February 27, 2007

RT0712

Computer Science 8 pages

Research Report

Exploiting patterns to reduce IP coefficient matrices

Takayuki Yoshizumi

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Exploiting patterns to reduce IP coefficient matrices

Takayuki Yoshizumi

Abstract

In this report, we present a novel method to reduce the non-zero factors in the coefficient matrices of integer programming problems by extracting coefficient patterns that appear frequently in the coefficient matrix and by replacing these patterns with new variables. Using this method, we can effectively reduce the memory requirements for maintaining the data structures, and for solving IP problems with IP solvers.

1. Introduction

We can now solve large integer programming (IP) problems with state-of-the-art IP solvers, such as CPLEX[1] [1], in practical calculation times. However, some large problems cannot be solved due to their memory requirements. For example, for an IP problem from an actual scheduling problem in the steel industry[2], the number of variables is in the hundreds of thousands, the number of constraints is in the tens of thousands, and the number of non-zero factors in the coefficient matrix is over one hundred million. To hold the data structure for such large problems, we need more than 1 GB of memory. To solve such a problem with an IP solver requires several GB of memory. Such memory requirements often exceed physical memory capacities, and sometimes even exceed the 32-bit address space of the most pervasive 32-bit Windows environments.

It is more effective to store only the non-zero factors of a coefficient matrix, not the entire matrix. The memory requirements to store the data structure for a problem depend on the number of non-zero factors. The memory requirements for an IP solver are proportional to the number of non-zero factors and are 2 to 5 times larger than the memory that stores the data structure. Reducing the number of non-zero factors has a large impact on the overall memory requirements.

While there is active research on speeding up IP solvers by adding redundant constraints or reformulating IP problems [3][4], so far there has been no research on methods to reduce the size of the problem (as the number of non-zero factors).

In this report, we present a novel method to reduce the number of non-zero factors in a coefficient matrix by extracting coefficient patterns that appear frequently in the coefficient matrix and replacing these patterns with new variables. Our method can effectively reduce the memory requirements for the data structures and for solving the IP problems with IP solvers.

---

[1] CPLEX is a registered trademark of ILOG.

2. An algorithm for reducing a coefficient matrix.

The method that we present in this report consists of two major algorithmic components. One clusters the coefficient matrix, and the other reduces the non-zero factors by using variable transformations. In the rest of this section, we describe these two components.

2.1. Coefficient matrix clustering

We produce coefficient matrix clusters that consist of subsets of the variables and constraints. The desirable conditions for these coefficient matrix clusters are:

- For each constraint included in a particular cluster, most of its coefficients are non-zero, or most of its coefficients are zero.
- For each variable included in a particular cluster, most of its coefficients are the same, or most of its coefficients are zero.

In general, it is difficult to find good coefficient matrix clusters. It is possible, however, to make good clusters by considering the nature of an actual problem, because IP problems are usually formulated from actual problems in the real world.

For example, IP constraints mapped from the same kind of constraint from an actual problem can form a constraint cluster. Those variables that represent the same event of an actual problem may form a variable cluster. Finally, we get coefficient matrix clusters as all of the pairs of a constraint cluster and a variable cluster. Figure 1 shows an example of coefficient matrix clusters made with this method. In this example, we have *k* variable clusters, *l* constraint clusters, and *kl* coefficient matrix clusters. (Since we can arbitrarily renumber the subscripts of variables and constraints in IP problems, we renumbered them in order that those elements that are included in the same cluster are sequentially next to each other.)

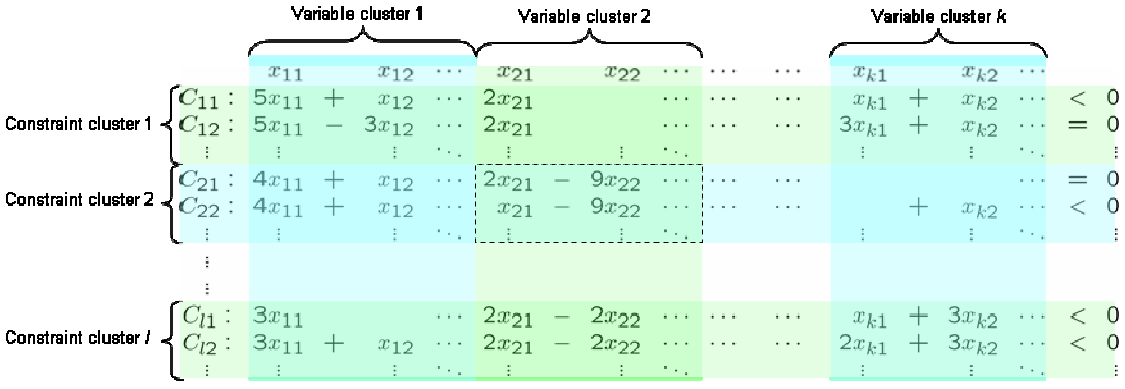

Figure 1: An example of coefficient matrix clusters.

2.2. Reducing non-zero factors by variable transformation

We seek to reduce the non-zero factors for each coefficient matrix cluster. Figure 2 shows an example of a coefficient matrix cluster. (For the sake of simplicity, the subscripts are renumbered from 1.)

$$
\begin{array}{lrcrcrcrcrcr}
C_1: & 5x_1 & + & x_2 & + & 2x_3 & - & 9x_4 & & & + & 7x_6 & + & x_7 \\
C_2: & 5x_1 & - & 3x_2 & + & 2x_3 & - & 9x_4 & + & 2x_5 & + & 7x_6 & + & 3x_7 \\
C_3: & x_1 & - & 3x_2 & + & 4x_3 & - & 9x_4 & + & 2x_5 & + & 7x_6 & + & x_7 \\
C_4: & 5x_1 & - & 2x_2 & & & & & & & + & 7x_6 & & \\
C_5: & 5x_1 & - & 3x_2 & & & - & 8x_4 & + & 2x_5 & + & 7x_6 & + & x_7
\end{array}
$$

Figure 2: An example of a coefficient matrix cluster

2.2.1. Reduction by dictionary compression

One of the basic ideas of data compression is the dictionary compression method, which replaces a pattern that appears frequently in the data set with a shorter pattern. This idea can be applied to compression of a coefficient matrix. Suppose that a common coefficient pattern that appears in multiple constraints is replaced by a new variable. In the case of the example shown at Figure 3, we can find a common pattern that is shown as six shaded regions.

$$
\begin{array}{lrcrcrcrcrcr}
C_1: & 5x_1 & + & x_2 & + & 2x_3 & - & 9x_4 & & & + & 7x_6 & + & x_7 \\
C_2: & 5x_1 & - & 3x_2 & + & 2x_3 & - & 9x_4 & + & 2x_5 & + & 7x_6 & + & 3x_7 \\
C_3: & x_1 & - & 3x_2 & + & 4x_3 & - & 9x_4 & + & 2x_5 & + & 7x_6 & + & x_7 \\
C_4: & 5x_1 & - & 2x_2 & & & & & & & + & 7x_6 & & \\
C_5: & 5x_1 & - & 3x_2 & & & - & 8x_4 & + & 2x_5 & + & 7x_6 & + & x_7
\end{array}
$$

Figure 3: Common pattern of the coefficient matrix cluster.

In the constraints of C_2, C_3, C_5, their coefficients for x_2, x_5, x_6 are same. Introducing the new variable *y*, we can define a new constraint for $C_y : -3x_2 + 2x_5 + 7x_6 = y$, which can be added to the IP formulation. Then we erase the terms for x_2, x_5, x_6 for the constraints of C_2, C_3, C_5, and add y to each of these constraints. These operations are mathematically equivalent. Finally, we get the revised coefficient matrix cluster, which is shown in Figure 4. While the number of variables and constraints has increased by one, the number of non-zero factors has decreased to a greater degree.

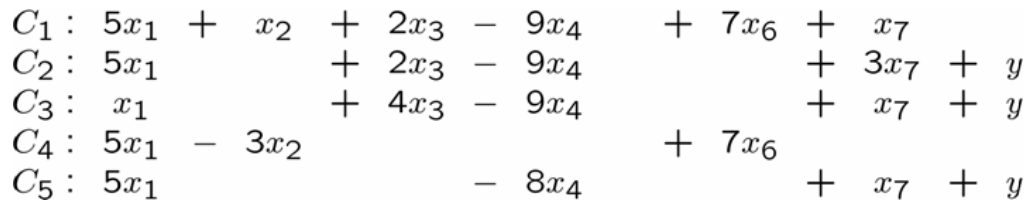

Figure 4: A coefficient matrix cluster after dictionary compression

When the number of variables that relate to the reduction is k and the number of constraints is l, we can reduce the number of non-zero factors by $(kl-k-l-1)$. The more constraints and variables we can extract from a coefficient matrix cluster, the more non-zero factors we can eliminate. In general, however, it is difficult to extract those subsets of the variables and constraints that maximize $(kl-k-l-1)$. (In practice, it is reasonable to use heuristic methods that can extract good subsets of the variables and constraints in practical time.)

Since a pattern must be exactly the same among the constraints, the maximum values of $(kl-k-l-1)$ will not be very much large in some problems. In such problems, the effectiveness of this method will tend to be limited.

2.2.2. Reduction by extracting the coefficients that appear most frequently

To solve the difficulties of dictionary compression, we present a novel method, which exploits the coefficients that appear most frequently.

First, we select constraints that will be candidates for elimination in a later step. We consider those constraints for which the number of non-zero factors is greater than or equal to half of the number of variables in the cluster as candidates. For the example shown at Figure 2, those constraints for which the number of non-zero factors is less than or equal to 3 will be excluded from the later step, because the number of variables of the cluster is 7. Only C_4 is excluded in this example. (The dark gray region of Figure 5 represents those elements that are excluded from the later steps.)

Next we select variables and their coefficients that will be candidates for elimination. For each variable, we select the coefficient that appears most frequently and count its number of occurrences. If the frequency of occurrences of the coefficient is less than or equal to half of the number of constraints in the cluster, its variables are excluded. In Figure 2, the number of constraints that will be candidates for elimination is 4, because C_4 was already excluded. We exclude those variables whose frequency of occurrence is less than or equal to 2. In this example, x_3 is excluded because its frequency of occurrence is 2. Variables that will be candidates for elimination are shown with half-tone meshing in Figure 5.

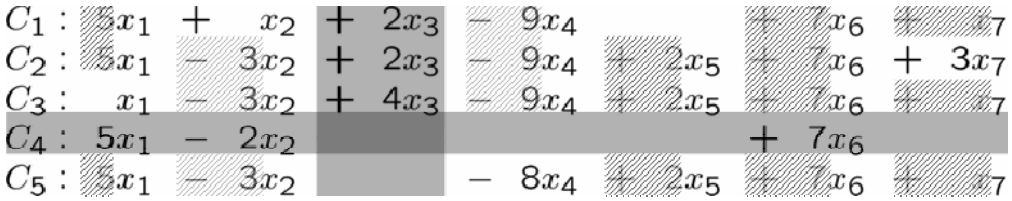

Figure 5: Extraction of components for reduction.

At this point, we introduce a new variable z, and define a linear combination of the variables which will be candidates for elimination as z:

$$C_z : 5x_1 - 3x_2 - 9x_4 + 2x_5 + 7x_6 + x_7 = z$$

This constraint is added to the IP problem, the variable z is added to each constraint that will be candidates for elimination, and the non-zero factors of those constraints are removed. Figure 6 shows the constraints after reduction of the non-zero factors. (The reason why some variables and constraints are excluded before reduction is that this transformation may increase the number of non-zero factors if they are not excluded.)

It is possible to efficiently reduce the non-zero factors among many constraints by defining new constraints that exploit the coefficients that appear most frequently. The number of non-zero factors that can be reduced by this method is larger than that by dictionary compression.

This method can, of course, be applied to linear programming (LP) and mixed integer programming (MIP) problems as well as to integer programming problems.

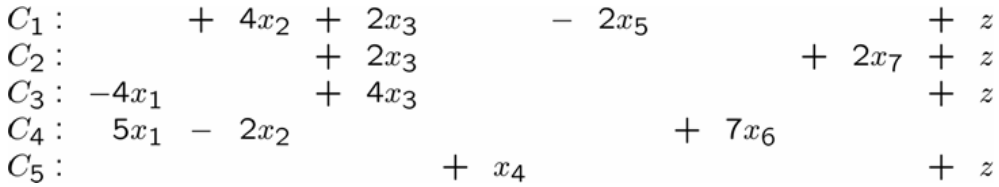

Figure 6: Constraints after transformation.

3. Experimental results

Table 1 shows experimental results for the IP problem formulated from an actual scheduling problem. We used CPLEX version 9.0 as the IP solver. We measure several indices for the original problem, for dictionary compression, and for the proposed method. The size of each IP problem is represented as the number of variables, of constraints, or of non-zero factors. The percentages in parenthesis represent the ratios of the size of the reduced problem to the original problem. MPS (Mathematical programming system) is a format to describe a mathematical programming problem such as IP, LP, or MIP. The MPS file size and the memory required for storing the data

structure of that problem are about the same size. The memory requirements for CPLEX are about 2 to 5 times larger than the MPS file sizes. We can roughly estimate the memory requirements from the MPS file sizes. The “reduction time” represents the time that is used for reducing non-zero factors. The “CPLEX time” represents the time for CPLEX. The termination condition for CPLEX is an optimality gap of 1% or a maximum running time of 2,000 seconds. “Total time” is the total running time, which consists of “reduction time”, “CPLEX time”, and others, such as file I/O, generating the IP data structure from the original scheduling problem, and so on.

Using the proposed method, the number of non-zero factors can be reduced down to less than 10 percent of the original problem while increasing the numbers of variables and constraints by only a few percent. The reduction efficiency of proposed method is about two times better than the dictionary compression method. This result shows that proposed method can effectively reduce the non-zero factors of IP problems.

For computational time, we believe the reduction time is practical because it is not very large compared with the running time of CPLEX and the total time. The Cplex times of both reduction methods are better than the time for the original IP problem. However the Cplex time of the proposed method is not always better than the dictionary compression method. One of the reasons is that sometimes adding new variables may make an IP problem more difficult for an IP solver even if the IP problems are mathematically exactly the same.

Table 1: Experimental results

instance	reduction method	#variables	#constraints	#non-zero factors	MPS file size(MB)	reduction time (sec)	CPLEX time	Total time
1	None	21,192	7,794	9,704,453	246	N/A	247	362
	Dictionary	21,270 (100.4%)	7,872 (101.0%)	1,924,051 (19.8%)	49	43	46	168
	Proposed	21,292 (100.5%)	7,894 (101.3%)	914,132 (9.4%)	24	53	85	213
2	None	63,576	21,086	29,113,359	738	N/A	335	593
	Dictionary	63,810 (100.4%)	21,320 (101.1%)	5,772,153 (19.8%)	148	129	91	400
	Proposed	63,876 (100.5%)	21,386 (101.4%)	2,742,396 (9.4%)	72	159	165	488
3	None	32,556	12,467	9,100,860	231	N/A	97	201
	Dictionary	32,720 (100.5%)	12,631 (101.3%)	2,701,976 (29.7%)	70	38	57	170
	Proposed	32,788 (100.7%)	12,699 (101.9%)	1,470,430 (16.2%)	39	49	27	147
4	None	97,668	35,651	27,302,580	693	N/A	496	739
	Dictionary	98,160 (100.5%)	36,143 (101.4%)	8,105,928 (29.7%)	209	114	74	371
	Proposed	98,364 (100.7%)	36,347 (102.0%)	4,411,290 (16.2%)	116	147	45	362
5	None	204,262	50,799	111,282,850	unsolvable (out of memory)			
	Dictionary	204,980 (100.4%)	51,517 (101.4%)	25,057,010 (22.5%)	636	486	timeout gap: 1.7%	3050
	Proposed	205,162 (100.4%)	51,699 (101.8%)	11,377,172 (10.2%)	296	606	timeout gap: 3.9%	3129

4. Conclusions

In this report, we present a novel method to reduce the non-zero factors in coefficient matrices for integer programming problems. Experimental results showed that our method can effectively reduce IP problem sizes. On some instances, memory requirements can be reduced to less than 10% of the

original problems.

In many instances, we can also reduce the computational time of an IP solver by problem size reductions. However this is not always true. We will investigate the relationships between problem size reduction and computation times of an IP solver.

References

[1] ILOG CPLEX, http://www.ilog.com/products/cplex/

[2] http://www.research.ibm.com/trl/projects/optimization/hsm_e.htm

[3] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing, 6(4):445--454, 1994.

[4] Ellis L. Johnson, George L. Nemhauser, Martin W.P. Savelsbergh, "Progress in Linear Programming-based Algorithms for Integer Programming: An Exposition", INFORMS Journal on Computing, Vol. 12, No. 1, Winter 2000