Constraint classification for mixed integer programming formulations

TECHNISCHE UNIVERSITEIT EINDHOVEN
Faculteit Wiskunde en Informatica

Memorandum COSOR 91-30

Constraint Classification for Mixed Integer Programming Formulations

G.L. Nemhauser
M.W.P. Savelsbergh
G.C. Sigismondi

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Eindhoven, November 1991
The Netherlands

Constraint Classification for Mixed Integer Programming Formulations

G.L. Nemhauser
Georgia Institute of Technology
School of Industrial and System Engineering
Atlanta, GA 30332-0205
USA

M.W.P. Savelsbergh
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

G.C. Sigismondi
Georgia Institute of Technology
School of Industrial and System Engineering
Atlanta, GA 30332-0205
USA

1. Introduction

The success of branch-and-cut algorithms for combinatorial optimization problems [Hoffman and Padberg 1985, Padberg and Rinaldi 1989] and large scale 0-1 linear programming problems [Crowder, Johnson, and Padberg, 1983] has lead to a renewed interest in mixed integer programming. The key idea of the branch-and-cut approach is reformulation. Problems are reformulated so as to make the difference in the objective function values between the solutions to the linear programming relaxation and the integer program as small as possible.

There are various ways to tighten the linear programming relaxation of an integer program. Preprocessing techniques [Hoffman and Padberg, 1991] try, among others things, to reduce the size of coefficients in the constraint matrix and to reduce the size of bounds on the variables. Constraint generation techniques [Crowder, Johnson and Padberg, 1983, Van Roy and Wolsey, 1986] try to generate strong valid inequalities.

Reformulation techniques should make the best possible use of the problem structure. It is beneficial to distinguish two modes of operation. General reformulation techniques, which are embedded in mixed integer programming systems such as ABC_OPT [Hoffman and Padberg 1989], MINTO [Savelsbergh, Sigismondi and Nemhauser 1991], MPSARX [Van Roy and Wolsey 1986], and OSL [IBM Corporation, 1990] try to identify problem structure based on an analysis of the constraint matrix. Problem specific reformulation techniques are based on an a priori investigation of the polyhedron associated with the set of feasible solutions.

The first step in the analysis of the constraint matrix is the classification of each constraint. The set of constraints is partitioned into a number of general types. The partition should be based on the specific structures that a system uses for its preprocessing, constraint generation, and branching strategies. As a result, there is no consensus on terminology and classification scheme. Generalized upper bound constraints, for instance, are defined as $\sum_{j \in S} x_j = 1$ in Nemhauser and Wolsey [1988] and as $\sum_{j \in S} x_j \leq 1$ in Wolsey [1990]

In this note, we define a classification scheme that is used in our system MINTO [Savelsbergh, Sigismondi and Nemhauser 1991]. Its purpose is to identify important classes to be used in preprocessing, constraint generation, branching, etc. We propose it as a general scheme to be evaluated, modified and then, hopefully, adopted, by the mixed integer programming community.

2. Constraint classification

A general mixed integer programming problem is of the form

$$\max \sum_{j\in B} c_j x_j + \sum_{j\in I} c_j x_j + \sum_{j\in C} c_j x_j$$

subject to

$$\begin{aligned} \sum_{j\in B} a_{ij} x_j + \sum_{j\in I} a_{ij} x_j + \sum_{j\in C} a_{ij} x_j \ \tilde{} \ b_i \quad & i=1,\ldots,m \\ 0 \le x_j \le 1 \quad & j\in B \\ l_{xj} \le x_j \le u_{xj} \quad & j\in I\cup C \\ x_j \in \mathbb{Z} \quad & j\in B\cup I \\ x_j \in \mathbb{R} \quad & j\in C \end{aligned}$$

where B is the set of binary variables, I is the set of integer variables, C is the set of continuous variables, the sense $\tilde{}$ of a constraint can be $\le$,$\ge$, or $=$, and the lower and upper bounds may be plus or minus infinity. See Nemhauser and Wolsey [1988] for a general treatment of the subject.

To classify constraints, we first distinguish binary variables from integer and continuous ones. Note that this is different from a variable classification that surely would separate integer and continuous variables if for no other reason than the need to do so in branching. However, we have not yet found a significant use of constraints that distinguish between integer and continuous variables, e.g., we do not use Gomory mixed integer cuts. We use the symbol y to indicate integer and continuous variables and x to indicate binary variables. Each constraint class will be an equivalence class with respect to complementing binary variables, i.e., if a constraint with term $a_j x_j$ is in a given class then the constraint with $a_j x_j$ replaced by $a_j(1-x_j)$ is also in the class. Consequently, the most general constraint that can appear in a mixed integer programming formulation can be represented as follows

$$\sum_{j\in B} a_j x_j + \sum_{j\in I\cup C} a_j y_j \ \tilde{} \ b,$$

where a_j for $j\in B$ and b are positive, and a_j for $J\in I\cup C$ are nonzero.

Furthermore, we distinguish variable bounds from simple bounds. In a constraint with a variable bound, there is a distinct binary variable that bounds all others if it is set to either 0 or 1. The most general constraint with a variable bound can be represented as follows

$$\sum_{j\in B} a_j x_j + \sum_{j\in I\cup C} a_j y_j \ \tilde{} \ a_k x_k,$$

where a_j for $j\in B$, a_j for $J\in I\cup C$, and a_k are all positive. Whenever we consider a constraint with a variable bound, we will assume that the distinct binary variable appears in the right-hand-side of the constraint and all other variables appear in the left-hand-side of the constraint.

The language we propose to define constraint classes uses six fields. The first field describes the type of variables that occur in the constraint. The second field specifies the number of variables in the constraint. The third field characterizes the coefficients of the variables in the constraint. The fourth field describes the type of bound. The fifth field specifies the sense of the bound. The sixth field characterizes the value of the bound.

The classification language consists of a set of rules that define allowable structures. Each rule defines a nonterminal symbol (classification or field) in terms of other nonterminal symbols and terminal symbols (values of fields, or 'tokens'); the symbol ∨ is used to represent an exclusive or. Each nonterminal symbol is enclosed in angular brackets. Each token is followed by a comment on its interpretation between square brackets. The token ° indicates the empty symbol; it is used to indicate a default value, which is usually either the simplest or the most appropriate value.

Each constraint class under consideration is defined by a number of tokens, some of which may be equal to °. For notational convenience, the tokens are represented as follows

$$\langle field\ 1\rangle^{\langle field\ 2\rangle}_{\langle field\ 3\rangle}\langle field\ 4\rangle\langle field\ 5\rangle_{\langle field\ 6\rangle}$$

The partitioning of the set of constraints into a number of classes, as done by mixed integer programming systems, is in fact nothing more than the identification of interesting special cases.

<classification> ::=

<type of variables>
<number of variables>
<coefficients of variables>
<type of bound>
<sense of bound>
<value of bound>

2.1. *Type of variables*

The first field specifies the type of variables that appear in the left-hand-side of the constraint, i.e., whether there are both binary and non-binary variables, just binary variables, or just non-binary variables.

<type of variables> ::= ° ∨ BIN ∨ MIX

°	[no binary variables]
BIN	[all binary variables]
MIX	[both binary and non-binary variables]

2.2. *Number of variables*

The second field specifies the number of variables that appear in the left-hand-side of the constraint, i.e., whether there is only a single variable or whether there is more than one variable. Note that in the case of a left-hand-side with both binary and non-binary variables there can never be a single variable.

<number of variables> ::= ° ∨ 1

°	[an arbitrary number of variables]
1	[a single variable]

2.3. *Coefficients of variables*
The third field specifies the coefficients of the variables that appear in the left-hand-side of the constraint, i.e., whether they all have coefficient c or whether they have arbitrary coefficients.

<coefficients of variables> ::= ° ∨ c

°	[arbitrary coefficients]
c	[all coefficients equal to c]

2.4. *Type of bound*
The fourth field specifies the type of bound. The type of bound can be either simple or variable. In constraints with a variable bound all variables in the left-hand-side have positive coefficients and there is a single binary variable with a positive coefficient in the right-hand-side.

<type of bound> ::= ° ∨ VAR

°	[simple bound]
VAR	[variable bound]

2.5. *Sense of bound*
The fifth field specifies the sense of bound. The sense of bound is determined by the sense of the constraint, which can be either ≤, =, or ≥.

<sense of bound> ::= UB ∨ EQ ∨ LB

UB	[upper bound]
EQ	[equal bound]
LB	[lower bound]

2.6. *Value of bound*
The sixth field specifies the value of the bound, i.e., whether it is equal to a specific value c or whether it can be an arbitrary value.

<value of bound> ::= ° ∨ c

°	[arbitrary bound]
c	[bound equal to c]

The language presented above defines a hierarchical structure of constraint classes. Figure 1 explicitly shows this hierarchical structure for the constraints with sense ≤. We now define for each constraint a unique constraint class by assigning it to the smallest class in the hierarchical structure that contains it. Table 1 illustrates the classification scheme by presenting classes for several types of constraints encountered in the literature.

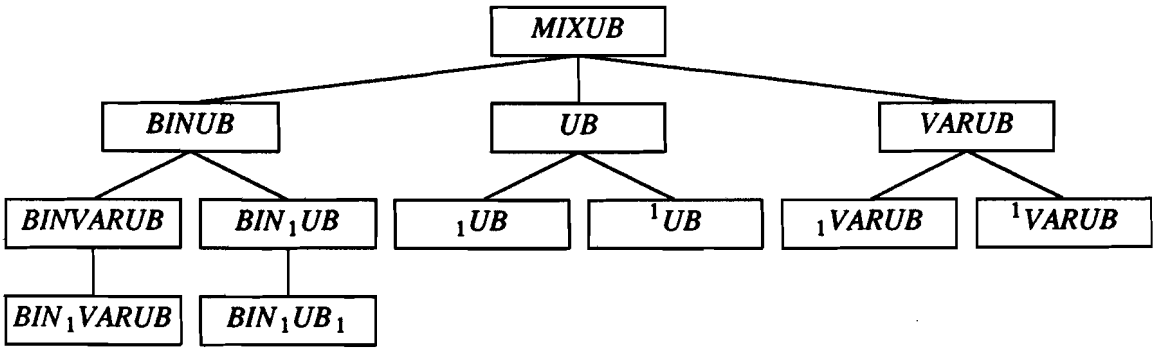

Figure 1. Hierarchical structure of less-than-or-equal constraints.

Classification	Inequality	Name
BIN_1EQ_1	$\sum_{j\in B} x_j=1$	generalized upper bound
BIN_1UB_1	$\sum_{j\in B} x_j\leq 1$	special ordered set
BIN_1UB	$\sum_{j\in B} x_j\leq k$ $(k\neq 1)$	invariant knapsack
BIN_1VARUB	$\sum_{j\in B} x_j\leq a_k x_k$	plant-location
BIN_1VARLB	$\sum_{j\in B} x_j\geq a_k x_k$	reverse plant-location
$BINUB$	$\sum_{j\in B} a_j x_j\leq a_k$	knapsack
^1VARUB	$a_j y_j\leq a_k x_k$	variable upper bound
^1VARLB	$a_j y_j\geq a_k x_k$	variable lower bound
^1UB	$a_j y_j\leq a_k$	simple upper bound
^1LB	$a_j y_j\geq a_k$	simple lower bound

Table 1. Examples of classifications.

References

H. Crowder, E.L. Johnson, M.W. Padberg (1983). Solving large-scale zero-one linear programming problems. *Oper. Res. 31*, 803-834.

K.L. Hoffman, M. Padberg (1985). LP-based combinatorial problem solving. *Annals of Oper. Res. 4*, 145-194.

K.L. Hoffman, M. Padberg (1989). *ABC_OPT: A branch-and-cut optimizer for solving large 0-1 linear programming problems.* (In preparation)

K.L. Hoffman, M. Padberg (1991). Improving LP-representation of zero-one linear programs for branch-and-cut. *ORSA J. Comput. 3*, 121-134.

IBM Corporation (1990). *Optimization Subroutine Library, Guide and Reference.*

G.L. Nemhauser, L.A. Wolsey (1988). *Integer Programming and Combinatorial Optimization.* Wiley, Chichester.

M. Padberg, G. Rinaldi (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review 33*, 60-100.

T. van Roy, L.A. Wolsey (1986). Solving mixed 0-1 programs by automatic reformulation. *Oper. Res. 35*, 45-57.

M.W.P. Savelsbergh, G.C. Sigismondi, G.L. Nemhauser (1991). *MINTO, a Mixed INTeger Optimizer*. Memorandum COSOR 91-18, Eindhoven University of Technology; also appeared as Computation Optimization Report 91-03, Georgia Institute of Technology.

L.A. Wolsey (1990). Valid inequalities for 0-1 knapsacks and MIPS with generalised upper bound constraints. *Discrete Applied Mathematics 29*, 251-261.

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science
PROBABILITY THEORY, STATISTICS, OPERATIONS RESEARCH
AND SYSTEMS THEORY
P.O. Box 513
5600 MB Eindhoven, The Netherlands

Secretariate: Dommelbuilding 0.03
Telephone : 040-473130

-List of COSOR-memoranda - 1991

Number	Month	Author	Title
91-01	January	M.W.I. van Kraaij W.Z. Venema J. Wessels	The construction of a strategy for manpower planning problems.
91-02	January	M.W.I. van Kraaij W.Z. Venema J. Wessels	Support for problem formulation and evaluation in manpower planning problems.
91-03	January	M.W.P. Savelsbergh	The vehicle routing problem with time windows: minimizing route duration.
91-04	January	M.W.I. van Kraaij	Some considerations concerning the problem interpreter of the new manpower planning system formasy.
91-05	February	G.L. Nemhauser M.W.P. Savelsbergh	A cutting plane algorithm for the single machine scheduling problem with release times.
91-06	March	R.J.G. Wilms	Properties of Fourier-Stieltjes sequences of distribution with support in [0,1).
91-07	March	F. Coolen R. Dekker A. Smit	Analysis of a two-phase inspection model with competing risks.
91-08	April	P.J. Zwietering E.H.L. Aarts J. Wessels	The Design and Complexity of Exact Multi-Layered Perceptrons.
91-09	May	P.J. Zwietering E.H.L. Aarts J. Wessels	The Classification Capabilities of Exact Two-Layered Peceptrons.
91-10	May	P.J. Zwietering E.H.L. Aarts J. Wessels	Sorting With A Neural Net.
91-11	May	F. Coolen	On some misconceptions about subjective probability and Bayesian inference.

COSOR-MEMORANDA (2)

91-12	May	P. van der Laan	Two-stage selection procedures with attention to screening.
91-13	May	I.J.B.F. Adan G.J. van Houtum J. Wessels W.H.M. Zijm	A compensation procedure for multiprogramming queues.
91-14	June	J. Korst E. Aarts J.K. Lenstra J. Wessels	Periodic assignment and graph colouring.
91-15	July	P.J. Zwietering M.J.A.L. van Kraaij E.H.L. Aarts J. Wessels	Neural Networks and Production Planning.
91-16	July	P. Deheuvels J.H.J. Einmahl	Approximations and Two-Sample Tests Based on P - P and Q - Q Plots of the Kaplan-Meier Estimators of Lifetime Distributions.
91-17	August	M.W.P. Savelsbergh G.C. Sigismondi G.L. Nemhauser	Functional description of MINTO, a Mixed INTeger Optimizer.
91-18	August	M.W.P. Savelsbergh G.C. Sigismondi G.L. Nemhauser	MINTO, a Mixed INTeger Optimizer.
91-19	August	P. van der Laan	The efficiency of subset selection of an almost best treatment.
91-20	September	P. van der Laan	Subset selection for an -best population: efficiency results.
91-21	September	E. Levner A.S. Nemirovsky	A network flow algorithm for just-in-time project scheduling.
91-22	September	R.J.M. Vaessens E.H.L. Aarts J.H. van Lint	Genetic Algorithms in Coding Theory - A Table for $A_3(n,d)$.
91-23	September	P. van der Laan	Distribution theory for selection from logistic populations.

COSOR-MEMORANDA (3)

91-24	October	I.J.B.F. Adan J. Wessels W.H.M. Zijm	Matrix-geometric analysis of the shortest queue problem with threshold jockeying.
91-25	October	I.J.B.F. Adan J. Wessels W.H.M. Zijm	Analysing Multiprogramming Queues by Generating Functions.
91-26	October	E.E.M. van Berkum P.M. Upperman	D-optimal designs for an incomplete quadratic model.
91-27	October	R.P. Gilles P.H.M. Ruys S. Jilin	Quasi-Networks in Social Relational Systems.
91-28	October	I.J.B.F. Adan J. Wessels W.H.M. Zijm	A Compensation Approach for Two-dimensional Markov Processes.
91-29	November	J. Wessels	Tools for the Interfacing Between Dynamical Problems and Models withing Decision Support Systems.
91-30	November	G.L. Nemhauser M.W.P. Savelsbergh G.C. Sigismondi	Constraint Classification for Mixed Integer Programming Formulations.