EURO Journal on Computational Optimization
https://doi.org/10.1007/s13675-020-00130-z

ORIGINAL PAPER

An exploratory computational analysis of dual degeneracy in mixed-integer programming

Gerald Gamrath[1,2] · **Timo Berthold**[3] · **Domenico Salvagnin**[4]

Received: 1 October 2019 / Accepted: 10 July 2020

Abstract
Dual degeneracy, i.e., the presence of multiple optimal bases to a linear programming (LP) problem, heavily affects the solution process of mixed integer programming (MIP) solvers. Different optimal bases lead to different cuts being generated, different branching decisions being taken and different solutions being found by primal heuristics. Nevertheless, only a few methods have been published that either avoid or exploit dual degeneracy. The aim of the present paper is to conduct a thorough computational study on the presence of dual degeneracy for the instances of well-known public MIP instance collections. How many instances are affected by dual degeneracy? How degenerate are the affected models? How does branching affect degeneracy: Does it increase or decrease by fixing variables? Can we identify different types of degenerate MIPs? As a tool to answer these questions, we introduce a new measure for dual degeneracy: the variable–constraint ratio of the optimal face. It provides an estimate for the likelihood that a basic variable can be pivoted out of the basis. Furthermore, we study how the so-called cloud intervals—the projections of the optimal face of the LP relaxations onto the individual variables—evolve during tree search and the implications for reducing the set of branching candidates.

Keywords Mixed integer programming · Dual degeneracy · Linear programming · Branch-and-bound

✉ Gerald Gamrath
gamrath@zib.de

Timo Berthold
timoberthold@fico.com

Domenico Salvagnin
salvagni@dei.unipd.it

1 Zuse Institute Berlin, 14195 Berlin, Germany

2 I[2]DAMO GmbH, Englerallee 19, 14195 Berlin, Germany

3 Fair Isaac Germany GmbH, Stubenwald-Allee 19, 64625 Bensheim, Germany

4 DEI, Via Gradenigo, 6/B, 35131 Padua, Italy

Published online: 01 August 2020

Mathematics Subject Classification 90C10 · 90C11 · 90C57

1 Introduction

In this paper, we analyze dual degeneracy emerging in linear programming (LP) relaxations solved during the solution process of generic *mixed-integer (linear) programs* (MIPs) in standard form:

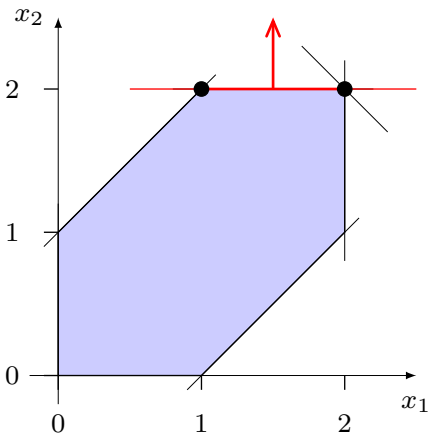
$$\min\{c^T x : Ax = b, x \geq 0, x_j \in \mathbb{Z}\ \forall j \in J\}$$

where $x \in \mathbb{R}^n$ is the solution vector, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $J \subseteq N = \{1, \ldots, n\}$. In the following, x may consist of both structural variables as well as artificial slack variables introduced to obtain this form. For the ease of presentation, we do not consider explicit upper bounds on the variables in our notation. However, all measures introduced in the following can be easily extended to that case.

MIPs are typically solved with the LP-based branch-and-bound algorithm (Land and Doig 1960; Dakin 1965), which recursively splits the problem into subproblems until those are easy to solve or can be shown to not contain a solution that improves the best one found so far. In this process, LP relaxations of the subproblems—obtained by omitting the integrality restrictions—are solved repeatedly. On the one hand, they provide solution candidates; on the other hand, the optimal LP value provides a lower bound on the optimal value of the current subproblem.

The algorithm of choice to solve LP relaxation during the branch-and-bound process is the simplex algorithm (Dantzig 1951; Lemke 1954; Beale 1954). It chooses a subset B of the variable index set $N = \{1, \ldots, n\}$ of size m such that the square matrix $A_{\cdot B}$, containing only the columns with index in B, is regular. Then, $x_B := A_{\cdot B}^{-1} b$ and $x_{N \setminus B} := 0$. Now, in each iteration of the simplex algorithm, a pivot operation is performed which exchanges one of the variables in B (called the *basic variables*) for one in $\bar{B} = N \setminus B$ (the *non-basic variables*). Given any basis B, the objective value of any LP solution can be expressed as the sum of the current LP objective and the sum of the products of the change in the value for each non-basic variable with its reduced cost $\bar{c}_j$, see Chvatal (1983). As a consequence, an LP solution with corresponding basis B is optimal if $\bar{c}_j \geq 0$ for all $j \in \bar{B}$, i.e., the reduced costs for all nonbasic variables are non-negative. The basis B also determines a solution to the dual LP $\max\{y^T b : y^T A \leq c, y \in \mathbb{R}^m\}$. A basis B is called *optimal* if both the associated primal and dual solutions are feasible, implying that both solutions are optimal for their respective problem.

Degeneracy describes the case that multiple simplex bases correspond to the same primal or dual solution of a linear program (LP). It has been a topic of interest since the invention of the simplex method, see, e.g., Orchard-Hays (1958) and Chvatal (1983). A solution to an LP is called *primal degenerate* if there are basic variables set to 0 in the solution. Those variables can be pivoted out of the basis to obtain a different basis that still represents the same primal solution. Primal degeneracy can be a consequence of redundant constraints, but may as well be inherent in the specific problem and unavoidable.



$$\begin{cases} \min -x_2 \\ x_1 - x_2 \leq 1 \\ x_1 + x_2 \leq 4 \\ x_1 - x_2 \geq -1 \\ 0 \leq x1 \leq 2 \\ 0 \leq x2 \leq 2 \end{cases}$$

Fig. 1 Illustration of primal and dual degenerate solutions

In this paper, we are focusing on *dual degeneracy*. An LP solution is called *dual degenerate* if the corresponding dual solution is (primal) degenerate. The dual solution is degenerate if one of the primal non-basic variables has a reduced cost of zero. These variables will be called *dual degenerate* in the following. A dual degenerate optimal LP solution implies that there might be alternative optimal solutions to this LP. More specifically, each dual degenerate variable can be pivoted into the basis without changing the objective value. Thus, multiple optimal bases are guaranteed to exist. If the variable that is pivoted out of the basis is not subject to primal degeneracy, i.e., was not at its bound before, this corresponds to a distinct primal optimal solution represented by the new basis.

An illustration of the two types of degeneracy is given in Fig. 1. The two black points (and all points on the line in between) are optimal solutions; thus, the two black points are both dual degenerate basic solutions. The black point on the left, with coordinates $(1, 2)$, lies at the intersection of exactly two hyperplanes, defined by constraints $x_2 \leq 2$ and $x_1 - x_2 \geq -1$: with only two structural variables (plus slack variables), this solution is primal non-degenerate. On the other hand, the black point on the right, with coordinates $(2, 2)$, lies at the intersection of three hyperplanes, associated to the three constraints $x_1 \leq 2$, $x_2 \leq 2$, and $x_1 + x_2 \leq 4$. Even if both structural variables are basic, the slack variable of one of those inequalities has to be basic as well, although the inequalities are all fulfilled with equality, meaning that one slack variables has value zero and is degenerate. Therefore, this solution is primal degenerate. Again, note that while in this simple two-dimensional example primal degeneracy is caused by the presence of a *redundant* constraint, $x_1 + x_2 \leq 4$, this is not true in higher dimensions, so primal degeneracy is in general unavoidable. Finally, the slack variable associated with $x_1 - x_2 \geq -1$ has a reduced cost of zero on the left black point, which is thus dual degenerate.

In case of dual degeneracy, which of the alternative optimal solutions is returned by a simplex-based LP solver is arbitrary and cannot be predicted in most cases. An exception is the case when the simplex algorithm employs the lexicographic pivot-

ing rule, which is, however, rarely used in practical applications due to its inferior performance. While obtaining a random optimal solution may be acceptable when solving a pure LP, it quickly becomes an issue for LP relaxations being solved during the solving process of a MIP solver. Such a solver takes many decisions based on the current LP solution, and, if this solution is just arbitrary among many possible ones, a slight change in the simplex implementation can lead to a very different LP solution being returned. As a consequence, the decisions that are taken based on the LP solution may change, and therewith the subsequent solving process can vary significantly. Such sensitivity to small, seemingly performance neutral, changes in the algorithm is called *performance variability* (Koch et al. 2011; Lodi and Tramontani 2013) and should typically be avoided. Taking into account dual degeneracy and potentially multiple optimal LP solutions may be a means to reduce performance variability and make the solver more stable.

Besides the reduction of performance variability, dual degeneracy can be exploited within a MIP solver to take better decisions. Rather than relying on the limited information that a single computed LP solution provides, it may be preferable to take into account multiple optimal LP solutions or to select a particular one among the set of optimal solutions. The primal solution polishing method, as implemented in SoPlex 3.1 (Gleixner et al. 2017a), is an example of the latter case, which mainly aims at increasing the integrality of the LP solution to aid primal heuristics and branching rules. On the other hand, concurrent root cut loops (Fischetti et al. 2016) are an example of the former case that exploits a small set of optimal solutions for the separation process. The pump-reduce procedure (Achterberg 2013) combines both ideas by trying to construct an LP solution with fewer fractionalities and using both the original as well as the updated LP solution for the selection of cutting planes. For primal degenerate LPs, dual variable picking (Bajgiran et al. 2017) tries to increase the number of reduced cost fixings by the investigation of multiple dual solutions. Multiple ways to exploit degeneracy in different algorithmic components of the MIP solver GUROBI are discussed by Achterberg (2018). Finally, the authors of this paper investigated branching improvements obtained by exploiting dual degeneracy and the knowledge of multiple optimal LP solutions (Berthold and Salvagnin 2013; Berthold et al. 2019a).

All these works are based on the claim that degeneracy is very common in real-world MIP instances. In this paper, we perform an empirical analysis of dual degeneracy in MIP instances in order to reinforce that claim and provide further motivation to consider degeneracy in more components of MIP solvers.

The paper is structured as follows. In Sect. 2 we introduce two degeneracy measures, and then use them to computationally analyze, at the end of the root node, how many models are affected by dual degeneracy and to which extent, using a collection of public benchmark instances. Then, in Sect. 3, we evaluate how degeneracy evolves during tree search, showing that, on average, the degree of dual degeneracy stays almost constant. Finally, in Sect. 4, we evaluate how dual degeneracy affects LP solution values, and comment on the implications of our findings on the branching variable selection. In particular, we show that a high dual degeneracy often allows a significant number of branching candidates to move to an integer value while staying on the optimal face, indicating that those are poor branching candidates, and this should be taken into account by branching strategies. Conclusions are drawn in Sect. 5.

2 Dual degeneracy measures

We will use two measures of dual degeneracy in the following. The first one is a classical measure in this context, and follows directly from the definition of dual degeneracy. The *degeneracy rate* counts the relative number of dual degenerate non-basic variables.

Definition 1 (*Degeneracy rate*) Let an LP $\min\{c^T x : Ax = b, x \geq 0\}$ be given as well as an optimal basis B and the corresponding non-basis $\bar{B}$ and reduced costs $\bar{c}$. Then, we call

$$\alpha = \frac{|\{j \in \bar{B} \mid \bar{c}_j = 0\}|}{|\bar{B}|}$$

the *degeneracy rate* of the basic solution.

The degeneracy rate α expresses which share of the non-basic variables may be pivoted into the basis, which potentially assigns a fractional solution value to these variables that are initially at their (integral) bound. However, it may be even more interesting to know that a basic variable with a fractional value may be pivoted out of the basis, making it integral. For example, such a variable is typically considered a bad candidate for branching as the dual bound of at least one child node will not be improved by branching on that variable.

When considering a particular instance, we would expect an increased degeneracy rate to result in a higher number of basic variables that can be pivoted out of the basis. However, the share of non-basic dual degenerate variables would be the same (50%) if one of only two non-basic variables were dual degenerate as it would be if 500,000 out of a million non-basic variables were dual degenerate. For the same number of constraint and thus, the same basis size, however, the latter will probably allow pivoting more of the current basic variables out of the basis.

This observation is our motivation for introducing a second dual degeneracy measure. It is based on an analysis of the optimal LP face, which can be obtained by fixing all variables—including slacks—to their current LP value whose reduced costs are non-zero, using the reduced costs associated with the starting optimal basis B. Then, one can use an appropriately modified objective function and optimize over the restricted LP to move to different optimal bases that potentially represent alternative LP optima, an approach used by several ideas to exploit dual degeneracy (Achterberg 2013; Berthold and Salvagnin 2013). The *variable–constraint ratio* of the optimal face is given by the quotient of the number of unfixed variables and constraints in the restricted problem.

Definition 2 (*Variable–constraint ratio*) Let an optimal basis B for the LP $\min\{c^T x : Ax = b, x \geq 0\}$ be given as well as the corresponding non-basis $\bar{B}$ and reduced costs $\bar{c}$. Then, we call

$$\beta = \frac{|\{j \in N \mid \bar{c}_j = 0\}|}{|B|}$$

the *variable–constraint ratio* of the basic solution.

Note that we write $|B|$ in the denominator, since the basis size is equal to the number of constraints. Since the basic variables have zero reduced costs by definition, the variable–constraint ratio is always 1 or larger. For highly degenerate problems, it can be arbitrarily large if the problem has many more variables than constraints.

While even a single dual degenerate variable can be sufficient to allow pivoting all basic variables out of the basis in turns, we expect the chance that many of the basic variables can be pivoted out of the basis to be higher the more dual degenerate non-basic variables there are per basic variable, i.e., the higher the variable–constraint ratio is.

Our computational analysis of dual degeneracy was performed using the academic MIP solver SCIP 5.0.1 (Achterberg 2007; Gleixner et al. 2017a) with SoPlex 3.1.1 (Wunderling 1996; Gleixner et al. 2017a) as the underlying LP solver. We used a development version of SCIP (git hash `0b5a1e8`) to determine and analyze the dual degeneracy of the LP solutions obtained during the MIP solving process.

The observations we describe in the following naturally depend on the solver and, in particular, the branching rule used for the experiments. We used reliability branching (Achterberg et al. 2005), because it is a state-of-the-art rule used (in slight variations) by many solvers. Therefore, we are optimistic that the results should transfer both to other solvers and other common branching rules.

As test set, we use the MMMC test set which is the union of different common MIP benchmark sets. It contains all instances from MIPLIB 3 (Bixby et al. 1998), MIPLIB 2003 (Achterberg et al. 2006), and the MIPLIB 2010 benchmark set (Koch et al. 2011) as well as the Cor@l test set (http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/), which mainly contains instances that users worldwide submitted to the NEOS server (Czyzyk et al. 1998). Duplicate instances were removed, leaving us with a total of 496 instances.

All computational experiments presented in the following were performed on a compute cluster with 48 compute nodes, running only one job per node at a time. Each compute node provided two Intel Xeon Gold 5122 CPUs at 3.60 GHz with 96 GB main memory. Computing detailed degeneracy information can be time consuming; therefore, we applied a time limit of 2 days per job.

To get a first impression of how these instances are affected by dual degeneracy, we analyze the final root LP solution of SCIP in the remainder of this section. That is the LP solution after presolving, node preprocessing, the cutting plane separation loop, and potential restarts. In the next section, we will examine the amount of degeneracy inherent in the LP solutions throughout the branch-and-bound tree search.

Note that we applied the default cutting plane separation methods of SCIP . We did an additional experiment where we disabled separation completely: however, this did not result in a significant difference concerning the observed degrees of degeneracy.

Figure 2 illustrates the degeneracy rate for the final root LP solutions. We disregard non-basic variables that have been fixed at the root node, independently of their reduced costs. The degeneracy rate is between 0% (if all non-basic unfixed variables have non-zero reduced cost) and 100% (if all non-basic unfixed variables have reduced cost zero). Each bar shows the number of instances with degeneracy rate in a certain 10% range (except for the first bar, all bars are left-open). Instances with the extreme degeneracy rates of 0% and 100% are highlighted in the respective bars by a darker color.

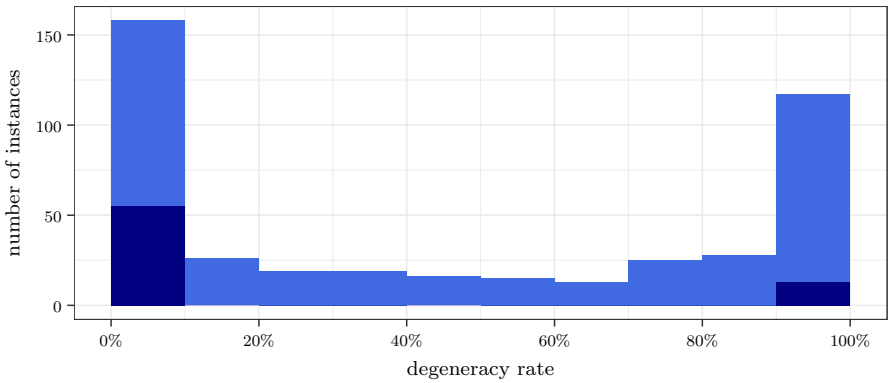


Fig. 2 Share of non-basic variables that are dual degenerate (final root LP). The darker parts represent instances with the extreme degeneracy rates of 0% and 100%

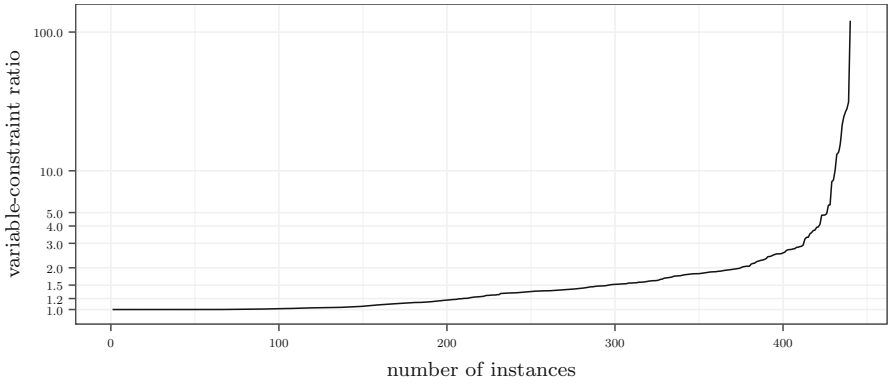


Fig. 3 Variable-constraint ratios (log scale) of the optimal face for instances in the MMMC test set (final root LP)

Out of the 496 instances, 56 instances are solved at the root node before degeneracy information is computed. Of the remaining 440 instances, only 55 instances show no dual degeneracy in the solution of the final root LP. On the other hand, 13 instances have only degenerate non-basic variables, i.e., have a degeneracy rate of 100%. These instances are pure feasibility problems with a zero objective function in the original model (6 instances), after presolving (4 instances), or after root processing (3 instances). For the majority of instances, degeneracy tends to cover either only a few or almost all variables: 158 instances have a degeneracy rate no larger than 10% while the rate is larger than 90% for 119 instances. The remaining instances distribute among the degeneracy rates from 10 to 90% with slightly fewer instances at medium ranges.

The variable–constraint ratios for the final root LP of the 440 instances from the MMMC test set not solved before degeneracy information was computed are presented in Fig. 3. There are 55 instances with a ratio of 1.0, which are the instances showing no degeneracy at all. 111 and 129 instances have ratios larger than 1 but no larger than

1.1, and larger than 1.1 but no larger than 1.5, respectively. Overall, 374 instances have a variable–constraint ratio of the optimal face no larger than 2.0 and 412 instances have a still reasonable ratio of 3.0 or smaller. On the other hand, there are 10 instances with a ratio larger than 10; one of them even has a ratio of 120.

Let us note that we were not able to identify a correlation between the degeneracy measures at the root node and the hardness of a problem or its performance variability. A reason for this may be that degeneracy measures can vary significantly during the tree search, as we will see in the following.

3 Evolution of dual degeneracy throughout the tree search

We observed degeneracy in the final root LPs of most of the regarded instances, but that alone is not enough to understand how degeneracy evolves during the tree search. In order to investigate this, we ran each instance with a node limit of 1 million, and computed the average degeneracy per depth level by first averaging over all nodes of one depth for each individual instance, and then averaging again over the instances averages for each depth level. Note that we disregard variables fixed by branching or domain propagation for the degeneracy computation and only compute the degeneracy share of the unfixed variables. The observed average degeneracy rate is almost constant during the tree search. At the root node, the average degeneracy rate is 45.8%, at depth 100, it is 45.7%. In between, it varies slightly between 44.4 and 49.4%.

While the average degeneracy rate is almost constant during the tree search across the testbed, it can evolve significantly on individual instances, in both directions. This is depicted in Fig. 4, which focuses on the change in the average degeneracy rate per depth level for the first 20 levels of the tree,[1] compared to the degeneracy of the final LP of the root node (after potential restarts). Each instance is represented by one point per depth level; the color of the point represents the root degeneracy of the instance. The x-axis represents the depth level; coordinates are jittered and points are drawn translucent to better display the density of instances at common degeneracy differences. The y-coordinate of a point represents the average difference between the degeneracy of that instance at the corresponding depth level to its degeneracy rate at the root node. Note that we use a square root scaling of the y-axis, i.e., we take the square root of the absolute difference, but keep the sign. This scaling provides a higher resolution for smaller values while still allowing for values with an absolute value smaller than one. In particular, it allows for zero values, as opposed to a log scaling.

We observe that many instances change their degeneracy rate during the tree search. There are roughly the same number of instances with increasing degeneracy rate at deeper levels as with decreasing degeneracy rate. The variance increases with the depth, which is reasonable as the deeper a subproblem is in the tree, the more different it is to the global problem. This explains why we observed an almost constant average.

[1] We chose this limit because the number of instances reaching this level is still reasonable: out of the 440 instances for which we computed the degeneracy at the root node, 327 reached a tree depth of at least 20.

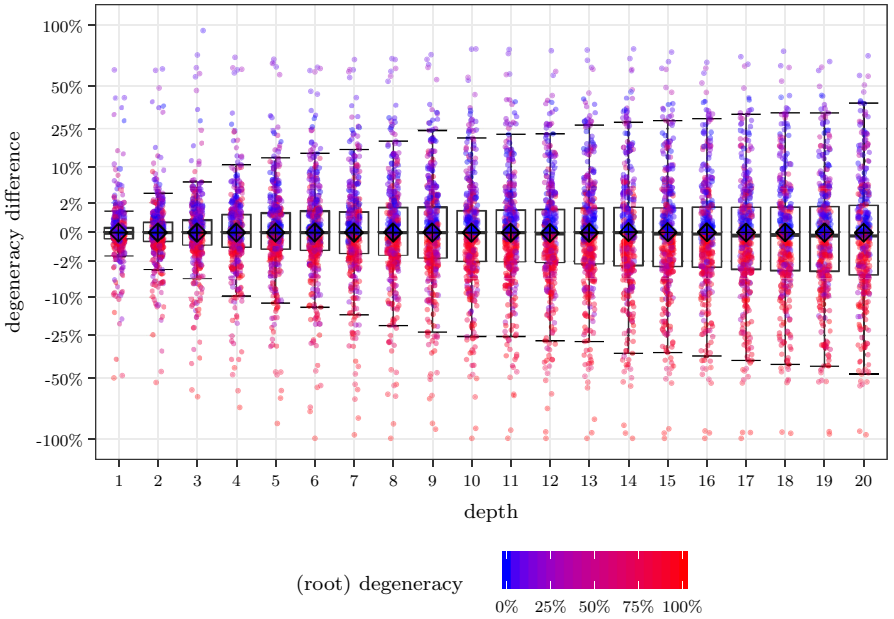


Fig. 4 Difference between average degeneracy and root node degeneracy per depth level. Each point represents one instance, their distribution is visualized by the underlying box plot and the average is represented by ⟠ (color figure online)

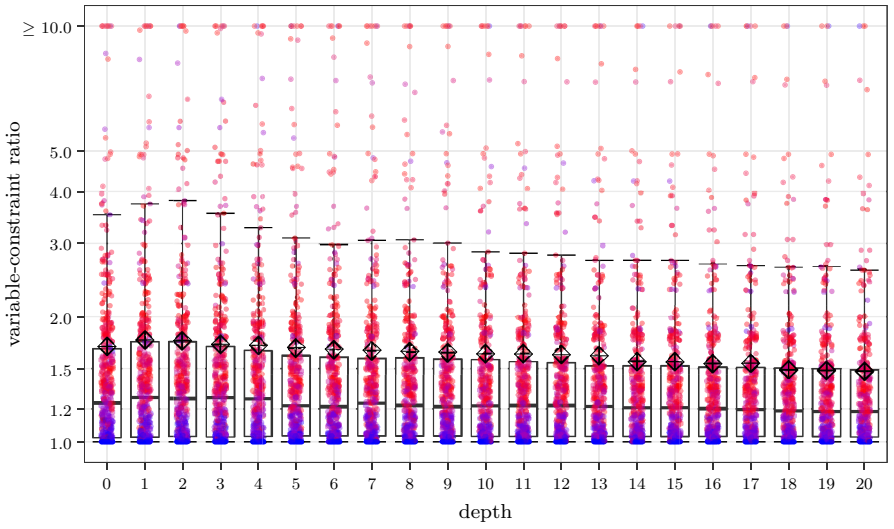


Fig. 5 Average variable–constraint ratio of the optimal face per depth level in the branch-and-bound tree. Each point represents one instance, their distribution is visualized by the underlying box plot, and the average is presented by ⟠ (color figure online)

In order to better evaluate the distribution of the points, Fig. 4 also shows a box plot. It shows that the median is around 0 for most of the depth levels; it reaches its highest absolute value at level 20, where it is at -0.03%. Half of the instances do not change their degeneracy rate a lot, as illustrated by the box. Up to depth level 5, 50% of the instances change their degeneracy rate by less than 1%, compared to the root node. Up to depth 10, the change is less than 2% for half of the instances. Even at level 20, the average change in degeneracy rate is between -4.3 and 1.7% for 50% of the instances. On the other hand, there are also 25.7% of the instances that change their average degeneracy rate by more than 10% at level 20, and almost 12% that change by more than 25%. Unsurprisingly, instances that reduce their degeneracy rate typically started with a high degeneracy rate at the root node, while those increasing the degeneracy rate often had comparably small degeneracy rates at the root LP.

Figure 5 shows the development of the variable–constraint ratio in the tree. As in the previous figure, each instance is represented by one point for each depth level, jittered to better show the number of instances in common regions. The y-coordinate corresponds to the variable–constraint ratio, using a $\log_{10}$ scale this time since all values are greater than or equal to 1.0. The color of each point displays the average degeneracy rate of this instance at the corresponding depth level. The y-axis is capped at 10 to allow a more detailed view of the most interesting region. There were 134 instance/depth combinations (coming from 27 different instances) with higher ratios.

We observe a tendency that the variable–constraint ratio slightly decreases when the depth increases. An exception is the larger ratio in depth one compared to the root node, which is consistently identified by average, median, and the quartiles in the box plot. An implementation detail of SCIP causes this increase: at the end of the root node, all cutting planes that are not tight are removed from the LP. Thus, the denominator of the variable constraint ratio is decreased for the following nodes. Therefore, the ratio increases. The box plot shows that 75% of the instances have an average variable–constraint ratio of less than 1.75 at depth 1 and 2. This number decreases by increasing depth level and is less than 1.5 for depth levels 19 and 20. We can also see that no more than 25% of the instances have a ratio smaller than 1.02, which means that for more than half of the instances, the ratio is in a reasonable range. We further observe that instances with a large variable–constraint ratio typically have a high degeneracy as well. There are exceptions, though, one of them being instance neos-495307, which has an average degeneracy rate of less than 4% at all depth levels. However, since its matrix dimensions are very unbalanced (it has more than 9000 variables and only 3 constraints), the variable constraint ratio is very large. It slowly decreases from 26 at the root node to 18 at depth 20.

This study of degeneracy allows us to give an answer to the question we posed at the beginning of Sect. 2 concerning the frequency of dual degeneracy in standard MIP models. For 87.5% of the instances, the root LP solution is subject to dual degeneracy. Throughout the tree, almost half of the non-basic variables are dual degenerate on average. At the same time, the variable–constraint ratio is larger than 1.2 for more than half of the instances up to depth level 17. Together, these numbers indicate that many variables can change their status from basic to non-basic or vice versa in alternative optimal LP solutions.

We conclude that dual degeneracy is prevalent in standard MIP models. It should therefore become standard to consider degeneracy in the development of LP-based algorithms for MIP solving, be it components of a solver or stand-alone methods, be it general purpose software or problem-specific applications.

In addition, the analysis above also shows that different models exhibit different manifestations of dual degeneracy, ranging from constant degeneracy to degeneracy affected (in either direction) by branching. As such, MIP solver components should be aware of the different scenarios and react accordingly, not just with the aim of handling existing degeneracy, but also trying to avoid introducing it.

4 The effect of dual degeneracy on LP solution values

So far, we demonstrated that dual degeneracy is very common in real-world problems. How many of the non-basic variables are affected by dual degeneracy is represented by the degeneracy rate. But being dual degenerate alone does not guarantee that a variable can indeed take different values on the optimal face. In this section, we investigate how often this happens and how this relates to the discussed degeneracy measures. In particular, we aim at answering the following two questions:

- How many of the integer variables can change their value while staying on the optimal face?
- How often can they even achieve an integer value on the optimal face?

We introduced the variable–constraint ratio as a second measure for dual degeneracy with the intent to capture those effects better than the degeneracy rate. Our analysis will show if this goal was accomplished.

The range in which a variable's value can be moved while staying on the optimal face is captured in the *cloud interval* introduced by Berthold and Salvagnin (2013). For a set $\mathcal{S}$ of alternative LP optima, the cloud interval $\mathcal{I}_j$ for an integer variable x_j is defined by

$$\mathcal{I}_j = \left[\min\{x'_j \mid x' \in \mathcal{S}\}, \max\{x'_j \mid x' \in \mathcal{S}\}\right].$$

We are especially interested in the cloud intervals of integer variables. In particular, we want to identify variables that can take different values in alternative LP optima, which corresponds to a cloud interval containing more than a single point. We call such a cloud interval *non-trivial*. Additionally, we are interested in the case that the cloud interval contains an integer value, because this implies that there is at least one optimal solution for which that variable is integral. This might have implications for solver components, e.g., branching on such variables should typically be avoided since the dual bound will not improve for at least one child node.

As already mentioned in Sect. 2, alternative LP optima can be computed by restricting the LP to the optimal face and optimizing auxiliary objective functions (Achterberg 2013). In this paper, we minimize and maximize each variable which is not yet fixed, in a similar fashion as optimization-based bound tightening (see, e.g., Zamora and Grossmann (1999), Caprara and Locatelli (2010), Gleixner et al. (2017b)). In the worst case,

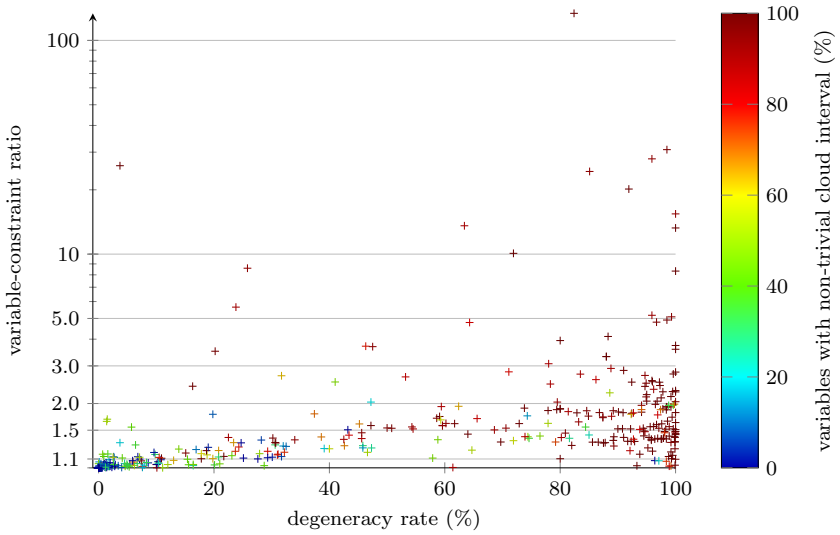


Fig. 6 Share of unfixed integer variables that can change their value on the optimal face, plotted by dual degeneracy rate (x-axis) and variable–constraint ratio of the optimal face (y-axis) (color figure online)

this requires to solve $2m + n$ auxiliary LPs: two LPs for each of the m basic variables and one LP for each non-basic variable. Non-basic variables are initially set to one of their bounds and can only move in one direction.

Using filtering techniques, see Gleixner et al. (2017b), the number of LPs to be solved can often be reduced. Nevertheless, it will typically be too large to use this approach in a practical application at every node of the branch-and-bound tree. From a theoretical point of view, however, this method is interesting since it determines the largest possible cloud intervals for each variable. Therefore, we will use this method for a deeper analysis of the effects of dual degeneracy in MIP models in the following. More practical methods to sample alternative LP optima are discussed by Berthold et al. (2019a).

Figure 6 shows how many of the integer variables that remained unfixed after restricting to the optimal face have a non-trivial cloud interval. Each instance is represented by one cross that is positioned according to the share of non-basic variables that are degenerate (x-coordinate) and the variable–constraint ratio of the optimal face (y-coordinate, log scale). The color encodes the relative number of integer variables with non-trivial cloud interval.

We see that a small degeneracy rate typically leads to a small variable–constraint ratio and a small number of integer variables that can change their value on the optimal face. That is the expected behavior, but there are outliers as well, typically with larger variable–constraint ratios, that result in a high share of variables with non-trivial cloud intervals. In general, a variable–constraint ratio of 2.0 or larger leads to non-trivial cloud intervals for the majority of integer variables. Out of the 63 instances that fall into this category, 26 have a non-trivial cloud interval for all unfixed integer variables, 48 for at least 90% of the variables. For high degeneracy rates, the variable–constraint

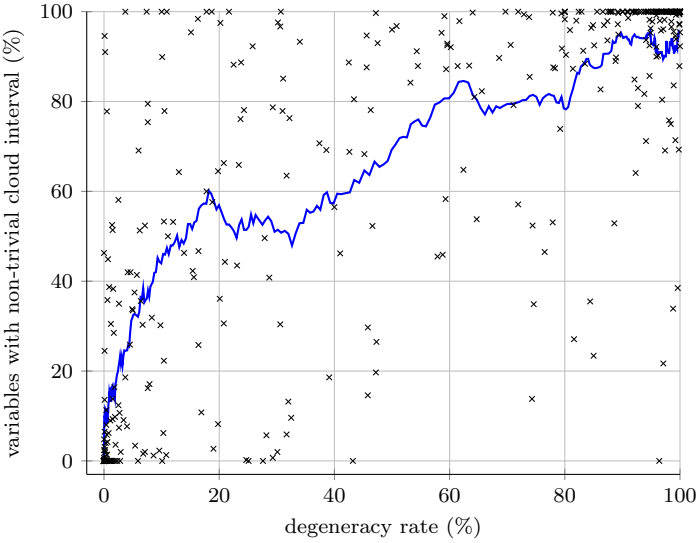


Fig. 7 Relation between dual degeneracy share (x-axis) and share of unfixed integer variables that can change their value on the optimal face (y-axis). The blue line shows the moving average over the 30 closest instances by degeneracy rate (color figure online)

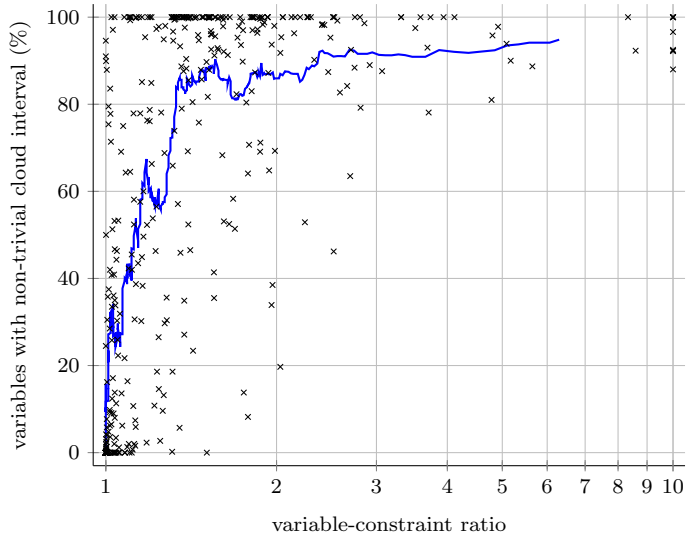


Fig. 8 Relation between variable–constraint ratio (x-axis) and share of unfixed integer variables that can change their value on the optimal face (y-axis). Ten instances with a variable–constraint ratio larger than 10 are printed with a ratio of 10 to improve readability. The blue line shows the moving average over 30 closest instances by degeneracy rate (color figure online)

ratios increase. The relative number of non-trivial cloud intervals increases as well. Out of the 152 instances with a degeneracy rate of 80% or higher, 81 can move all unfixed integer variable on the optimal face. Out of the same set of 152 instances, 124

can move at least 90% of the variables. As a comparison: out of the 256 instances that have a variable–constraint ratio less than 2.0 and a degeneracy rate of less than 80%, only 33 have a non-trivial cloud interval for at least 90% of the unfixed integer variables.

The one outlier in the upper left area is again instance neos-495307 with a degeneracy rate of 3.7% and a variable–constraint ratio of 25.9. Consequently, all 352 integer variables at the root node with reduced costs 0 have non-trivial cloud intervals. We would not have expected this based on the degeneracy rate alone, which illustrates the usefulness of the variable–constraint ratio as a degeneracy measure.

Next we analyze how the percentage of variables that do not take a distinct value on the optimal face is correlated to either the degeneracy rate or the variable–constraint ratio. To depict this correlation, we use moving averages. In Figs. 7 and 8, each instance is represented by a cross, with degeneracy rate and the variable–constraint ratio being the x-coordinate, respectively, and the percentage of variables with a non-trivial cloud interval being the y-coordinate. The blue line shows a moving average taken over 30 consecutive points according to the quantity on the x-axis. Note that Fig. 7 uses a linear scale and Fig. 8 uses a logarithmic scale.

The crosses in Fig. 7 indicate that there are two clusters: one around "zero degeneracy, all cloud intervals singular" and one at "degeneracy larger than 90%, all variables have non-trivial cloud intervals." In between, they appear to be all over the place at a first glance. The moving average reveals that there is a clear correlation between the degeneracy rate and the percentage of instances with non-trivial cloud interval: the larger the degeneracy, the higher the chance that a variable can take multiple values in an optimal LP solution.

Note that in Fig. 8, the majority of instances have a variable–constraint ratio between 1 and 2, but it can grow up to 133 for individual instances, hence we decided to truncate it at 10. The relation between the variable–constraint ratio and the percentage of variables with non-trivial cloud intervals can be separated into two parts: Until a ratio of about 1.5, there is a clear correlation: the more the variable–constraint ratio deviates from one, the higher the chance that a variable can take multiple values in an optimal LP solution. When the variable–constraint ratio is large enough (greater than 1.5) the average percentage of variables with a non-trivial cloud interval stabilizes at around 90%. It is noticeable that there are no crosses in the lower right half of Fig. 8. Once the variable–constraint ratio grows large enough, it is almost guaranteed that a certain amount of cloud intervals are non-trivial. There is only one instance with a ratio larger than 1.5 and less than 10% variables with non-trivial cloud intervals and there are only two instances with a variable–constraint ratio larger than 2 and less than 50% variables with non-trivial cloud intervals.

Figure 9 illustrates the share of integer variables with non-trivial cloud intervals per depth level of the branch-and-bound tree for all instances reaching depth 20. The color scale is similar to the one in Fig. 6 but split into 10% buckets with extra buckets for candidate reductions of 0% and 100%.

As for the degeneracy rate, the extreme cases are the common ones at the root node. At the root node, almost 25% of the instances have non-trivial cloud intervals for all unfixed integer variables. For half of the instances, at least 80% of the unfixed integer variables have non-trivial cloud intervals. The other extreme—no integer variable can

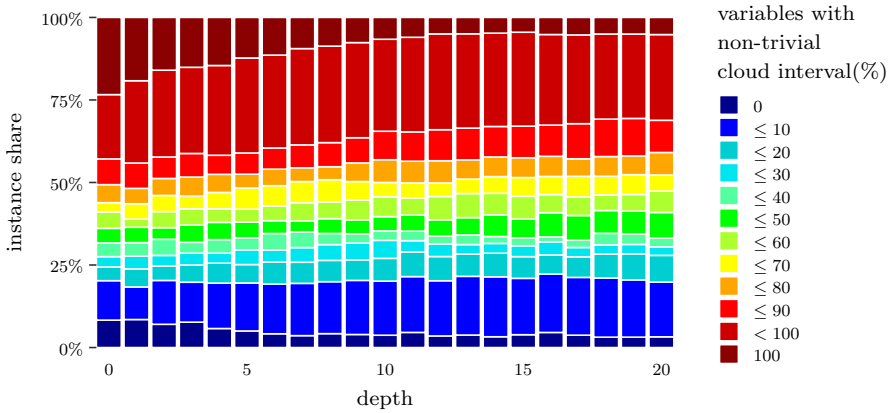


Fig. 9 Share of integer variables with non-trivial cloud interval by depth level. Each bar is split into multiple pieces according to the relative number of instances that have non-trivial cloud intervals for a share of variables in a particular range (color figure online)

be moved on the optimal face—can be observed for 8.3% of the instances. About 25% of the instances have non-trivial cloud intervals for less than 20% of the variables. The remaining instances are split among the other buckets.

For the deeper levels of the tree, we again averaged over all nodes in the respective level for each individual instance before averaging over the instances. We see that the two extreme cases happen less frequently in deeper levels of the tree. At depth 20, only 3.2% of the instances cannot move any unfixed integer variable on the optimal face for all nodes at this depth. This decrease, however, comes with an increase in the number of instances where more than 0% and less than 10% of the unfixed integer variable have a non-trivial cloud interval. The sum of these two cases is almost constant over the different depth levels in the tree and stays at about 20% of the instances.

For instances with a larger number of non-trivial cloud intervals, the picture looks similar. At depth level 20, only 5% of the instances have non-trivial cloud intervals for all unfixed integer variables. In return, the number of instances where not all, but at least 80% of the variables have non-trivial cloud intervals increases. Still, their share together with the extreme case of all variables having non-trivial cloud intervals reduces to 40.9%. As a result, there are more instances where some, but not (close to) all or none, of the unfixed integer variables have a non-trivial cloud interval. These observations are in line with the slightly decreased variable–constraint ratio in deeper levels we observed in Sect. 3.

We have seen that for many instances, most of the variables that remained unfixed when fixing to the optimal LP face have a non-trivial cloud interval. On the one hand, those are non-basic dual degenerate variables that can be pivoted into the basis and can then move away from their bound. On the other hand, also the basic variables can often be moved to different values. That case is of particular interest since the fractional basic variables play an important role in many algorithms within MIP solvers, in particular, for branching and cutting.

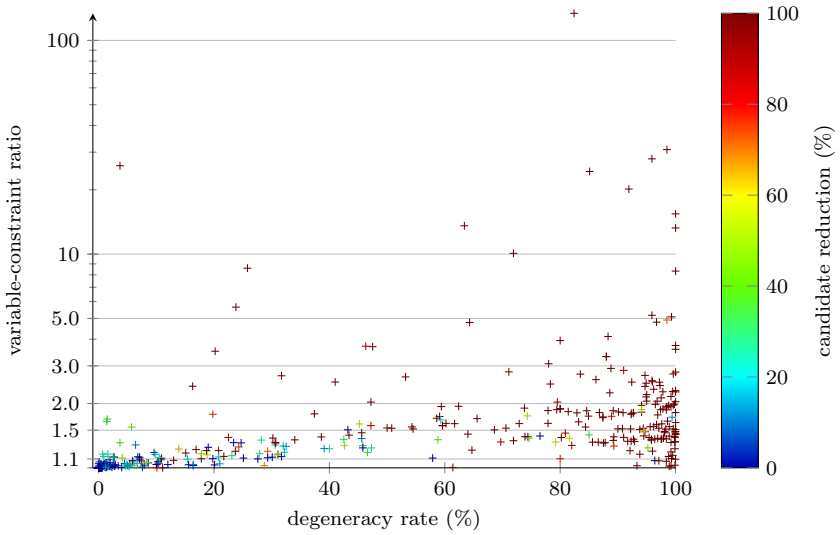


Fig. 10 Share of branching candidates with integer value in their cloud interval, plotted by dual degeneracy share (x-axis) and variable–constraint ratio of the optimal face (y-axis) (color figure online)

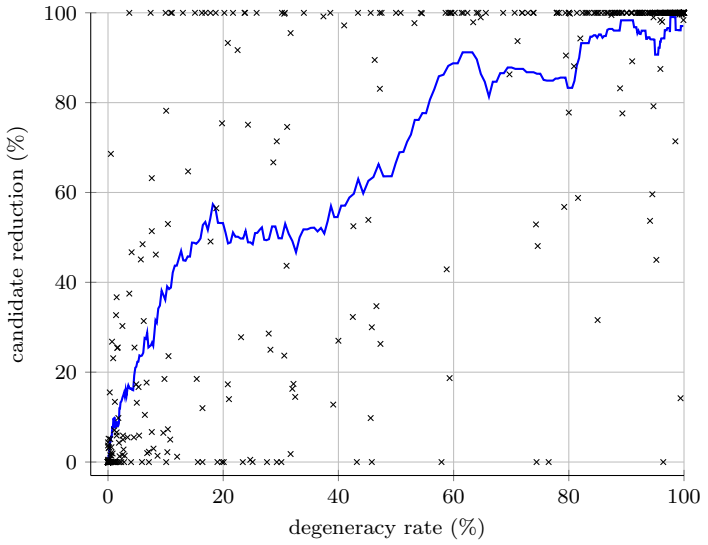


Fig. 11 Relation between dual degeneracy share (x-axis) and share of branching candidates with integer value in their cloud interval (y-axis). The line shows the moving average over 30 closest instances by degeneracy rate

For branching, it is useful to know whether a fractional variable can be moved to an integer value, i.e., whether it has an integer value in its cloud interval. If this is the case, the branching on this variable does not not improve the dual bound for at least one child node and the variable would thus not be considered for branching by

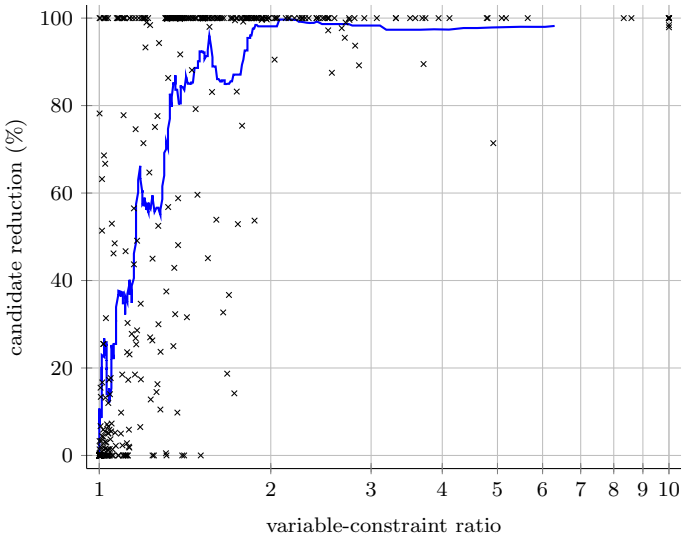


Fig. 12 Relation between variable–constraint ratio (x-axis) and share of branching candidates with integer value in their cloud interval (y-axis). Ten instances with a variable–constraint ratio larger than 10 are printed with a ratio of 10 to improve readability. The line shows the moving average over 30 closest instances by degeneracy rate

many branching rules. Therefore, we call the share of fractional branching candidates with integer value in their cloud intervals the *branching candidate reduction*, see the following formal definition.

Definition 3 (*Branching candidate reduction*) Let a MIP $\min\{c^T x : Ax = b, x \geq 0, x_j \in \mathbb{Z}\ \forall j \in J\}$ be given, together with an optimal LP solution $x^\star$ and cloud intervals $\mathcal{I}_j$, $j \in J$. Let $F(x^\star) := \{j \in J \mid x_j^\star \notin \mathbb{Z}\}$ be the set of branching candidates, i.e., the set of integer variables with fractional value in $x^\star$. Then, the *branching candidate reduction* ρ is defined by

$$\rho = \frac{|\{j \in F(x^\star) \mid \mathcal{I}_j \cap \mathbb{Z} \neq \emptyset\}|}{|F(x^\star)|}.$$

Intuitively, a large branching candidate reduction is favorable as it allows to restrict the set of promising branching candidates. A special case in this context is a candidate reduction of 100%. This proves that no branching candidate will improve the dual bound of both child nodes.

Figure 10 shows the branching candidate reduction at the root node for the instances of the MMMC test set. As in Fig. 6, each instance is represented by one cross positioned according to degeneracy rate and variable–constraint ratio with the color encoding the relative reduction in the number of branching candidates. Note that the variable base set is a different one than in Fig. 6, as we only consider branching candidates and disregard non-basic variables.

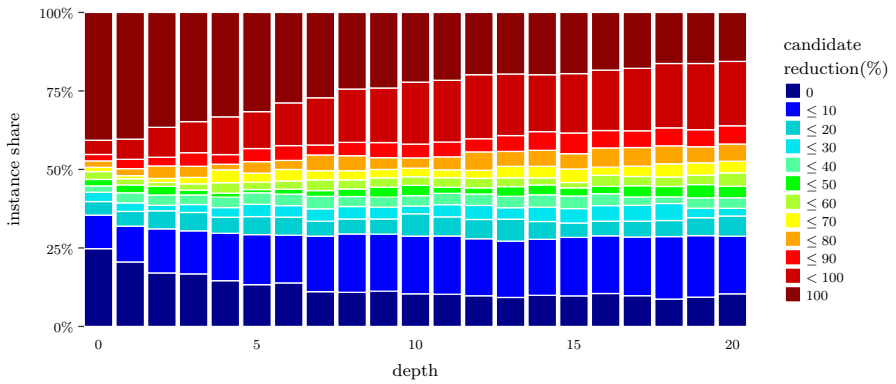


Fig. 13 Branching candidate reduction by depth level. Each bar is split into multiple pieces according to the share of instances with branching candidate reduction in a particular range (colour figure online)

As for the non-trivial cloud intervals, a variable–constraint ratio of 2.0 or larger leads to very high candidate reductions. Out of the 63 instances that fall into this category, 79.4% can move all branching candidates of the final root LP to an integer value while retaining LP optimality.[2] On one instance, the candidate reduction amounts to 71.4%, while the remaining ones show reductions of around 90% and more. For large degeneracy rates we observe high branching candidate reductions as well. Out of the 152 instances with a degeneracy rate of 80% or higher, 127 have an optimal face in which every initial branching candidate can take an integer value.

Figures 11 and 12 again relate the candidate reduction to the degeneracy rate and the variable–constraint ratio, respectively, by showing crosses for individual instances and a moving average (blue line). They confirm observations that we made before: We see a strong correlation between the degeneracy rate and the candidate reduction, but also between the variable–constraint ratio and the candidate reduction, for a variable–constraint ratio less than two. For instances with a variable–constraint ratio larger than two, the moving average is almost constant at about 98%, and except for one single instance, we can always achieve a candidate reduction of more than 85%.

As before, we consider the branching candidate reduction for different depth levels in the tree up to depth 20, see Fig. 13. At the root node, almost 25% of the instances do not have a single fractional variable that ca be moved to an integer value on the optimal face. That number is considerably larger than the number of instances for which all unfixed integer variables have trivial cloud intervals, see Fig. 9. Also the other extreme, a branching candidate reduction of 100%, is more often observed than that all unfixed integer variables have non-trivial cloud intervals. It is the case for almost 40% of the instances. The remaining instances are split among the other buckets with more instances going into the more extreme buckets.

As for non-trivial cloud intervals, the two extreme cases happen less frequently in deeper levels of the tree. Only 11.3% of the instances have a branching candidate reduction of 0% at all nodes in depth 20. At the same time, however, the number of

[2] Note, however, that this does not mean that all variables can be moved to an integral value at the same time.

instances with more than 0% and less than 10% candidate reduction increases. The sum of these two cases is almost constant over the different depth levels in the tree. For large candidate reductions, the picture looks similar. At depth level 20, only 16.5% of the instances have a candidate reduction of 100%, while the share of instances with a candidate reduction between 90 and 100% increases from 6 to 18.7%. Nevertheless, their sum decreases by more than 10%. Consequently, there are more instances for which some, but not (close to) all or none of the candidates can be moved to an integer point. When it comes to exploiting degeneracy within MIP solvers, this may actually be beneficial, see our discussion of reducing the set of branching candidates. Thus, taking degeneracy into account may be even more important and promising in deeper levels of the tree. Here, degeneracy is often still present to almost the same extent as at the root node but the candidate reduction is less extreme.

5 Conclusions

We performed a computational analysis of dual degeneracy in MIP instances and demonstrated that it is very common in practical instances from standard MIP problem collections. To this end, we introduced a new metric, the variable–constraint ratio, and combined it with the share of degenerate non-basic variables to obtain an improved measure of dual degeneracy. The majority of the analyzed instances tend to have an extreme behavior with respect to dual degeneracy, i.e., either almost all or only very few variables are affected.

During the branch-and-bound process, the average degree of dual degeneracy is almost constant. However, we regularly observed large changes in the grade of dual degeneracy for individual instances in deeper levels of the tree. For some instances, branching increases dual degeneracy, while for others, it is decreased. Additionally, we analyzed the effect of dual degeneracy on the uncertainty of the LP solution. Instances with high dual degeneracy measures often allow many variables to vary their value across different optimal LP solutions. Perhaps most importantly, a high dual degeneracy often leads to a large number of branching candidates obtaining integer values in alternative LP optima, which indicates that they are poor candidates for branching. Again, both effects tend to occur in extreme manifestations, but get less extreme at deeper levels of the tree.

The omnipresence of dual degeneracy implies that MIP solvers should take degeneracy into account; the changes in the grade of degeneracy advocate that the dual degeneracy of instances should be measured continuously and not only for the original formulation. A prerequisite for that is an effective measure of dual degeneracy. The combination of two measures that we introduced in this paper fulfills that role and captures the impact of dual degeneracy well, as we demonstrated in our analysis. It has already been successfully applied to improve the hybrid branching rule (Achterberg and Berthold 2009) of SCIP, see Berthold et al. (2019a), and in recent work on local rapid learning (Berthold et al. 2019b).

Acknowledgements Open Access funding provided by Projekt DEAL. The work for this article has been conducted within the *Research Campus Modal* funded by the German Federal Ministry of Education and

Research (Fund Numbers 05M14ZAM, 05M20ZBM). We would like to thank the two anonymous reviewers for constructive feedback that improved the quality of the presentation.

References

Achterberg T (2007) Constraint integer programming. Ph.D. thesis, Technische Universität Berlin. http://nbnresolving.de/urn:nbn:de:0297-zib-11129

Achterberg T (2013) LP relaxation modification and cut selection in a MIP solver. US Patent 08463729

Achterberg T (2018) Exploiting Degeneracy in MIP. In: Presentation slides from Aussois workshop 2018. www.iasi.cnr.it/aussois/web/uploads/2018/slides/achterbergt.pdf

Achterberg T, Berthold T (2009) Hybrid branching. In: van Hoeve WJ, Hooker JN (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, 6th international conference, CPAIOR 2009, lecture notes in computer science, vol 5547. Springer, pp 309–311

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. Oper Res Lett 33:42–54

Achterberg T, Koch T, Martin A (2006) MIPLIB 2003. Oper Res Lett 34(4):1–12. https://doi.org/10.1016/j.orl.2005.07.009

Bajgiran OS, Cire AA, Rousseau LM (2017) A first look at picking dual variables for maximizing reduced cost fixing. In: Salvagnin D, Lombardi M (eds) Integration of AI and OR techniques in constraint programming. Springer, pp 221–228

Beale EML (1954) An alternative method for linear programming. Math Proc Cambr Philos Soc 50(4):513–523. https://doi.org/10.1017/S0305004100029650

Berthold T, Salvagnin D (2013) Cloud branching. In: Gomes C, Sellmann M (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems. Springer, Berlin, pp 28–43

Berthold T, Gamrath G, Salvagnin D (2019a) Exploiting dual degeneracy in branching. Technical report 19-17, ZIB, Takustr. 7, 14195 Berlin

Berthold T, Stuckey PJ, Witzig J (2019b) Local rapid learning for integer programs. In: Rousseau LM, Stergiou K (eds) Integration of AI and OR techniques in constraint programming. Springer, pp 67–83

Bixby RE, Ceria S, McZeal CM, Savelsbergh MW (1998) An updated mixed integer programming library: MIPLIB 3.0. Optima 58:12–15

Caprara A, Locatelli M (2010) Global optimization problems and domain reduction strategies. Math Progr 125:123–137. https://doi.org/10.1007/s10107-008-0263-4

Chvatal V (1983) Linear programming. Macmillan, New York City

COR@L: MIP instances. http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/

Czyzyk J, Mesnier M, Moré J (1998) The NEOS server. IEEE Comput Sci Eng 5(3):68–75

Dakin RJ (1965) A tree-search algorithm for mixed integer programming problems. Comput J 8(3):250–255

Dantzig GB (1951) Maximization of a linear function of variables subject to linear inequalities. In: Koopmans T (ed) Activity analysis of production and allocation. Wiley, pp 339–347

Fischetti M, Lodi A, Monaci M, Salvagnin D, Tramontani A (2016) Improving branch-and-cut performance by random sampling. Math Program Comput 8(1):113–132

Gleixner A, Eifler L, Gally T, Gamrath G, Gemander P, Gottwald RL, Hendel G, Hojny C, Koch T, Miltenberger M, Müller B, Pfetsch ME, Puchert C, Rehfeldt D, Schlösser F, Serrano F, Shinano Y, Viernickel JM, Vigerske S, Weninger D, Witt, JT, Witzig J (2017a) The SCIP optimization suite 5.0. Technical report 17-61, ZIB, Takustr. 7, 14195 Berlin

Gleixner AM, Berthold T, Müller B, Weltge S (2017b) Three enhancements for optimization-based bound tightening. J Glob Optim 67(4):731–757. https://doi.org/10.1007/s10898-016-0450-4

Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2011) MIPLIB 2010: mixed integer programming library version 5. Math Program Comput 3(2):103–163

Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. Econometrica 28(3):497–520

Lemke CE (1954) The dual method of solving the linear programming problem. Naval Res Logist Q 1(1):36–47. https://doi.org/10.1002/nav.3800010107

Lodi A, Tramontani A (2013) Performance variability in mixed-integer programming. In: Theory driven by influential applications, chap. 1. INFORMS, pp 1–12. https://doi.org/10.1287/educ.2013.0112

Orchard-Hays W (1958) Evolution of linear programming computing techniques. Manag Sci 4(2):183–190

Wunderling R (1996) Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin

Zamora JM, Grossmann IE (1999) A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. J Glob Optim 14:217–249. https://doi.org/10.1023/A:1008312714792