Incremental and Encoding Formulations for Mixed Integer Programming

Sercan Yıldız[a], Juan Pablo Vielma[b,c,*]

[a]*Tepper School of Business, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, United States*
[b]*Sloan School of Management, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA 02139, United States*
[c]*Department of Industrial Engineering, University of Pittsburgh, 3700 O'Hara Street, Pittsburgh, PA 15261, United States*

Abstract

The standard way to represent a choice between n alternatives in Mixed Integer Programming is through n binary variables that add up to one. Unfortunately, this approach commonly leads to unbalanced branch-and-bound trees and diminished solver performance. In this paper, we present an encoding formulation framework that encompasses and expands existing approaches to mitigate this behavior. Through this framework, we generalize the incremental formulation for piecewise linear functions to any finite union of polyhedra with identical recession cones.

Keywords: Mixed Integer Programming, Disjunctive Programming, Formulations

1. Introduction

A textbook approach to selecting among n discrete alternatives in a Mixed Integer Programming (MIP) problem is to utilize n binary variables $\{y_i\}_{i=1}^n$ with the additional requirement that they add up to one. In particular, this is captured by the set $D^n := \Delta^n \cap \{0,1\}^n$ where $\Delta^n := \{y \in \mathbb{R}^n_+ : \sum_{i=1}^n y_i = 1\}$. For instance, given a finite set $\{a_1, \ldots, a_n\} \subset \mathbb{R}$, we may model the constraint $x \in \{a_1, \ldots, a_n\}$

*Corresponding author. Tel.:+1 617 324 1204
Email addresses: `syildiz@andrew.cmu.edu` (Sercan Yıldız), `jvielma@mit.edu` (Juan Pablo Vielma)

via the MIP formulation given by

$$x = \sum_{i=1}^{n} a_i y_i, \quad y \in D^n. \tag{1}$$

This approach is quite simple and, as we discuss in Section 3, it can be easily extended to more general settings. Unfortunately, although the approach is usually quite effective, it has some unfavorable properties that can slow down branch-and-bound based MIP solvers. These properties stem from the way the formulation is affected by standard variable branching in a branch-and-bound algorithm. While fixing a given binary variable y_i to one (usually denoted *up-branching*) fixes all other variables to zero, fixing the same variable to zero (usually denoted *down-branching*) does not impose any constraints on the other variables (unless $n = 2$). It is well-known that this behavior leads to unbalanced branch-and-bound trees and may result in long solution times.

To this date, there have been three central approaches to dealing with the branching issues that arise in formulations based on D^n. The first approach replaces the usual variable branching strategy with a more sophisticated constraint branching scheme [1]. For instance, Beale and Tomlin [2] introduced SOS1 branching as a very effective branching scheme for D^n and other related sets. Unfortunately, if SOS1 branching is not implemented using modern techniques, it can be slower than variable branching in state-of-the-art solvers [3]. One way to circumvent this problem is to use binary variables to simulate a specialized constraint branching scheme [4, 5]. The second approach is normally employed in scheduling problems where selecting option i corresponds to executing a certain activity in period i. In this setting, we can use an alternative set of binary variables that instead indicate whether the activity is executed in period i or before. These new variables are sometimes called *by-variables* and lead to balanced branch-and-bound trees that can significantly reduce solution times [6–12]. The third approach proposes to model n alternatives with a number of binary variables that is logarithmic in n [5, 13–17]. This approach tends to generate more balanced branch-and-bound trees, but, as we will see in Section 2, they are not immune to imbalance issues either. Nevertheless, formulations that are built using a logarithmic number of binary variables usually provide a significant computational advantage.

The encoding formulations that we describe in Section 2 provide a general framework in which all three approaches can be viewed. In Section 3, we use this framework to come up with an incremental formulation for the finite union of polyhedra with identical recession cones. We end the paper with a note on the selection of encodings in Section 4. In the remainder, we let $[n] := \{1, \ldots, n\}$ with the understanding that $[0] = \emptyset$. We use $\mathbf{0}^n \in \mathbb{R}^n$ to refer to the n-dimensional vector of all zeros and $e^{i,n} \in \mathbb{R}^n$ to refer to the i^{th} n-dimensional standard unit vector.

2. Encoding Formulations and Branch-and-Bound

The basis for our analysis will be a reformulation of D^n which has been part of the mathematical programming folklore since at least 1972 [13] and which has recently been re-discovered by several authors [5, 15, 17]. This formulation requires a set of vectors $\left\{b^i\right\}_{i=1}^n \subseteq \{0,1\}^m$ such that $b^i \neq b^j$ for any $i \neq j$. For any such set, it models $y \in D^n$ as

$$\sum_{i=1}^n y_i = 1, \quad u = \sum_{i=1}^n b^i y_i, \quad u \in \{0,1\}^m, \quad y \in \mathbb{R}^n_+. \tag{2}$$

A key theoretical strength of (2) is the tightness of its LP relaxation. In this regard, a MIP formulation has the strongest possible property when all extreme points of its LP relaxation obey the corresponding integrality requirements. Such formulations are referred to as *locally ideal* by Padberg [18] and Padberg and Rijal [19]. We note here that (2) is locally ideal for any choice of $\left\{b^i\right\}_{i=1}^n \subseteq \{0,1\}^m$ as long as $b^i \neq b^j$ for any $i \neq j$ [16, 17].

Formulation (2) may seem wasteful as it contains more variables and constraints than the original definition of D^n. However, depending on the choice of $\left\{b^i\right\}_{i=1}^n$, (2) can present a computational advantage by significantly reducing the number of binary variables or by leading to more balanced branch-and-bound trees. One reason for this advantage stems from the fact that variable branching on u induces a constraint branching scheme on y in the spirit of [4, 5]. More specifically, we have that up-branching on the variable u_j fixes $y_i = 0$ for all i such that $b^i_j = 0$ while down-branching on u_j fixes $y_i = 0$ for all i such that $b^i_j = 1$. The effectiveness of this induced branching scheme will of course depend on the choice of $\left\{b^i\right\}_{i=1}^n$. We now describe three such choices and analyze the branching schemes they induce. We will refer to the resulting formulations as *encoding formulations* since one can think of $\left\{b^i\right\}_{i=1}^n$ as an encoding of a selection among the alternatives in the sense that $u = b^i$ if and only if alternative i is selected.

We obtain the simplest encoding when we set $m = n$ and $b^i = e^{i,n}$. In this case, (2) reduces to

$$\sum_{i=1}^{n} y_i = 1, \quad u_i = y_i, \ \forall\, i \in [n], \qquad (3)$$
$$u \in \{0,1\}^n, \quad y \in \mathbb{R}^n_+,$$

which is nothing more than a redundant reformulation of D^n. We can easily see that the induced branching scheme is just the standard variable branching on y and, hence, we do not obtain any benefits. We refer to this choice as the *unary encoding formulation*.

A more meaningful choice appears in the development of logarithmic-sized formulations. For ease of exposition, let us assume that $n = 2^k$ for some positive integer $k \in \mathbb{Z}_+$ although the approach can easily be adapted to more general cases as well. In this setting, we can simply let $m = k$ and $\left\{b^i\right\}_{i=1}^{n} = \{0,1\}^m$. This choice transforms (2) into a formulation with a logarithmic number of binary variables which was the original case studied in [5, 13, 15, 17]. The induced constraint branching scheme is not quite clear, but we can easily see that it creates balanced branch-and-bound trees by observing that we have

$$\left|\left\{i \in [n] : b^i_{j'} = 0,\ b^i_j = \overline{u}_j,\ \forall\, j \in J\right\}\right| =$$
$$\left|\left\{i \in [n] : b^i_{j'} = 1,\ b^i_j = \overline{u}_j,\ \forall\, j \in J\right\}\right|$$

for all $j' \in [m]$, $J \subseteq [m] \setminus \{j'\}$ and $\overline{u} \in \{0,1\}^m$. Here, J can be considered as the subset of $\left\{u^j\right\}_{j=1}^{m}$ which has been fixed to some value at a particular node of the branch-and-bound tree. Together with the reduced number of variables, this property can lead to a significant computational advantage. We refer to this choice as the *binary encoding formulation*.

While the binary encoding formulation induces a non-trivial branching scheme, as noted in [5], it does not induce the traditional SOS1 constraint branching scheme. To construct a formulation that induces SOS1 branching on y, we can set $m = n - 1$, $b^1 = \mathbf{0}^m$ and $b^i = \sum_{j=1}^{i-1} e^{j,m}$ for all $i \in \{2, \ldots, n\}$. In this case, we have that up-branching on the variable u_j fixes $y_i = 0$ for all $i \leq j$ while down-branching on u_j fixes $y_i = 0$ for all $i > j$. Thus, we recover precisely the SOS1 branching scheme of Beale and Tomlin [2]. We refer to this choice as the *incremental encoding formulation* because it can be used to construct a formulation for piecewise-linear functions that is known as the incremental model in the literature.

To illustrate the potential advantage of the incremental encoding formulation, we study the behavior of a simple branch-and-bound algorithm on the different formulations of the following simple problem. For a given $a \in (\mathbb{Z} \setminus \{0\})^n$ such that $a_i < a_{i+1}$ for all $i \in [n-1]$, consider the problem

$$\begin{aligned} \min \quad & |x| && \text{(4a)} \\ \text{s.t.} \quad & && \\ & x \in \{a_1, \ldots, a_n\}. && \text{(4b)} \end{aligned}$$

We can use (2) to model the discrete alternatives constraint (4b) and usual linear programming tricks to linearize the objective function. Applying these techniques leads us to the MIP formulation of (4) given by

$$\begin{aligned} \min \quad & t && \text{(5a)} \\ \text{s.t.} \quad & && \\ & x \leq t, \quad -x \leq t && \text{(5b)} \\ & x = \sum_{i=1}^{n} a_i y_i, \quad \sum_{i=1}^{n} y_i = 1, \quad y \in \mathbb{R}_+^n, && \text{(5c)} \\ & u = \sum_{i=1}^{n} b^i y_i, \quad u \in \{0,1\}^m. && \text{(5d)} \end{aligned}$$

Note that, in (5), we have modeled the discrete alternatives constraint and linearized the objective function independently. While it is possible to construct a stronger formulation for this problem by considering both aspects at the same time, we refrain from this because our intent here is to evaluate the effectiveness of the formulations as they would be used in practice where different portions of an optimization problem are modeled independently and then combined. The following proposition shows that the incremental encoding formulation never requires branching on more than a single variable to solve this problem. In contrast, there are choices of n and a for which the unary and binary encoding formulations may need branching on up to $n/2$ and $\log_2 n$ variables, respectively.

Proposition 2.1.

1. *For any n and choice of $a \in (\mathbb{Z} \setminus \{0\})^n$, the incremental encoding version of* (5) *can be solved by a branch-and-bound algorithm by branching on a single variable.*

2. *If n is even, $a_i < 0$ for all $i \leq n/2$ and $a_i > 0$ for all $i > n/2$, then a branch-and-bound algorithm solving the unary encoding version of* (5) *requires branching on at least $n/2$ variables.*
3. *If n is a power of two, $a_i < 0 < a_n$ for all $i \leq n-1$, then a branch-and-bound algorithm solving the binary encoding version of* (5) *requires branching on at least $\log_2 n$ variables.*

Proof. Let $i^* \in [n]$ be such that $a_{i^*} < 0 < a_{i^*+1}$. It is clear that the optimal solution to (4) is either $x = a_{i^*}$ or $x = a_{i^*+1}$. Recall that, in the incremental encoding formulation, we have $b^i_{i^*} = 0$ for all $i \leq i^*$ and $b^i_{i^*} = 1$ for all $i > i^*$. Hence, up-branching on the single variable u_{i^*} fixes $y_i = 0$ for all $i \leq i^*$ while down-branching on u_{i^*} fixes $y_i = 0$ for all $i > i^*$. When projected on to the x-space, the LP relaxation of (5) in the first branch corresponds to the interval $[a_{i^*+1}, a_n]$ and has the optimal solution $x = a_{i^*+1}$. Similarly, the LP relaxation in the second branch projects down to $[a_1, a_{i^*}]$ and has the optimal solution $x = a_{i^*}$. This proves Claim 1.

Now, suppose a branch-and-bound algorithm can solve the unary encoding version of (5) by branching on the variables $\{u_i\}_{i\in K}$ where $K \subset [n]$ and $|K| < n/2$. As we have observed before, up-branching on the variable u_i fixes $y_i = 1$ and $y_j = 0$ for all $j \neq i$ in the unary encoding formulation. Hence, the optimal value of (5) at any leaf reached by up-branching on a variable u_i must equal $|a_i| > 0$. On the other hand, the LP relaxation of (5) at the leaf reached by down-branching on all $\{u_i\}_{i\in K}$ has the optimal value 0 since there exist distinct $i', j' \in [n] \setminus K$ such that $i' \leq n/2$ and $j' > n/2$ and the projection of the LP relaxation on to the x-space contains 0. This leaf could have been pruned by neither integrality nor bound and must still be active. This contradiction proves Claim 2.

The proof of Claim 3 follows an outline similar to that of Claim 2. Let $l := \log_2 n$ and suppose a branch-and-bound algorithm can solve the binary encoding version of (5) by branching on the variables $\{u_i\}_{i\in K}$ where $K \subset [l]$. At termination, the branch-and-bound tree has a leaf at which $u = b^n$ is a feasible assignment. Let $j' \in [l] \setminus K$. Let $i' \in [n]$ be such that $b^{i'}_{j'} = 1 - b^n_{j'}$ and $b^{i'}_j = b^n_j$ for all $j \neq j'$. It follows that $u = b^{i'}$ is a feasible assignment at this leaf as well. The fact that $a_{i'} < 0$ implies that the projection of the LP relaxation of (5) on to the x-space at this leaf contains 0. Again, we conclude that this leaf could have been pruned by neither integrality nor bound. This completes the proof of Claim 3. □

3. Incremental Formulations

An interesting behavior of the incremental encoding formulation is that it naturally induces the order $u_1 \geq u_2 \geq \ldots \geq u_{n-1}$. This ordering condition is also shared by the *by-variable* formulations [6–12], which explains why they lead to balanced branch-and-bound trees. Moreover, it is explicitly imposed in other more complicated ad-hoc formulations for piecewise-linear functions [20–23] and probabilistic constraints [24, 25]. We now show that all these formulations can be obtained through a generic procedure.

The first step in this procedure is to note that we can use D^n to formulate more general discrete alternatives by using the following formulation introduced by Lowe [26] and Jeroslow and Lowe [27].

Proposition 3.1 ([26, 27]). *Let $\{P_i\}_{i=1}^n$ be a finite family of polyhedra such that there exists $\left\{r^k\right\}_{k=0}^{R-1} \subset \mathbb{R}^d$ and $\left\{v_i^j\right\}_{j=0}^{V_i-1} \subset \mathbb{R}^d$ for $i \in [n]$ such that $P_i := \operatorname{conv}\left\{v_i^j\right\}_{j=0}^{V_i-1} + \operatorname{cone}\left\{r^k\right\}_{k=0}^{R-1}$ for all $i \in [n]$. Then a MIP formulation of $x \in \bigcup_{i=1}^n P_i$ is given by*

$$x = \sum_{i=1}^{n} \sum_{j=0}^{V_i-1} \lambda_i^j v_i^j + \sum_{k=0}^{R-1} \mu^k r^k, \tag{6a}$$

$$\sum_{j=0}^{V_i-1} \lambda_i^j = y_i, \ \forall\, i \in [n], \tag{6b}$$

$$\mu \geq 0, \quad \lambda_i \geq 0, \ \forall\, i \in [n], \quad y \in D^n. \tag{6c}$$

Formulation (6) is locally ideal [26–28]. We can combine this standard formulation with the generic encoding formulation (2) of D^n to obtain the following generalization of a formulation for piecewise-linear functions introduced in [16].

Corollary 3.2. *Let $\{P_i\}_{i=1}^n$ be a finite family of polyhedra such that there exists $\left\{r^k\right\}_{k=0}^{R-1} \subset \mathbb{R}^d$ and $\left\{v_i^j\right\}_{j=0}^{V_i-1} \subset \mathbb{R}^d$ for $i \in [n]$ such that $P_i := \operatorname{conv}\left\{v_i^j\right\}_{j=0}^{V_i-1} + \operatorname{cone}\left\{r^k\right\}_{k=0}^{R-1}$ for all $i \in [n]$. Then a MIP formulation of $x \in \bigcup_{i=1}^n P_i$ is given by*

$$x = \sum_{i=1}^{n} \sum_{j=0}^{V_i-1} \lambda_i^j v_i^j + \sum_{k=0}^{R-1} \mu^k r^k, \tag{7a}$$

$$\sum_{i=1}^{n} \sum_{j=0}^{V_i-1} \lambda_i^j = 1, \quad u = \sum_{i=1}^{n} \sum_{j=0}^{V_i-1} b^i \lambda_i^j, \tag{7b}$$

$$\mu \geq 0, \quad \lambda_i \geq 0, \forall\, i \in [n], \quad u \in \{0,1\}^m . \tag{7c}$$

The fact that (7) is locally ideal follows directly from the similar result on (2). For the specific case of the incremental encoding, we can use simple algebraic manipulations to obtain the following formulation which explicitly considers the order property among the u variables.

Proposition 3.3. *Let $\{P_i\}_{i=1}^n$ be a finite family of polyhedra such that there exists $\left\{r^k\right\}_{k=0}^{R-1} \subset \mathbb{R}^d$ and $\left\{v_i^j\right\}_{j=0}^{V_i-1} \subset \mathbb{R}^d$ for $i \in [n]$ such that $P_i := \operatorname{conv}\left\{v_i^j\right\}_{j=0}^{V_i-1} + \operatorname{cone}\left\{r^k\right\}_{k=0}^{R-1}$ for all $i \in [n]$. Then a MIP formulation of $x \in \bigcup_{i=1}^n P_i$ is given by*

$$\begin{aligned} x = v_1^0 &+ \sum_{i=1}^{n-1} u_i \left(v_{i+1}^0 - v_i^{V_i-1}\right) \\ &+ \sum_{i=1}^{n} \sum_{j=1}^{V_i-1} \delta_i^j \left(v_i^j - v_i^0\right) + \sum_{k=0}^{R-1} \mu^k r^k, \end{aligned} \tag{8a}$$

$$u_i \leq \delta_i^{V_i-1}, \ \forall\, i \in [n-1], \tag{8b}$$

$$\sum_{j=1}^{V_{i+1}-1} \delta_{i+1}^j \leq u_i, \ \forall\, i \in [n-1], \tag{8c}$$

$$\sum_{j=1}^{V_1-1} \delta_1^j \leq 1, \tag{8d}$$

$$\mu \geq 0, \quad \delta_i \geq 0, \forall\, i \in [n], \quad u \in \{0,1\}^{n-1} . \tag{8e}$$

Proof. We exhibit a bijection which preserves the values of the x variables between points satisfying (8) and the incremental encoding version of (7). The claim then follows from Corollary 3.2. In the rest of the proof, we refer to the incremental encoding version of (7) as simply (7). Let $(\overline{x}, \overline{u}, \overline{\lambda}, \overline{\mu})$ be a solution to (7). Define $\overline{u}_0 := 1$ and $\overline{u}_n := 0$ and observe that (7b) can be rewritten as

$\overline{u}_i = \sum_{l=i+1}^{n} \sum_{j=0}^{V_i-1} \overline{\lambda}_l^j$ for all $i \in \{0, 1, \ldots, n-1\}$ when the vectors $\left\{b^i\right\}_{i=1}^n$ are chosen according to incremental encoding. Let $\overline{\delta}_i^j := \overline{\lambda}_i^j$ for all $j \in [V_i - 2]$ and $\overline{\delta}_i^{V_i-1} := \overline{u}_i + \overline{\lambda}_i^{V_i-1}$ for all $i \in [n]$. The validity of (8b)-(8e) for $(\overline{x}, \overline{u}, \overline{\delta}, \overline{\mu})$ follows from the definition of $\overline{\delta}$ and the non-negativity of $\overline{\lambda}$. To verify that (8a) also holds, let $\overline{r} = \sum_{k=0}^{R-1} \overline{\mu}^k r^k$ and observe

$$\begin{aligned}
\overline{x} &= \sum_{i=1}^{n} \sum_{j=0}^{V_i-1} \overline{\lambda}_i^j v_i^j + \overline{r} \\
&= \sum_{i=1}^{n} \left(\left(\overline{u}_{i-1} - \overline{u}_i - \sum_{j=1}^{V_i-1} \overline{\lambda}_i^j \right) v_i^0 + \sum_{j=1}^{V_i-1} \overline{\lambda}_i^j v_i^j \right) + \overline{r} \\
&= \sum_{i=1}^{n} \left((\overline{u}_{i-1} - \overline{u}_i) v_i^0 + \sum_{j=1}^{V_i-1} \overline{\lambda}_i^j (v_i^j - v_i^0) \right) + \overline{r} \\
&= v_1^0 + \sum_{i=1}^{n-1} \overline{u}_i (v_{i+1}^0 - v_i^0) + \sum_{i=1}^{n} \sum_{j=1}^{V_i-1} \overline{\lambda}_i^j (v_i^j - v_i^0) + \overline{r} \\
&= v_1^0 + \sum_{i=1}^{n-1} \overline{u}_i (v_{i+1}^0 - v_i^0) \\
&\quad + \sum_{i=1}^{n} \left(\sum_{j=1}^{V_i-2} \overline{\delta}_i^j (v_i^j - v_i^0) + (\overline{\delta}_i^{V_i-1} - \overline{u}_i)(v_i^{V_i-1} - v_i^0) \right) + \overline{r} \\
&= v_1^0 + \sum_{i=1}^{n-1} \overline{u}_i (v_{i+1}^0 - v_i^{V_i-1}) + \sum_{i=1}^{n} \sum_{j=1}^{V_i-1} \overline{\delta}_i^j (v_i^j - v_i^0) + \overline{r}.
\end{aligned}$$

To prove the reverse inclusion, let $(\tilde{x}, \tilde{u}, \tilde{\delta}, \tilde{\mu})$ be a solution to (8). Define $\tilde{u}_0 := 1$ and $\tilde{u}_n := 0$ and let $\tilde{\lambda}_i^j := \tilde{\delta}_i^j$ for all $j \in [V_i - 2]$, $\tilde{\lambda}_i^{V_i-1} := \tilde{\delta}_i^{V_i-1} - \tilde{u}_i$ and $\tilde{\lambda}_i^0 := \tilde{u}_{i-1} - \sum_{j=1}^{V_i-1} \tilde{\delta}_i^j$ for all $i \in [n]$. It is not difficult to see that these definitions imply $\tilde{u}_i = \sum_{l=i+1}^{n} \sum_{j=0}^{V_i-1} \tilde{\lambda}_l^j$ for all $i \in \{0, 1, \ldots, n-1\}$. Furthermore, the equalities in the first part of the proof continue to hold. Thus, $(\tilde{x}, \tilde{u}, \tilde{\lambda}, \tilde{\mu})$ satisfies (7a)-(7b). To complete the proof, it is enough to show that $\tilde{\lambda}$ is non-negative. However, this follows directly from its definition and (8b)-(8c). □

Formulation (8) is locally ideal. To see this, observe that the linear mapping defined in the proof of Proposition 3.3 is in fact a bijection between points satisfying the *LP relaxations* of (8) and the incremental encoding version of (7). Given any solution to the LP relaxation of (8) in which the u variables have fractional values, this mapping associates with it a solution to the LP relaxation of (7) with the same u values. However, this cannot be an extreme point solution to the LP relaxation of (7) by the simple fact that (7) is locally ideal. Hence, it can be expressed as the convex combination of other solutions to the LP relaxation of (7). Mapping these solutions back to (8) and using the continuity of linear mappings shows that the LP relaxation of (8) cannot have any extreme point solutions that violate the integrality restrictions.

Formulation (8) generalizes a formulation introduced by Wilson [23] for piecewise-linear functions in two ways. First, it presents a direct extension from piecewise-linear functions to the union of a finite number of arbitrary polyhedra with identical recession cones. Second, it enjoys a broader scope of applicability by eliminating the need for a topological condition that was required for the validity of Wilson's formulation.

We may obtain other incremental formulations through the following rather straightforward lemma.

Lemma 3.4. *The incremental encoding formulation for $y \in D^n$ is equivalent to*

$$u \in \{0,1\}^{n-1}, \quad u_i \geq u_{i+1}, \; \forall\, i \in [n-2], \tag{9a}$$

$$y_1 = 1 - u_1, \quad y_n = u_{n-1}, \tag{9b}$$

$$y_i = u_{i-1} - u_i, \; \forall\, i \in \{2, \ldots, n-2\}. \tag{9c}$$

Given any formulation that requires $y \in D^n$, we can replace this constraint with (9a) and eliminate every occurrence of y through the relationships (9b)-(9c). For instance, following this procedure leads us to the formulation for probabilistic constraints studied in [24, 28] from a standard formulation introduced in [29–31]. We refer the readers to [28] for details on this specific transformation.

4. Creating and Selecting Encodings

In this paper, we have concentrated on unary, binary and incremental encodings. However, encoding formulations in the spirit of (7) can be constructed for any selection of distinct binary vectors $\left\{b^i\right\}_{i=1}^n$. Furthermore, even for the incremental and binary encodings, there is still a choice to be made. For instance, there is more than one way to order all vectors in $\{0,1\}^m$ to define $\left\{b^i\right\}_{i=1}^n$ and different orders lead to different versions of the binary encoding formulation. Similarly, while the order in the definition of $\left\{b^i\right\}_{i=1}^n$ is fixed for the incremental formulation, if we change the order in which the polyhedra $\{P_i\}_{i=1}^n$ are indexed, we will also obtain different variants of the incremental version of (7). It is beyond the scope of this paper to analyze general encoding construction schemes, but to illustrate the potential variety of encodings, we now present a hierarchy of encodings that encompasses the incremental encoding in one extreme and the binary encoding in the other.

To simplify presentation, we assume in the rest of this section that n is a power of 2. The hierarchy we propose is indexed by $h \in \{0, 1, \ldots, \log_2 n\}$, which can be considered as the number of bits of $\left\{b^i\right\}_{i=1}^n$ that behave as in the binary encoding. Let $t_l := n/2^l$. For a given value of h, we set $m = h + t_h - 1$ and define the vectors $\left\{b^i\right\}_{i=1}^n \subset \mathbb{R}^m$ in the encoding as follows: For all $l \in [h]$, let $b_l^i = 1$ if there exists $p \in [\lceil 2^{l-1} \rceil]$ such that $i \in \{2(p-1)t_l + 1, \ldots, (2p-1)t_l\}$ and $b_l^i = 0$ otherwise. For all $s \in [t_h - 1]$, let $b_{h+s}^i = 1$ if there exists $p \in [\lceil 2^{h-1} \rceil]$ such that $i \in \{2(p-1)t_h + s + 1, \ldots, (2p-1)t_h + s\}$ and $b_{i,h+s} = 0$ otherwise. Figure 1 shows this hierarchy of encodings in matrix form (we present a matrix whose columns are $\left\{b^i\right\}_{i=1}^n$) for $n = 8$ and $h \in \{0, 1, 2, 3\}$. Note that p indexes the blocks of contiguous 1's in each row in the definition above. It can be seen that setting $h = 0$ produces the incremental encoding whereas setting $h = \log_2 n$ (or even $h = \log_2 n - 1$) produces the binary encoding.

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(a) $h = 0$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

(b) $h = 1$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(c) $h = 2$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

(d) $h = 3$

Figure 1: Encodings in the hierarchy for $n = 8$.

Finally, we note that the performance of different encodings is hard to predict and can vary with specific instances. For example, [5, 16] present computational results which show that formulations for piecewise-linear functions based on the binary encoding can significantly outperform those based on the unary and incremental encodings. However, in a different set of problems, [32] reports that the incremental formulation performs better than the binary one. Fortunately, general encoding formulations in the form of (7) offer a convenient way to evaluate the performance of different encoding choices. While the incremental formulation (8) could have a computational advantage over the incremental encoding version of (7), the latter could provide a simpler preliminary test to evaluate the potential advantage of implementing the former.

References

[1] D. Ryan, B. Foster, An integer programming approach to scheduling, in: A. Wren (Ed.), Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, North-Holland, 1981, pp. 269–280.

[2] E. Beale, J. Tomlin, Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, in: OR 69: Proceedings of the Fifth International Conference on Operational Research, pp. 447–454.

[3] J. Vielma, A. Keha, G. Nemhauser, Nonconvex, lower semicontinuous piecewise linear optimization, Discrete Optimization 5 (2008) 467–488.

[4] J. Appleget, R. Wood, Explicit-constraint branching for solving mixed-integer programs, volume 12 of *Operations Research / Computer Science Interfaces Series*, Kluwer, 2000, pp. 245–261.

[5] J. Vielma, G. Nemhauser, Modeling disjunctive constraints with a logarithmic number of binary variables and constraints, Mathematical Programming 128 (2011) 49–72.

[6] D. Bricker, Reformulation of special ordered sets for implicit enumeration algorithms with applications in nonconvex separable programming, AIIE Transactions 9 (1977) 195–203.

[7] E. Lin, D. Bricker, On the calculation of true and pseudo penalties in multiple choice integer programming, European Journal of Operational Research 55 (1991) 228–236.

[8] W. Ogryczak, A note on modeling multiple choice requirements for simple mixed integer programming solvers, Computers & Operations Research 23 (1996) 199–205.

[9] C. Souza, L. Wolsey, Scheduling Projects with Labour Constraints, Relatório Técnico IC-97-22, IC - UNICAMP, 1997.

[10] C. Cavalcante, C. C. de Souza, M. Savelsbergh, Y. Wang, L. Wolsey, Scheduling projects with labor constraints, Discrete Applied Mathematics 112 (2001) 27–52.

[11] R. Möhring, A. Schulz, F. Stork, M. Uetz, On project scheduling with irregular starting time costs, Operations Research Letters 28 (2001) 149–154.

[12] E. Lin, D. Bricker, Connecting special ordered inequalities and transformation and reformulation technique in multiple choice programming, Computers & Operations Research 29 (2002) 1441–1446.

[13] D. Sommer, Computational experience with the ophelie mixed integer code, 1972. Talk presented at the International TIMS Conference, Houston.

[14] T. Ibaraki, Integer programming formulation of combinatorial optimization problems, Discrete Mathematics 16 (1976) 39–52.

[15] H. Li, H. Lu, Global optimization for generalized geometric programs with mixed free-sign variables, Operations Research 57 (2009) 701–713.

[16] J. Vielma, S. Ahmed, G. Nemhauser, Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions, Operations Research 58 (2010) 303–315.

[17] W. Adams, S. Henry, Base-2 expansions for linearizing products of functions of discrete variables, Operations Research 60 (2012) 1477–1490.

[18] M. Padberg, Approximating separable nonlinear functions via mixed zero-one programs, Operations Research Letters 27 (2000) 1–5.

[19] M. Padberg, M. Rijal, Location, Scheduling, Design and Integer Programming, volume 3 of *International Series in Operations Research & Management Science*, Springer, 1996.

[20] H. Markowitz, A. Manne, On the solution of discrete programming problems, Econometrica 25 (1957) 84–110.

[21] G. Dantzig, On the significance of solving linear-programming problems with some integer variables, Econometrica 28 (1960) 30–44.

[22] G. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, 1963.

[23] D. Wilson, Polyhedral Methods for Piecewise-Linear Functions, Ph.D. thesis, University of Kentucky, Lexington, KY, USA, 1998.

[24] J. Luedtke, S. Ahmed, G. Nemhauser, An integer programming approach for linear programs with probabilistic constraints, Mathematical Programming 122 (2010) 247–272.

[25] J. Vielma, S. Ahmed, G. Nemhauser, Mixed integer linear programming formulations for probabilistic constraints, Operations Research Letters 40 (2012) 153–158.

[26] J. Lowe, Modelling with Integer Variables, Ph.D. thesis, Georgia Institute of Technology, 1984.

[27] R. Jeroslow, J. Lowe, Modeling with integer variables, Mathematical Programming Studies 22 (1984) 167–184.

[28] J. Vielma, Mixed integer linear programming formulation techniques, Optimization Online (2012). `http://www.optimization-online.org/DB_HTML/2012/07/3539.html`.

[29] E. Balas, On the convex-hull of the union of certain polyhedra, Operations Research Letters 7 (1988) 279–283.

[30] R. Jeroslow, A simplification for some disjunctive formulations, European Journal of Operational Research 36 (1988) 116–121.

[31] C. Blair, Representation for multiple right-hand sides, Mathematical Programming 49 (1990) 1–5.

[32] B. Geißler, A. Martin, A. Morsi, L. Schewe, Using piecewise linear functions for solving MINLPs, in: J. Lee, S. Leyffer (Eds.), Mixed Integer Nonlinear Programming, volume 154 of *The IMA Volumes in Mathematics and its Applications*, Springer, 2012, pp. 287–314.