A Total Unimodularity Based Branch-and-Bound Method for Integer Programming

Jun Wang
Department of Management Science and Engineering
Qingdao University
Qingdao, Shandong, China
Email: jwang.qdu@gmail.com

Mei Long
Department of Systems Engineering and Engineering Management
Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Email: mlong@se.cuhk.edu.hk

Duan Li
Department of Systems Engineering and Engineering Management
Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Email: dli@se.cuhk.edu.hk

***Abstract*— We consider a novel solution method for integer programming problems with a 0-$\pm$1 constraint matrix, which include Set Covering Problem (SCP) and Satisfiability Problem (SAT) as special cases. Specifically, a branching rule is designed to actively make a satisfaction of total unimodularity (TU), thus speeding up the convergence of the search process in the branch-and-bound method. This TU-based branch-and-bound method is tested for a family of random 3-SAT problems and is compared with the IP solver of *ILOG CPLEX*. Computational experience shows that this new solution scheme is efficient.**

I. Introduction

While a wide variety of real-world decision making problems can be formulated as integer programming problems, a large percentage of integer programming problems has been proven to be NP-hard. We consider in this paper the following class of linear integer programming problems:

$$\begin{aligned}(IP) \qquad & \min c^T x \\ & \text{s.t. } Ax \geq b, \\ & \quad x \in Z_+^n,\end{aligned}$$

where all elements of A are 0, -1 or 1 and vector b is integral.

The most widely adopted solution scheme for (IP) is the branch and bound method, in which the bounds are often derived by linear programming relaxations and a common branch rule is to split a parent problem into a family of children problems by fixing the most fractional variable [16]. In the solution process of the branch and bound method, attaining optimality for a branch is one of the most efficient rules to prune a sub-tree. Therefore, the capability to achieve an integer solution for the linear programming relaxation of subproblems, generated in the process of branching and bounding, impacts significantly on the efficiency of the branch-and-bound method.

Whether or not a solution to linear programming is integer is closely related to a property of the constraint matrix, *total unimodularity (TU)*. A matrix is totally unimodular if all its subdeterminants are either 0 or ± 1. Hoffman and Kruskal [10] proved that a sufficient condition that the optimal solution to linear programming relaxation of (IP) is integer for all integral column vector b is that A is totally unimodular. In addition, as proved in [15], if any one of the matrices A, A^T, $-A$, (A, A), or (A, I) is totally unimodular, so are all the others.

TU matrices are highly desirable in discrete optimization because they assure an integral solution to linear programming. Although checking whether a matrix is TU is polynomial solvable [12], the constraint matrix A of (IP) is rarely TU, especially when the size of A is large. In general, the requirement that matrix A is TU is too restricted to be applied in integer programming.

Set Covering Problem (SCP) can be formulated as the integer programming problem (IP) where $x \in \{0,1\}^n$, A is a 0-1 matrix and all entries of b are ones. SCP has many practical applications, such as crew scheduling and service center allocation. Both exact algorithms [2][3] and heuristic methods [4][8] have been proposed for SCP in the literature. The following SCP example is investigated to illustrate the main idea of this paper,

$$\begin{aligned}& \min z = x_1 + 2x_2 + x_3 + 2x_4 \\ & \text{s.t. } \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} x \geq \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \\ & \quad x \in \{0,1\}^4.\end{aligned} \tag{1}$$

The constraint matrix A of the example is obviously not TU. Solving the linear relaxation of (1) yields an optimal solution $x^* = (0, 0.7, 0.7, 0.3)^T$, in which three variables, x_2, x_3 and x_4, are not integers and can be selected as the next branching variable. Note that matrix A is still not TU after removing the second column, while A becomes TU after removing the third or fourth column. The subproblems generated by fixing x_3 or x_4 at 0 or 1 are therefore integer solvable. Therefore, choosing x_3 or x_4 as a branching variable is better than choosing x_2. A 0-$\pm$1 matrix can be always reduced to a totally unimodular one by removing some columns, i.e., fixing a set of variables. Thus, those variables with higher possibility in making the reduced constraint matrix TU should be identified as branching variables with higher priorities.

1-4244-0387-1/06/$20.00

The above discussion motivates us to devise a mechanism that ranks the decision variables based on their capability in actively making the satisfaction of TU, and then to propose a branching rule accordingly.

The Satisfiability Problem (SAT) is a fundamental problem in mathematical logic, inference, and computing theory. SAT plays an important role in solving a large family of *NP*-complete problems. We present a novel equivalent integer programming model for SAT in Section II. We then devise a TU-based branching rule in Section III. We describe in Section IV the TU-based branch-and-bound method in details. Finally, we summarize in Section V the computational experience and conclude the paper.

II. Satisfiability Problem and its Integer Programming Model

Consider a set of Boolean variables $X = \{x_1, x_2, \cdots, x_n\}$. Both x_i and its complement $\overline{x}_i$ are referred to as *literals* in the language of logic. A *clause* is a disjunction of literals, in which each literal occurs only once. The *conjunctive normal form (CNF)* formula is a conjunction of clauses. For example, we consider $X = \{x_1, x_2, x_3, x_4\}$. A CNF formula is given by

$$F = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee \overline{x}_4), \tag{2}$$

where $\vee$ represents *logical or*, $\wedge$ represents *logical and*, and $x_1 \vee x_2$, $\overline{x}_1 \vee x_2 \vee \overline{x}_3$, $\overline{x}_2 \vee \overline{x}_4$ are three clauses of F.

Given a set X and a CNF formula, the SAT problem is to find whether there exists a true value of F associated with a certain assignment of all logical variables. The SAT problem in (2) is satisfiable as a true value of F can be realized by setting $x_1 = x_2 = x_3 = true$ and $x_4 = false$. When all clauses of a CNF formula have at most r literals, the problem is referred as r-SAT problem. 2-SAT problem is polynomial solvable [1][14] and, essentially, a linear-time algorithm [7] was proposed in 1976. However, no polynomial-time algorithm has been found to solve 3-SAT problem. Among r-SAT problems, 3-SAT problem is the smallest NP-complete problem with complexity $O(2^n)$. When the ratio of clause number to variable number is approximatively equal to 4.25, 3-SAT problem becomes especially hard to handle [5]. In the literature, some solution algorithms have been proposed to solve the SAT problem [6][9][11]. In this paper, we will focus on 3-SAT problems instead of general SAT problems.

The clauses of CNF could be transformed into a set of linear constraints. After replacing complements $\bar{x}_i$ by $1 - x_i$ and logical or ($\vee$) by addition operation ($+$), a clause is transformed to be a real-valued function that has to possess a value greater than and equal to one. For example, CNF F in (2) can be dealt with as follows,

$$\begin{aligned} x_1 \vee x_2 \quad &\Rightarrow x_1 + x_2 \geq 1, \\ \bar{x_1} \vee x_2 \vee \bar{x}_3 \quad &\Rightarrow (1 - x_1) + x_2 + (1 - x_3) \geq 1, \\ \bar{x_2} \vee \bar{x_4} \quad &\Rightarrow (1 - x_2) + (1 - x_4) \geq 1. \end{aligned}$$

To solve this SAT example is equivalently to check whether there is a solution to the following set of inequalities,

$$\begin{cases} x_1 + x_2 \geq 1, \\ -x_1 + x_2 - x_3 \geq 1 - 2, \\ -x_2 - x_4 \geq 1 - 2. \end{cases} \tag{3}$$

By introducing a binary variable vector, s, we can form the following integer programming problem for SAT,

$$\begin{aligned} (IP1) \qquad & \min \ \mathbf{1}^\top s, \\ & \text{s.t. } Ax + Is \geq \mathbf{1} - b, \\ & \quad x \in \{0,1\}^n, \ s \in \{0,1\}^m, \end{aligned} \tag{4}$$

where $A = [a_{ij}]_{m \times n}$ is a 0-$\pm$1 matrix, $\mathbf{1}$ is an m-dimensional vector with all entries being ones, I is an $m \times m$ identity matrix, and $b = [b_i]_m$ with b_i being the number of complement literals in the i-th clause.

There is another way to transform an SAT problem to a set of linear constraints, in which some additional constraints, such as $x_j + \overline{x}_j = 1$, are introduced instead of replacing the complements directly. For example, CNF F in (2) can be reformulated as the following constraint set,

$$\begin{cases} x_1 + x_2 \geq 1, \\ \overline{x}_1 + x_2 + \overline{x}_3 \geq 1, \\ \overline{x}_2 + \overline{x}_4 \geq 1, \\ x_1 + \overline{x}_1 = 1, \\ x_2 + \overline{x}_2 = 1, \\ x_3 + \overline{x}_3 = 1, \\ x_4 + \overline{x}_4 = 1. \end{cases} \tag{5}$$

Similar to problem $(IP1)$, an integer programming problem can be constructed as follows for SAT,

$$\begin{aligned} (IP2) \qquad & \min \ \mathbf{1}^\top s, \\ & \text{s.t. } A(x, \overline{x}) + I_m s \geq \mathbf{1}, \\ & \quad I_n x + I_n \overline{x} = \mathbf{1}, \\ & \quad x \in \{0,1\}^n, \ \overline{x} \in \{0,1\}^n, \ s \in \{0,1\}^m, \end{aligned} \tag{6}$$

where $A = (A^+, A^-)^\top$, A^+ and A^- are $m \times n$ 0-1 matrices, which represent the coefficient matrices of x and $\overline{x}$ respectively, and $I_m(I_n)$ is the identity matrix with dimension $m(n)$. The following theorem is evident.

Theorem 1: An SAT problem is satisfiable if and only if the optimal value of $(IP1)$ or $(IP2)$ is zero. When the optimal value of $(IP1)$ or $(IP2)$ is zero, the optimal solution of $(IP1)$ or $(IP2)$ forms a feasible solution to the primal SAT problem.

Theorem 1 implies that an SAT problem can be equivalently transformed into an integer programming problem $(IP1)$ or $(IP2)$. The right-hand-sides in both forms (4) and (6) are clearly integral. Thus, when considering integral property of the integer programming model for SAT, it is sufficient to investigate the TU property of matrix A in (4) or (6) .

III. Rationales in Ranking Variables

Before devising a mechanism to rank variables, we first investigate properties of TU for 2×2 0-±1 matrices and 3×3 0-1 matrices, respectively.

A. TU of 2×2 0-±1 Matrix

For 2×2 0-±1 matrices, among the total 81 (=3^4) possible cases, only the following 8 cases are not TU,

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix}, \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix},$$
$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}, \begin{pmatrix} -1 & 1 \\ -1 & -1 \end{pmatrix}.$$

Observation 1: For a 2×2 0-±1 matrix, if any entry is zero, the matrix must be TU.

Observation 2: A 2×2 0-±1 matrix is not TU if and only if two entries in one row (column) have the same signs, while two entries in the other row (column) have different signs.

B. TU of 3×3 0-1 Matrix

For 3×3 0-1 matrices, among the total 512 (= 2^9) possible cases, only the following 6 cases are not TU,

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

Observation 3: For a 3×3 0-1 matrix, the matrix is not TU if and only if each row or column has exact one zero element.

C. The Branching Rule

How to find a branching rule such as to generate a reduced TU constraint matrix as quickly as possible by fixing variables is the major research subject in this research. Based on the above recognitions of the characteristics associated with the non-TU cases for 2×2 0-±1 and 3×3 0-1 matrices, we define the following non-TU degree of a variable to be a degree of eliminating the non-TU cases in the constraint matrix by fixing the variable. In other words, the level of non-TU degree indicates the ability of a variable to remove non-TU structure of the constraint matrix by fixing this variable. A branching rule is then developed by ranking the variables according to their non-TU degrees.

Consider a 0-±1 matrix A generated from ($IP1$). For each given pair of two columns, i and j, let e^1_{ij} denote the number of rows with same elements, $(1,1)$ and $(-1,-1)$, and e^2_{ij} the number of rows with different elements, $(1,-1)$ and $(-1,1)$. Define $e_{ij} = e^1_{ij} \times e^2_{ij}$ which gives out a non-TU degree contributed by coupling x_i and x_j. Define further $w_i = \sum_j e_{ij}$, which represents a non-TU degree of variable x_i. We choose the variable x_{i^*} with the largest positive non-TU degree as a branching variable, then set $e_{i^*j} = 0$ and recalculate w_i for the unranked variables. The next branching variable will be found accordingly.

Note that not all variables can be ranked according to the values of w_i's since after several iterations all remaining unranked variables may have zero w_i values. We then construct ($IP2$) for the SAT problem with the reduced constraint matrix. For each given triple of three columns, i, j and k which form a submatrix A_{ijk}, let e^1_{ijk} denote the number of rows of A_{ijk} that have a single zero element on the left, $(0,1,1)$, e^2_{ijk} the number of rows of A_{ijk} that have a single zero element in the middle, $(1,0,1)$, and e^3_{ijk} the number of rows of A_{ijk} that have a single zero element on the right, $(1,1,0)$. Let $e_{ijk} = e^1_{ijk} \times e^2_{ijk} \times e^3_{ijk}$ that gives out a non-TU degree contributed by submatrix A_{ijk}. Let $w_i = \sum_{jk} e_{ijk}$, which represents a non-TU degree of x_i. We choose the variable x_{i^*} with the largest positive non-TU degree as the next branching variable. We then set $e_{i^*jk} = 0$, recalculate w_i of the unranked variables and go to the next iteration.

After the above procedures, if the variables have not been ranked completely, we choose an arbitrary branching rule to order the remaining unranked variables.

Let S be a permutation of $\{1,\ldots,n\}$ to record an order of variables to be fixed. The following procedure is designed to obtain the branching order of variables for SAT problem.

Procedure 1: (Branching order for SAT problem)

Step 0. Convert SAT problem into ($IP1$), set $S = \emptyset$.

Step 1. Count the numbers e^1_{ij} and e^2_{ij} for all pairs i and j.

Step 2. Calculate $e_{ij} = e^1_{ij} \times e^2_{ij}$ for all $i \neq j$, and let $e_{ij} = 0$ for $i = j$.

Step 3. Calculate $w_i = \sum_j e_{ij}$ for all i. If all $w_i = 0$, convert the reduced SAT problem with the variables not ranked in S into ($IP2$) and go to Step 6.

Step 4. Identify

$$i^* = \arg\max_i \ w_i$$

and add x_{i^*} into the branching order list, i.e., $S = S \cup \{i^*\}$. If all variables are ranked in S, terminate the procedure.

Step 5. Set all $e_{i^*j} = 0$, go back to Step 2.

Step 6. Count the numbers e^1_{ijk}, e^2_{ijk} and e^3_{ijk} for all triples i, j, k.

Step 7. Calculate $e_{ijk} = e^1_{ijk} \times e^2_{ijk} \times e^3_{ijk}$ for all $i \neq j \neq k$, and let $e_{ijk} = 0$ for $i = j$ or $i = k$ or $j = k$.

Step 8. Calculate $w_i = \sum_{j,k} e_{ijk}$ for all i. If all $w_i = 0$, terminate the procedure.

Step 9. Identify

$$i^* = \arg\max_i \ w_i$$

and add x_{i^*} into the branching order, i.e., $S = S \cup \{i^*\}$. If all variables are ranked in S, terminate the procedure.

Step 10. Set all $e_{i^*jk} = 0$, go back to Step 8.

Procedure 1 gives out a branching order S, which ranks all or part of the variables.

IV. TU-based Branch-and-Bound Method

Employing the generated branching order of variables as a branching rule, the TU-based branch-and-bound method is developed to solve the integer programming problems. Utilizing the above branching order procedure, we propose the following TU-based branch-and-bound method for SAT problem.

Algorithm 1: (TU-based Branch-and-Bound Method)

Step 1. Transform the SAT problem into integer programming problems ($IP1$) and ($IP2$).

Step 2. Obtain the branching order S.

Step 3. The width-first search strategy is used within the framework of the branch-and-bound method to solve ($IP1$) or ($IP2$) according to the branching rule S. At any branch node with an integer optimal solution achieved, if the optimal value is zero, the algorithm terminates and the SAT problem is satisfiable; Otherwise, continue.

Step 4. If no active node exists, the algorithm terminates and the SAT problem is unsatisfiable; Otherwise, continue.

Step 5. Apply the branch-and-bound method with an arbitrary branching order to the remaining variables unranked in S.

V. Computational Experience and Conclusions

The TU-based branch-and-bound method developed in this research has been coded in C Language under UNIX platform, while *ILOG CPLEX* is called as a linear programming solver. Uniform Random-3-SAT is a family of 3-SAT problems obtained by randomly generating 3-CNF formula with uniform distribution. The method is tested on four types of Uniform Random-3-SAT problems with n×m=20×91, 50×218, 75×325 and 100×430 (m/n≈4.25). Table I summarizes the computational results of 100 instances for each problem. The inter programming solver of *ILOG CPLEX* is one of the most powerful integer programming solvers, which is also tested on the same problems. The computational results of *ILOG CPLEX* are summarized in Table II. Comparing the data in Table I and II, the TU-based branch-and-bound method seems to be more efficient than *CPLEX IP* solver.

n× m	Number of nodes			CPU time (Sec.)		
	Ave.	Min.	Max.	Ave.	Min.	Max.
20× 91	10.89	1	37	0.02	0.00	0.06
50× 218	92.83	1	343	0.39	0.04	1.31
75× 325	703.98	8	6502	4.86	0.07	37.07
100× 430	3165.91	12	17478	37.18	0.42	164.44

TABLE I
STATISTICS OF TU BASED BRANCH-AND-BOUND METHOD IN SOLVING 3-SAT

n× m	Number of nodes			CPU time (Sec.)		
	Ave.	Min.	Max.	Ave.	Min.	Max.
20× 91	9.27	1	41	0.04	0	0.09
50× 218	144.56	1	692	0.58	0.02	2.46
75× 325	964.99	1	12472	5.55	0.07	37.36
100× 430	4139.81	15	27900	39.84	1.18	222.64

TABLE II
STATISTICS OF *CPLEX IP* IN SOLVING 3-SAT

TU is a highly desirable property for integer programming in the literature. However, TU is rarely satisfied in practice. We devise a mechanism to generate a branching order to actively make the satisfaction of TU in early stage of the implementation of the branch-and-bound method, thus facilitating the search process for integer programming.

Acknowledgment

This research was partially supported by the Research Grant Council of Hong Kong, Grant CUHK2050337.

References

[1] J. H. Aho and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
[2] E. Balas and M. C. Carrera, "A Dynamic Subgradient-based Branch-and-bound Procedure for Set Covering," *Operations Research*, Vol. 44, pages 975-890, 1996.
[3] J. E. Beasley, "An Algorithm for Set Covering Problems," *European Journal of Operations Resaerch*, Vol. 31, pages 85-93, 1987.
[4] A. Caprara, M. Fischetti and P. Toth, "A Heuristic Method for the Set Covering Problem," *Operations Research*, Vol. 47, No.5, pages 730-743, 1999.
[5] J. M. Crawford and L.D. Auton, "Experimental Result on the Crossover Point in Random 3-SAT," *Artificial Intelligence*, Vol. 8, pages 31-57, 1996.
[6] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of ACM*, Vol. 7, 201-215, 1960.
[7] A. I. S. Even and A. Shamir, "On the Complexity of Timetable and Multi-commodity Flow problems," *SIAM, Journal of Computing*, Vol. 5, No. 4, pages 691-703, 1976.
[8] M. L. Fisher and P. Kedia, "Optimal Solutions of Set Covering/partioning Problems Using Dual Heuristics," *Management Science*, Vol.36, pages 674-688, 1990.
[9] J. Gu, "Local search for satisfiability (SAT) problem", *IEEE Trans. on System, Man, and Cybernetics*, Vol. 23, No. 4, pages 1108-1129, 1993.
[10] A. J. Hoffman and J.B. Kruskal, "Integral Boundary Points of Convex Polyhedra," *Linear Inequalities and Related Systems*, Princeton University Press, princeton, NJ, pages 223-246, 1965.
[11] C. Li, "A constraint-based approach to narrow search trees for satisfiability," *Information processing letters*, Vol. 71, pages 75-80, 1999.
[12] R. G. Parker and R. L. Rardin, *Discrete Optimization* Academic Press, London, 1988.
[13] J. N. Reingold and N. Deo, *Combiatorial Algorithms: Theory and Practice*, Prentice-Hall, 1977.
[14] T. Schaefer, "The Complexity and Satisfiability Problem," *Proceedings in the Theth Annual ACM Symposium, Theroy of Computer Science*, pages 216-226, 1978.
[15] A. F. Veinott and G.B. Dantzig, "integral Extreme Points," *SIAM Review*, Vol. 10, pages 371-372, 1968.
[16] L. A. Wolsey, *Integer Programming*, John Wiley & Sons, New York, 1998.