# How important are branching decisions: fooling MIP solvers

Pierre Le Bodic* and George L. Nemhauser

April 21, 2014

**Abstract**

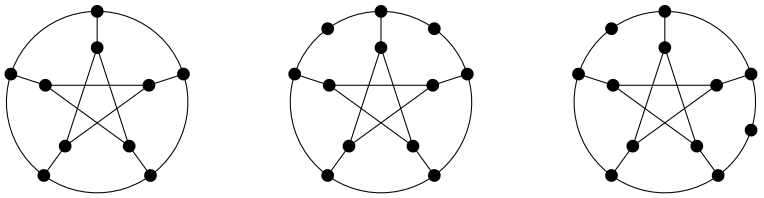
We show the importance of selecting good branching variables by exhibiting a family of instances for which an optimal solution is both trivial to find and provably optimal by a fixed-size branch-and-bound tree, but for which state-of-the-art Mixed Integer Programming solvers need an increasing amount of resources. The instances encode the edge-coloring problem on a family of graphs containing a small subgraph requiring four colors, while the rest of the graph requires only three.

## 1 Introduction

Mixed Integer Programming (MIP) solvers depend on branching rules to implicitly search the solution space. Numerous experimental results (see e.g. [2]) provide a good notion of their performances. However, little literature has been dedicated to theoretical results on MIP branching. By contrast, branching in satisfiability (SAT) solvers have been studied in a theoretical setting. Liberatore [10] has proven that choosing a branching candidate that minimizes the tree size is NP-hard. Ouyang [13] provides a family of instances of increasing sizes for which a fixed-size search tree exists. Unfortunately, the findings of Ouyang [13] do not translate in the context of MIP solving, since the given SAT instances – once written in an appropriate format – are trivially solved by current MIP solvers.

Our study is analogous to Ouyang's, but in a MIP setting. We design a family of increasingly large IP instances encoding the edge-coloring problem. These instances have trivial feasible solutions. Moreover, we prove that there exists for each instance (of arbitrary size) a fixed-size branch-and-bound (B&B) tree that proves optimality for these solutions. We then give experimental results showing that current MIP solvers need an increasing amount of resources to prove optimality. Finally, we explain this behavior for SCIP, a state-of-the-art open-source MIP solver.

*lebodic@gatech.edu



(a) Petersen graph (b) Graph $P1$ (c) Graph $P2$

Figure 1: The Petersen graph and two variations, $P1$ and $P2$. The first two graphs have $\chi' = 4$, while $P2$ has $\chi' = 3$.

# 2 Instances

We build IP instances encoding the *chromatic index problem* on specific *input graphs* using a simple *mathematical model*.

## 2.1 The chromatic index problem

Let $G$ be a simple graph. A *proper edge coloring* (we suppose all colorings are proper throughout the article) of $G$ is such that no two adjacent edges are assigned the same color. The *chromatic index* $\chi'$ is the minimum number of colors such that there exists an edge coloring of $G$. Vizing's theorem [18, 5] states that the chromatic index is either $\Delta$ or $\Delta + 1$, where $\Delta$ is the maximum degree of a vertex in $G$. The problem of determining whether $\chi'$ equals $\Delta$ or $\Delta + 1$ is NP-complete [7], even if $\Delta = 3$.

## 2.2 Input graphs

We consider only graphs with $\Delta = 3$. The Petersen graph [14, 6], shown in Figure 1a, is a graph that has $\Delta = 3$ and $\chi' = 4$.

We build two graphs similar to the Petersen graph, $P1$ and $P2$ (see Figures 1b and 1c), in such a way that only $P1$ retains the property that $\chi' = 4$.

**Lemma 1.** *The chromatic index of* $P1$ *is* 4*, and the chromatic index of* $P2$ *is* 3.

*Proof.* Consider the inner $C_5$ (cycle graph on 5 vertices) of the Petersen graph as shown in Figure 1a. Without loss of generality, color two adjacent edges of this $C_5$ with colors 1 and 2. Its only 3-edge-coloring is then $1, 2, 1, 2, 3$, and there is a unique 3-edge-coloring of the edges between the inner $C_5$ and the outer $C_5$. Consider now the outer $C_5$ and let $a, b, c, d, e$ be its edges. Two adjacent edges, say $a$ and $b$, must both have colors different from 2 and 3. The same is true for two other edges, say $c$ and $d$, with colors 1 and 3. The remaining edge $e$ and its two adjacent edges, $a$ and $d$, must all have a color different than 3. With these constraints, each set of edges $\{a, b\}, \{c, d\}, \{a, d, e\}$ thus locally prevents a 3-coloring from existing. Consider the operation that consists in *dividing* an

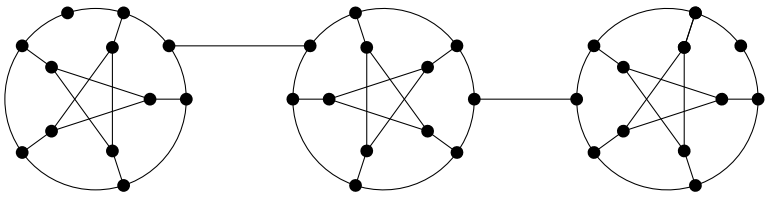


Figure 2: The graph $G_3$. From left to right, $G_3$ contains $P1$ and two $P2$'s as a subgraph. $P1$ being a subgraph and $\Delta = 3$ implies $\chi' = 4$.

edge $uv$ by adding a vertex $w$, thus creating edges $uw$ and $wv$. Consider the graphs in which exactly two edges among $a, b, c, d, e$ are divided once. Only three such choices make the graph 3-edge-colorable, namely those that divide $\{a, c\}$, $\{a, d\}$ or $\{b, d\}$. In each choice, both edges are non-adjacent: this graph is $P2$. The other graphs are $P1$. □

Let $k$ be a positive integer, and let $G_k$ be an input graph of size $k$. Start building $G_k$ by adding one $P1$ and $k-1$ $P2$'s in a disconnected fashion. As is, $G_k$ contains $2k$ vertices of degree 2. We add $k-1$ edges between the vertices of degree 2, so that $G_k$ is connected, and such that one vertex of $P1$ (and thus exactly one vertex among all of the $P2$'s) still has degree 2. The graph $G_k$ has $12k$ vertices and $18k-1$ edges. Since $G_k$ contains $P_1$ as a subgraph, and $\Delta = 3$, its chromatic index is 4. The graph $G_3$ is given in Figure 2.

## 2.3 Mathematical model

Given $G = (V, E)$, let $C = \{1, \ldots, \Delta + 1\}$ be the set of possible colors. We use a simple formulation for the chromatic index problem:

$$
\begin{array}{lllll}
\min_{x,c} & \sum_{i \in C} c^i & & & \\
& \sum_{i \in C} x_e^i = 1 & \forall e \in E & & \\
& \sum_{u \in V, e=uv} x_e^i \leq c^i & \forall v \in V, & \forall i \in C & \\
& c^i \in \{0, 1\} & & \forall i \in C & \\
& x_e^i \in \{0, 1\} & \forall e \in E, & \forall i \in C &
\end{array}
$$

Variables $c^i$ encode whether color $i$ is used or not. Variables $x_e^i$ encode whether color $i$ is assigned to edge $e$. The objective function minimizes the number of colors used. The second set of constraints ensures each edge is assigned a color. The third set of constraints enforces a proper coloring.

This formulation has $(\Delta+1)(|E|+1)$ variables, $|E|+(\Delta+1)|V|$ constraints and $(\Delta + 1)(3|E| + |V|)$ non-zero coefficients. Note that more elaborated MIP techniques have been designed to solve the edge-coloring problem (see e.g. [12, 8, 9]). However, this simple model is sufficient for our purposes.

# 3 Fooling MIP solvers

Let $I_k$ be the IP instances resulting from applying the model in Section 2.3 to the graphs $G_k$ given in Section 2.2. In this section we study the behavior of MIP solvers on the instances $I_k$ as $k$ grows. In these instances, finding the optimal solution is trivial. Indeed, every MIP solver we have tested finds the optimal solution at the root of the B&B tree. This is not surprising, as finding a 4-edge-coloring to the graphs $G_k$ can be done in linear time [17]. Thus we assume that the optimal solution is found or available at the root node. Furthermore, for any $k \geq 1$, the continuous relaxation of $I_k$ has value 3 [16, p. 63], which means the absolute gap is 1 at the root. The B&B task is thus to close that gap. Note that the objective function is integral, thus a node with an LP optimal value greater than 3 can be fathomed.

We will first prove that there exists for all $k \geq 1$ a fixed-size B&B tree that proves optimality for $I_k$. We will then produce experimental results using SCIP, CPLEX and GUROBI, showing that each of them needs an increasing amount of resources as $k$ increases. We will subsequently explain this behavior by looking at the branching rule of SCIP. Finally, we will verify that tuning solver parameters produces only limited performance improvements.

## 3.1 A fixed-size branch-and-bound tree

We prove that the B&B tree of $I_1$ also proves optimality for $I_k$, for $k \geq 2$:

**Theorem 1.** *Given an optimal solution, there is a fixed-size B&B tree for $I_k$, for all $k \geq 1$.*

*Proof.* Let $T$ be a complete branch-and-bound tree (i.e. one that proves optimality) for the instance $I_1$. Start solving $I_k$ by building the tree $T'$ following the branching decisions taken in $T$. Observe that the global dual bound of $T$ is 4, i.e. every leaf in $T$ has an LP optimal value of at least 4 (with $\infty$ meaning infeasibility by convention). Since all constraints of $I_1$ are contained in or implied by $I_k$, the LP optimum at each leaf of $T'$ is greater or equal than the LP optimum at its counterpart leaf in $T$.

□

## 3.2 Experimental results

We present numerical results for three solvers: CPLEX 12.6.0.0, GUROBI 5.6.0 and SCIP 3.1.0 (with the CPLEX LP solver). All experiments have been conducted on the same machine, with a single thread per solver. Instance $I_k$ has $72k$ variables, $64k$ constraints and $264k$ non-zero coefficients, up to small constants. For each instance $I_k$, ten equivalent programs have been generated by random row permutations (to reduce the impact of *performance variability* [11]). Table 1 reports the number of instances solved, together with the number of nodes and the time in seconds used by each solver.

| | CPLEX | | | GUROBI | | | SCIP | | |
|---|---|---|---|---|---|---|---|---|---|
| Size (k) | s | n | t | s | n | t | s | n | t |
| 1 | 10 | 11 | 0 | 10 | 21 | 0 | 10 | 12 | 0 |
| 2 | 10 | 15 | 0 | 10 | 23 | 0 | 10 | 19 | 1 |
| 4 | 10 | 38 | 0 | 10 | 30 | 1 | 10 | 41 | 4 |
| 8 | 10 | 59 | 0 | 10 | 50 | 3 | 10 | 79 | 10 |
| 16 | 9 | 302 | 3 | 10 | 84 | 15 | 10 | 263 | 23 |
| 32 | 7 | 213 | 11 | 10 | 175 | 47 | 10 | 419 | 48 |
| 64 | 9 | 50 | 26 | 10 | 1921 | 424 | 9 | 1328 | 178 |
| 128 | 8 | 276 | 79 | 7 | 1470 | 1098 | 10 | 6542 | 808 |
| 256 | 6 | 1366 | 564 | 7 | 699 | 4182 | 8 | 6225 | 2041 |
| 512 | 2 | 3265 | 1700 | 7 | 198 | 3586 | 6 | 6125 | 6347 |
| 1024 | 2 | 1509 | 5501 | 3 | 112 | 16943 | 0 | - | - |

Table 1: Number of instances solved (s), and, for the instances solved, the geometric means of the number of nodes (n) and time in seconds (t) with a 5 hour time limit, depending on the size of the graph $G_k$.

As $k$ grows, and as long as all instances are solved, the number of nodes required increases for all solvers. At $k = 16$ and above, some solvers start running out of time, and not all of the 10 instances are solved. Concomitantly, the number of nodes may then decrease. Indeed, as $k$ grows, the time required to solve each node increases. As a consequence, the solver either gets lucky and takes good early branching decisions, yielding a small tree, or it runs out of time before fully exploring the tree.

As a comparison, we have also conducted experiments in the case $\chi' = 3$, by replacing the subgraph $P1$ by $P2$ in $G_k$ (i.e. it now contains $k$ subgraphs $P2$). Table 2 provides results in the same setup as Table 1. These instances are clearly harder to solve than the ones with $\chi' = 4$, i.e. closing the dual gap is easier than closing the primal gap. Indeed, for these instances, there is a trivial linear-size B&B tree, since there exists a 3-coloring, which objective value matches the dual bound at the root, but it is not clear if a fixed-size tree exists. So in this respect, MIP solvers are not completely fooled.

### 3.3 Branching rule analysis

We study the behavior of the default branching rule of the open-source code SCIP, because we know exactly how it performs. It is called *reliability pseudocost branching* [4, 2, 3], and mostly relies on *strong branching* and *pseudocost branching*. Strong branching assigns high scores to variables producing high dual bounds (when minimizing) in the children of the current node. Pseudocost branching is a *history-based* rule that tries to predict the score that strong branching would compute. It is an inexpensive alternative to strong branching, but it needs a history of previous branchings to be reliable. In reliability

| | CPLEX | | | GUROBI | | | SCIP | | |
|---|---|---|---|---|---|---|---|---|---|
| Size (k) | s | n | t | s | n | t | s | n | t |
| 1 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 1 | 0 |
| 2 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 1 | 0 |
| 4 | 10 | 1 | 0 | 10 | 0 | 0 | 10 | 2 | 1 |
| 8 | 10 | 0 | 0 | 10 | 0 | 0 | 10 | 12 | 4 |
| 16 | 10 | 144 | 2 | 10 | 17 | 6 | 10 | 267 | 21 |
| 32 | 10 | 2308 | 26 | 10 | 4086 | 175 | 10 | 32463 | 528 |
| 64 | 6 | 72153 | 1157 | 3 | 61730 | 963 | 4 | 202926 | 4635 |
| 128 | 0 | - | - | 0 | - | - | 0 | - | - |

Table 2: Results in the same setup as Table 1, except that graphs are modified so that $\chi' = 3$.

pseudocost branching, strong branching is performed when pseudocosts are considered unreliable. Pseudocosts can then be updated using the strong branching information.

For instances $I_k$, it turns out that pseudocosts never help. Indeed, recall that a lower bound to the LP at any node is 3. Suppose strong branching tests a particular candidate variable: either both child LPs have value 3, or at least one of them cannot lead to a strictly better feasible solution (since the objective function is integral and a solution of value 4 is known). Such a child is considered infeasible, even if its LP has objective value in $)3, 4]$, and pseudocosts are then not updated. Pseudocosts are thus systematically 0 for all variables throughout all runs, and therefore provide no information on branching candidates.

The first tie-breaking strategy of reliability pseudocost branching is *conflict analysis* [1]. Conflict analysis extracts information from infeasible nodes by considering all variable bound changes (typically, branching decisions) leading to that node, and by finding a small subset of these changes such that the resulting subproblem is still infeasible. The conflict score of a variable increases when it is involed in a newly created conflict, and it decays otherwise.

At the root node, heuristics may produce some conflicts, but conflict analysis will still mostly be uninitialized, leading to a branching decision that – in our experiments – systematically varies depending on the row permutations. Note that two other tie-breaking rules exist should conflict analysis fail to select a single best candidate, but they both exhibit a similar behavior.

Initially, all variables have an equal probability of being branched on, but for each good variable (i.e. from the $P1$ gadget) there are $k-1$ bad ones. Branching randomly eventually creates infeasible nodes, which populates the conflict pool. It is important to point out that conflicts may not include any good variable, and in fact, most infeasible nodes are in practice the result of an improper coloring of two adjacent edges: this can happen at any vertex of the graph. Since branching is initially random, the conflicts created by branching are also randomly created. Consequently, the conflict pool is randomly populated and

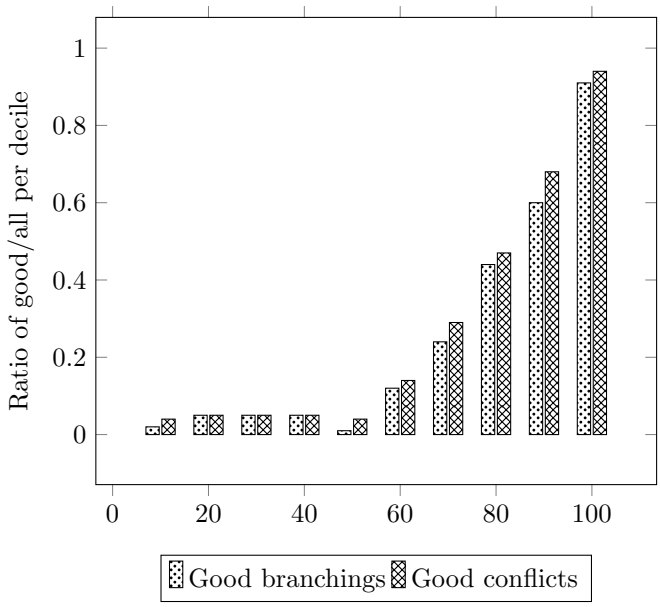
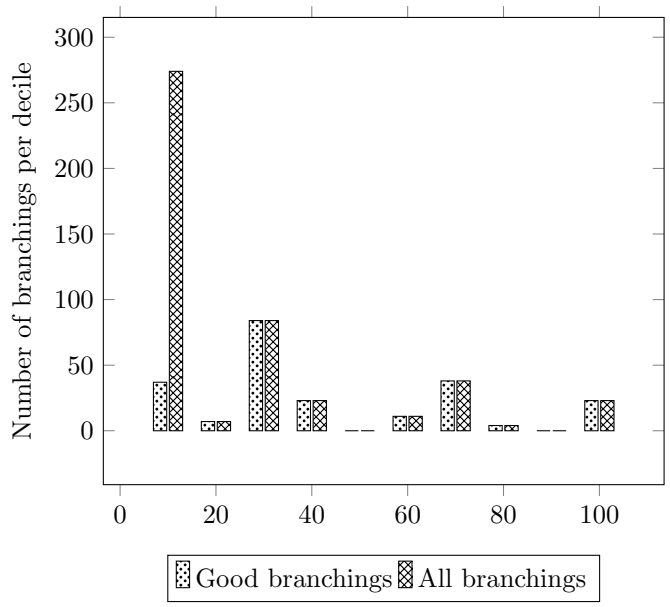



Figure 3: Ratio of good branchings and good conflicts per decile of the B&B run, averaged over 20 instances with $k = 256$.

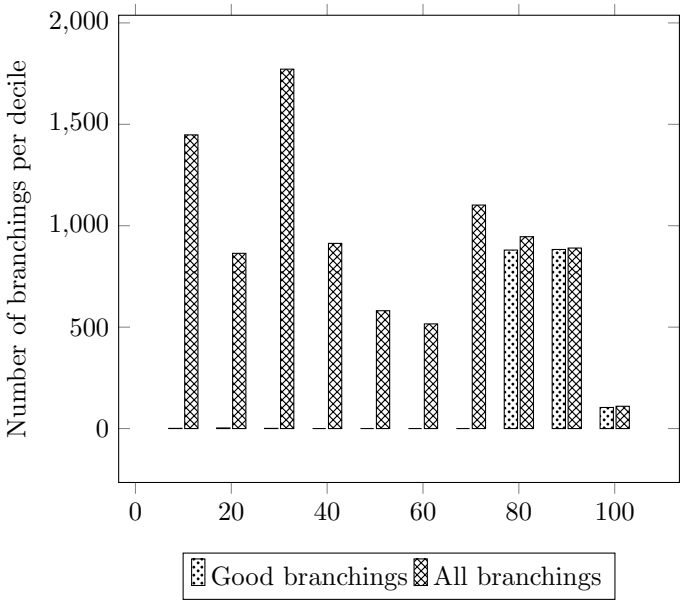
branching remains random. Eventually, a conflict with a set of good variables is created, and some good variables will then be among the best candidates. By branching on good variables, it is very likely that other good conflicts will appear and increase the score of good variables. In practice, we indeed observe a *snowballing* effect: as soon as good conflicts are found, good branchings are performed, which in turns generate good conflicts. Figure 3 gives the ratio of good branchings and good conflicts during B&B runs, and shows that the two are extremely correlated. Figure 4 provides the map of the good branchings for an easy and a hard instance. In our testbed, most of the instances have the profile of the hard instance. This statistical evidence supports the previous analysis of the branching rule. We do not know to what extent this analysis applies to the branching rule of CPLEX and GUROBI, but the high variability of the results for a given $k$ also suggests a very random behavior.

### 3.4 Further experiments

We have performed additional tests on instances $I_k$ by tuning parameters based on problem knowledge and the analysis of Section 3.3. Since we know the best solution is available at the root, and that reducing the dual gap is hard, we have tuned the balance between feasibility and optimality using the parameter *emphasis mip* (set to 3) of CPLEX and *MIPFocus* (set to 3) of GUROBI, that both focus on improving the dual bound rather than the primal bound. In




(a) Instance solved in 679 nodes.




(b) Instance solved in 17581 nodes.

Figure 4: Map of good branchings and all branchings per decile of the B&B run, for an easy instance and a hard one, both of size $k = 256$.

| Setting | CPLEX +probing | | | CPLEX +dual bound | | | GUROBI +dual bound | | |
|---|---|---|---|---|---|---|---|---|---|
| Size (k) | s | n | t | s | n | t | s | n | t |
| 1 | 0 | 0 | - | 0 | 0 | - | 0 | -24 | - |
| 2 | 0 | 0 | - | 0 | 0 | - | 0 | +4 | - |
| 4 | 0 | 0 | - | 0 | 0 | - | 0 | -3 | +200 |
| 8 | 0 | 0 | - | 0 | -5 | - | 0 | -50 | +167 |
| 16 | 0 | 0 | 0 | 0 | +30 | +67 | 0 | -36 | +40 |
| 32 | 0 | 0 | 0 | 0 | +376 | +336 | 0 | -15 | +104 |
| 64 | 0 | -16 | -8 | +1 | +40 | +31 | 0 | -79 | +18 |
| 128 | +1 | +66 | +8 | 0 | -7 | +42 | +3 | -53 | +197 |
| 256 | +1 | -60 | -48 | -2 | -78 | -29 | +1 | +48 | +101 |
| 512 | -1 | -24 | -37 | -1 | -20 | -48 | -7 | - | - |
| 1024 | 0 | -28 | -72 | +2 | -19 | -71 | -3 | - | - |

Table 3: Parameter tuning with CPLEX and GUROBI. The number of instances solved (s) is reported as the difference with the corresponding column of Table 1. The number of nodes (n) and the time in seconds (t) are relative variations in %.

| Setting | SCIP +conflicts | | | SCIP - strong branching | | | SCIP inference | | |
|---|---|---|---|---|---|---|---|---|---|
| Size (k) | s | n | t | s | n | t | s | n | t |
| 1 | 0 | +17 | - | 0 | 0 | - | 0 | +708 | - |
| 2 | 0 | -21 | 0 | 0 | +16 | 0 | 0 | +658 | -100 |
| 4 | 0 | -56 | -25 | 0 | -2 | 0 | 0 | +559 | -75 |
| 8 | 0 | -70 | -10 | 0 | +18 | +10 | 0 | +257 | -70 |
| 16 | 0 | -46 | +17 | 0 | -24 | -9 | 0 | +41 | -70 |
| 32 | 0 | +102 | +69 | 0 | +29 | -2 | 0 | +51 | -63 |
| 64 | +1 | -34 | +17 | +1 | +48 | +17 | +1 | +201 | -16 |
| 128 | 0 | -76 | -13 | -2 | -46 | -40 | 0 | +137 | +29 |
| 256 | +1 | -52 | +60 | -2 | +16 | -2 | -3 | +295 | +75 |
| 512 | +1 | -59 | +57 | 0 | +15 | -14 | -3 | +47 | -38 |
| 1024 | 0 | - | - | 0 | - | - | 0 | - | - |

Table 4: Parameter tuning with SCIP, with the same notations as Table 3.

CPLEX, we have also run the tests using *mip strategy probe* (set to 3. Probing is a preprocessing technique, see e.g. [15]). The results of these three distinct parameter changes are reported in Table 3.

Based on the analysis, we have tried more specific parameters in SCIP. The first test removes all limits (e.g. LP iterations) to the resources allocated to conflict analysis, and enables it in all implemented cases. The second test reduces the extent of strong branching (namely, the maximum number of branching candidates investigated and the total LP iterations are reduced by a factor of 5 and 2 respectively). The third test uses inference branching [2], a SAT solving rule that does not take the objective function into account, instead of reliability pseudocost branching. The results are reported in Table 4.

For CPLEX, both probing and giving priority to dual bound improvement pay off for large values of $k$. On the other hand, giving priority to dual bound improvement decreases overall performance for GUROBI. For SCIP, emphasizing conflict analysis drastically reduces the number of nodes created, but at a steep price in terms of computation time. Reducing the resources allocated to strong branching do not seem to improve running times substantially. (Note that completely disabling strong branching leads to an overall 91% increase in the number of nodes and a 15% increase in time. This is most likely due to the fact that strong branching can detect infeasibility earlier than other branching rules, creating fewer nodes.) Inference branching produces many more nodes, and does not yield consistent benefits in terms of computing time.

In light of these results, it seems unlikely that any amount of parameter tuning will enable either solver to produce trees of near-constant size for increasing values of $k$.

# 4 Conclusion

We exhibit a family of MIP instances for which solvers use an arbitrarily higher amount of resources than theoretically needed. To the best of our knowledge, this is the first result of this kind in a MIP setting. Every solver and configuration tested presents similar behavior, which leads us to believe that new techniques (e.g., exploiting structure or symmetry, for instance during preprocessing, branching or cut generation) may be needed for MIP solvers to efficiently solve these instances.

# References


[1] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4 – 20, 2007.

[2] T. Achterberg. *Constraint Integer Programming.* PhD thesis, 2007.

[3] T. Achterberg and T. Berthold. Hybrid branching. In WJ. Hoeve and J. N. Hooker, editors, *Integration of AI and OR Techniques in Constraint*

*Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 309–311. Springer, 2009.

[4] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42 – 54, 2005.

[5] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[6] D. A. Holton and J. Sheehan. *The Petersen Graph*. Cambridge University Press, 1993.

[7] I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

[8] J. Lee and J. Leung. A comparison of two edge-coloring formulations. *Operations Research Letters*, 13(4):215 – 223, 1993.

[9] J. Lee, J. Leung, and S. Vries. Separating type-I odd-cycle inequalities for a binary-encoded edge-coloring formulation. *Journal of Combinatorial Optimization*, 9(1):59–67, 2005.

[10] P. Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1–2):315 – 326, 2000.

[11] A. Lodi and A. Tramontani. Performance variability in mixed-integer programming. In Topaloglu H, editor, *Tutorials in Operations Research*, volume 10, chapter 1, pages 1–12. 2013.

[12] G. L. Nemhauser and S. Park. A polyhedral approach to edge coloring. *Oper. Res. Lett.*, 10(6):315–322, 1991.

[13] M. Ouyang. How good are branching rules in DPLL? *Discrete Applied Mathematics*, 89(1-3):281–286, 1998.

[14] J. Petersen. Sur le théorème de Tait. *L'Intermédiaire des Mathématiciens*, 5, 1898.

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

[16] E. R. Scheinerman and D. H. Ullman. *Fractional Graph Theory – A Rational Approach*. Wiley, 1997.

[17] S. Skulrattanakulchai. 4-edge-coloring graphs of maximum degree 3 in linear time. *Inf. Process. Lett.*, 81(4):191–195, 2002.

[18] V. G. Vizing. On an estimate of the chromatic class of a $p$-graph. *Diskret. Analiz*, 3:25–30, 1964.