$, respectively. This structure changes based upon a and b, and the change is limited to multiplying some rows and/or columns by -1. These embedded matrices are clearly totally unimodular (TUM). Because multiplying a column or row of a TUM matrix by -1 preserves the TUM property, all $P^{LR}_{ab_e}$ and $P^{LR}_{ab_i}$ are TUM matrices. Since β_i and $\beta_j \in \mathbb{Z}_+$ and all matrices are TUM, then every extreme point is integer. □

The next lemma proves that every feasible integer solution in T_p is contained in exactly one of its children's nodes. This is accomplished by splitting the feasible region into four quadrants.

Lemma 2: *If T_p is any node in OBA's branching tree, then $\mathbb{Z}^n \cap P_p = \mathbb{Z}^n \cap (P^{LL_e}_p \cup P^{GL_e}_p \cup P^{LG_e}_p \cup P^{GG_e}_p \cup P^{LL_i}_p \cup P^{GL_i}_p \cup P^{LG_i}_p \cup P^{GG_i}_p)$.*

Proof: Let T_p be any node in OBA's branching tree. Because each of the eight branching polyhedra is a subset of P_p, $\mathbb{Z}^n \cap (P^{LL_e}_p \cup P^{GL_e}_p \cup P^{LG_e}_p \cup P^{GG_e}_p \cup P^{LL_i}_p \cup P^{GL_i}_p \cup P^{LG_i}_p \cup P^{GG_i}_p) \subseteq \mathbb{Z}^n \cap P_p$.

To show the opposite direction and for contradiction, assume there exists an $x' \in \mathbb{Z}^n \in P_p$ such that $x' \notin (P^{LL_e}_p \cup P^{GL_e}_p \cup P^{LG_e}_p \cup P^{GG_e}_p \cup P^{LL_i}_p \cup P^{GL_i}_p \cup P^{LG_i}_p \cup P^{GG_i}_p)$ where $\beta_i = \lfloor x^{*T_p}_i \rfloor$ and $\beta_j = \lfloor x^{*T_p}_j \rfloor$.

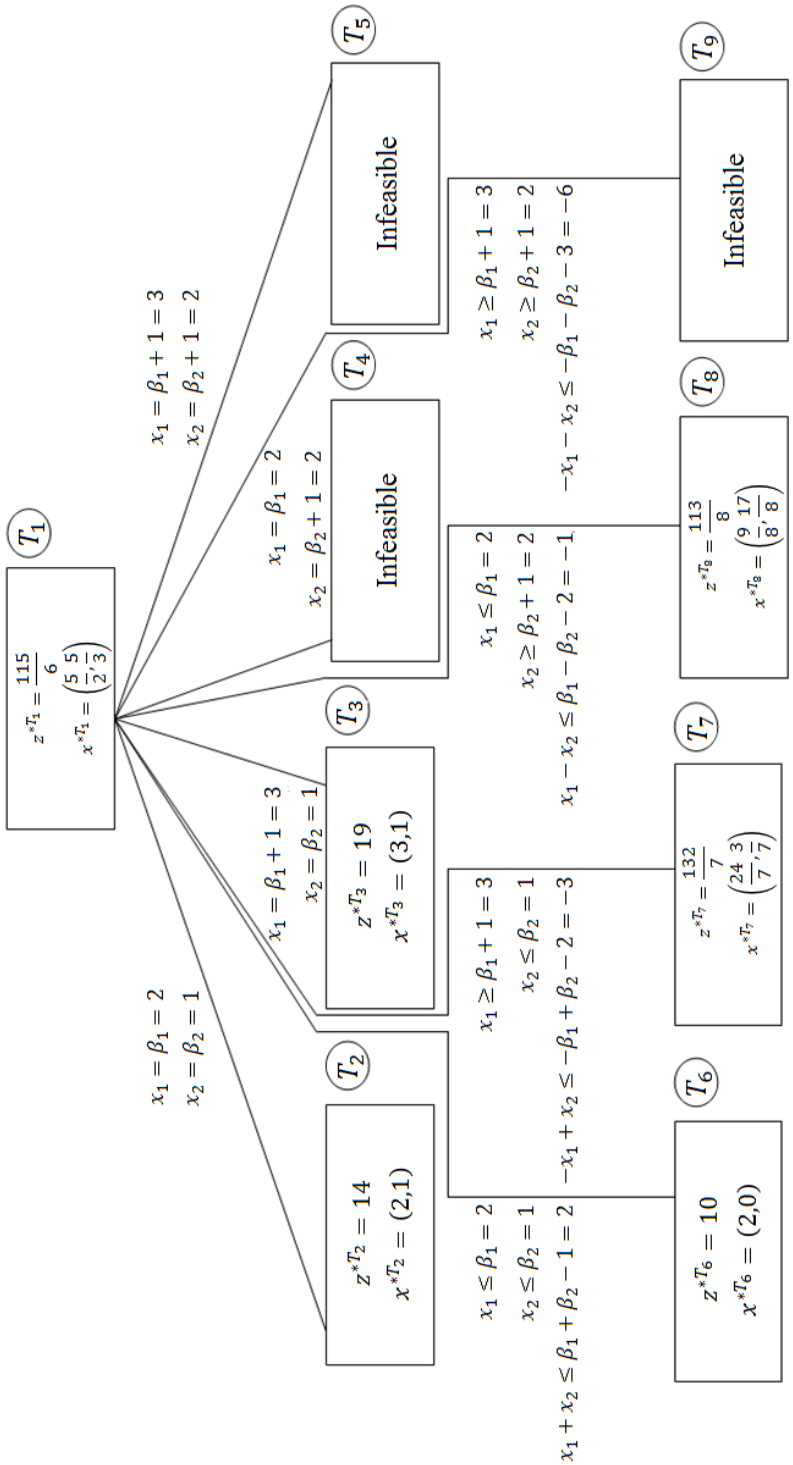


Figure 3: OBA's enumeration tree for Example 1

Clearly, $x'_i \leq \beta_i$ or $x'_i \geq \beta_i + 1$, and $x'_j \leq \beta_j$ or $x'_j \geq \beta_j + 1$. Due to symmetry, it is sufficient to only consider the case $x'_i \leq \beta_i$ and $x'_j \leq \beta_j$. If $x'_i = \beta_i$ and $x'_j = \beta_j$, then x' satisfies $P_p^{LL_e}$, a contradiction to $x' \notin P_p^{LL_e}$. Thus, $x'_i + x'_j \leq \beta_i + \beta_j - 1$. However, this solution satisfies all three constraints of $P_p^{LL_i}$, which is also contradiction. Thus, every feasible integer solution in T_p is in one of its eight children. □

The following theorem shows that OBA correctly solves any bounded IP and terminates in finite time. The proof utilizes strong induction.

Theorem 1: *OBA correctly solves any bounded IP within a finite number of steps.*

Proof: Consider the following bounded IP with $n = 1$ as the base case:

$$\begin{aligned} &\text{maximize } z = c_1 x_1 \\ &\text{subject to} \quad Ax_1 \leq b \\ &\qquad\qquad\quad x_1 \leq u_1 \\ &\qquad\qquad\quad x_1 \in \mathbb{Z}_+. \end{aligned}$$

Since the above IP has only one variable, this problem can be reduced to:

$$\begin{aligned} &\text{maximize } z = c_1 x_1 \\ &\text{subject to} \quad l'_1 \leq x_1 \leq u'_1 \\ &\qquad\qquad\quad x_1 \in \mathbb{Z}_+. \end{aligned}$$

The solution to this IP is trivial and is one of three cases: (1) infeasible if $\lfloor u'_1 \rfloor < l'_1$; (2) $x_1 = \lfloor u'_1 \rfloor$ if $c_1 \geq 0$; (3) $x_1 = \lceil l'_1 \rceil$ if $c_1 < 0$. Observe that case (3) can be transformed into case (2) by substituting $x'_1 = -x_1$. Consequently, assume $c_1 \geq 0$.

Consider case (1) where $\lfloor u'_1 \rfloor < l'_1$. If $u'_1 < l'_1$, then solving OBA's T_1 node is infeasible. If not, then the solution to OBA's T_1 node is $z = c_1 u'_1$ and $x_1 = u'_1 \notin \mathbb{Z}_+$. Since the IP has only one variable, OBA branches on x_1 twice. Thus, OBA creates the following eight children:

$T_1^{LL_e} = T_2$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 = \lfloor u'_1 \rfloor$, $x_1 = \lfloor u'_1 \rfloor, x_1 \geq 0$;

$T_1^{LG_e} = T_3$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 = \lfloor u'_1 \rfloor$, $x_1 = \lfloor u'_1 \rfloor + 1, x_1 \geq 0$;

$T_1^{GL_e} = T_4$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 = \lfloor u'_1 \rfloor + 1$, $x_1 = \lfloor u'_1 \rfloor, x_1 \geq 0$;

$T_1^{GG_e} = T_5$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 = \lfloor u'_1 \rfloor + 1$, $x_1 = \lfloor u'_1 \rfloor + 1, x_1 \geq 0$;

$T_1^{LL_i} = T_6$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 \leq \lfloor u'_1 \rfloor$, $x_1 \leq \lfloor u'_1 \rfloor, 2x_1 \leq 2\lfloor u'_1 \rfloor - 1, x_1 \geq 0$;

$T_1^{LG_i} = T_7$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 \leq \lfloor u'_1 \rfloor$, $x_1 \geq \lfloor u'_1 \rfloor + 1, 0 \leq -2, x_1 \geq 0$;

$T_1^{GL_i} = T_8$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 \geq \lfloor u'_1 \rfloor + 1$, $x_1 \leq \lfloor u'_1 \rfloor, 0 \leq -2, x_1 \geq 0$;

$T_1^{GG_i} = T_9$, which is maximize $z = c_1 x_1$ subject to $l'_1 \leq x_1 \leq u'_1, x_1 \geq \lfloor u'_1 \rfloor + 1$, $x_2 \leq \lfloor u'_1 \rfloor + 1, -2x_1 \leq -2\lfloor u'_1 \rfloor - 3, x_1 \geq 0$.

Since $l'_1 \geq \lfloor u'_1 \rfloor$, T_2, T_3, T_4, T_6, T_7 and T_8 are infeasible. Because $\lfloor u'_1 \rfloor + 1 > u'_1$, T_5 and T_9 are infeasible. Thus, OBA correctly identifies that the IP is infeasible for this case.

If $\lfloor u'_1 \rfloor \geq l'_1$, then T_1's solution is $z^{*T_1} = c_1 u'_1$ and $x_1^{*T_1} = u'_1$. If $u'_1 \in \mathbb{Z}_+$, then OBA terminates and correctly identifies an optimal solution. If not, OBA branches on x_1 twice creating eight children that are identical to the children from the previous case.

Because $l'_1 \leq \lfloor u'_1 \rfloor$, the solution to T_2 is $z^{*T_2} = c_1 \lfloor u'_1 \rfloor$ and $x_1^{*T_2} = \lfloor u'_1 \rfloor$. Clearly, T_2's solution is integer, this node is fathomed, and z^{best} and x^{best} are updated. Since $\lfloor u'_1 \rfloor + 1 > u'_1$, T_3, T_4, T_5, T_7, T_8, and T_9 are infeasible. Solving T_6 results in either T_6 being infeasible or an optimal solution $z^{*T_6} = c_1(\lfloor u'_1 \rfloor - \frac{1}{2}) < z^{best}$ and $x_1^{*T_6} = \lfloor u'_1 \rfloor - \frac{1}{2}$. In either case, T_6 is fathomed, OBA terminates, and correctly reports the optimal solution.

In the special case where $c_1 = 0$, solving T_1 may result in an infinite number of values for $x_1^{*T_1}$. If the solution to T_1 is integer, then T_1 is fathomed and OBA terminates. If not and the IP has an integer solution, then the solution to T_2, T_5, or both are integer, which also correctly terminates OBA. If T_2 and T_5 are both infeasible, then no integer solution exists and all the branched nodes are infeasible and fathomed. Thus, in every case, OBA correctly solves all bounded IPs with $n = 1$, which concludes the base case.

By strong induction, assume OBA correctly solves a bounded IP with $n - 1$ or fewer variables where $n \geq 2$. Now consider a bounded IP of the form maximize $z = c^T x$ subject to $Ax \leq b, 0 \leq x \leq u, x \in \mathbb{Z}_+^n$. OBA solves T_1 and if its linear relaxation solution is either integer or infeasible, then OBA terminates and correctly reports the optimal solution or that the IP is infeasible. If not, OBA selects an i and $j \in \{1, ..., n\}$ with $i \neq j$ such that $x_i^{*T_1} \notin \mathbb{Z}_+$, and OBA creates the following eight children:

$$
\begin{aligned}
T_1^{LL_e} = T_2,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i = \lfloor x_i^{*T_1} \rfloor,\\
&\quad x_j = \lfloor x_j^{*T_1} \rfloor, 0 \leq x \leq u;\\
T_1^{LG_e} = T_3,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i = \lfloor x_i^{*T_1} \rfloor,\\
&\quad x_j = \lfloor x_j^{*T_1} \rfloor + 1, 0 \leq x \leq u;\\
T_1^{GL_e} = T_4,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i = \lfloor x_i^{*T_1} \rfloor + 1,\\
&\quad x_j = \lfloor x_j^{*T_1} \rfloor, 0 \leq x \leq u;\\
T_1^{GG_e} = T_5,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i = \lfloor x_i^{*T_1} \rfloor + 1,\\
&\quad x_j = \lfloor x_j^{*T_1} \rfloor + 1, 0 \leq x \leq u;\\
T_1^{LL_i} = T_6,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i \leq \lfloor x_i^{*T_1} \rfloor,\\
&\quad x_j \leq \lfloor x_j^{*T_1} \rfloor, x_i + x_j \leq \lfloor x_i^{*T_1} \rfloor + \lfloor x_j^{*T_1} \rfloor - 1, 0 \leq x \leq u;\\
T_1^{LG_i} = T_7,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i \leq \lfloor x_i^{*T_1} \rfloor,\\
&\quad x_j \geq \lfloor x_j^{*T_1} \rfloor + 1, x_i - x_j \leq \lfloor x_i^{*T_1} \rfloor - \lfloor x_j^{*T_1} \rfloor - 2, 0 \leq x \leq u;\\
T_1^{GL_i} = T_8,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i \geq \lfloor x_i^{*T_1} \rfloor + 1,\\
&\quad x_j \leq \lfloor x_j^{*T_1} \rfloor, -x_i + x_j \leq -\lfloor x_i^{*T_1} \rfloor + \lfloor x_j^{*T_1} \rfloor - 2, 0 \leq x \leq u;\\
T_1^{GG_i} = T_9,&\ \text{which is maximize } z = c^T x \text{ subject to } Ax \leq b, x_i \geq \lfloor x_i^{*T_1} \rfloor + 1,\\
&\quad x_j \geq \lfloor x_j^{*T_1} \rfloor + 1, -x_i - x_j \leq -\lfloor x_i^{*T_1} \rfloor - \lfloor x_j^{*T_1} \rfloor - 3, 0 \leq x \leq u.
\end{aligned}
$$

Nodes T_2, T_3, T_4, and T_5 force x_i to a fixed integer value. Through substitution, each of these nodes become linear relaxation problems with $n - 1$ or fewer variables.

Nodes T_6, T_7, T_8, and T_9 change either the upper or lower bound of both x_i and x_j. If either of these changes force x_i or x_j to a fixed integer value, then that particular variable can be removed from the IP through substitution. Thus, the node's linear relaxation problem has $n - 1$ or fewer variables. After branching on x_i a total of $u_i + 1$ times, the upper and

lower bound of x_i are identical or the node is infeasible. By the pigeon hole principle, some variable is forced to a particular value. When this occurs, the node becomes a linear relaxation problem with at most $n-1$ variables.

Every node in OBA's tree eventually becomes the linear relaxation problem of an IP with $n-1$ or fewer variables. Applying OBA to each of these nodes correctly solves the respective IP, by the induction assumption. OBA combines each of these results in the larger tree structure and reports the correct solution or that the problem is infeasible. □

Since OBA solves any bounded IP in finite time, the question remains whether or not OBA is more effective than traditional BB. In some situations OBA should be less efficient than traditional BB. For instance, if an IP has only binary decision variables, then four of OBA's children are redundant in the branching tree. Another issue occurs when a parent's linear relaxation problem is almost infeasible or has a solution near z^{best}. In such a scenario, any type of branching should create children that are immediately fathomed. Since OBA generates eight children per parent and BB creates only two children, OBA may spend four times more effort on infeasible or pointless nodes.

The primary theoretical benefit of OBA is an immediate reduction in dimension by branching. In standard branching, the dimension of the feasible linear relaxation space of each child is frequently identical to the dimension of the feasible linear relaxation space of its parent. In contrast, the dimension of the feasible linear relaxation space of each of OBA's equality children (four of the eight children) is strictly less than the dimension of its parent's feasible linear relaxation space. This dimension reduction is formally shown in Theorem 2.

Theorem 2: *Given a bounded IP, if OBA creates children at T_p, then the dimension of the feasible linear relaxation space of $T_p^{LL_e}$, $T_p^{GL_e}$, $T_p^{LG_e}$, and $T_p^{GG_e}$ is strictly less than the dimension of T_p's feasible linear relaxation space.*

Proof: Given a bounded IP, consider an iteration of OBA. Assume node T_p is branched upon creating $T_p^{LL_e}$, $T_p^{GL_e}$, $T_p^{LG_e}$, and $T_p^{GG_e}$. Since T_p is branched, there exists an $x_i^{*T_p} \notin \mathbb{Z}_+$. Thus, $P_p^{LL_e}$, $P_p^{GL_e}$, $P_p^{LG_e}$, and $P_p^{GG_e}$ all require either $x_i = \lfloor x_i^{*T_p} \rfloor$ or $x_i = \lfloor x_i^{*T_p} \rfloor + 1$. The four children are symmetric, so it is sufficient to only consider $P_p^{LL_e}$.

Assume $P_p^{LL_e} \neq \emptyset$. Consider the vector $v = x^{*T_p^{LL_e}} - x^{*T_p}$. Clearly, every feasible solution of $T_p^{LL_e}$'s linear relaxation problem satisfies $x_i = \lfloor x_i^{*T_p} \rfloor$. Thus, the vector v is not a feasible vector in $P_p^{LL_e}$. However, the vector v is contained in T_p's feasible linear relaxation space. Therefore, $\dim(P_p^{LL_e}) \leq \dim(P_p) - 1$.

If $P_p^{LL_e} = \emptyset$, then $\dim(P_p^{LL_e}) = -1$. Since there exists a feasible solution to T_p, $\dim(P_p) \geq 0$, and the result follows. □

Since OBA reduces the dimension of the linear relaxation space, one would expect OBA to identify infeasible nodes or quality integer solutions earlier in the branching tree than BB. The next section describes a computational study that validates this claim.

4 Computational Study

The computational study was performed on an Intel® Core^{TM} i7-6700 3.4GHz processor with 32 GB of RAM. The goal is to test and compare the quality of OBA's branching structure

to CPLEX's, a mathematical programming solver (IBM ILOG CPLEX Optimization Studio, 2016), branching algorithm under the same computational conditions. In this study, OBA was implemented in C++ using Microsoft Visual Studio with CPLEX 12.7.

4.1 Implementation and Instances

Because CPLEX has been developed and improved over decades, OBA cannot match the efficiency of the data retrieval and storage. As a result, only the number of nodes evaluated is considered in this computational study. This is reasonable as both OBA's and CPLEX's branching steps trivially take $O(n)$ time and constant space, and $O(d)$ to load constraints where d is the depth of the node being evaluated.

Computational experiments evaluated OBA for the first $500,000$ nodes. When comparing to CPLEX's branching algorithm, three results were analyzed. The first result compares the quality of the first integer solution found by both OBA and CPLEX. That is, does OBA find an integer solution with fewer nodes and a better objective function value than CPLEX? The second result determines the number of nodes required by CPLEX to match or surpass the objective function value of the first integer solution found by OBA. Given the best integer solution obtained by OBA in its first $500,000$ nodes, the third result obtains the number of nodes required by CPLEX to match or surpass the quality of this particular solution.

To guarantee both OBA and CPLEX run under the same computational conditions, many parameters were modified in CPLEX. The traditional branch and cut searching strategy was selected for CPLEX and all cuts were disabled (clique cuts, cover cuts, disjunctive cuts, flow cover cuts, flow path cuts, Gomory fractional cuts, generalized upper bound cover cuts, implied bound cuts, lift-and-project cuts, multi-commodity flow cuts, mixed integer rounding cuts, and zero-half cuts). In addition, heuristics were not applied. Observe that this approximates CPLEX's branch and cut algorithm to traditional BB.

Presolve during preprocessing and node presolve were also disabled along with row and column reductions, coefficient reductions, dependent row reductions, symmetry breaking reductions, and problem matrix scaling. For both OBA and CPLEX, depth first left was determined as the node search strategy and branching direction. In addition, CPLEX's traditional dive strategy was selected, statistics about the mixed integer program (MIP) kappa were not collected, no priority order was given for the MIP optimization, and no probing was performed on variables.

Other important parameters were left at default such as the MIP emphasis (balance optimality and feasibility), parallel optimization mode (CPLEX decides whether to invoke deterministic or opportunistic search), and branching variable selection (CPLEX chooses the variable to branch). Computational experiments within CPLEX also stored node files in the hard drive instead of RAM to avoid running out of memory. Parameters not mentioned in this paper were either left at default settings or did not impact CPLEX's branch and cut algorithm.

This computational study solved 270 benchmark multiple knapsack instances from the OR-Library Beasley (1990). Problems sets are named as *mknapcb1*, *mknapcb2*, ..., *mknapcb9* and each set has 30 instances with 100, 250, and 500 variables, 5, 10, and 30 constraints. These instances were created by Chu and Beasley Chu and Beasley (1998) and take the form of maximize $z = \sum_{l=1}^{n} c_l x_l$, subject to $\sum_{l=1}^{n} a_{k,l} x_l \leq b_k$ for all $k \in \{1, ..., m\}$ and $x_l \in \{0, 1\}$ for all $l \in \{1, ..., n\}$. From the aforementioned comment, these

instances were changed to general integer decision variables. That is, $x_l \in \mathbb{Z}_+$ for all $l \in \{1, ..., n\}$.

These problems have $a_{k,l} \in \mathbb{Z}_+$, randomly generated, and uniformly distributed between 0 and $1,000$. Each right-hand side value b_k was calculated as $\lfloor \delta \sum_{l=1}^{n} a_{k,l} \rfloor$ where δ is the tightness ratio. Furthermore, each cost coefficient c_l was generated as $\sum_{k=1}^{m} a_{k,l} + \lfloor 500\gamma_l \rfloor$ where γ_l is a uniform random number between 0 and 1. For each problem set, 10 instances exist with $\delta = 0.25$, 10 with $\delta = 0.50$, and 10 with $\delta = 0.75$.

4.2 Results and Analysis

Results for problems sets *mknapcb1*, *mknapcb2*, *mknapcb3*, and *mknapcb4* are not presented because these instances are small and solved to optimality, on average, in less than 10 seconds. Problems sets *mknapcb5*, *mknapcb6*, and *mknapcb7* are considered medium sized problems and are jointly compared. Problems sets *mknapcb8* and *mknapcb9* are considered large complex problems and are also jointly compared.

Tables 3-7 (Appendix) present the results obtained for instances in problems sets *mknapcb5*, *mknapcb6*, *mknapcb7*, *mknapcb8*, and *mknapcb9*, respectively. Let # denote the particular data set instance. Columns *1st Integer* presents the z value, node number, and optimality gap $\Delta = \left(\frac{z^*}{z}\right) - 1$ of the first integer solution obtained by both OBA and CPLEX. Column *Best Integer* describes the z value and node number of the best integer solution found by OBA within the first half a million nodes. Columns *Match OBA's 1st Integer* and *Match OBA's Best Integer* show the number of nodes evaluated by CPLEX to obtain a z value that is at least as good as the z value of OBA's first and best integer solutions, respectively.

The optimal objective function value z^* was obtained by solving each instance with CPLEX at default settings. Instances where z^* is denoted with a † corresponds to the best z value obtained with a time limit of 10 hours. All remaining instances are optimally solved.

When analyzing the results of instances in problem set *mknapcb5* (Table 3), OBA found the first integer solution, on average, after 212 nodes with an optimality gap of 1.2%. On the other hand, CPLEX found its first integer solution after $2,189$ nodes with an optimality gap of 3.9%. Not only did OBA found a better solution faster than CPLEX, but it also required CPLEX over a 160 thousand nodes to achieve OBA's first integer solution. After half a million nodes, OBA's best integer solution was found, on average, within $333,358$ nodes while CPLEX required on average more than 1.3 million nodes to match OBA's best found integer solution, which is approximately 1 million more nodes than OBA.

The reader can easily make the same analysis for problems sets *mknapcb6* and *mknapcb7* in Tables 4 and 5. Table 1 summarizes the results from Tables 3-5. Observe that the results in Table 1 correspond to the average from all 30 instances from Tables 3-5. In this case, OBA found the first integer solution, on average, with 90% fewer nodes and such a solution is 56% closer to the optimal solution than CPLEX. Moreover, CPLEX's branch and cut algorithm required more than 200 thousand nodes to achieve the same (or better) first integer solution, and more than 3.4 million nodes to match the best integer solution from OBA on these medium sized instances.

For *mknapcb8* (Table 6), OBA found the first integer solution, on average, after 179 nodes with an optimality gap of 2.2% while CPLEX found its first integer solution after $2,392$ nodes with an optimality gap of 7.0%. In addition, CPLEX matched OBA's first and best integer solutions after more than 5 million and half a billion nodes, respectively.

Table 1 Summary of results from problems set *mknapcb5*, *mknapcb6*, and *mknapcb7*

Instance	OBA up to 500,000 Nodes			CPLEX			
	1st Integer		Best Integer	1st Integer		Match OBA's 1st Integer	Match OBA's Best Integer
	Node	Δ	Node	Node	Δ	Node	Node
mknapcb5	212	1.2%	333,358	2,189	3.9%	163,613	1,313,906
mknapcb6	454	0.7%	303,789	4,480	3.0%	470,510	4,963,618
mknapcb7	74	5.4%	290,005	859	9.5%	22,635	5,006,540
Average	**247**	**2.4%**	**309,051**	**2,510**	**5.5%**	**218,919**	**3,761,355**

The most complex instances in this computational study are from *mknapcb9* (Table 7). OBA's first integer solution was found after 327 nodes with an optimality gap of 1.3%, and CPLEX's first integer solution was found after $4,763$ nodes with an optimality gap of 5.5%, on average. In this case, CPLEX had to evaluate more than 300 million and 1.3 billion nodes to achieve OBA's first and best integer solutions, on average.

Overall, problems sets *mknapcb8* and *mknapcb9* evaluated on average 93% fewer nodes to obtain a first integer solution that is 73% closer to the optimal solution than CPLEX (Table 2). Furthermore, CPLEX had to search over 180 million nodes and 980 million nodes to achieve an integer solution that is at least as good as OBA's first and best integer solutions, respectively. Results from these large complex instances show some of OBA's potential to identify quality integer solutions in nonbinary IPs.

Table 2 Summary of results from problems set *mknapcb8* and *mknapcb9*

Instance	OBA up to 500,000 Nodes			CPLEX			
	1st Integer		Best Integer	1st Integer		Match OBA's 1st Integer	Match OBA's Best Integer
	Node	Δ	Node	Node	Δ	Node	Node
mknapcb8	179	2.2%	338,610	2,392	7.0%	5,167,623	586,002,366
mknapcb9	327	1.3%	280,626	4,763	5.5%	357,681,991	1,376,964,911
Average	**253**	**1.7%**	**309,618**	**3,578**	**6.3%**	**181,424,807**	**981,483,638**

Based on the computational results presented, one may imply that OBA finds quality integer solutions more quickly than CPLEX's branch and cut algorithm (or simply an approximation of BB). That is, CPLEX had to search over 3 million more nodes than OBA to achieve the same (or better) solution when solving medium sized instances from the OR-Library. This comparison is more pronounced when analyzing large complex instances from the OR-Library, since CPLEX had to search almost 1 billion more nodes to match OBA's best integer solution. Consequently, this paper suggests using OBA early in the branching tree and transitioning to standard branching. Alternately, one could implement OBA's equality nodes for concepts such as warm start solution or random diving.

5 Conclusion and Future Research

This paper introduced a polyhedral branching technique, referred to as the octanary branching algorithm (OBA). Traditional branch and bound (BB) generates two child nodes

for every unfathomed parent node, while OBA creates eight children. Theoretically, OBA solves any bounded integer program. Furthermore, OBA forces the dimension of the feasible linear relaxation space of four of the eight children to be strictly less than the dimension of its parent's feasible linear relaxation space. This dimension reduction should enable OBA to identify integer solutions earlier in its branching tree.

Computational experiments evaluated the initial tree of OBA and CPLEX's branching algorithm (an approximation of BB). In both medium sized and large complex instances from the OR-Library, OBA found better integer solutions earlier in the branching tree. In the most complex instances, CPLEX searched over 1 billion more nodes than OBA to find the same quality integer solution. Thus, OBA should be used in place of BB for the first portion of the branching tree, be implemented as a warm start solution, or be applied as a diving strategy.

The work presented in this paper also generates a variety of new research questions. Since OBA finds quality integer solutions faster than BB, would OBA be significantly more effective at a random diving process? What other branching polyhedra can be more effective at solving integer programs. For instance, the four children from OBA that create equality constraints represent an "inner layer" of the feasible region close to the parent's linear relaxation solution, while the four children with inequality constraints represent the "outer layer." Any additional layers would be further from the parent's linear relaxation solution and would likely have significantly worse objective function values. Would a branching scheme that creates more than two layers perform better than OBA?

Acknowledgment

This research was funded in part by the Kansas Technology Enterprise Corporation, MOE Academic Research Fund under Grant No. 2016-T2-1-170, and National Research Foundation under Grant No. NRF-NRFF2018-0.

References

Abraham, J.N. and Rao, K.S. (2008) 'Production planning through flow network optimisation and mixed integer linear programming models in a petroleum refinery', *International Journal of Operational Research*, Vol. 3, No. 3, pp.315–335

Achterberg, T., Koch, T. and Martin, A. (2005) 'Branching rules revisited', *Operations Research Letters*, Vol. 33, No. 1, pp.42–54

Albashabsheh, N.T. and Heier Stamm, J.L. (2019) 'Optimization of lignocellulosic biomass-to-biofuel supply chains with mobile pelleting', *Transportation Research Part E: Logistics and Transportation Review*, Vol. 122, No. 1, pp.545–562

Antunes, C.H., Martins, A.G. and Brito, I.S. (2004) 'A multiple objective mixed integer linear programming model for power generation expansion planning', *Energy*, Vol. 29, No. 4, pp.613–627

Bailey, J.P. (2012) *Octanary Branching Algorithm*. Mater's thesis, Kansas State University, Manhattan KS, USA

Bailey, J, Iwen, M.A. and Spencer, C.V. (2012) 'On the design of deterministic matrices for fast recovery of Fourier compressible functions', *SIAM Journal on Matrix Analysis and Applications*, Vol. 33, No. 1, pp.263–289

Balas, E. and Zemel, E. (1978) 'Facets of the knapsack polytope from minimal covers', *SIAM Journal on Applied Mathematics*, Vol. 34, No. 1, pp.119–148

Beasley, J.E. (1990) 'OR-library: distributing test problems by electronic mail', *The Journal of the Operational Research Society*, Vol. 41, No. 11, pp.1069–1072

Ben-Arieh, D., Easton, T. and Choubey, A.M. (2009) 'Solving the multiple platforms configuration problem', *International Journal of Production Research*, Vol. 47, No. 7, pp.1969–1988

Benders, J.F. (1962) 'Partitioning procedures for solving mixed-variables programming problems', *Numerische Mathematik*, Vol. 4, No. 1, pp.238–252

Carrion, M. and Arroyo, J.M. (2006) 'A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem', *IEEE Transactions on Power Systems*, Vol. 21, No. 3, pp.1371–1378

Chu, P.C. and Beasley, J.E. (1998) 'A genetic algorithm for the multidimensional knapsack problem', *Journal of Heuristics*, Vol. 4, No. 1, pp.63–86

Chvátal, V. (1973) 'Edmonds polytopes and a hierarchy of combinatorial problems', *Discrete Mathematics*, Vol. 4, No. 4, pp.305–337

Conejo, A.J., Castillo, E., Mínguez, R. and García-Bertrand, R. (2006) *Decomposition Techniques in Mathematical Programming*, Springer-Verlag, Berlin Heidelberg, Germany

Conforti, M., Cornuejols, G. and Zambelli, G. (2014) *Integer Programming*, Springer, Switzerland

Dantzig, G.B. and Wolfe, P. (1960) 'Decomposition principle for linear programs', *Operations Research*, Vol. 8, No. 1, pp.101–111

Dantzig, G.B. and Wolfe, P. (1961) 'The decomposition algorithm for linear programs', *Econometrica*, Vol. 29, No. 4, pp.767–778

Delli, U. and Sinha, A.K. (2019) 'Parallel computation framework for optimizing trailer routes in bulk transportation', *Journal of Industrial Engineering International*, pp.1–11 [online] https://link.springer.com/article/10.1007/s40092-019-0308-8 (Accessed 17 June 2019)

Duong, V.H. and Bui, N.H. (2018) 'A mixed-integer linear formulation for a capacitated facility location problem in supply chain network design', *International Journal of Operational Research*, Vol. 33, No. 1, pp.32–54

Easton, T. and Lee, J. (2012) 'Quaternary hyperplane branching with internally generated cutting planes for solving integer programmes', *International Journal of Operational Research*, Vol. 14, No. 3, pp.366–385

Easton, T., Hooker, K. and Lee, E.K. (2003) 'Facets of the independent set polytope', *Mathematical Programming*, Vol. 98, No. 1-3, pp.177–199

Ergünes, B., Özdamar, L., Demir, O. and Gülcan, N. (2017) 'A partitioning algorithm for the mixed integer nonlinear programming problem', *International Journal of Operational Research*, Vol. 28, No. 2, pp.201–215

Ford, L.R. and Fulkerson, D.R. (1958) 'A suggested computation for maximal multi-commodity network flows', *Management Science*, Vol. 5, No. 1, pp.97–101

Gifford, T., Opicka, T., Sinha, A., Brink, D.V., Gifford, A. and Randall, R. (2018) 'Dispatch optimization in bulk tanker transport operations', *INFORMS Journal on Applied Analytics*, Vol. 48, No. 5, pp.403–421

Gilmore, P.C. and Gomory, R.E. (1961) 'A linear programming approach to the cutting-stock problem', *Operations Research*, Vol. 9, No. 6, pp.849–859

Gilmore, P.C. and Gomory, R.E. (1963) 'A linear programming approach to the cutting-stock problem-part ii', *Operations Research*, Vol. 11, No. 6, pp.863–888

Gomory, R.E. (1969) 'Some polyhedra related to combinatorial problems', *Linear Algebra and its Applications*, Vol. 2, No. 4, pp.451–558

Hamilton, W.F. and Moses, M.A. (1973) 'An optimization model for corporate financial planning', *Operations Research*, Vol. 21, No. 3, pp.677–692

Heier Stamm, J.L., Serban, N., Swann, J. and Wortley, P. (2017) 'Quantifying and explaining accessibility with application to the 2009 H1N1 vaccination campaign', *Health Care Management Science*, Vol. 20, No. 1, pp.76–93

Hickman, R. and Easton, T. (2015) 'Merging valid inequalities over the multiple knapsack polyhedron', *International Journal of Operational Research*, Vol. 24, No. 2, pp.214–227

Hickman, R. and Easton, T. (2015) 'On merging cover inequalities for multiple knapsack problems', *Open Journal of Optimization*, Vol. 4, No. 4, pp.141–155

IBM ILOG CPLEX Optimization Studio (2016) Version 12.7 [online] http://www-01.ibm.com/software/info/ilog/ (Accessed 17 June 2019)

IBM ILOG CPLEX Optimization Studio CPLEX User's Manual (2016) Version 12.7 [online] https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/pdf/usrcplex.pdf (Accessed 17 June 2019)

Jörnsten, K.O. and Värbrand, P. (1991) 'A hybrid algorithm for the generalized assignment problem', *Optimization*, Vol. 22, No. 2, pp.273–282

Karp, R.M. (1972) 'Reducibility among Combinatorial Problems', in Miller, R.E. et al. (Eds), *Complexity of Computer Computations*, Plenum, New York NY, USA, pp.85–103

Kashkoush, M.N., Shalaby, M.A. and Abdelhafiez, E.A. (2012) 'A mixed-integer model for two-dimensional polyominoes strip packing and tiling problems', *International Journal of Operational Research*, Vol. 15, No. 4, pp.391–405

Kose, E. and Karabay, S. (2016) 'Mathematical programming model proposal to solve a real-life public sector facility location problem', *International Journal of Operational Research*, Vol. 26, No. 1, pp.1–12

Krokhmal, P., Uryasev, S. and Zrazhevsky, G. (2002) 'Risk management for hedge fund portfolios: a comparative analysis of linear rebalancing strategies', *The Journal of Alternative Investments*, Vol. 5, No. 1, pp.10–29

Land, A.H. and Doig, A.G. (1960) 'An automatic method of solving discrete programming problems', *Econometrica*, Vol. 28, No. 3, pp.497–520

Lee, E.K., Fox, T. and Crocker, I. (2003) 'Integer programming applied to intensity-modulated radiation therapy treatment planning', *Annals of Operations Research*, Vol. 119, No. 1-4, pp.165–181

Limpianchob, C. (2017) 'Integrated of harvesting and production planning in aromatic coconut supply chain using mixed-integer linear programming', *International Journal of Operational Research*, Vol. 30, No. 3, pp.360–374

Linderoth, J.T. and Savelsbergh, M.W.P. (1999) 'A computational study of search strategies for mixed integer programming', *INFORMS Journal on Computing*, Vol. 11, No. 2, pp.173–187

Lübbecke, M.E. and Desrosiers, J. (2005) 'Selected topics in column generation', *Operations Research*, Vol. 53, No. 6, pp.1007–1023

Lustig, M., Donoho, D. and Pauly, J.M. (2007) 'Sparse MRI: the application of compressed sensing for rapid MR imaging', *Magnetic Resonance in Medicine*, Vol. 58, No. 6, pp.1182–1195

Lustig, M., Donoho, D.L., Santos, J.M. and Pauly, J.M. (2008) 'Compressed sensing MRI: a look at how CS can improve on current imaging techniques', *IEEE Signal Processing Magazine*, Vol. 25, No. 2, pp.72–82

Ma, J. (2009) 'Single-pixel remote sensing', *IEEE Geoscience and Remote Sensing Letters*, Vol. 6, No. 2, pp.199–203

Ma, J. (2010) 'Compressed sensing for surface characterization and metrology', *IEEE Transactions on Instrumentation and Measurement*, Vol. 59, No. 6, pp.1600–1615

Mehrotra, S. and Li, Z. (2011) 'Branching on hyperplane methods for mixed integer linear and convex programming using adjoint lattices', *Journal of Global Optimization*, Vol. 49, No. 4, pp.623–649

Morrison, D.R., Jacobson, S.H., Sauppe, J.J. and Sewell, E.C. (2016) 'Branch-and-bound algorithms: a survey of recent advances in searching, branching, and pruning', *Discrete Optimization*, Vol. 19, No. 1, pp.79–102

Muggy, L. and Heier Stamm, J.L. (2017) 'Dynamic, robust models to quantify the impact of decentralization in post-disaster health care facility location decisions', *Operations Research for Health Care*, Vol. 12, No. 1, pp.43–59

Nemhauser, G.L. (2012) 'Column generation for linear and integer programming', *Documenta Mathematica*, Extra Volume: Optimization Stories, pp.65–73

Nemhauser, G.L and Wolsey, L.A. (1999) *Integer and Combinatorial Optimization*, John Wiley and Sons, New York NY, USA

Noor-E-Alam M., Todd, B. and Doucette, J. (2014) 'Integer linear programming model for grid-based wireless transmitter location problems', *International Journal of Operational Research*, Vol. 22, No. 1, pp.48–64

Pendharkar, P.C. and Rodger, J.A. (2006) 'Information technology capital budgeting using a knapsack problem', *International Transactions in Operational Research*, Vol. 13, No. 4, pp.333–351

Ryan, D. and Foster, B. (1981) 'An integer programming approach to scheduling', in Wren, A. (Ed), *Computer Scheduling of Public Transport*, North-Holland Publishing Company, Amsterdam, Netherlands, pp.269–280

Salam, A., Bandaly, D. and Defersha, F.M. (2015) 'Optimising the design of a supply chain network with economies of scale using mixed integer programming', *International Journal of Operational Research*, Vol. 10, No. 4, pp.398–415

Singh, G., Sier, D., Ernst, A.T., Gavriliouk, O., Oyston, R., Giles, T. and Welgama, P. (2012) 'A mixed integer programming model for long term capacity expansion planning: a case study from the Hunter Valley Coal Chain', *European Journal of Operational Research*, Vol. 220, No. 1, pp.210–224

Sinha, A.K., Davich, T. and Krishnamurthy, A. (2016) 'Optimisation of production and subcontracting strategies', *International Journal of Production Research*, Vol. 54, No. 8, pp.2377–2393

Stahl, J.E., Kong, N., Shechter, S.M., Schaefer, A.J. and Roberts, M.S. (2005) 'A methodological framework for optimally reorganizing liver transplant regions', *Medical Decision Making*, Vol. 25, No. 1, pp.35–46

Subramanian, R., Scheff, R.P., Quillinan, J.D., Steve Wiper, D. and Marsten, R.E. (1994) 'Coldstart: fleet assignment at Delta Air Lines', *INFORMS Journal on Applied Analytics*, Vol. 24, No. 1, pp.104–120

Toffolo, T.A.M., Santos, H.G., Carvalho, M.A.M. and Soares, J.A. (2016) 'An integer programming approach to the multimode resource-constrained multiproject scheduling problem', *Journal of Scheduling*, Vol. 19, No. 3, pp.295–307

Vitor, F.T. (2015) *Improving the Solution Time of Integer Programs by Merging Knapsack Constraints with Cover Inequalities*. Mater's thesis, Kansas State University, Manhattan KS, USA

Vitor, F. and Easton, T. (2016) 'Merged knapsack cover inequalities for the multiple knapsack problem', in *Proceedings of the 2016 Industrial and Systems Engineering Research Conference*, Institute of Industrial and Systems Engineers, Anaheim CA, USA, pp.607–612

Vitor, F. and Easton, T. (2019) 'Approximate and exact merging of knapsack constraints with cover inequalities', to appear in *Optimization*

Vuthipadadon, S. and Olafsson, S. (2007) 'An integer programming approach for scheduling inbound calls in call centres', *International Journal of Operational Research*, Vol. 2, No. 4, pp.414–428

Walker, R.J. (1960) 'An enumerative technique for a class of combinatorial problems', in *Proceedings of Symposia in Applied Mathematics*, American Mathematical Society, Providence RI, USA, pp.91–94

Yin, J., Yang, L., Tang, T., Gao, Z. and Ran, B. (2017) 'Dynamic passenger demand oriented metro train scheduling with energy-efficiency and waiting time minimization: mixed-integer linear programming approaches', *Transportation Research Part B: Methodological*, Vol. 97, No. 1, pp.182–213

Zhan, Y. and Zheng, Q.P. (2018) 'A multistage decision-dependent stochastic bilevel programming approach for power generation investment expansion planning', *IISE Transactions*, Vol. 50, No. 8, pp.720–734

Appendix

Table 3 Results obtained with instances from problem set *mknapcb5* - 250 variables × 10 constraints

δ	#	z^*	OBA up to 500,000 Nodes					CPLEX						
			1st Integer			Best Integer		1st Integer			Match OBA's 1st Integer		Match OBA's Best Integer	
			z	Node	Δ	z	Node	z	Node	Δ	z	Node	z	Node
0.25	1	73,707	72,592	153	1.5%	73,498	240,867	67,063	2,089	9.9%	72,671	77,628	73,528	634,295
	2	68,262	66,709	202	2.3%	67,778	263,153	61,467	2,166	11.1%	66,748	31,049	67,879	1,844,822
	3	65,872	64,692	287	1.8%	65,526	475,661	62,007	1,968	6.2%	65,164	3,980	65,663	84,350
	4	73,113	71,213	148	2.7%	72,946	443,133	70,488	2,117	3.7%	71,282	6,071	72,970	834,873
	5	67,810	66,318	144	2.2%	67,547	295,202	59,823	2,213	13.4%	66,389	21,805	67,591	663,873
	6	67,404	66,456	314	1.4%	66,884	310,974	61,490	2,102	9.6%	66,595	16,594	66,888	72,989
	7	67,248	66,472	153	1.2%	66,980	68,371	64,297	2,204	4.6%	66,532	16,738	66,999	1,719,410
	8	74,030	72,856	327	1.6%	73,951	331,459	72,742	1,971	1.8%	73,087	2,342	74,030	242,537
	9	67,880	66,113	180	2.7%	67,371	375,879	65,456	2,009	3.7%	66,883	2,512	67,771	80,882
	10	72,857	71,477	139	1.9%	72,558	248,007	70,669	2,065	3.1%	71,481	2,399	72,662	382,718
0.50	11	139,865	138,939	135	0.7%	139,630	436,193	137,360	2,025	1.8%	138,939	17,990	139,706	210,065
	12	133,070	131,709	241	1.0%	132,874	443,125	130,009	2,331	2.4%	131,911	65,467	132,887	2,313,373
	13	142,345	141,632	144	0.5%	142,031	265,309	139,723	1,998	1.9%	141,693	127,821	142,178	359,889
	14	149,425	148,560	211	0.6%	149,019	174,119	146,420	2,006	2.1%	148,931	5,701	149,021	20,734
	15	136,335	135,021	126	1.0%	135,845	485,726	135,390	2,041	0.7%	135,390	2,041	135,886	50,645
	16	135,985	134,600	162	1.0%	135,639	290,764	132,229	2,051	2.8%	134,688	2,658	135,672	160,957
	17	137,032	135,864	400	0.9%	136,954	260,574	132,737	2,171	3.2%	135,896	164,138	136,975	1,050,886
	18	130,965	129,595	275	1.1%	130,383	375,663	126,846	2,432	3.2%	129,891	19,065	130,438	898,862
	19	138,210	137,150	162	0.8%	137,860	342,392	133,339	2,069	3.7%	137,284	6,215	138,184	625,142
	20	135,413	133,447	135	1.5%	135,052	307,799	132,716	2,290	2.0%	133,464	2,589	135,091	201,691
0.75	21	202,153	200,185	189	1.0%	201,582	488,196	191,621	2,375	5.5%	200,186	58,146	201,690	175,868
	22	198,119	196,870	180	0.6%	197,492	474,267	194,128	2,430	2.1%	197,198	289,345	197,501	556,581
	23	202,936	202,010	144	0.5%	202,715	6,414	201,357	2,236	0.8%	202,304	23,060	202,817	1,329,543
	24	205,247	203,782	211	0.7%	204,862	464,862	201,930	2,539	1.6%	204,475	3,543	204,866	780,325
	25	219,813	219,028	647	0.4%	219,770	380,795	217,574	2,163	1.0%	219,275	67,193	219,813	369,825
	26	209,091	207,181	228	0.9%	208,680	367,118	206,567	1,920	1.2%	207,902	2,126	208,682	97,360
	27	209,163	207,557	144	0.8%	209,002	468,097	205,567	2,059	1.7%	208,445	2,365	209,052	83,623
	28	214,848	213,274	188	0.7%	214,758	469,826	211,742	2,438	1.5%	213,608	35,856	214,848	1,726,624
	29	204,351	203,301	126	0.5%	204,271	377,095	195,465	2,396	4.5%	203,382	923,155	204,351	6,642,405
	30	201,084	200,182	274	0.5%	200,745	69,714	190,183	2,786	5.7%	200,206	2,908,797	200,754	15,202,044
Average				**212**	**1.2%**		**333,358**		**2,189**	**3.9%**		**163,613**		**1,313,906**

Table 4 Results obtained with instances from problem set *mknapcb6* - 500 variables × 10 constraints

δ	#	z^*	OBA up to 500,000 Nodes					CPLEX						
			1st Integer			Best Integer		1st Integer			Match OBA's 1st Integer		Match OBA's Best Integer	
			z	Node	Δ	z	Node	z	Node	Δ	z	Node	z	Node
0.25	1	147,197	145,400	260	1.2%	146,418	404,036	141,411	4,119	4.1%	145,401	12,968	146,603	45,442
	2	149,902	149,001	615	0.6%	149,396	164,448	139,494	4,266	7.5%	149,018	1,463,764	149,415	8,640,676
	3	144,130	142,831	1,273	0.9%	143,685	371,396	137,536	4,541	4.8%	143,082	3,788,071	143,706	18,744,800
	4	143,311	141,850	323	1.0%	142,910	488,932	141,164	4,119	1.5%	141,936	13,209	142,939	165,536
	5	140,504	138,788	251	1.2%	139,815	162,278	135,689	4,218	3.5%	138,811	13,015	139,846	230,244
	6	145,890	144,399	623	1.0%	145,448	266,648	139,073	4,426	4.9%	144,562	86,685	145,496	457,516
	7	153,398	152,058	287	0.9%	152,967	254,172	148,361	4,110	3.4%	152,063	26,711	153,035	1,486,118
	8	142,329	140,506	269	1.3%	141,774	469,839	136,912	4,064	4.0%	140,553	6,005	141,882	164,341
	9	141,017	138,583	278	1.8%	140,099	384,847	129,732	4,804	8.7%	138,694	116,190	140,148	788,606
	10	139,187	137,683	260	1.1%	138,680	443,963	134,778	4,538	3.3%	137,738	358,414	138,706	24,944,759
0.50	11	277,975	276,892	407	0.4%	277,752	428,284	274,292	4,269	1.3%	277,024	45,899	277,795	20,930,632
	12	288,022	286,533	269	0.5%	287,484	225,749	274,133	5,465	5.1%	286,553	690,535	287,497	3,724,401
	13	286,686	285,279	430	0.5%	286,288	287,656	283,224	4,336	1.2%	285,337	10,743	286,463	209,647
	14	304,563 †	303,527	599	0.3%	304,400	240,027	299,535	4,525	1.7%	303,777	129,194	304,563	2,651,173
	15	283,784	281,844	305	0.7%	283,194	407,946	271,111	5,095	4.7%	281,845	96,119	283,197	426,148
	16	282,947	281,381	520	0.6%	282,399	251,086	275,206	4,433	2.8%	281,417	17,123	282,563	203,609
	17	299,363 †	297,518	465	0.6%	298,708	110,241	295,152	4,195	1.4%	297,527	16,184	298,733	164,134
	18	290,833 †	289,249	899	0.5%	290,313	263,169	287,451	4,435	1.2%	289,503	165,587	290,319	4,707,322
	19	294,267	292,973	360	0.4%	293,792	348,482	284,766	4,609	3.3%	292,989	798,115	293,832	5,057,984
	20	291,991 †	290,621	359	0.5%	291,489	342,045	290,883	3,987	0.4%	290,883	3,987	291,536	12,049
0.75	21	433,589	431,947	251	0.4%	433,152	47,736	426,265	4,580	1.7%	431,988	173,550	433,219	3,664,415
	22	474,526	473,210	527	0.3%	474,228	247,046	463,065	4,230	2.5%	473,313	126,841	474,338	1,167,227
	23	413,945 †	412,000	269	0.5%	413,152	333,775	407,632	4,151	1.5%	412,440	4,486	413,214	76,218
	24	455,080	453,827	781	0.3%	454,623	138,716	447,112	4,402	1.8%	453,966	98,584	454,655	1,139,117
	25	422,465 †	421,386	747	0.3%	422,033	459,395	418,945	4,314	0.8%	421,396	27,534	422,061	374,065
	26	422,780 †	420,862	314	0.5%	422,132	494,950	395,042	6,251	7.0%	420,903	5,275,679	422,260	7,085,052
	27	444,754	443,435	621	0.3%	444,329	14,257	439,599	4,344	1.2%	443,734	16,246	444,349	163,138
	28	426,245 †	425,059	322	0.3%	425,739	315,001	412,900	4,542	3.2%	425,227	124,490	425,867	587,501
	29	413,149 †	411,574	358	0.4%	412,929	322,298	406,758	4,926	1.6%	411,811	403,583	412,936	40,543,979
	30	437,295 †	435,995	367	0.3%	436,922	425,254	433,152	4,119	1.0%	436,067	5,800	436,933	352,696
Average				**454**	**0.7%**		**303,789**		**4,480**	**3.0%**		**470,510**		**4,963,618**

Table 5 Results obtained with instances from problem set *mknapcb7* - 100 variables × 30 constraints

δ	#	z^*	OBA up to 500,000 Nodes					CPLEX						
			1st Integer			Best Integer		1st Integer			Match OBA's 1st Integer		Match OBA's Best Integer	
			z	Node	Δ	z	Node	z	Node	Δ	z	Node	z	Node
0.25	1	22,563	20,910	96	7.9%	22,048	282,496	19,559	808	15.4%	21,279	1,253	22,075	182,195
	2	22,495	20,468	61	9.9%	21,837	66,150	19,136	811	17.6%	20,652	1,280	22,004	214,567
	3	21,285	19,703	78	8.0%	21,052	267,199	19,536	806	9.0%	19,945	948	21,067	3,434,602
	4	22,142	20,143	51	9.9%	21,810	324,847	17,865	738	23.9%	20,281	930	21,843	40,536
	5	22,523	20,432	56	10.2%	21,926	348,289	19,352	740	16.4%	20,443	15,345	21,927	1,424,154
	6	22,711	21,273	69	6.8%	22,423	18,896	19,675	840	15.4%	21,345	9,050	22,485	1,080,408
	7	22,306	20,622	60	8.2%	21,881	319,005	19,877	798	12.2%	21,081	7,243	22,052	263,207
	8	22,121	20,086	91	10.1%	21,614	243,483	20,398	767	8.4%	20,398	767	21,622	149,952
	9	23,006	20,111	70	14.4%	22,731	168,942	20,682	756	11.2%	20,682	756	22,960	73,243,087
	10	22,228	20,355	64	9.2%	21,367	439,509	20,017	798	11.0%	20,685	1,784	21,639	104,992
0.50	11	44,178	41,962	69	5.3%	43,796	197,611	40,991	885	7.8%	41,962	2,881	43,829	12,061,075
	12	45,524	44,544	69	2.2%	45,023	483,668	43,356	804	5.0%	44,559	21,978	45,129	211,516
	13	46,545	45,078	123	3.3%	46,236	128,187	44,014	821	5.8%	45,120	14,222	46,286	3,809,827
	14	45,568	44,236	172	3.0%	45,363	319,757	43,584	838	4.6%	44,315	18,910	45,369	2,823,144
	15	43,634	41,453	51	5.3%	42,946	124,747	37,512	1,012	16.3%	41,484	37,393	42,950	24,774,086
	16	45,408	43,317	51	4.8%	44,961	314,093	42,160	879	7.7%	43,341	1,705	45,015	957,484
	17	43,940	41,957	51	4.7%	43,257	223,380	40,816	837	7.7%	42,235	1,016	43,258	800,297
	18	46,327	44,554	78	4.0%	45,555	361,552	43,590	858	6.3%	44,562	5,301	45,690	59,011
	19	45,784	43,987	60	4.1%	45,343	239,389	42,864	946	6.8%	44,007	2,718	45,437	1,299,784
	20	45,353	43,984	140	3.1%	44,772	322,386	41,188	925	10.1%	44,174	96,425	44,842	915,313
0.75	21	66,000	64,405	61	2.5%	65,508	445,081	63,555	860	3.8%	64,631	6,458	65,539	2,796,348
	22	68,990	66,937	60	3.1%	68,162	444,884	65,118	981	5.9%	67,416	21,871	68,165	214,460
	23	66,017	63,889	105	3.3%	65,284	426,920	62,367	848	5.9%	63,918	2,854	65,344	888,482
	24	68,371 †	66,570	60	2.7%	67,610	427,812	64,405	963	6.2%	66,894	2,193	67,618	247,244
	25	68,748	66,579	60	3.3%	68,007	208,577	65,911	843	4.3%	66,903	912	68,111	3,566
	26	70,462	69,183	69	1.8%	70,195	225,420	66,325	930	6.2%	69,279	15,733	70,260	3,656,236
	27	68,280	66,071	60	3.3%	67,506	147,930	64,655	954	5.6%	66,204	19,104	67,697	10,429,359
	28	67,523 †	66,164	60	2.1%	66,977	376,280	62,993	849	7.2%	66,540	2,969	67,011	48,932
	29	68,845	67,689	60	1.7%	68,130	398,936	62,327	975	10.5%	67,792	344,284	68,156	726,357
	30	67,951	66,009	60	2.9%	67,234	404,713	60,997	914	11.4%	66,052	20,754	67,251	3,335,988
Average				**74**	**5.4%**		**299,005**		**859**	**9.5%**		**22,635**		**5,006,540**

Table 6 Results obtained with instances from problem set *mknapcb8* - 250 variables × 30 constraints

δ	#	z^*	OBA up to 500,000 Nodes					CPLEX						
			1st Integer			Best Integer		1st Integer			Match OBA's 1st Integer		Match OBA's Best Integer	
			z	Node	Δ	z	Node	z	Node	Δ	z	Node	z	Node
0.25	1	59,621 †	57,291	238	4.1%	58,281	418,277	55,901	2,159	6.7%	57,579	3,970	58,556	30,974
	2	61,832 †	59,482	193	4.0%	60,661	432,819	58,235	2,284	6.2%	59,761	241,003	60,983	9,361,270
	3	60,884 †	58,179	199	4.6%	59,680	201,502	53,550	2,213	13.7%	58,532	40,208	59,696	212,533
	4	60,557 †	58,751	162	3.1%	59,965	360,062	56,673	1,814	6.9%	58,920	153,925	60,012	3,349,592
	5	59,922 †	58,062	296	3.2%	59,316	431,650	53,261	2,229	12.5%	58,130	74,191	59,321	25,528,718
	6	61,138 †	59,031	162	3.6%	60,229	403,647	57,770	2,142	5.8%	59,041	20,057	60,350	791,285
	7	59,487 †	57,523	154	3.4%	58,413	372,762	53,937	2,187	10.3%	57,545	115,892	58,459	123,661,291
	8	59,581 †	57,343	154	3.9%	58,413	179,141	54,474	2,327	9.4%	57,699	230,331	58,420	11,173,187
	9	60,248 †	57,390	135	5.0%	58,854	205,167	56,239	2,195	7.1%	57,542	29,703	58,902	415,108
	10	59,369 †	57,457	188	3.3%	58,115	389,568	53,262	2,185	11.5%	57,493	1,550,945	58,263	29,882,426
0.50	11	123,737 †	121,691	153	1.7%	122,802	222,030	113,357	2,536	9.2%	121,727	61,427,184	122,804	1,439,091,923
	12	122,749 †	120,505	135	1.9%	121,733	354,292	112,613	2,505	9.0%	120,518	1,805,925	121,743	22,288,055
	13	121,243 †	119,021	153	1.9%	120,266	481,706	109,538	2,725	10.7%	119,086	23,015,553	120,349	428,130,021
	14	119,807 †	118,214	162	1.3%	119,011	353,939	112,569	2,383	6.4%	118,229	3,465,708	119,035	33,755,940
	15	119,707 †	118,097	274	1.4%	118,721	471,758	109,095	2,459	9.7%	118,172	3,012,828	118,768	73,488,696
	16	124,196 †	122,632	135	1.3%	123,687	245,641	117,617	2,425	5.6%	122,689	5,888,968	123,697	655,519,803
	17	121,050 †	118,850	188	1.9%	120,136	498,264	112,228	2,464	7.9%	118,924	229,932	120,264	13,127,795
	18	120,441 †	118,192	175	1.9%	119,447	439,024	117,690	2,254	2.3%	118,294	3,829	119,468	410,032
	19	123,069 †	120,408	144	2.2%	122,289	389,766	116,391	2,448	5.7%	120,419	3,262,601	122,364	87,413,277
	20	119,131 †	116,905	207	1.9%	118,078	190,533	111,129	2,331	7.2%	117,007	172,653	118,117	3,065,817
0.75	21	182,715 †	180,578	135	1.2%	181,627	304,677	177,356	2,382	3.0%	180,890	199,687	181,885	1,468,332
	22	179,562 †	177,003	157	1.4%	178,379	450,158	164,034	3,294	9.5%	177,087	29,691,477	178,383	10,321,410,907
	23	185,068 †	182,708	148	1.3%	183,674	128,500	179,166	2,509	3.3%	182,712	63,955	183,776	761,077
	24	181,479 †	179,523	228	1.1%	180,534	329,330	175,529	2,509	3.4%	179,594	199,855	180,592	12,885,772
	25	184,176 †	182,162	153	1.1%	183,045	344,363	180,140	2,165	2.2%	182,276	6,916	183,104	97,125
	26	178,353 †	176,641	144	1.0%	177,150	331,781	169,835	2,560	5.0%	176,844	252,247	177,202	684,944
	27	186,157 †	184,911	348	0.7%	185,268	95,810	181,763	2,232	2.4%	184,911	1,031,079	185,357	3,063,620
	28	180,562 †	178,426	135	1.2%	179,690	426,151	171,378	2,519	5.4%	178,658	12,790,422	179,693	1,119,807,227
	29	183,767 †	181,508	180	1.2%	182,440	257,994	171,151	2,597	7.4%	181,684	1,766,511	182,606	5,186,890
	30	183,276 †	181,241	135	1.1%	182,265	447,989	173,634	2,732	5.6%	181,245	4,281,147	182,292	3,154,007,336
	Average			**179**	**2.2%**		**338,610**		**2,392**	**7.0%**		**5,167,623**		**586,002,366**

Table 7 Results obtained with instances from problem set *mknapcb9* - 500 variables × 30 constraints

δ	#	z^*	OBA up to 500,000 Nodes					CPLEX						
			1st Integer			Best Integer		1st Integer			Match OBA's 1st Integer		Match OBA's Best Integer	
			z	Node	Δ	z	Node	z	Node	Δ	z	Node	z	Node
0.25	1	127,693 †	124,367	260	2.7%	126,064	305,483	120,894	4,404	5.6%	124,551	248,739	126,118	15,658,421
	2	124,242 †	122,127	327	1.7%	122,840	199,416	112,929	4,369	10.0%	122,404	38,597,251	122,913	285,684,357
	3	127,833 †	124,736	359	2.5%	125,637	393,546	118,690	4,333	7.7%	124,765	142,985	125,641	533,866
	4	123,839 †	121,652	390	1.8%	122,605	370,105	114,321	4,480	8.3%	121,683	2,533,114	122,758	158,319,818
	5	128,114 †	125,893	269	1.8%	126,666	275,097	123,463	4,286	3.8%	125,938	471,931	126,733	3,140,039
	6	125,058 †	122,331	523	2.2%	123,622	486,553	120,542	4,153	3.7%	122,403	246,404	123,640	16,285,578
	7	124,438 †	121,719	323	2.2%	122,817	449,776	114,868	4,260	8.3%	121,774	80,323	122,842	1,260,076
	8	125,905 †	122,778	251	2.5%	124,189	412,275	120,465	4,257	4.5%	122,987	149,447	124,235	2,665,781
	9	123,712 †	121,437	381	1.9%	122,334	241,061	115,679	4,415	6.9%	121,664	94,955	122,395	1,057,281
	10	126,469 †	123,635	444	2.3%	124,539	489,603	111,022	4,652	13.9%	123,717	3,350,698	124,580	31,799,563
0.50	11	251,107 †	249,076	323	0.8%	249,644	6,664	243,777	4,901	3.0%	249,080	88,001,886	249,669	584,762,024
	12	252,364 †	250,012	251	0.9%	251,079	221,133	234,665	5,038	7.5%	250,038	218,970,988	251,112	1,034,427,1[illegible]
	13	252,660 †	250,693	327	0.8%	251,303	49,625	233,238	5,245	8.3%	250,836	42,422,864	251,315	82,188,918
	14	247,335 †	243,980	471	1.4%	245,497	420,778	228,258	5,718	8.4%	245,065	2,904,194,208	245,510	3,624,241,24[illegible]
	15	253,276 †	250,616	318	1.1%	251,522	299,877	245,532	4,774	3.2%	250,667	42,957,658	251,530	5,430,130,01[illegible]
	16	247,356 †	244,391	251	1.2%	245,877	151,564	241,494	4,394	2.4%	244,449	8,840	245,889	21,551,556
	17	251,999 †	248,904	314	1.2%	250,126	407,700	232,889	5,236	8.2%	249,047	20,115,368	250,165	376,447,224
	18	250,200 †	247,712	251	1.0%	248,883	318,642	240,854	4,715	3.9%	247,720	6,937,381	248,993	29,148,990
	19	251,999 †	249,600	390	1.0%	250,564	300,553	238,581	4,668	5.6%	249,601	6,629,753	250,564	60,962,763
	20	252,204 †	249,659	368	1.0%	250,746	119,270	237,346	4,911	6.3%	249,677	10,530,486	250,848	102,120,967
0.75	21	376,175 †	374,532	287	0.4%	374,860	131,365	365,169	4,663	3.0%	374,535	71,802,201	374,869	198,924,587
	22	371,446 †	368,810	260	0.7%	369,977	116,809	361,436	4,666	2.8%	368,825	4,833,578	370,139	343,661,012
	23	385,447 †	383,248	260	0.6%	384,391	178,186	376,734	4,937	2.3%	383,258	42,698,124	384,544	1,092,147,639
	24	379,404 †	376,785	296	0.7%	377,680	230,294	365,993	5,630	3.7%	376,885	3,764,660,458	377,695	6,612,572,260
	25	380,682 †	378,729	332	0.5%	379,277	160,484	363,385	4,841	4.8%	378,741	60,306,638	379,362	386,518,889
	26	376,813 †	374,642	251	0.6%	375,513	270,267	358,387	5,118	5.1%	374,673	2,834,248,988	375,540	8,481,378,482
	27	376,557 †	374,442	260	0.6%	375,472	483,973	354,029	5,090	6.4%	374,477	6,774,908	375,536	529,253,057
	28	385,745 †	383,161	341	0.7%	384,144	270,185	379,588	4,596	1.6%	383,204	1,137,453	384,173	55,194,768
	29	377,726 †	376,013	314	0.5%	376,976	190,226	363,131	5,117	4.0%	376,044	540,771,975	376,993	9,288,330,397
	30	371,856 †	369,599	403	0.6%	370,354	468,271	360,517	5,025	3.1%	369,607	16,540,124	370,389	2,458,580,591
	Average			**327**	**1.3%**		**280,626**		**4,763**	**5.5%**		**357,681,991**		**1,376,964,911**