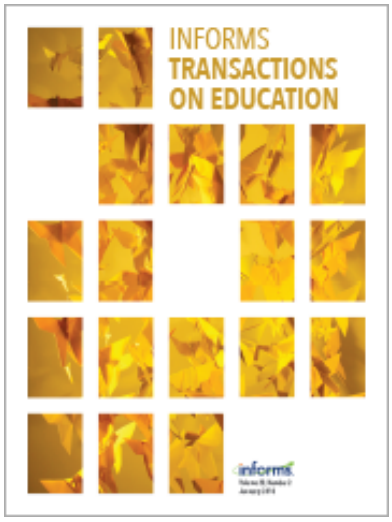

INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Teaching Use of Binary Variables in Integer Linear Programs: Formulating Logical Conditions

Scott P. Stevens, Susan W. Palocsay

To cite this article:
Scott P. Stevens, Susan W. Palocsay (2017) Teaching Use of Binary Variables in Integer Linear Programs: Formulating Logical Conditions. INFORMS Transactions on Education 18(1):28-36. https://doi.org/10.1287/ited.2017.0177

http://pubsonline.informs.org/journal/ited/

INFORMS TRANSACTIONS ON EDUCATION
Vol. 18, No. 1, September 2017, pp. 28–36
ISSN 1532-0545 (online)

Teaching Use of Binary Variables in Integer Linear Programs: Formulating Logical Conditions

Scott P. Stevens,[a] **Susan W. Palocsay**[a]

[a] Computer Information Systems and Business Analytics Department, James Madison University, Harrisonburg, Virginia 22807
Contact: stevensp@jmu.edu (SPS); palocsw@jmu.edu (SWP)

Received: August 3, 2016
Revised: February 7, 2017; March 18, 2017; March 26, 2017
Accepted: April 4, 2017
Published Online in Articles in Advance: August 31, 2017

https://doi.org/10.1287/ited.2017.0177

Abstract. Binary variables are often needed in linear programming models to indicate whether particular alternatives should be implemented and to impose logical relations among decisions. However, it is usually not obvious to students how to use binary variables to transform conditional statements of logic into linear relations. We propose to address this difficulty with a simple two-step approach. It provides rules for decomposing a conditional requirement into a group of elementary implications and then translating each of these into linear constraints. Pre- and post-test results from a sample of undergraduate business students are presented to support the effectiveness of this pedagogical approach.

Supplemental Material: Supplemental files available online at https://doi.org/10.1287/ited.2017.0177.

Keywords: optimization modeling • integer programming • binary variables • logical conditions

1. Introduction

A quantitative approach to problem solving involves more than just numbers: It requires creating a mathematical model of the situation being analyzed. Unfortunately, this modeling step is frequently a major stumbling block for students. A comprehensive study by Powell and Willemain (2007) revealed a number of obstacles. For example, students place too much emphasis on the numerical data given in the problem. They also underuse abstract variables and ignore the relationships between those variables. These findings are complemented by discoveries from mathematical educators (e.g., Lester 2013) and neuroscientists investigating algebraic problem solving (Lee et al. 2007).

Powell and Willemain suggest that students in Operations Research/Management Science (OR/MS) classes should be taught how to model without data, to think abstractly, to monitor their progress, and to follow a systematic process. How to accomplish these teaching goals is far from obvious, even when the range of modeling problems is narrowed to linear programming (LP) or mixed integer programming (MIP) problems. LP formulation has long been recognized as one of the most difficult topics to teach in OR/MS (Murphy and Panchanadam 1999). In a recent study, Williams et al. (2016) reported that only 26% of the students in an undergraduate management science course correctly formulated a two-decision product mix LP. Similarly, we have observed poor performance for LP models involving 0–1 variables. Yet an extensive literature search did not uncover any empirical evidence of specific problems related to integer programming (IP) formulation. As a result, we began investigating obstacles to modeling logical conditions and searching for ways to help our students overcome them.

Initially, we concentrated on the question of why students struggle with LP models involving 0–1 variables. Certainly, arithmetic or algebra with such variables is straightforward, i.e., they can be treated as any other variables. The difficulty in working with them appears to arise because a binary variable in a pure or mixed IP almost always codes for a Boolean variable. A Boolean variable records the truth or falsity (the "truth value") of some proposition, like "Factory A is used this month." We could represent this truth value with a 0–1 variable (e.g., A) by following the convention that the variable equals 1 if the proposition is true and a 0 if the proposition is false. With a bit of ingenuity, it is possible to write *algebraic* relations among such binary variables that code for *logical* relations among propositions.

Some relations of this type can be thought of as restrictions of ordinary linear constraints to variables that are binary. These are fairly straightforward to model, e.g., "Both A and B are true" ($A + B = 2$) and "At least one of C, D and E is true" ($C + D + E \geq 1$). For other types of logical relations, especially those involving implication, this approach breaks down. In representing these relations, students must examine binary variables while alternating between two different perspectives. The variables represent numbers and satisfy constraints that are algebraic relations. Simultaneously, they represent the truth values of logical propositions. It is not surprising, then, that Williams (2013) found that students were unable to "capture the required restrictions by 0–1 variables with logical constraints" and noted that "it is very easy to model a restriction incorrectly" (p. 177). He also found a general lack of awareness of the modeling power provided by binary variables.

We partially address this challenge by focusing on a broad class of logical implications. We provide a parsimonious system that consists of three easily remembered rules, yet which is powerful enough to handle many compound implications, e.g., "If both A and B are true, then at least one of C, D and E is true." Two "decomposition rules" break up if-then statements containing AND or OR connectives into sets of less complicated if-then statements. Subsequent application of a "translation rule" results in one or more binary constraints that encode the desired logical relations.

For completeness and to support our pedagogical approach, we also include other well known 0–1 logical constraint forms. Finally, we compare student proficiency before and after exposure to our system to collect data on formulation difficulties with logical conditions and to demonstrate the system's observed effectiveness. Plans for further extensions to the methodology are also discussed. For background purposes, we begin with an overview of relevant literature on instructional guidance for modeling applications that use binary variables.

2. Literature Review

Stevens and Palocsay (2004) linked LP formulation issues to cognitive difficulties and documented errors in students' algebra word problem solving strategies. They presented a system in which a student generates an LP model from a word problem by first creating a "measurable quantity representation" (p. 43) of the problem and then applying a mechanical rule to "translate" this representation into its mathematical form. Classroom testing showed that students taught with this methodology performed significantly better than their more conventionally taught peers, both on model formulation and on post-optimality analysis. Brown and Dell (2007) used a similar approach to the modelling challenge , providing students with "formulettes." These rules allow a variety of conditions expressed in natural language to be converted into their mathematical counterparts. Their list includes conditions that are modeled in terms of binary variables, which can take on only the values of 0 or 1. In fact, fully half of their 24 formulettes deal with these simple 0–1 variables.

Introductory OR/MS textbooks typically present binary variables in the context of specialized applications of IP modeling. Common examples include capital budgeting, fixed-cost, and location decisions (e.g., Anderson et al. 2016, Winston and Albright 2016). More advanced course materials often address tasks such as the knapsack problem, set packing, partitioning, and covering. These are usually considered from the perspective of combinatorial optimization with an emphasis on efficient solution algorithms (Nemhauser and Wolsey 1999). Anderson et al. (2016) describe straightforward constraint forms using binary variables for multiple-choice, mutually exclusive, k out of n alternatives, and co-requisite requirements.

Williams (2013) gives a formal treatment of common propositional connectives in Boolean notation and demonstrates translation into IP constraints via a series of examples. He also discusses (separately) computational issues associated with these formulations in terms of number of variables and constraints. Chen et al. (2010) provide additional coverage of practical applications using 0–1 variables in optimization models. They address an extensive set of problem features, including transformation of piecewise linear functions and functions with products of binary and continuous variables, as well as non-simultaneous constraints. However, given the limited mathematical background of our students, we found these expositions to be too abstract for teaching purposes. Instead, our focus is the modeling skills of the more typical undergraduate student, one who lacks expert knowledge of logic and IP theory. Based on our extensive teaching experience in an undergraduate business program, we believe that the methodology presented here will allow more students to gain proficiency in IP modeling with binary variables.

The system we propose can be viewed as an extension of the work of Brown and Dell (2007) whose IP formulettes include rules for representing a variety of logical conditions as constraints. Our system allows many of their logical relation formulettes to be generated straightforwardly from one set of only three rules; our system also substantially extends and generalizes the results of those formulettes. In addition, we offer a single, simple rule that encapsulates and generalizes their formulettes for variable upper bounds and semi-continuous variables.

3. Rules for Using Binary Variables in Integer Linear Programs

3.1. Representing Pure Form Implications

Begin with a set of propositions $A_1, A_2, \ldots, A_n$, each of which could be true or false. For example, A_i might be "Speaker i is present." If we combine these with the logical operator AND, we create A_1 AND A_2 AND $\cdots$ AND A_n, a compound proposition that is true if and only if all of the individual A_i are true. We will refer to this compound by the shorthand "ALL A_i." In our example, ALL A_i would mean that all of the speakers are present. Similarly, we could have used the OR operator instead of AND, creating "A_1 OR A_2 OR $\cdots$ OR A_n," a compound proposition that is true if and only if at least one of the A_i is true. In our example, this would mean that at least one of the speakers is present. We will refer to this second compound proposition with the shorthand "SOME A_i." Of course, there may only be a single A, in which case "ALL A_i" and "SOME A_i" have the same meaning as just "A."

Next we add to our collection the implication operator $\rightarrow$. $A \rightarrow B$ (read "A implies B" or "If A, then B") is true unless A is true and B is false. If we use $\rightarrow$ to join two compounds of the SOME A_i or ALL A_i varieties, we obtain one of nine forms as indicated below:

$$\left\{\begin{array}{c} A \\ \text{SOME } A_i \\ \text{ALL } A_i \end{array}\right\} \quad \rightarrow \quad \left\{\begin{array}{c} B \\ \text{SOME } B_j \\ \text{ALL } B_j \end{array}\right\}$$

We define any of the nine forms above (such as ALL $A_i \rightarrow$ SOME B_j) as a *pure form implication* (Plastria 2002). For clarity, propositions in the left hand expression will be represented with A_i, $i = 1$ to n, while those in the right-hand expression will be represented with B_j, $j = 1$ to m. When translating propositions into constraints, we follow the usual convention of coding these propositions by binary variables with the same name, and setting the binary variable to 1 if and only if the corresponding proposition is true.

We now present a set of three simple rules that will allow any pure form implication to be translated into a set of linear constraints. In most cases we can represent a pure form implication with a single constraint. Two *decomposition* rules break down a pure form implication into a collection of simpler implications. Then the *translation* rule converts these simpler implications into linear relations among binary variables. Each rule is accompanied by an optional mnemonic that may help students in recall and application of the rule (Lesser 2011, Putnam 2015).

We begin with the decomposition rules, which eliminate SOME A_i on the left of the implication and/or ALL B_j on the right.

> Decomposition (The "SOME Left, ALL Right" Rules)
> Rule D1: If the *left* side of an implication is SOME A_i, break up the statement into n statements, one for each A_i. The right-hand side (RHS) of the implication, whatever it is, remains unchanged in this process.
> So $(A_1$ OR A_2 OR $A_3) \rightarrow (B_1$ OR $B_2)$ becomes a set of three implications, i.e., $A_1 \rightarrow (B_1$ OR $B_2)$, $A_2 \rightarrow (B_1$ OR $B_2)$, and $A_3 \rightarrow (B_1$ OR $B_2)$.
> Rule D2: If the *right* side of an implication is ALL B_j, break up the statement into m statements, one for each B_j. The left-hand side (LHS) of the implication, whatever it is, remains unchanged in this process.
> So $A \rightarrow (B_1$ AND $B_2)$ becomes $A \rightarrow B_1$ along with $A \rightarrow B_2$.
> Mnemonic: Imagine arriving late to a pizza party. "SOME *left*? ALL *right*!"

Why they work: In Rule D1, the original proposition says that if any of the A_i is true, so is the consequent. The new propositions, taken together, say exactly the same thing. In Rule D2, the original proposition says that if the premise is true, each B_j is true. Thus, these new propositions, taken together, are equivalent.

Next is the translation rule, which converts these implications into linear constraints using binary variables. Explicitly, it provides only the translation of (ALL A_i) $\rightarrow$ (SOME B_j), but it is more versatile than it appears. If $n = 1$, it provides a translation for $A \rightarrow$ (SOME B_j). If $m = 1$, it provides a translation for (ALL A_i) $\rightarrow B$. Finally, if $n = m = 1$, it translates $A \rightarrow B$.

> Translation (The "Awesome ALL—SOME" Rule)
> Rule T: The statement (ALL A_i) $\rightarrow$ (SOME B_j) can be represented by
>
> $$\Sigma A_i \leq \Sigma B_j + n - 1$$
>
> where n is the number of A_i. (Of course, this rule also applies when there is only one A or only one B.)
> Mnemonic: "Add the ALL, then add the SOME, then add the ALL count minus 1."

Why it works: If any of the A_i is false, then ΣA_i is at most $n - 1$, so the constraint puts no restrictions on the B_j, since ΣB_j is always at least 0. If all of the A_i are true, then the left side equals n, which reduces the constraint to $\Sigma B_j \geq 1$; that is, at least one B_j must be true.

Rules D1, D2, and T are all that are needed to correctly represent any pure form implication as a collection of linear expressions of binary variables, as summarized in Table 1. We demonstrate by applying them to several problems.

Example 1 (Williams 2013, p. 174). "If either of the products A or B (or both) are manufactured, then at least

Table 1. Translation of Pure Form Implications

Pure form implication	Rules	Translation result
$A \rightarrow B$	T with $m = n = 1$	$A \le B$
$A \rightarrow$ SOME B_j	T with $n = 1$	$A \le \Sigma B_j$
$A \rightarrow$ ALL B_j	D2 then T with $m = n = 1$	$A \le B_j$ for $j = 1$ to m
SOME $A_i \rightarrow B$	D1 then T with $m = n = 1$	$A_i \le B$ for $i = 1$ to n
SOME $A_i \rightarrow$ SOME B_j	D1 then T with $n = 1$	$A_i \le \Sigma B_j$ for $i = 1$ to n
SOME $A_i \rightarrow$ ALL B_j	D1 then D2 then T with $n = m = 1$	$A_i \le B_j$ for $i = 1$ to n and $j = 1$ to m
ALL $A_i \rightarrow B$	T with $m = 1$	$\Sigma A_i \le B + n - 1$
ALL $A_i \rightarrow$ SOME B_j	T	$\Sigma A_i \le \Sigma B_j + n - 1$
ALL $A_i \rightarrow$ ALL B_j	D2 then T with $m = 1$	$\Sigma A_i \le B_j + n - 1$ for $j = 1$ to m

one of the products C, D or E must also be manufactured."

This is a SOME → SOME form: SOME one of the first collection implies SOME one of the second collection. Rule D1 ("SOME left") says to break this up with each of the two left-hand events generating its own proposition as follows:

$$A \rightarrow (C \text{ OR } D \text{ OR } E)$$
$$B \rightarrow (C \text{ OR } D \text{ OR } E)$$

Rule T (with $n = 1$) applies to each of these, to directly obtain

$$A \le C + D + E$$
$$B \le C + D + E$$

Example 2. "If all three projects are to go forward, either Assembly Line 1 or Assembly Line 2 (or both) must be assigned to them."

This is an ALL → SOME form: ALL projects imply SOME line. Rule T applies with $n = 3$. Remembering the ≤ relation, we "add the ALL, then add the SOME, then add the ALL count minus 1" to obtain

$$\text{PROJ1} + \text{PROJ2} + \text{PROJ3} \le \text{LINE1} + \text{LINE2} + 2$$

Example 3. "If Job 1 and Job 2 are both started on time, then both must be finished on time."

This is an ALL → ALL form: ALL on time starts imply ALL on time finishes. By Rule D2 ("ALL *right*"), it is equivalent to

$$(\text{STARTON1 AND STARTON2}) \rightarrow \text{FINON1}$$
$$(\text{STARTON1 AND STARTON2}) \rightarrow \text{FINON2}.$$

Each of these needs Rule T with $n = 2$ to obtain

$$\text{STARTON1} + \text{STARTON2} \le \text{FINON1} + 1$$
$$\text{STARTON1} + \text{STARTON2} \le \text{FINON2} + 1.$$

In this example, there was only one term in the SOME expression.

We extend the reach of these rules by pointing out to our students that if A is a binary variable, then $1 - A$ is true precisely when A is false. $1 - A$ thus translates the complement (negative) of a proposition. Our rules treat this binary complement as if it were any other binary variable. This allows us to expand our pure form implications to include NO C_i, meaning "none of the C_i are true," because this is logically equivalent to ALL(NOT C_i), meaning "all of the C_i are untrue." We apply this observation to a requirement in a facility location decision problem as follows:

Example 4 (Williams 2009, p. 51, Paraphrased)**.** If warehouses are built at any of locations $P1$, $P2$, $P3$ or $P4$, then no warehouses can be built at locations $Q1$, $Q2$, $Q3$, $Q4$ or $Q5$.

This is a SOME $P_i \rightarrow$ NO Q_j problem, which really means SOME $P_i \rightarrow$ ALL(NOT Q_j). Recalling that NOT Q_j is translated as $1 - Q_j$, we perform a partial translation of the proposition to get the intermediate form SOME $P_i \rightarrow$ ALL$(1 - Q_j)$. Both the SOME *left* and ALL *right* decomposition rules can now be applied in succession to break this down into 20 statements of the form $P_i \rightarrow (1 - Q_j)$. Each of these becomes a constraint of the form $P_i \le 1 - Q_j$ by the translation rule with $m = n = 1$, giving an answer equivalent to Williams' formulation.

Our rules can also be used to directly produce the trio of constraints in two of Brown and Dell's formulettes for binary variables, as explained below.

Example 5 (Brown and Dell 2007, p. 155)**.** "$Z3$ can be produced if and only if a machine $Z1$ and a worker $Z2$ are available."

Two-way implications like this naturally decompose into two inferences: Here $Z3 \rightarrow (Z1 \text{ AND } Z2)$ along with $(Z1 \text{ AND } Z2) \rightarrow Z3$. Using the ALL *right* decomposition rule, the first of these becomes $Z3 \rightarrow Z1$ and $Z3 \rightarrow Z2$, and then becomes $Z3 \le Z1$ and $Z3 \le Z2$ via the translation rule. For the second one, the translation rule immediately gives $Z1 + Z1 \le Z3 + 1$.

Example 6 (Brown and Dell 2007, p. 155)**.** "Project $Z3$ can be funded if and only if project $Z1$ or project $Z2$, or both projects are funded."

Again, this is two inferences, i.e., $Z3 \rightarrow (Z1 \text{ OR } Z2)$ and $(Z1 \text{ OR } Z2) \rightarrow Z3$. For the first, a direct application

of the translation rule gives $Z3 \le Z1 + Z2$. Applying the SOME *left* decomposition rule to the second inference results in $Z1 \rightarrow Z3$ along with $Z2 \rightarrow Z3$. Once again, the translation rule (simplest case) transforms these into $Z1 \le Z3$ and $Z2 \le Z3$.

Our goal is to help students to correctly represent logical inferences as constraints involving binary variables. Although applying our rules results in a correct formulation, it says nothing about the comparative efficiency of variants in terms of solution time. For example, the decomposition and translation rules often result in a set of k inequality constraints $\text{LHS}_i \le \text{RHS}_i$ for $i = 1$ to k, all of which are identical on the LHS or the RHS. When this occurs, a mathematically-equivalent constraint could be constructed by combining all of the LHS , then all of the RHS, resulting in the $\Sigma\ \text{LHS}_i \le \Sigma\ \text{RHS}_i$.

For instance, in Example 1, we obtained the constraints $A \le C + D + E$ and $B \le C + D + E$. These combine to $A + B \le 2(C + D + E)$, and this single constraint correctly encodes the requirement that if A or B is produced, then C or D or E must be produced as well. Yet using this more compact form (rather than the two original constraints) increases the computational effort required to solve the problem by allowing more fractional solutions to the LP relaxation. For example, $A = 1$, $C = 0.5$, $B = D = E = 0$ is feasible in the linear relaxation of the combined constraint but is forbidden in the original, two-constraint representation. It is well known that there are significant advantages to solving the expanded formulation, particularly when m or n is large (Hooker 2011, Williams 2013).

The translation rule itself can be expressed in terms of a more restrictive set of constraints by introducing two new binary variables, Z_A and Z_B. Z_A is true if and only if ALL A_i is true, while Z_B is true if and only if SOME B_j is true. ALL $A_i \rightarrow$ SOME B_j can thus be "unpacked" into $Z_A \rightarrow Z_B$, $Z_A \rightarrow$ ALL A_i, ALL $A_i \rightarrow Z_A$, $Z_B \rightarrow$ SOME B_j, and SOME $B_j \rightarrow Z_B$. Using the decomposition rules, the translation rule with $n = 1$, and the translation rule with $m = 1$ results in the following equivalent representation of ALL $A_i \rightarrow$ SOME B_j:

$$Z_A \le Z_B, Z_A \le A_i \quad \text{for all } i,\ \Sigma A_i \le Z_A + n - 1,\ Z_B \le \Sigma B_j, \text{ and } B_j \le Z_B \text{ for all } j.$$

However, as we are limited to teaching introductory optimization modeling in a spreadsheet environment, we developed the translation rule to enable these students to more easily create valid constraints.

Further discussion of reformulating IPs to improve computational efficiency would be appropriate in advanced undergraduate and graduate OR courses. At this level, modeling principles can be given in the context of branch-and-bound and cutting-plane algorithms with corresponding mathematical theory. Implementation in commercial solvers then becomes an important topic, requiring additional study of methods commonly used for pre-processing (Trick 2005). Specific guidelines on state-of-the-art optimizers are also available for instruction aimed at practitioners (e.g., see techniques demonstrated by Klotz and Newman 2013).

3.2. Other Common Constraint Forms

Some essential logical constraint types are not pure form implications. We cover the most important of these in class before presenting the decomposition and translation rules. Although this material is not new and overlaps with Brown and Dell (2007), we include a brief discussion here to support the description of our pedagogy and assessment in Section 4.

"Counting" constraints: Covering, packing, partitioning and knapsack constraints. In the constraint forms discussed thus far, a binary variable codes for the truth value of a proposition, and the constraint itself represents a logical implication. We noted that this duality of the arithmetic and logical perspectives seems to be a stumbling block for many students. In some important constraint types, however, the logical perspective is not required. In these, the binary variables can be thought of as counting how many times an activity is performed, i.e., 0 or 1 time. For these constraints, the same thought process used in creating LP constraints leads directly to the IP constraints. Some of these constraint forms are shown in Table 2.

The first three extend obviously to the requirement that at least k, at most k, or exactly k of these variables are true. For a sample application, see the case cited in Winch and Yurkiewicz (2014). It contains several questions that require students to use counting constraints to construct an optimal class schedule (e.g., take exactly two of four finance courses).

In the same way, a knapsack constraint can be thought of as an ordinary LP constraint in which the variables are restricted to the values of 0 and 1. If we wish to impose the condition that the total weight of goods shipped be no more than 800 pounds, the constraint

$$100\ \text{TYPE1} + 250\ \text{TYPE2} + 400\ \text{TYPE3} + 500\ \text{TYPE4} \le 800$$

is appropriate for four kinds of goods that weigh 100, 250, 400, and 500 pounds per unit, respectively. The four variables, of course, represent the number of units of each type of good shipped. This constraint makes sense with continuous nonnegative variables, but it makes equally good sense if the variables are restricted to binary values. The latter case is a knapsack constraint in which one unit of each item is available and that unit is shipped or not shipped. The IP models in

Table 2. Some Counting Constraints with Binary Variables

English interpretation of requirement	Logical form	Constraint form
At least one of the A_i is true (covering)	SOME A_i, or (A_1 OR A_2 OR $\cdots$ OR A_n)	$\Sigma A_i \geq 1$
At most one of the A_i is true (packing)		$\Sigma A_i \leq 1$
Exactly one of the A_i is true (partitioning)	A_1 XOR A_2 (when $n = 2$)	$\Sigma A_i = 1$
All of the A_i are true	ALL A_i, or (A_1 AND A_2 AND $\cdots$ AND A_n)	$\Sigma A_i = n$
The A_i are all true or all false	$A_1 \leftrightarrow A_2$, or "A_1 if and only if A_2" (when $n = 2$)	$A_1 = A_2 = \cdots = A_n$

Gordon and Erkut (2004) provide several nice examples of counting constraints in assigning volunteers to shifts and gates at a folk festival.

Combining the knapsack idea with the observation that $1 - C$ translates the proposition "NOT C" gives a useful result. We call it the two right-hand sides (2RHS) rule:

Two Right-Hand Sides

Rule 2RHS: We want a constraint that will bound a linear expression by a constant, but the size of that constant depends on a binary variable A. The bound is to be C_T when A is true, but C_F when A is false. That is,

$$(\textit{linear expression}) \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} \begin{cases} C_T & \text{if } A = 1 \\ C_F & \text{if } A = 0 \end{cases}$$

for one of the three relational operators $\leq$, $=$, or $\geq$.

This is accomplished by replacing the right-hand side of the constraint with $C_T A + C_F(1 - A)$.

Mnemonic: "A True" value times "A" + "A False" value times "not A"

Direct computation with $A = 0$ and $A = 1$ establishes the result. The RHS can be written in other forms, but this one has a particularly nice mnemonic. The RHS is constructed as (*bound for A true*) times "A" + (*bound for A false*) times "NOT A." The resulting expression can then be simplified. This result slightly generalizes the "variable upper bound" formulette of Brown and Dell (2007), which explicitly addresses only the case when $C_F = 0$ and the relation is less-than-or equal-to.

Some problems might omit a value for C_T or for C_F. This is usually not a problem. Computationally or by applying domain knowledge about the problem, find constants C_{low} and C_{high} that provide practical lower and upper bounds on the value of the linear expression. Then, if C_T or C_F is missing, we replace the missing value by C_{high} (for a $\leq$ constraint), or by C_{low} (for a $\geq$ constraint) before applying the 2RHS rule. The values of C_{high} and C_{low} should be no more extreme than they need to be, or solution time may be compromised due to an expanded feasible region. Camm (1990) demonstrates the increasing computational requirements from setting C_{high} to unnecessarily large values in a fixed charge problem.

When an equality constraint has a missing C_T or C_F, we decompose it into a $\leq$ constraint and a $\geq$ constraint (since $X = Y$ if and only if $X \leq Y$ and $Y \leq X$). We then apply the 2RHS rule to each of these inequalities. An example of the various cases for applying this technique is provided in the supplemental material (Part I). For a more complex situation, consider the following problem:

Example 7 (Chen et al. 2010, p. 74)**.** "John's broker suggests the following two combinations: either choose two from stocks 1, 2, 3, and 4 or choose at least two stocks from stocks 3, 4, 5 and 6."

We let X and Y refer to the two combinations, respectively, and use the binary variables X, Y, and S_1 through S_6 to represent whether each option or stock is selected. Since John is limited to choosing only one of the combinations, $X + Y = 1$ is required.

If Option Y is chosen, it forces S_1 and S_2 to be 0 and at least two of the remaining stocks to be chosen. Otherwise, it imposes no restrictions. Thus:

$$S_1 + S_2 \leq \begin{cases} 0 & \text{if } Y = 1 \\ 2 & \text{if } Y = 0 \end{cases} \quad \text{and}$$

$$S_3 + S_4 + S_5 + S_6 \geq \begin{cases} 2 & \text{if } Y = 1 \\ 0 & \text{if } Y = 0 \end{cases}$$

Note the use of 2 for C_F in the first constraint and 0 for C_F in the second. These translate by the 2RHS rule to $S_1 + S_2 \leq 2(1 - Y)$ and $S_3 + S_4 + S_5 + S_6 \geq 2Y$, respectively.

If Option X is chosen, we get similar restrictions. Now it is S_5 and S_6 that must be 0, and the sum of the other four variables must be *exactly* 2:

$$S_5 + S_6 \leq \begin{cases} 0 & \text{if } X = 1 \\ 2 & \text{if } X = 0 \end{cases} \quad \text{and}$$

$$S_1 + S_2 + S_3 + S_4 = \begin{cases} 2 & \text{if } X = 1 \\ \text{anything} & \text{if } X = 0. \end{cases}$$

The first of these translates, as in the work above, to $S_5 + S_6 \leq 2(1 - X)$, but note that no single number makes sense for the C_F value in the second constraint. This equality constraint must be represented by a pair of inequalities:

$$\begin{cases} 2 & \text{if } X = 1 \\ 0 & \text{if } X = 0 \end{cases} \leq S_1 + S_2 + S_3 + S_4 \leq \begin{cases} 2 & \text{if } X = 1 \\ 4 & \text{if } X = 0. \end{cases}$$

Note that the inequalities force the sum to be exactly 2 if $X = 1$ and allow it to be anything between 0 and 4, otherwise. That is, if $X = 0$, no restriction is imposed. Two applications of the 2RHS rule convert this to

$$2X \le S_1 + S_2 + S_3 + S_4 \le 2X + 4(1 - X).$$

Obviously, the final expression could be simplified. If desired, one could also eliminate the constraint $X + Y = 1$ and the variable Y, replacing Y everywhere with $1 - X$. The constraint above is an example of a "pinching constraint," where the quantity $S_1 + S_2 + S_3 + S_4$ is caught between two bounds, the sizes of which depends on the binary variable X. This is a generalization of Brown and Dell's "semi-continuous variable" formulette, which addresses only the case in which an expression must be in the closed interval $[a, b]$ when $X = 1$ and must be 0 when $X = 0$.

4. Assessment Results

We conducted experiments in the Fall and Spring semesters of the 2015–2016 academic year to compare student performance before and after their exposure to our rule-based methodology. Institutional Review Board approval was obtained for undergraduate students taking an advanced mathematical modeling course as part of a new business analytics program. Pre-requisites were calculus, statistics, and management science courses at the introductory level. Class sizes were 15 and 18 in the Fall and Spring, respectively; a majority of the students were pursuing a major in business. Both classes were taught by the same instructor.

Students were initially given two out-of-class reading assignments, i.e., the relevant sections of Winston and Albright (2016) and a handout written by the instructor (supplemental material, Part I). The handout discussed the idea of a Boolean variable, the 2RHS rule, and how $A \rightarrow B$ can be modeled. In the next class period, the instructor gave a 10 minute presentation, briefly reviewing the handout material and giving an example of an ALL $A_i \rightarrow B$ constraint, but not presenting it as a rule. After this review, the students were given an in-class exercise.

The exercise began with a scenario about a college student and included a list of binary variable definitions relevant to her decisions and activities, such as CAMPUS representing the proposition that she goes to campus today. This was followed by 20 propositions about her behavior, such as "If Rebecca goes to CLASS 8 or CLASS 10 today, then it's MONDAY." (Binary variable names were capitalized in the propositions.) Students were to convert these propositions into the appropriate constraints. The first 10 propositions included two counting constraints and eight pure form implications. The remaining 10 propositions began with nine pure form implications that involved the NOT construction, then ended with a knapsack constraint. Students were encouraged to begin with statements 1–10 and statement 20, and to proceed to the NOT questions (11–19) as time permitted.

Table 3. Overall Assessment Results

Class	n	Original mean	Retest mean	Mean improvement	Median improvement
Fall 2015	15	6.13	12.67	6.54	6.5
Spring 2016	18	7.58	12.78	5.29	5
Combined	33	6.92	12.73	5.81	6

After this class, students were given another reading assignment, i.e., an instructor-written handout presenting the decomposition and translation rules with examples (supplemental material, Part II). When the class met the next time, the instructor presented a 10-minute summary of the new materials, then gave the students the same 20-proposition exercise to complete. Students scored one point for each proposition that they correctly represented by constraints. Results are shown in Table 3.

Based on the nonparametric Wilcoxon signed rank test, we conclude that median test score improves at least 3 points (15%) after exposure to the new technique with a p-value of less than 0.001. Three of the 33 students saw their scores drop by a single point on the retest, and one of these three admitted not doing the second reading assignment. The remaining 30 students all did better on the retest, improving by 1 to 11 points. Most students at least doubled their original scores.

During the Fall session, we observed extremely low scores for questions 11–20 on the original exercise. On average, students attempted only 1.4 of these 10 questions and answered only 0.47 of them correctly. Thinking that perhaps we had not given the students enough time for the exercise, we extended the 35 minutes allowed in the Fall to 50 minutes in the Spring. Performance on the original exercise improved in the Spring and students got a couple more of the questions from 11–20 correct, as indicated in Table 4. However, both groups had nearly the same average on the retest (12.67 and 12.78); the time constraint did not appear to be an issue in either semester . Still, it could be argued that the improvement in student performance on the retest might be due primarily to the time limitations in the fall class.

To address this, we ran two more hypothesis tests, the first restricting attention to the first 10 questions only, and the second restricting attention to the Spring semester only. Restricting to the first 10 questions, based on the Wilcoxon signed rank test, we conclude that median performance increases on the retest by more than 1 point, or 10% ($p < 0.01$). Restricting to the Spring semester only, we again conclude that median

Table 4. Breakdown of Assessment Results by Questions

Class	*Q* 1–10 original mean	*Q* 1–10 retest mean	*Q* 1–10 mean improvement	*Q* 11–20 original mean	*Q* 11–20 retest mean	*Q* 11–20 mean improvement
Fall 2015	5.67	7.73	2.07	0.47	4.93	4.47
Spring 2016	4.97	7.44	2.47	2.61	5.33	2.72
Combined	5.29	7.58	2.29	1.64	5.15	3.57

performance improves by more than 3 points (15%) after exposure to the new techniques with a p-value of 0.012.

The propositions in questions 11–20 are more complex than those in questions 1–10 due to the inclusion of NOT constructions, such as "If Rebecca does not get a LIFT and doesn't ride the BUS today, then she STAYS_HOME or WALKS." Not surprisingly, the average on this question set is lower than on questions 1–10.

Additional data were collected on time used by the students in the Spring class. Given a 50-minute limit, individual students took from 21 to 50 minutes to complete the exercise with an average of 37 minutes. Overall accuracy improved from 7.6 to 12.8 correct, about a 68% increase in the number of correct answers. We found their progress in speed and precision to be encouraging. However, we did not control for length of exposure to IP modeling by testing performance with our methodology given first.

5. Conclusion

Informal feedback from students taught using the rules for decomposition and translation was very positive. They were particularly impressed by the ease of formulating constraints for logical implications after first categorizing the left and right sides according to the template in Section 3.1. This satisfies our goal of teaching business students how to consistently approach optimization applications that require modeling with binary variables. Now that we have the pedagogical tools to make this material accessible to our students, we want to identify more real-world examples and cases with complex IP models to further develop their mastery. While our assessment results are promising, additional research to test the effectiveness of this methodology on a larger and more diverse body, perhaps using a new instrument, is appropriate.

To expand the scope of our system, we are currently developing supplementary rules for more complicated formulation tasks. For instance, it is possible to generalize the implications in Table 1 to include the option of SOME k for "at least k" of the individual A_i and/or B_i are true. We are also developing tools that allow compound logical propositions such as (A_1 OR A_2) OR (B_1 AND B_2) to be represented by a small family of constraints, using (in this example) the representations of (X OR Y) and (X AND Y) as a starting point.

Optimization is the key component of "prescriptive" analytics for recommending a preferred strategy or plan. A recent Gartner report indicates that recognition of the value of optimization among business analytics leaders is growing: "By 2018, optimization will no longer be a niche discipline; it will become a best practice for leading organizations to address a wide range of complex business decisions" (Gartner 2015, p. 1). As corporations seek a higher level of analytical maturity, expertise in optimization modeling will become critical to improving business processes. We need to ensure that OR/MS students have the requisite knowledge and skills to build and implement optimization-based solutions.

To meet these demands, students must also have proficiency in methods for solving IP models. Brown and Dell (2007) present their formulettes with binary variables before instruction on solution techniques. We believe that our rule-based system can be similarly used to effectively introduce the modeling flexibility that is possible with these variables. At a minimum, it should be followed by study of the basic branch-and-bound method with discussion of the computational difficulties associated with IPs that have 0–1 restrictions. From there, advanced OR/MS students can proceed to review of underlying theoretical concepts (Williams 2013) and recently developed techniques used by commercial solvers (Klotz and Newman 2013, Newman and Weiss 2013). For spreadsheet-based OR/MS courses, pre-processing and probing options are available in the Analytic Solver Platform (Frontline Systems 2016) add-in to improve solution times for problems with binary variables. Augmentation of logical condition modeling with awareness of potential pitfalls and the efficient use of current technology should further extend the impact of optimization on practical applications.

Acknowledgments

We are grateful to the editor, associate editor, and reviewers for their comments and suggestions.

References

Anderson DR, Sweeney DJ, Williams TA, Camm JD, Cochran JJ, Fry MJ, Ohlmann JW (2016) *An Introduction to Management Science: Quantitative Approaches to Decision Making*, 14th ed. (South-Western Cengage Learning, Mason, OH).

Brown GB, Dell RF (2007) Formulating integer linear programs: A rogue's gallery. *INFORMS Trans. Ed.* 7(2):153–159.

Camm JD (1990) Cutting big M down to size. *Interfaces* 20(5):61–66.
Chen D-S, Batson RG, Dang Y (2010) *Applied Integer Programming: Modeling and Solution* (John Wiley & Sons, Hoboken, NJ).
Frontline Systems, Inc. (2016) *Frontline Solvers Reference Guide, Version 2016-R3.*
Gartner (2015) Market guide for optimization solutions. Accessed July 23, 2017, www.gartner.com.
Gordon L, Erkut E (2004) Improving volunteer scheduling for the Edmonton folk festival. *Interfaces* 34(5):367–376.
Hooker J (2011) Formulating good MILP models. Cochran J, ed. *Wiley Encyclopedia of Operations Research and Management Science*, (John Wiley & Sons, Hoboken, NJ)
Klotz E, Newman AM (2013) Practical guidelines for solving difficult mixed integer linear programs. *Surveys Oper. Res. Management Sci.* 18(1–2):18–32.
Lee K, Lim Z, Yeong S, Ng S, Venkatraman V, Chee M (2007) Strategic differences in algebraic problem solving: Neuroanatomical correlates. *Brain Res.* 1155:163–171.
Lesser LM (2011) On the use of mnemonics for teaching statistics. *Model Assisted Statist. Appl.* 6(2):151–160.
Lester F (2013) Thoughts about research on mathematical problem-solving instruction. *Math. Enthusiast* 10(1–2):245–278.
Murphy FH, Panchanadam V (1999) Using analogical reasoning and schema formation to improve the success in formulating linear programming models. *Oper. Res.* 47(5):663–674.
Nemhauser G, Wolsey L (1999) *Integer and Combinatorial Optimization.* (John Wiley & Sons, New York).
Newman AM, Weiss M (2013) A survey of linear and mixed-integer optimization tutorials. *INFORMS Trans. Ed.* 14(1):26–38.
Plastria F (2002) Formulating logical implications in combinatorial optimization. *Eur. J. Oper. Res.* 140(2):338–353.
Powell S, Willemain T (2007) How novices formulate models. Part I: Qualitative insights and implications for teaching. *J. Oper. Res. Soc.* 58(8):983–995.
Putnam AL (2015) Mnemonics in education: Current research and applications. *Translational Issues Psych. Sci.* 1(2):130–139.
Stevens SP, Palocsay SW (2004) A translation approach to teaching linear program formulation. *INFORMS Trans. Ed.* 4(3):38–54.
Trick M (2005) Formulations and reformulations in integer programming. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Springer, Berlin), 366–379.
Williams HP (2009) *Logic and Integer Programming* (Springer, New York).
Williams HP (2013) *Model Building in Mathematical Programming*, 5th ed. (John Wiley & Sons, Chichester, UK).
Williams JAS, Rankin M, Gallamore K, Reid R (2016) Beyond model formulation: Assessment of novices graphing, interpreting, and writing about their model and solution. *INFORMS Trans. Ed.* 17(1):13–19.
Winch JK, Yurkiewicz J (2014) Class scheduling with linear programming. *INFORMS Trans. Ed.* 15(1):143–147.
Winston WL, Albright SC (2016) *Practical Management Science*, 5th ed. (Cengage Learning, Boston).