Noname manuscript No.
(will be inserted by the editor)

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

Gerald Gamrath · Timo Berthold · Domenico Salvagnin

Received: date / Accepted: date

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? In this course, 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

Mathematics Subject Classification (2010) 90C10 · 90C11 · 90C57

Gerald Gamrath
Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
E-mail: gamrath@zib.de

Timo Berthold
Fair Isaac Germany GmbH, Stubenwald-Allee 19, 64625 Bensheim, Germany
E-mail: timoberthold@fico.com

Domenico Salvagnin
DEI, Via Gradenigo, 6/B, 35131 Padova, Italy
E-mail: salvagni@dei.unipd.it

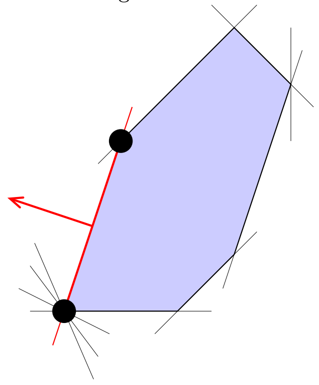
1 Introduction

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., [21,13]. A solution to an LP is called *primal degenerate* if there are basic variables set to one of their bounds 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.

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 reduced costs 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 Figure 1. The upper black point lies at the intersection of exactly two hyperplanes defined by constraints. With only two structural variables (plus slack variables), this solution is primal non-degenerate. The lower black point lies at the intersection of five hyperplanes. Even if both structural variables are basic, three of the slack variables of those inequalities have to be basic as well although the inequalities are fulfilled with equality, meaning that the slack variables have value zero and are degenerate. Therefore, this solution is primal degenerate. Both black points and all points on the line in between are optimal solutions; thus, the two black points are both dual degenerate basic solutions. For the

Fig. 1 Illustration of primal and dual degenerate solutions.



upper point, for example, the slack variable of the constraint drawn as a thin black line is non-basic with reduced cost zero, since this constraint has non-zero slack in the lower optimal solution.

In case of dual degeneracy, which of the alternative optimal solutions is returned by the LP solver is arbitrary and cannot be predicted in most cases. An exception is the use of the lexicographic pivoting rule in the simplex algorithm, which is, however, not 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 mixed integer programming (MIP) solver. A MIP 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 algorithm can lead to a very different LP solution being returned. As a conseuqence, 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* [19,20] and should typically be avoided. Taking into account dual degeneracy and potentially multiple optimal LP solutions may be a mean to reduce performance variability and make the solver more stable.

Besides the reduction of performance variability, exploiting dual degeneracy and the knowledge of multiple LP optima can be used within a MIP solver to take better decisions. Rather than relying on 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 [17], is an example for the latter case which mainly aims at increasing integrality of the LP solution to aid primal heuristics and branching rules. On the other hand, concurrent root cut loops [16] are an example for the former case that exploits a small set of optimal solutions for the separation process. The pump-reduce procedure [2] 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 [7] 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 in [3]. Finally, the authors of this paper investigated branching improvements obtained by exploiting dual degeneracy and the knowledge of multiple optimal LP solutions [9,8].

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.

2 Dual degeneracy measures

Let us investigate how common dual degeneracy is in real-world MIPs. For that, we performed computational experiments using the academic MIP solver SCIP 5.0.1 [1,17] with SoPlex 3.1.1 [22,17] as underlying LP solver. As test set, we use the MMMC test set which contains all instances from MIPLIB 3 [11], MIPLIB 2003 [6], and the MIPLIB 2010 benchmark set [19] as well as the Cor@l test set [14], which mainly contains instances that users worldwide submitted to the NEOS server [15]. Duplicate instances were removed, leaving us with a total of 496 instances. The observations we describe in the following naturally depend on the branching rule used for the experiments. We used reliability branching [5] and provided the optimum as a cutoff bound to focus on the branching performance. We chose that branching rule because it is a state-of-the-art rule used (in slight variations) by many solvers. Still, we are optimistic that the results should transfer to other common branching rules.

To get a first impression, we focused on the degeneracy in the final root LP solution of SCIP, i.e., 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.

We will use two measures of dual degeneracy in the following. The first one is a classical measure in that context and follows directly from the definition of dual degeneracy. The *degeneracy rate* counts the relative number of dual degenerate non-basic variables. Figure 2 illustrates this number for the final root LP solutions. Thereby, 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.

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). Overall, it looks like 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.

In order to gain a deeper understanding of dual degeneracy, we introduce a second dual degeneracy measure. It is motivated by the observation that a typical way to exploit dual degeneracy is to investigate the optimal face of the

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 %.

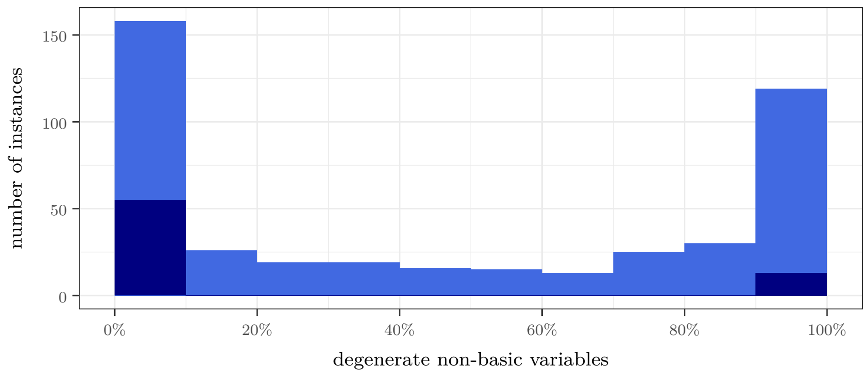
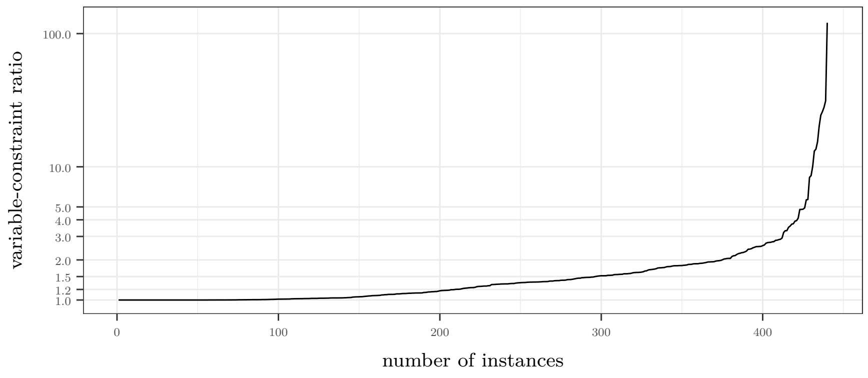


Fig. 3 Variable-constraint ratios of the optimal face for instances in the MMMC test set (final root LP).



LP relaxation in order to generate additional points lying on this face. The LP is reduced to the optimal face by fixing to their current value all variables—including slacks–whose reduced costs are non-zero, using the reduced costs associated with the starting optimal basis. Then, optimizing over the restricted LP with changed objective functions allows to move to different optimal bases that potentially represent alternative LP optima. Therefore, the size of this restricted problem is of interest when investigating dual degeneracy. We denote the number of unfixed variables in the restricted problem by $\bar{n}$ and the number of constraints by m and call $\beta = \frac{\bar{n}}{m}$ the *variable-constraint ratio* of the optimal face. This ratio is 1 if no non-basic variable is dual degenerate, as only the basic variables remain unfixed. It increases as the number of dual degenerate non-basic variables increases.

The share of non-basic dual degenerate variables as illustrated in Figure 2 expresses how many 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. The larger the variable-constraint ratio is, the higher we expect the probability to be that a basic variable can be pivoted out of the basis. While this is also true for the share of non-basic dual degenerate variables when looking at a particular instance, it does not allow to easily compare instances with different problem sizes. As an example, 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 a fixed basis size, however, the latter will probably allow pivoting more of the current basic variables out of the basis.

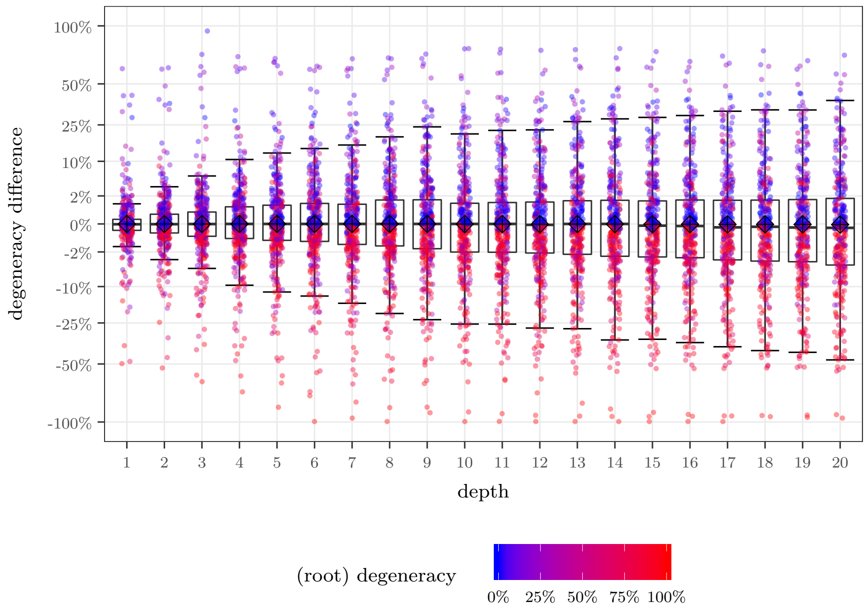
The variable-constraint ratios for the final root LP of the 440 instances from the MMMC test set not solved before degeneracy information was computed are presented in Figure 3. There are 55 instances with a ratio of 1.0, which are the instances showing no degeneracy at all. 111 and 129 instances have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

3 Evolution of dual degeneracy throughout the tree search

We observed degeneracy in the final root LPs of most of the regarded instances, but how does the degeneracy change during the tree search? In order to investigate this, we ran each instance with a node limit of 1 million and a time limit of 2 days. We computed the average degeneracy per depth level in the branch-and-bound tree by first averaging over all nodes of one depth for each instance and then over the instances 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 %.

However, is the average degeneracy rate also constant for individual instances? This question is answered by Figure 4, which focuses on the change in the average degeneracy rate per depth level, compared to the degeneracy of the final LP of the root node (after potential restarts). It shows the first 20 levels of the tree. 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 ◈.

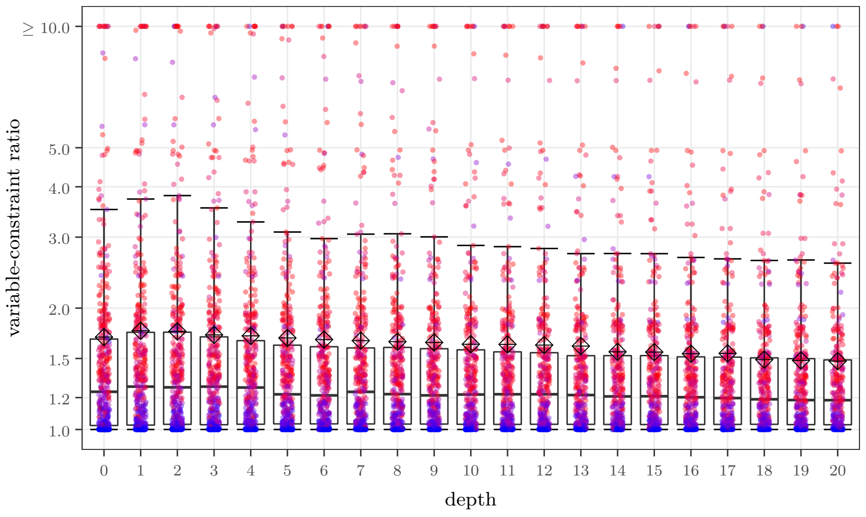


Each instance is represented by one point for each depth level; the color of the point represents the root degeneracy of the instance. The x-axis represents the depth level; coordinates are jittered, and points are drawn translucent to better display the density of instances at common degeneracy differences. 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 1, in particular, zero values, as opposed to a log scaling. We observe that many instances change their degeneracy rate during the tree search and that there are roughly the same number of instances with increasing degeneracy rate at deeper levels as with decreasing degeneracy rate. The variance increases with a higher depth in the tree, which is reasonable as the deeper a subproblem is in the tree, the more different it is to the global problem. In order to better evaluate the distribution of the points, Figure 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; at level 10, by no more than 2 %. Even at level 20, the average change in degeneracy rate of 50 % of the instances is between −4.3 % and 1.7 %. On the other hand, there are also 25.7 % of the instances that change their average degeneracy rate by more than 10 % in either direction 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 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 as all values are no smaller than 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 although we had 134 instance/depth combinations (of 27 different instances) with higher ratios.

We observe a tendency for slightly decreasing variable-constraint ratios with higher depths. An exception is the higher 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 so that the ratio increases. The box plot

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 $\diamond$.



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 below 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 also observe that instances with a high 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 high and 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 Section 2 concerning the frequentness 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. On the other hand, 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 both many non-basic variables can become basic in alternative optimal LP solutions as well as many of the basic variables are non-basic in alternative optima.

We conclude that dual degeneracy is prevalent in standard MIP models, and this should provide a strong motivation to consider it within MIP solvers. In addition, the analysis above also shows that different models exhibit different kinds of degeneracy, ranging from constant degeneracy to degeneracy affected (in either direction) by branching. As such, MIP solving 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 our degeneracy measures. In particular, we are trying to answer the two following questions. How many of the integer variables can change their value while staying on the optimal face? And how often can they even achieve an integer value on the optimal face? We introduced the variable-constraint ratio as a 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 in [9]. For a set $\mathcal{S}$

of alternative LP optima, the cloud interval $\mathcal{I}_j$ for an integer variable x_j is defined as

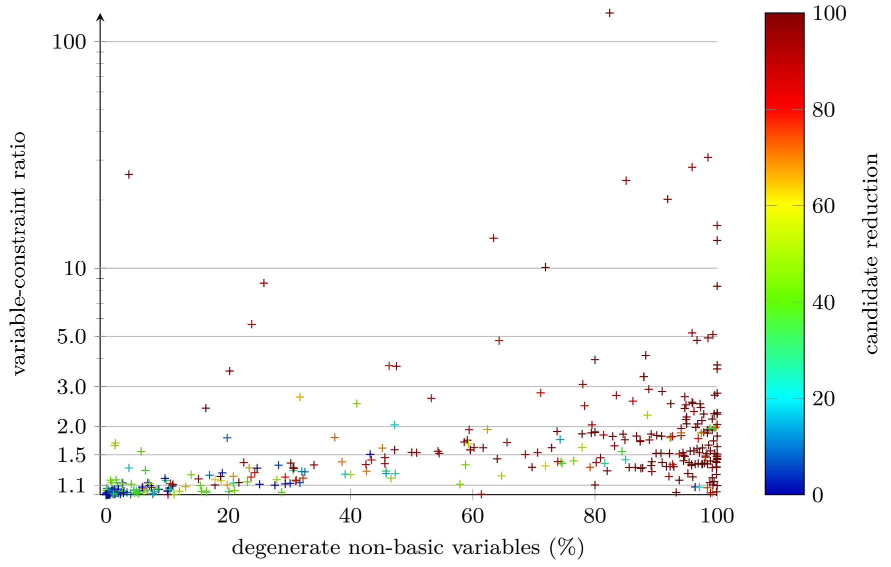
$$\mathcal{I}_j = \left[\min\{x'_j \mid x' \in \mathcal{S}\}, \max\{x'_j \mid x' \in \mathcal{S}\}\right].$$

We are particularly interested in the cloud intervals of integer variables. In particular, we want to identify variables that can be moved while staying on the optimal face, which corresponds to a non-trivial cloud interval with $|\mathcal{I}_j| > 0$. 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 as they are guaranteed to not improve the dual bound for at least one child node.

In order to compute alternative LP optima, the LP can be restricted to the optimal face by fixing all variables (including slacks) whose reduced costs are non-zero in the starting optimal basis. The validity of this approach follows directly from the fact that the objective value of any LP solution can be expressed as the sum of the current LP objective and the product of the current reduced cost with the change in the value of every non-basic variable, see [13]. Then, auxiliary objective functions can be used to move to different bases and generate alternative LP optima [2]. In this paper, we minimize and maximize each variable which is not yet fixed: this is what optimization-based bound tightening techniques do (see, e.g., [23,12]), but applied to the optimal face. In the worst case, 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 as they are initially set to one of their bounds and can only move in one direction. Using filtering techniques, see [18], the number of LPs to be solved can often be reduced, but typically stays 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 in [8].

Figure 6 shows how many of the integer variables that remained unfixed after fixing 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. First, we can 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

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

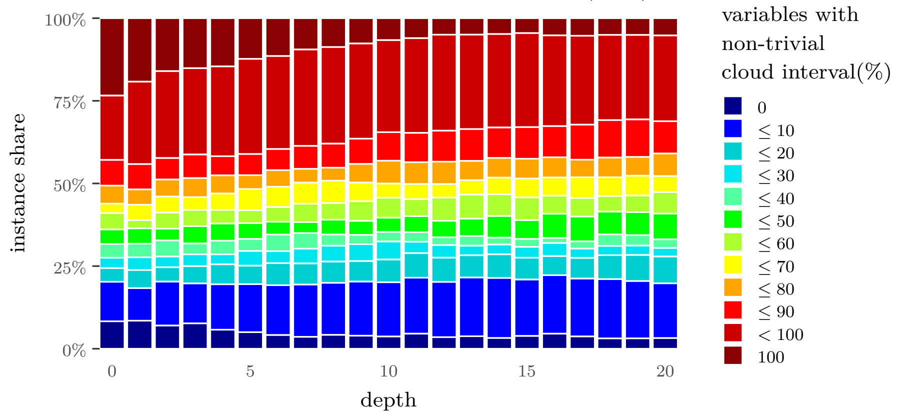


a non-trivial cloud interval for all unfixed integer variables, 48 for at least 90 % of the variables. For high degeneracy rates, the variable-constraint ratios increase as does the relative number of non-trivial cloud intervals. Out of the 152 instances with a degeneracy rate of 80 % or higher, 81 can move all unfixed integer variable on the optimal face, while this holds for 90 % of the variables for 124 of the instances. As a comparison: out of the 256 instances that have a variable-constraint ratio smaller 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.

Figure 7 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 Figure 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—

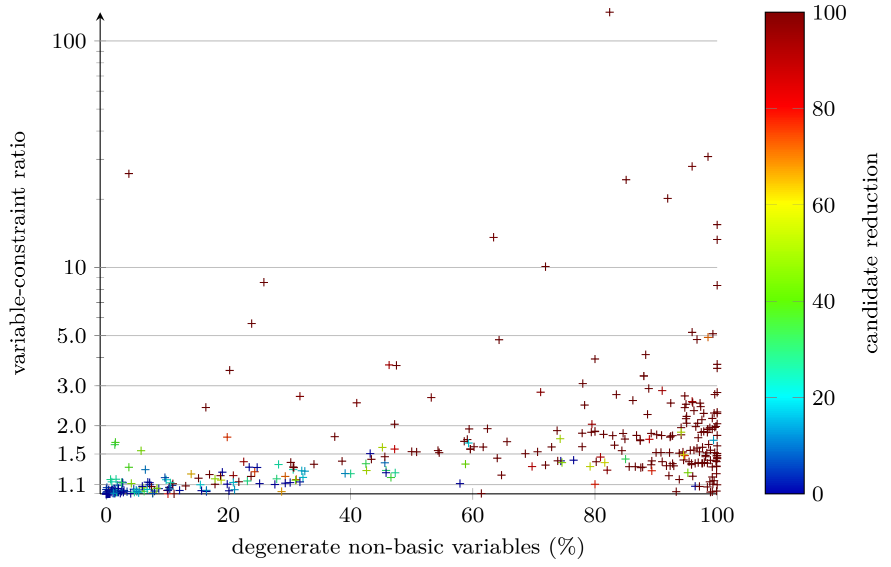
Fig. 7 Share of integer variables with non-trivial cloud interval by depth level. Each bar is split into multiple pieces according to the relative number of instances that have non-trivial cloud intervals for a share of variables in a particular range (color).



no integer variable can 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 few (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 that level for each instance before averaging over the instances. We can see that the two extreme cases happen less frequently in deeper levels of the tree. At depth 20, only 3.2 % of the instances can move no unfixed integer variable on the optimal face at all nodes in 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 stays almost the same over the different depth levels in the tree and stays at about 20 % of the instances. For instances with high number of non-trivial cloud intervals, the picture looks similar. At depth level 20, only 5 % of the instances have a 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, but still, their share together with the extreme case of 100 % 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 Section 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

Fig. 8 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).

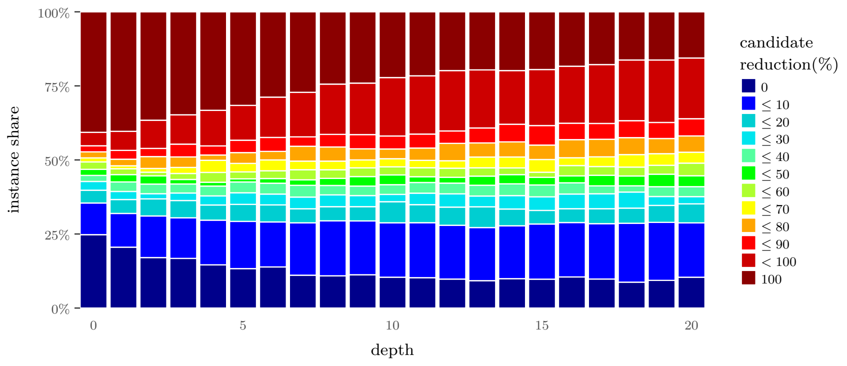


branching. In case of branching, it is particularly interesting to know whether a fractional variable can be moved to an integer value, i.e., whether it has an integer value in its cloud interval. If this is the case, the variable would typically not be considered for branching by many branching rules since it is already guaranteed to not improve the dual bound for at least one child node. Therefore, we call the share of fractional branching candidates with integer value in their cloud intervals the *branching candidate reduction*. 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. Nevertheless, branching needs to be performed. Therefore, knowledge about an integer value in the cloud intervals cannot be used to restrict the branching candidate set anymore, but branching rules may profit from that knowledge in different ways, see [8].

Figure 8 shows the branching candidate reduction at the root node for the instances of the MMMC test set. As in Figure 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 Figure 6, as we only consider branching candidates and disregard non-basic variables.

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

Fig. 9 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 (color).



root LP to an integer value while retaining LP optimality.[1] On one instance, the candidate reduction amounts to 71.4 %, while the remaining ones show reductions of around 90 % and more. For high 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.

Again, we regard the branching candidate reduction for different depth levels in the tree up to depth 20, see Figure 9. 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 where all unfixed integer variables have trivial cloud intervals, see Figure 7. 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 instances with more than 0 % and less than 10 % candidate reduction increases. The sum of these two cases stays almost the same over the different depth levels in the tree. For high 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 % so that there are more instances where some, but not (close to) all or none of the candidates can be moved to an integer point. When it comes

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

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, where degeneracy is often still present to almost the same extend 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 nonbasic variables to obtain an improved measure of dual degeneracy. The majority of the analyzed instances tend to behave extreme 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 stays 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 [4] of SCIP, see [8], and in recent work on local rapid learning [10].

Acknowledgments

The work for this article has been conducted within the *Research Campus Modal* funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM).

References

1. Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007). http://vs24.kobv.de/opus4-zib/frontdoor/index/index/docId/1018
2. Achterberg, T.: LP relaxation modification and cut selection in a MIP solver (2013). US Patent 08463729
3. Achterberg, T.: Exploiting Degeneracy in MIP. Presentation slides from Aussois Workshop 2018. www.iasi.cnr.it/aussois/web/uploads/2018/slides/achterbergt.pdf (2018)
4. Achterberg, T., Berthold, T.: Hybrid branching. In: W.J. van Hoeve, J.N. Hooker (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, pp. 309–311. Springer (2009)
5. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33**, 42–54 (2005)
6. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters **34**(4), 1–12 (2006). DOI 10.1016/j.orl.2005.07.009
7. Bajgiran, O.S., Cire, A.A., Rousseau, L.M.: A first look at picking dual variables for maximizing reduced cost fixing. In: D. Salvagnin, M. Lombardi (eds.) Integration of AI and OR Techniques in Constraint Programming, pp. 221–228. Springer International Publishing (2017)
8. Berthold, T., Gamrath, G., Salvagnin, D.: Exploiting dual degeneracy in branching. Tech. Rep. 19-17, ZIB, Takustr. 7, 14195 Berlin (2019)
9. Berthold, T., Salvagnin, D.: Cloud branching. In: C. Gomes, M. Sellmann (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 28–43. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
10. Berthold, T., Stuckey, P.J., Witzig, J.: Local Rapid Learning for Integer Programs. In: L.M. Rousseau, K. Stergiou (eds.) Integration of AI and OR Techniques in Constraint Programming, pp. 67–83. Springer International Publishing (2019)
11. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.: An updated mixed integer programming library: MIPLIB 3.0. Optima (58), 12–15 (1998)
12. Caprara, A., Locatelli, M.: Global optimization problems and domain reduction strategies. Mathematical Programming **125**, 123–137 (2010). doi:10.1007/s10107-008-0263-4
13. Chvatal, V.: Linear programming. Macmillan (1983)
14. COR@L: MIP Instances. http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/
15. Czyzyk, J., Mesnier, M., Moré, J.: The NEOS server. Computational Science & Engineering, IEEE **5**(3), 68–75 (1998). http://www.neos-server.org/neos/
16. Fischetti, M., Lodi, A., Monaci, M., Salvagnin, D., Tramontani, A.: Improving branch-and-cut performance by random sampling. Mathematical Programming Computation **8**(1), 113–132 (2016)
17. Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Serrano, F., Shinano, Y., Viernickel, J.M., Vigerske, S., Weninger, D., Witt, J.T., Witzig, J.: The SCIP Optimization Suite 5.0. Tech. Rep. 17-61, ZIB, Takustr. 7, 14195 Berlin (2017)
18. Gleixner, A.M., Berthold, T., Müller, B., Weltge, S.: Three enhancements for optimization-based bound tightening. Journal of Global Optimization **67**(4), 731–757 (2017). DOI 10.1007/s10898-016-0450-4
19. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010 – Mixed Integer Programming Library version 5. Mathematical Programming Computation **3**(2), 103–163 (2011). http://miplib.zib.de
20. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. In: Theory Driven by Influential Applications, chap. 1, pp. 1–12. INFORMS (2013). DOI 10.1287/educ.2013.0112

21. Orchard-Hays, W.: Evolution of linear programming computing techniques. Management Science **4**(2), 183–190 (1958)
22. Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin (1996)
23. Zamora, J.M., Grossmann, I.E.: A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. Journal of Global Optimization **14**, 217–249 (1999). doi:10.1023/A:1008312714792