Operations Research

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

Technical Note—Solving Integer Programming Problems by Aggregating Constraints

Kenneth E. Kendall, Stanley Zionts,

OPERATIONS RESEARCH, Vol. 25, No. 2, March-April 1977

Solving Integer Programming Problems by Aggregating Constraints

KENNETH E. KENDALL
University of Wisconsin, Milwaukee, Wisconsin

STANLEY ZIONTS
State University of New York, Buffalo, New York

(Received original January 1973; final, July 1976)

Integer programming problems with bounded variables can be solved by combining the constraints into an equivalent single constraint. This note presents a refinement to earlier work that reduces the size of the coefficients in the equivalent constraint and points out advantages as well as computational considerations for solving problems by this method.

A NUMBER of methods have been proposed for reducing an integer programming problem with bounds on the variables to a single-constraint knapsack problem. These include works of Anthonisse [1, 2], Bradley [4, 5], Elmaghraby and Wig [6], Glover [7], Glover and Woolsey [8], Hammer and Rudeanu [9], Mathews [10], and Padberg [11]. The basic idea of all of these methods is to find an equivalent single-constraint problem that is ostensibly easier to solve than the original multi-constraint problem. (One of the approaches, that of Bradley [5], includes a method for combining the remaining constraint with the objective function, leaving only the bounds on variables.) In this note we present a theorem that gives a new result that appears to integrate some of the previous work, and we show that single-constraint problems having relatively small coefficients can be found by means of the result. We then use it to develop some simpler and more practical results. We conclude with some observations about the possible implementation of such methods.

1. AN OVERVIEW OF CONSTRAINT AGGREGATION SCHEMES AND A CONSTRAINT AGGREGATION THEOREM

A naive constraint aggregation scheme was proposed by Padberg [11] and independently by us. Consider a constraint set

$$Ax = b, \qquad 0 \leqq x \leqq u, \qquad x \text{ integer}, \qquad (1)$$

where A is an m by n matrix, x and u are n by 1 vectors, and b is an m by 1 vector. A, μ, and b are made up of finite real constants. Without loss of generality, we assume A, b, and u to consist only of integers. Define the vector y as a 1 by m row vector: $y = (1, M, M^2, \cdots, M^{m-1})$, where M is a sufficiently large integer. Now the following constraint set (which, except for the bounds on x, is a single constraint) is equivalent to (1):

$$yAx = yb, \qquad 0 \leqq x \leqq u, \quad x \text{ integer.} \tag{2}$$

This method generally leads to enormous coefficients in the constraints and is consequently useful only as a means for thinking about constraint aggregation schemes. We omit the proof but note that, by virtue of the bounds on the variable x and of the integrality and finiteness of the elements of the matrix A and the vector b, the procedure permits a scalar representation of a vector relationship, thereby preserving the original relationships. The less naive methods described below accomplish the same result but achieve smaller (and therefore more desirable) coefficients.

Most of the current constraint aggregation schemes combine two constraints into one and then combine a third with the resulting constraint, and so on until all constraints have been combined. A few methods combine all the constraints at one time, although the calculations are effectively the same as in a one-at-a-time series of calculations.

Let the two constraints be $\sum_{j=1}^{j=n} a_{ij}x_j = b_i$, $i = 1, 2$, with conditions $0 \leqq x_j \leqq u_j$, x_j integer. Define the function $g_i(x) = b_i - \sum_{j=1}^{j=n} a_{ij}x_j$ for any set of values x_j integer, $0 \leqq x_j \leqq u_j$. The vector x is a solution to the ith constraint if and only if $g_i(x) = 0$.

A valid set of multipliers λ_1 and λ_2 generates an equation $\lambda_1 g_1(x) + \lambda_2 g_2(x) = 0$. We are interested only in multipliers for which the latter equation implies $g_1(x) = g_2(x) = 0$. By rewriting the combined equation, we have

$$g_1(x) = (-\lambda_2/\lambda_1)g_2(x), \tag{3}$$

and by using the bounds

$$b_i - \sum_{j=1}^{j=n} a_{ij}^{+}u_j \leqq g_i(x) \leqq b_i - \sum_{j=1}^{j=n} a_{ij}^{-}u_j,$$

where $a_{ij}^{+} = \max\{a_{ij}, 0\}$ and $a_{ij}^{-} = \min\{a_{ij}, 0\}$, we see that $\lambda_1 = 1$ and $\lambda_2 > \max\{b_1 - \sum_{j=1}^{j=n} a_{1j}^{-}u_j, -b_1 + \sum_{j=1}^{j=n} a_{1j}^{+}u_j\}$ is a legitimate set of multipliers. (We may obtain another set of multipliers by exchanging subscripts.) Thus, for the smallest absolute nonzero value of $g_2(x)$ we require the value of $g_1(x)$ for equality to be outside its feasible range. Theorem 1 gives a strong though not a dominant set of multipliers.

THEOREM 1. *If* $0 \leqq x \leqq u$ *and integer, the constraints*

$$\sum_{j=1}^{j=n} a_{1j}x_j = b_1 \qquad (g_1(x) = 0) \tag{4}$$

and $\sum_{j=1}^{j=n} a_{2j}x_j = b_2 \quad (g_2(x) = 0)$ (5)

are equivalent to

$$\lambda_1 g_1(x) + \lambda_2 g_2(x) = 0 \tag{6}$$

if the integer multipliers λ_i *satisfy the following conditions:*

1. *For any integer solution values* $0 \leqq x_j \leqq u_j$, $j = 1, \cdots, n$, *either* λ_1 *does not divide* $g_2(x)$ *or* λ_2 *does not divide* $g_1(x)$.
2. λ_1 *and* λ_2 *are relatively prime. (Their greatest common divisor is one.)*

Proof. The proof that (4) and (5) imply (6) is trivial. To prove that (6) implies (4) and (5), write (6) as $g_1(x) = (-\lambda_2/\lambda_1)\,g_2(x)$. By definition the left-hand side of the above equation is integer; consequently, so is the right-hand side. Now, by virtue of condition 2, $g_2(x)/\lambda_1$ is integer, and so is $g_1(x)/\lambda_2$. But this contradicts condition 1 unless $g_1(x) = g_2(x) = 0$, thereby proving the theorem.

Theorem 1 may not appear to be much value at first glance. We may, however, use it in conjunction with the basic requirement that

$$g_1(x) \neq -(\lambda_2/\lambda_1)g_2(x) \tag{7}$$

unless both $g_1(x)$ and $g_2(x)$ are zero. Theorem 1 and (7) give us a systematic (but computationally horrible) way of enumerating to find good multipliers. For a particular set of relatively prime multipliers, we can first check to see if all solutions satisfy condition 1 of Theorem 1. If condition 1 is not satisfied, the multipliers will still be valid if we can show that (7) holds.

Methods proposed by Anthonisse [1, 2], Bradley [4], and Glover and Woolsey [8] use relatively prime values for λ_1 and λ_2 outside the range of values specified by the upper and lower bounds of each equation. These methods consequently satisfy conditions 1 and 2

On the contrary, we propose that multipliers smaller than those generated by others can be found efficiently by obtaining values of $g_1(x)$ and $g_2(x)$ that cannot occur within the range of values for $g_1(x)$ and $g_2(x)$ as specified by the bounds of each equation. Such values are valid multipliers λ_2 and λ_1, provided that they satisfy the conditions of Theorem 1.

The first method is to choose $\lambda_1 > b_2 - \sum_{j=1}^{j=n} a_{2j}^- u_j - \min_{a_{2j}\neq 0}\{|a_{2j}|\}$ and $\lambda_2 > b_1 - \sum_{j=1}^{j=n} a_{1j}^- u_j - \min_{a_{1j}\neq 0}\{|a_{1j}|\}$ such that $(b_2 - \sum_{j=1}^{j=n} a_{2j}^- u_j)/\lambda_1$ and $(b_1 - \sum_{j=1}^{j=n} a_{1j}^- u_j)/\lambda_2$ are not integers and λ_1 and λ_2 are relatively prime.

The second method is slightly more involved.

(a) Enumerate a few values of $g_i(x) = b_i - \sum_{j=1}^{j=n} a_{ij}x_j$ for $0 \leqq x_j \leqq u_j$ beginning with the solution $x_j = u_j$ if $a_{ij} < 0$ and $x_j = 0$ otherwise, so that $g_i(x)$ strictly decreases and no values in the sequence are omitted.

(Alternatively, the enumeration may be begun from the lower bound $b_i - \sum_{j=1}^{j=n} a_{ij}^{+} u_j$, making the necessary changes.) The first few values are easy to enumerate because we increase or decrease the variable whose $|a_{ij}|$ is minimum and so on, using a lexicographical ordering scheme.

(b) Choose λ_1 greater than the smallest $g_2(x)$ enumerated and λ_2 greater than the smallest $g_1(x)$ enumerated, $\lambda_1 \neq$ any $g_2(x)$, $\lambda_2 \neq$ any $g_1(x)$, λ_1 and λ_2 are relatively prime, and $\lambda_1 > (b_2 - \sum_{j=1}^{j=n} a_{2j}^{-} u_j)/2$ and $\lambda_2 > (b_1 - \sum_{j=1}^{j=n} a_{1j}^{-} u_j)/2$.

How effective these multipliers are in practice remains to be seen. To illustrate the methods, we use an example presented by Balas [3]. Since it is necessary to work with equalities, slack variables are added and the constraints are written as follows:

$$\begin{aligned}
&\min z = 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5 \\
&\quad f_1: -x_1 + 3x_2 + 5x_3 + x_4 + 4x_5 + x_6 = -2 \\
&\quad f_2: 2x_1 - 6x_2 + 3x_3 + 2x_4 - 2x_5 + x_7 = 0 \\
&\quad f_3: x_2 - 2x_3 + x_4 + x_5 + x_8 = -1 \\
&\qquad\qquad x_j = 0 \text{ or } 1 \qquad j = 1, \cdots, 5 \\
&\qquad\qquad x_j \geqq \text{ and integer} \qquad j = 6, 7, 8.
\end{aligned}$$

(A weak set of upper bounds for the slack variables is given by $(x_6, x_7, x_8) \leqq (5, 8, 1)$. A stronger set yielding better multipliers may be found by solving a linear programming problem maximizing sequentially each slack variable.)

If we represent the solution vector as $(x_1, \cdots, x_5)$, only solutions (01100) and (11100) satisfy the constraints.

Using Theorem 1, (7), and enumeration, we find that $\lambda_1 = 7$ and $\lambda_2 = 6$ are valid multipliers for combining f_1 and f_2. The combined constraint is

$$f_0 = 5x_1 - 15x_2 - 17x_3 + 5x_4 + 16x_5 + 7x_6 + 6x_7 = -14,$$

from which we generate tighter upper bounds $u_6 = 2$ and $u_7 = 3$. Further enumerating with f_0 and f_3, we find that $\lambda_0 = 1$ and $\lambda_3 = 5$ are valid multipliers that yield the aggregate constraint

$$5x_1 - 10x_2 - 27x_3 + 10x_4 - 26x_5 + 7x_6 + 6x_7 + 5x_8 = -19.$$

(Even smaller coefficients may be attainable, but we limited our search to positive multipliers.)

The first of the suggested methods gives $\lambda_1 = 9$ and $\lambda_2 = 7$. Using f_0 to represent the combined equation, we find that $\lambda_0 = 2$ and $\lambda_3 = 17$. The second of the suggested methods also gives $\lambda_1 = 9$ and $\lambda_2 = 7$. Thus, the combined constraint is

$$f_0 : 5x_1 - 15x_2 - 24x_3 + 5x_4 + 22x_5 + 9x_6 + 7x_7 = -18$$
$$(x_j \leqq 1, j = 1, \cdots, 5, x_6 \leqq 2, x_7 \leqq 3)$$

and the remaining constraint is f_3. The enumeration of a few values of $g_i(x)$ are $g_0(x) = \{21, 16, 14, 12, 11, \cdots\}$, $g_3(x) = \{1, 0, -1, -2, \cdots\}$. We therefore choose $\lambda_0 = 2$, $\lambda_3 = 13$, which yields

$$10x_1 - 17x_2 - 74x_3 + 23x_4 + 57x_5 + 18x_6 + 14x_7 + 13x_8 = -49$$

The order in which constraints are combined makes a difference. (For instance, if we first combine constraints 1 and 3 and then combine constraint 2 with the result, we generate an aggregate constraint having larger coefficients than those obtained in the example.) Thus, assuming smaller coefficients in the aggregate constraint are desirable, further study in selecting the order of the constraints in the aggregation process is appropriate.

In addition, we may take advantage of the fact that our methods 1 and 2 generate smaller multipliers when coefficients of one are not found on variables with upper bounds greater than one. A strategy of combining the first and second constraints, then the third and fourth constraints, and so on, each pair becoming a new constraint and then aggregating the resulting set of constraints, may be appropriate. Also, it would appear to be worthwhile, as proposed by Zionts [12], to scan the constraints before combining and after each aggregation to tighten upper bounds, and to determine implied values of variables.

2. DISCUSSION

All of the aggregation schemes have the same disadvantage—the coefficients become too large to be stored as an integer in a single computer word. Since it is necessary to maintain accuracy in an integer programming problem, a multiple precision package of subroutines (subroutines that store an integer in more than one computer word and execute the basic operations of add, subtract, and multiply in that manner) must be utilized. This set of subroutines would be a major part of a computer program using an aggregation approach.

One advantage of these methods is the simplicity of solving the single-constraint problem that is generated once the constraints are aggregated. We should emphasize, however, that a single-equality constraint problem is much more difficult to solve than a single-inequality constraint problem. A second major advantage derives from the recursive nature of the algorithm since it is necessary to store only two constraints in core memory at a time. Furthermore, the objective function need not be considered until all the constraints are combined into one.

The possibility of writing an efficient computer program using a constraint aggregation algorithm and a multi-precision package of subroutines is not farfetched. Even though multi-precision requires more than one computer word per coefficient, the storage requirements for the coefficients of the aggregate constraint will be much less than the storage requirements for all of the constraints of the original problem. Such a program would be particularly valuable to a user who is restricted by the amount of available core storage. This method would allow a person using a time-sharing (and consequently "core-sharing") system or a limited memory mini-computer to solve reasonably large problems that are impossible to solve using the integer programming codes currently available.

ACKNOWLEDGMENT

The authors wish to thank the referees for their helpful comments.

REFERENCES

1. J. M. Anthonisse, "A Note on Reducing a System to a Single Equation, Preliminary Report," Stichting Mathematisch Centrum, Amsterdam, December 1970.
2. J. M. Anthonisse, "A Note on Equivalent Systems of Linear Diophantine Equations," *Opns. Res.* **17**, 167–177 (1973).
3. E. Balas, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Opns. Res.* **13**, 517–546 (1965).
4. G. H. Bradley, "Heuristic Solution Methods and Transformed Integer Linear Programming Problems," Report No. 43, Department of Administrative Sciences, Yale University, New Haven, Conn., March 1971.
5. G. H. Bradley, "Transformation of Integer Programs to Knapsack Problems," *Discrete Math.* **1**, 29–45 (1971).
6. S. E. Elmaghraby and M. K. Wig, "On the Treatment of Stock Cutting Problems as Diophantine Programs," Report No. 61, North Carolina University and Corning Glass Research Center, Chapel Hill, N.C., May 1970.
7. F. Glover, "New Results for Reducing Integer Linear Programs to Knapsack Problems," University of Colorado, Management Science Report Series No. 72-7, Boulder, Col., April 1972.
8. F. Glover and R. E. Woolsey, "Aggregating Diophantine Constraints," *Z. Opns. Res.* **16**, 1–10 (1972).
9. P. L. Hammer and S. Rudeanu, *Boolean Methods in Operations Research and Related Areas*, Berlin, Springer Verlag, 1968.
10. G. B. Mathews, "On the Partition of Numbers," *Proc. London Math. Soc.* **28**, 486–490 (1896).
11. M. W. Padberg, "Equivalent Knapsack-type Formulations of Bounded Integer Linear Programs: An Alternative Approach," *Naval Res. Log. Quart.* **19**, 699–708 (1972).
12. S. Zionts, "Generalized Implicit Enumeration Using Bounds on Variables for Solving Linear Programs with Zero-One Variables," *Naval Res. Log. Quart.* **19**, 165–181 (1972).