Computers & Operations Research 38 (2011) 1143–1152

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Faster integer-feasibility in mixed-integer linear programs by branching to force change

Jennifer Pryor, John W. Chinneck *

Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada K1S 5B6

ARTICLE INFO

Available online 27 October 2010

Keywords:
Mixed-integer programming
Branching
Integer feasibility

ABSTRACT

Branching in mixed-integer (or integer) linear programming requires choosing both the branching variable and the branching direction. This paper develops a number of new methods for making those two decisions either independently or together with the goal of reaching the first integer-feasible solution quickly. These new methods are based on estimating the probability of satisfying a constraint at the child node given a variable/direction pair. The surprising result is that the first integer-feasible solution is usually found much more quickly when the variable/direction pair with the smallest probability of satisfying the constraint is chosen. This is because this selection forces change in many candidate variables simultaneously, leading to an integer-feasible solution sooner. Extensive empirical results are given.

1. Introduction

Mixed-integer linear programs (MILP) are composed of a linear objective function and linear constraints over a set of variables, some or all of which are restricted to take on integer or binary values ("integer" is assumed to include "binary" as a special case hereafter). The most popular solution approach is branch and bound, supplemented with cuts and various other heuristics such as local searching (see e.g. Johnson et al. [1]). Nodes in the resulting search tree are variations on the original model with tightened bounds on the integer variables and/or added cut constraints. The bounding formula applied at a node in the search tree consists of the solution of a *linear programming (LP) relaxation* of the modified version of the original model represented by the node. The LP relaxation is a linear programming solution of the node model that simply ignores the integer restrictions on the variables.

A branch and bound node that is chosen for further expansion has an LP relaxation solution in which at least one of the integer variables does not have an integer value; such integer variables are *candidates* for branching. A candidate variable is chosen for branching and two child nodes are created: *branching up* adjusts the bound on the branching variable to be no less than the current value rounded up, while *branching down* adjusts the bound on the branching variable to be no more than the current value rounded down. If there are k candidate variables, then there are $2k$ ways to proceed from the current node to the next node in the usual depth-first exploration of the search tree. The heuristic used for choosing which of the $2k$ potential child nodes to explore next can have a major impact on the speed of the MILP solution.

There are two main ways to *branch*, i.e. to choose the child node to explore next. The most common approach is to first choose the candidate variable, and then choose the *branching direction* (i.e. decide whether to branch up or down). A great deal of research exists on heuristics for choosing the candidate variable, but there is surprisingly little research on the best way to choose the branching direction once the candidate variable has been selected. The second approach is to choose the branching variable and the branching direction simultaneously. There is relatively little research on techniques in this category.

This paper addresses the question of the best branching heuristic (i.e. the heuristic that most often reaches the first feasible solution fastest), given the node that is to be expanded. Exploring this question sheds some light on the characteristics of MILP models that affect how well various branching techniques work. Influential characteristics include the presence or absence of equality constraints, the inclusion of "multiple choice" constraints, and the fraction of inequality constraints that are violated by adjusting the branching variable in the up vs. down directions. The analysis uncovers some important general principles in branching.

The metric of interest in this paper is speed in reaching the first integer-feasible solution. This is important for several reasons. First, several classes of MILP problems do not have an objective function and require only a feasible solution. Second, in very large MILPs it is wise to reach an incumbent solution early so that at least one integer-feasible solution is in hand should the time limit be reached, and because an incumbent is then available for pruning the developing tree, thereby reducing the overall solution time.

* Corresponding author.
E-mail addresses: jpryor@sce.carleton.ca (J. Pryor), chinneck@sce.carleton.ca (J.W. Chinneck).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.10.025

Third, if the node selection heuristic is effective, then reaching an integer-feasible descendent of the chosen node quickly should lead to an optimum solution more quickly.

There are a variety of branching heuristics, though most are oriented towards fast optimality as opposed to fast integer-feasibility; see e.g. Achterberg et al. [2] for an overview and assessment. Many algorithms first choose the branching variable and then the branching direction, using separate algorithms for each decision. Given the branching variable, the most common direction selection heuristics are: branch up always, branch down always, or branch to the closest integer. Branching to the farthest integer is also sometimes used. A number of commercial solvers include a parameter that allows the user to make any of these choices, along with the choice to let the solver decide the branching direction using its own heuristic.

Previous research provides no definite conclusions about which branching direction heuristic is best. Meyer et al. [3] studied branching direction selection in the context of optimizing the placement of radioactive seeds for cancer treatment. They compared the branch up, branch down, and closest integer direction selection heuristics in combination with different node selection, variable selection and scaling techniques, as well as with variations on the MILP model itself. They conclude that the branching direction heuristic has a significant impact on solution times, but could not identify an overall best heuristic: each method performed better or worse depending on the scaling method used. For example, branching down worked well with aggressive scaling, and branching up worked best with standard scaling.

Jariwala [4] studied branch and bound solutions to the dynamic layout problem, comparing the usual three branching direction selection heuristics: branch up, branch down, and closest integer. He concludes that fixing the branching variable direction selection to either branch up or branch down is more effective than using the closest integer heuristic for this problem. Bernatzki et al. [5] looked at branching direction selection heuristics in a MILP model for optimizing scrap combination for steel production. They test branching up and branching down on the binary variables, and concluded that branching down performs best for this model.

Driebeek [6] developed a well-known branching heuristic that selects both the branching variable and the branching direction. Tomlin [7] extended Driebeek's idea by considering the integrality of variables. The resulting Driebeek and Tomlin method is a penalty method that estimates the potential degradation of the objective function value due to selecting a candidate variable, as estimated by performing a dual simplex pivot, which generates a lower bound on the bound improvement possible if a given candidate variable is selected as the branching variable [8]. Degradation bounds are calculated separately for branching up and for branching down. The largest degradation bound is used to choose the branching variable, and once chosen, the smaller of the bounds for that variable is used to choose the branching direction. This method is designed to reach optimality quickly, rather than integer feasibility. It is the default heuristic for branching variable and direction selection in the GLPK MILP solver [9] that is used in the experiments reported in this paper.

There are a variety of specialized algorithms for reaching integer-feasibility quickly in MILPs that are not solely branching heuristics. The pivot-and-shift algorithm [10] has a first phase that seeks integer feasibility using a variety of special techniques including rounding, specialized pivots and small neighbourhood searches. The OCTANE heuristic [11] for binary integer programs uses the intersection of the improving direction with the extended facets of the solution hyper-octagon to identify good binary solutions to try. The feasibility pump [12] alternates linear programming solutions with rounding. This paper concentrates on branching-related methods.

The general folklore is that branching in the up direction is usually best. This is definitely true in the case of so-called *"multiple choice"* constraints, which are composed entirely of binary variables and have the form $x_1+x_2+x_3+\cdots+x_n\ \{\leq,=\}1$. For these constraints, branching up on the branching variable (i.e. setting it to 1) forces all other variables in the constraint to zero, hence all variables take on integer values simultaneously. On the other hand, branching down on the branching variable (i.e. setting it to 0) allows the other variables to take on non-integer values, hence fewer, if any, are forced to integer values. Branching up on a variable in a multiple choice constraint is decidedly preferable for reaching integer-feasibility quickly.

While not guaranteed, each branch in a branch and bound solution will more often than not force the branching variable to an integer value. To reach an integer feasible solution more quickly it is desirable to force as many additional candidate variables as possible to integer values at each branch. How to do this is straightforward in the case of multiple choice constraints, but not so obvious for other classes of constraints. However because every candidate variable must be forced to an integer value to reach an integer-feasible solution it is obviously a poor idea to branch in such a way that few candidate variables are affected. The generalization of this idea is that branching should force as many candidate variables as possible to change their values, whether or not it can be guaranteed that they will change to integer values: some *may* be forced to integer values, thereby speeding the solution.

This brings us to the central theme of this paper, namely branching to force change in the candidate variable values. This is extremely effective in the case of multiple choice constraints, where branching up forces all candidate variables in the constraint to integer values simultaneously. This principle has not been previously articulated as a central motivation in branching heuristics. Most branching variable selection heuristics concentrate on forcing change in the value of the objective function. The single exception is the active constraints branching variable selection method of Patel and Chinneck [13], which concentrates on choosing the branching variable that has the greatest impact on the active constraints in the current LP-relaxation solution. Patel and Chinneck's *Method A* chooses the candidate variable that appears in the largest number of active constraints. In so doing it is also choosing the candidate variable that affects the values of the largest number of other candidate variables via their involvement in the active constraints. This is a very effective heuristic, outperforming state of the art commercial MILP solvers in reaching the first integer-feasible solution quickly. We will return throughout the paper to this theme of choosing the branching variable and branching direction so as to force change in the values of numerous candidate variables.

2. Experimental setup

The conclusions in this paper are based on extensive computational experimentation. The experimental conditions are described below.

The open-source GLPK MILP solver version 4.28 [9] was modified extensively to test a variety of existing and novel branching heuristics. All parameters were set at their default values with the following exceptions:

- *Stopping conditions*: solutions ran for a maximum of 2 h, or stopped earlier upon finding the first integer-feasible solution.
- *Node selection*: depth-first, except where noted.
- *Branching variable selection and branching direction selection*: as required for the experiment at hand.

Hardware consisted of four computers running Windows XP: a Pentium 4 CPU at 3.40 GHz with 1 GB of RAM, an Intel Core 2 CPU at