GRADIENT METHODS FOR BINARY INTEGER PROGRAMMING

Chien-Yuan Huang[1], Ta-Chung Wang[2]
Institute of Civil Aviation, National Cheng Kung University,
No.1, University Road, Tainan City 701, Taiwan (R.O.C.)
TEL: +886-6-2757575 ext.63620
[1]fbifa22@hotmail.com, [2]tachung@mail.ncku.edu.tw

Abstract:
Integer Programming (IP) is a common problem class where decision variables represent indivisibility or yes/no decisions. IP can also be interpreted as discrete optimizations which are extensively used in the areas of routing decisions, fleet assignment, and aircraft/aircrew scheduling. Solving IP models is much harder than solving Linear Programming (LP) models because of their intrinsically combinatorial nature. In addition, many "real-world" problems have a huge problem size that makes solving the related IP-based model time-consuming. In this paper, we propose a gradient method for binary IP that enable large problems to be solved in a short amount of time. The algorithm is partitioned into two phases. In phase 1 of the algorithm, integrality constraints are modified and the original problem is solved by the LP algorithm. In phase 2, all integrality constraints are modified again and added back. The Lagrange multipliers are then introduced to form a Lagrange dual problem. We then use the gradient method to quickly search for the optimal or nearly optimal solution. After developing the algorithm, we show the effectiveness of this approach by applying it to various kinds of binary IP problems with discussions about the reduction in computation time and the gap between objective functions.

Keywords: Integer Programming; Gradient Methods; Lagrange Dual Problems.

1. Introduction

Integer Programming (IP) problems are based on Linear Programming (LP) problems with extra requirements that each decision variable be an integer value representing indivisibility, e.g. aircraft, manpower, and facilities. It is thus possible to formulate models that are more realistic and meaningful. In addition, the integer variables used are often restricted to binary values in order to help companies make crucial yes/no decisions, and there are a vast number of applications of this method over very wide range of real-world problems. However, IP problems are intrinsically hard to be solved than that of LP problems.

In this paper, we propose a gradient method for binary integer programming (BIP), which is a totally different approach to the currently ubiquitous branch-and-bound technique. By using this method, we can

overcome the inherent weaknesses of the branch-and-bound algorithm (Taha, 2007), by avoiding spending too much time calculating linear programming relaxations (LPRs) and keeping track of the best integer feasible solution at every branching node. In fact, in the approach presented in this work we calculate the modified LPR just once, and then the algorithm proceeds to phase 2 for fast solution searching.

2. Computational Bottlenecks

The well-established algorithm for solving IP problems is the "branch-and-bound" algorithm, which adopts a very simple concept, "divide and conquer", to avoid complete enumeration. This concept is based on the tree search which divides the original problem into all possible sub-problems and then solves all of these.

An important feature of the IP model is its ability to capture nonlinearities and nonconvexities (Nemhauser, 1994). Although most IP problems have a finite number of feasible solutions, this number can grow exponentially. Also, due to the integer restrictions, the solution space of IP models are lattices rather than convex. Moreover, branch-and-bound algorithms do not offer the same advantages as the simplex method (Dantzig, 1963) in its ability to finding optimal solutions. IP problems are potentially exponential in computational complexity, and, Sugden (1992) noted that some IPs are intrinsically hard, no matter what branch-and-bound heuristic is used to fathom the search tree. Unless special steps are taken, the number of sub-problems (branches) can grow exponentially. In addition, another critical problem arises in branch-and-bound algorithms, which is that there is a fundamentally combinatorial nature embedded in IP, so that different node and variable choices yield different search trees. This feature of high uncertainty may result in extremely unpredictable computational time. Consequently, in general, it is impossible to determine the best tree search strategy. One can thus conclude that branch-and-bound algorithms possess a high degree of non-determinism with regard to their performance (MITRA et al., 1997), and even though such algorithms can be proved to terminate with an integer solution in a finite number of steps, it is difficult to predict the necessary running time. IP models are thus classified as *NP-hard*, and are much more difficult solve than LP problems.

3. Algorithm

The standard binary integer programming (BIP) problem can be stated as follows:

$$\min. \quad C^T x \tag{1}$$

$$s.t. \quad a_{ij} x_j - b_i \le 0,\ i = 1,\ldots,m,\ j = 1,\ldots,n \tag{2}$$

$$d_{lj} x_j - e_l = 0,\ l = 1,\ldots,p,\ j = 1,\ldots,n \tag{3}$$

$$x_j \in binary$$

where x_j is the decision variable, C is the vector of cost coefficient, and (2) and (3) are the constraints to be satisfied for an optimal solution.

The proposed algorithm is partitioned into two phases. In phase 1 of the algorithm, the binary integer restrictions are first relaxed as

$$0 \le x_j \le 1. \tag{4}$$

Then the problem is solved by the revised simplex method or the interior point method (Nemirovski & Todd, 2008), depending on the problem structure. By the efficient LP algorithm, we solve this tightened LPR in the hope of finding a good or nearly good initial point quickly as an input of phase 2. In phase 2 of the algorithm, the constraints of binary variables are first modified as follows

$$x_j(x_j - 1) = 0 \tag{5}$$

to ensure that the binary integer restrictions can have a specific mathematical representation to be incorporated into the objective function with Lagrange multipliers. Then, by introducing the Lagrange multipliers to all other constraints, the original BIP problem has transferred into a Lagrange dual problem. The resultant Lagrange dual function is

$$L(x, u_i, v_l, w_j) = C^T x + \sum_{i=1}^{m} u_i (a_{ij} x_j - b_i) + \sum_{l=1}^{p} v_l (d_{lj} x_j - e_l) + \sum_{j=1}^{n} w_j (x_j (x_j - 1)) \tag{6}$$

where u_i, v_l, w_j are Lagrange multipliers, and $u_i \ge 0$.

To determine the optimal solution, the Karush-Kuhn-Tucker (KKT) necessary optimality conditions should be satisfied. Since this problem has become an unconstrained optimization problem. We can thus use the gradient method for fast searching the optimal or nearly optimal solutions. Note that the KKT sufficient conditions for optimality are not considered in this paper. Readers who are interested in optimality conditions could refer to (Bazaraa & Shetty, 1979).

In addition, to further refine the proposed algorithm, we also take into account of conjugate gradient (CG) methods to avoid searching in directions that have been searched before. Since the original search direction (the negative gradient) may have a rapid change in its following search direction, and lead to a poor performance especially on some large-scale problems. Yuan (2009) proposed a hybrid method combining the FR conjugate gradient method (Fletcher & Reeves, 1964) and the WYL conjugate gradient method (Wei et al., 2006). This hybrid method has been proved having the sufficient descent property under the strong Wolfe-Powell (SWP) line search rule. The great performance is attributed to the CG update parameter

$$(\beta^k)^H = q_1 (\beta^k)^{FR} + q_2 (\beta^k)^{WYL}, \tag{7}$$

where $q_1 \ge 0, q_2 \ge 0$. But, Yuan did not suggest that how to choose the parameters q_1 and q_2 in the algorithm.

In phase 2 of the proposed algorithm, we also adopt the hybrid CG method to update the primal and dual variables, and also investigate the influence on the performance of different choices of q_1 and q_2. The algorithm is described as follows.

Step 1: Input the primal and dual initial point x^0 and λ^0, where $\lambda = (u, v, w)$. Choose $q_1 \geq 0, q_2 \geq 0$.

Step 2: Set $k = 0$, $d^0 = -g_x^0 = -\nabla_x L(x^0, \lambda^0)$, $\hat{d}^0 = g_\lambda^0 = \nabla_\lambda L(x^0)$

Step 3: If $\left\| g_x^0 \right\|_2 \leq \varepsilon$ ($\varepsilon = 10^{-4}$), then stop; otherwise go to step 4.

Step 4: Compute step-size α_x^k and $\hat{\alpha}_\lambda^k$ by some line search rules.

Step 5: Update $x^{k+1} = x^k + \alpha_x^k d^k$ and $\lambda^{k+1} = \lambda^k + \hat{\alpha}_\lambda^k \hat{d}^k$

Step 6: Compute $g_x^{k+1} = \nabla_x L(x^{k+1}, \lambda^{k+1})$, $g_\lambda^{k+1} = \nabla_\lambda L(x^{k+1})$. If $\left\| g_x^{k+1} \right\|_2 \leq \varepsilon$, then stop.

Step 7: Update the CG parameters

$$(\beta_x^k)^{FR} = \frac{(g_x^{k+1})^T g_x^{k+1}}{(g_x^k)^T g_x^k}, \quad (\beta_\lambda^k)^{FR} = \frac{(g_\lambda^{k+1})^T g_\lambda^{k+1}}{(g_\lambda^k)^T g_\lambda^k}$$

$$(\beta_x^k)^{WYL} = \frac{(g_x^{k+1})^T (g_x^{k+1} - (\left\| g_x^{k+1} \right\| / \left\| g_x^k \right\|) g_x^k)}{(g_x^k)^T g_x^k}, \quad (\beta_\lambda^k)^{WYL} = \frac{(g_\lambda^{k+1})^T (g_\lambda^{k+1} - (\left\| g_\lambda^{k+1} \right\| / \left\| g_\lambda^k \right\|) g_\lambda^k)}{(g_\lambda^k)^T g_\lambda^k}$$

Step 8: Update the search direction $d^{k+1} = -g_x^{k+1} + (q_1 (\beta_x^k)^{FR} + q_2 (\beta_x^k)^{WYL}) d^k$,

$$\hat{d}^{k+1} = g_\lambda^{k+1} + (q_1 (\beta_\lambda^k)^{FR} + q_2 (\beta_\lambda^k)^{WYL}) \hat{d}^k$$

Step 9: Set $k = k + 1$, and go to Step 4.

4. Experimental Results

The experimental results for the proposed gradient methods and the standard branch-and-bound technique are presented in the following. Table 1 gives some statistics for the testing models. All codes were written in MATLAB (7.10.0) running on a personal computer (Intel Core i5 CPU 760 @ 2.80 GHz, 3.49 GB RAM).

Tables 2 and 3 summarize the computational time and computational results. The first column in both tables reports the model name; TBB is the computational (CPU) time of the branch-and-bound algorithm; TLP is the computational (CPU) time of the phase 1 procedures; TGD is the computational (CPU) time of the phase 2 gradient method; TT reports the total time of the proposed algorithm; and speed-up is calculated by TBB/TT. Table 3 gives the computational results. GDx reports the final gradient of x when the process has converged; LPRgap is the objective value gap between the phase 1 LPR and the branch-and-bound algorithm; GDgap is the gap between the proposed algorithm and the exact solution; and Rgap is the final gap between the rounding solution and the exact solution. If the final solution is sufficiently close to zero or one (e.g. 0.001 or 0.999), then the solution is round to the nearby binary value.

Table 2 and Table 3 illustrate the value of the proposed gradient methods. The optimal solution is obviously achieved in a shorter time than needed for the branch-and-bound algorithm. Although in case M-01, the final gap still has 25.79 %, the branch-and-bound algorithm takes over than 96 hours CPU time to solve this problem while the proposed method only takes about 14.63 seconds CPU time to approximate the

Table 1: Model statistics (binary integer programming).

Model	Number of rows	Number of columns	Number of non-zeros
T-01	12	12	57
T-02	33	48	568
T-03	118	27	378
M-01	1999	200	3998

Table 2: Computational time.

Model	TBB (s)	TLP (s)	TGD (s)	TT (s)	Speed-up
T-01	0.5156	0.3281	0.0313	0.3594	1.4529
T-02	0.6094	0.3125	0.1406	0.4531	1.3449
T-03	496.1563	0.3594	6.8594	7.2188	68.7311
M-01	Over 345600	1.7813	12.8438	14.6251	Over 590.77

Table 3: Computational results.

Model	GDx	LPRgap (%)	GDgap (%)	Rgap (%)
T-01	8.9532e-5	58.33	1.4286e-3	0
T-02	9.4397e-5	60.87	4.3478e-4	0
T-03	3.4856e-5	27.78	11.2183	11.11
M-01	9.5349e-5	37.11	25.7387	25.79

solution. The overall computational performance of the proposed gradient method is significantly better than that of the branch-and-bound algorithm.

In the special case M-01, we noted that the phase 1 LPR produces a solution with the value equals to 0.5 on each x. Since the value 0.5 just locates at the middle point of zero and one, any move to the binary value becomes a crucial decision. For this reason, we modify again the constraints of binary variables by multiplying a multiplicative constant y on (5) as follows.

$$yx_j(x_j - 1) = 0 \tag{8}$$

Through this way, we can force the solution converging at the binary value.

Table 4: Types of different choices of parameters q_1 and q_2.

Type	1	2	3	4	5	6	7	8	9
q_1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
q_2	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

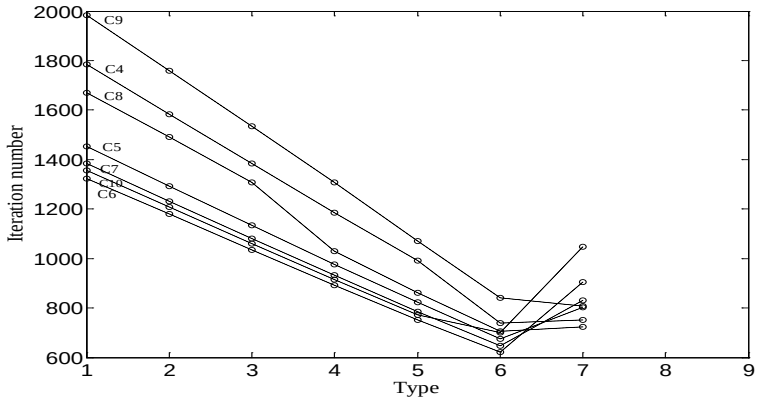

Figure 1: Performance of different choices of parameters q_1 and q_2.

To further refine the algorithm, we also investigate different choices of the initial point of dual variables that can have a great impact on the performance. We use models from Table 1 and generate extra models by setting different multiplicative constant. Through the experiment, we suggest that the initial point of dual variables corresponding to the modified binary constraints should set closer to zero.

Another issue related to the CG performance is that the best choice of parameters q_1 and q_2 are not yet known so far. In order to find out what parameters q_1 and q_2 should be chosen to drive the hybrid CG algorithm performing the best performance, we investigate different type of choices of parameters. Table 4 illustrates the testing types of different choices of the parameters. Fig. 1 shows the iteration number versus different types of choices of the parameters on each case. Most cases diverged while using type 8 and type 9 parameters. Therefore, to further improve the performance, we could choose the recommended parameters $q_1 = 0.6$ and $q_2 = 0.4$.

5. Conclusions and Future Work

In this paper we construct a new solver, and demonstrate that the proposed gradient method has better performance than the ubiquitous branch-and-bound technique. In addition, we have found the best or at least good enough choices of the initial value of dual variables and the preliminarily parameters in the hybrid CG method. These can significantly improve the performance of the algorithm. In the future, more BIP models should be tested for further testing. The different modified binary constraints should be investigated to further reduce the gap between the approximate and exact solution. Moreover, different solution searching routes generated by asynchronous updating the primal and dual variables will be taken into account. Finally, the error analysis between an approximate solution and an exact solution will also be investigated.

References

Bazaraa, M.S. & Shetty, C.M. (1979). Nonlinear programming: theory and algorithms. New York: Wiley.

Dantzig, G.B. (1963). Linear programming and extensions. Princeton, NJ: Princeton University Press

Fletcher, R. & Reeves, C.M. (1964). Function minimization by conjugate gradients. The Computer Journal, 7(2), 149-154.

MITRA, G., HAI, I. & HAJIAN, M.T. (1997). A distributed processing algorithm for solving integer programs using a cluster of workstations. Parallel Computing, 23, 733-753.

Nemhauser, G.L. (1994). The age of optimization: solving large-scale real-world problems. Operations Research, 42(1), 5-13.

Nemirovski, A.S. & Todd, M.J. (2008). Interior-point methods for optimization. Acta Numerica, 17, 191-234.

Sugden, S.J. (1992). A class of direct search methods for nonlinear integer programming. Gold Coast, Bond University.

Taha, H.A. (2007). Operations research: an introduction. Upper Saddle River, N.J.: Pearson Prentice Hall.

Wei, Z., Yao, S. & Liu, L. (2006). The convergence properties of some new conjugate gradient methods. Applied Mathematics and Computation, 183(2), 1341-1350

Yuan, G. (2009). A conjugate gradient method for unconstrained optimization problems. International Journal of Mathematics and Mathematical Sciences, 2009, 1-14.