On the Complexity of Finding Shortest Variable Disjunction Branch-and-Bound Proofs

Max Gläser[0000−0001−6150−9431] and Marc E. Pfetsch[0000−0002−0947−7193]

Department of Mathematics, TU Darmstadt, Germany
{glaeser,pfetsch}@mathematik.tu-darmstadt.de

Abstract. We investigate the complexity of finding small branch-and-bound trees using variable disjunctions. We first show that it is not possible to approximate the size of a smallest branch-and-bound tree within a factor of $2^{\frac{1}{5}n}$ in time $2^{\delta n}$ with $\delta < \frac{1}{5}$, unless the strong exponential time hypothesis fails. Similarly, for any $\varepsilon > 0$, no polynomial time $2^{(\frac{1}{2}-\varepsilon)n}$-approximation is possible, unless P = NP. We then discuss that finding small branch-and-bound trees generalizes finding short treelike resolution refutations. Therefore, hardness results, in particular non-automatizability results, transfer from this setting. Finally, we show that computing the size of a smallest branch-and-bound tree is #P-hard. Similar results hold for estimating the size of the tree produced by branching rules like most-infeasible branching.

Keywords: binary program · branch-and-bound proof · counting problem.

1 Introduction

The currently dominant strategy to solve mixed-integer linear programs is the branch-and-cut method. Besides the addition of cutting planes, the core is formed by branch-and-bound, based on linear programming (LP) relaxations. The nodes are usually created by variable branching, i.e., a disjunction on the variable bounds. Many branching rules, i.e., methods to choose the branching variable at each node, are known, see, e.g., [2, 1, 8] to mention a few.

This raises the desire to evaluate the performance of branching rules in comparison to the best possible, i.e., to the smallest size of a tree. In this paper, we investigate the theoretical complexity of computing the size of a smallest branch-and-bound tree based on variable branching. We concentrate on proving optimality by assuming that an objective cut using the optimal value is integrated into the system. Then the goal is to estimate the size of a smallest tree to prove infeasibility of a linear inequality system with integrality requirements on the variables. Moreover, we restrict attention to pure branch-and-bound algorithms, i.e., no cutting plane separation, domain reduction, presolving is used.

We first prove in Section 3 that it is not possible to approximate the size of a smallest branch-and-bound tree using variable branching within a factor of $2^{\frac{1}{5}n}$ in time $2^{\delta n}$ with $\delta < \frac{1}{5}$, where n is the number of variables, unless the strong

exponential time hypothesis fails (Theorem 2). The same argument can be used to show that unless P = NP, no polynomial time algorithm can approximate the size of smallest tree within $2^{\frac{1}{2}-\varepsilon}$ for any $\varepsilon > 0$ (Theorem 1). This result significantly strengthens the hardness of approximation within a factor of 2 in Hendel et al. [15].

However, can the smallest size be approximated, if polynomial time in the size of the instance is spent for every node of the smallest tree? This is captured by the notion of automatizability of proof systems, i.e., a proof can be produced in polynomial time in the size of the instance and of a shortest proof. In Section 4, we transfer hardness results from the literature to show that branch-and-bound with variable branching is not automatizable under reasonable hardness assumptions. However, it is quasi-automatizable, i.e., a proof can be produced in quasi-polynomial time in the size of the instance and smallest proof.

In Section 5, we prove that computing the size of a smallest branch-and-bound tree is #P-hard (Theorem 6). To the best of our knowledge, such a hardness result is novel, even across all commonly considered proof systems. Furthermore, for a binary program it is #P-hard to compute the size of the tree produced by many branching rules from practice (e.g., the most-infeasible branching rule), given suitable tie-breaking. Since one can construct a SAT-formula for which satisfying assignments correspond to leaves of the resulting tree [18, 22], this task is actually in #P (when solving LPs with a polynomial time algorithm). Due to the famous theorem of Toda [23], neither of the two aforementioned estimation problems can be solved in polynomial time, even when given access to an oracle from the polynomial hierarchy, unless the polynomial hierarchy collapses.

Note that Valiant and Vazirani [26] show that every problem in #P has a polynomial-time randomized approximation scheme, assuming access to an NP-oracle, making a result by Sipser [21] and Stockmeyer [22] explicit. However, we do not have a reason to believe that computing the size of a smallest branch-and-bound tree lies in #P. We note that Le Bodic and Nemhauser [20] consider an abstract model in which branching on a particular variable improves the value of the LP-bound for the two children by a fixed amount and show that computing the size of a shortest tree in their model is (weakly) NP-hard.

Branch-and-bound algorithms in practice almost always cut off the optimal LP solution of the currently considered node by branching on a fractional variable. Thus, our results also hold for LP-based branch-and-bound trees.

Note that pure branch-and-bound described above produces mixed results: On the one hand, it solves random binary programs (with a fixed number of constraints) in polynomial time [12], on the other hand, it takes exponential time to prove infeasibility of matching polytopes with an objective cut [6, Theorem 2.2], even though matching is solvable in polynomial time.

2 Preliminaries

2.1 Binary Programs and Branch-and-Bound Proofs

We consider binary programs (BP) of the following form

$$\max\{c^\top x \,:\, Ax \leq b,\ x \in \{0,1\}^n\},$$

where $n \in \mathbb{N}$, $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$. We define $[n] \coloneqq \{1,\dots,n\}$.

To concentrate the investigation on the ability of branch-and-bound to prove optimality (or infeasibility), we assume that the problems are bounded and we know the optimal values of feasible instances, which are integrated by an objective cut into the problem. As a consequence, we are aiming for proving infeasibility of the system

$$Ax \leq b, \quad x \in \{0,1\}^n. \tag{P}$$

To fix notation, we formalize bound-and-bound proofs. A *(variable disjunction) branch-and-bound proof (of infeasibility)* of (P) or *branch-and-bound refutation* of (P) is a rooted binary directed tree T with the following properties:

- Every non-leaf node N is labeled with some variable x_i (the *branching variable* for N), one of the outgoing edges from N is labeled by the *variable fixing* $x_i = 0$ and is called the *left child* (or $x_i = 0$-*child*) of N and the other outgoing edge from N is labeled by $x_i = 1$ and is called the *right child* (or $x_i = 1$-*child*) of N.
- To each node N there corresponds a linear program (P^N), which is the LP-relaxation of (P) strengthened by all the variable fixings which occur as labels on the path from the root to N and (P^N) is infeasible if and only if N is a leaf. We say N is *(in)feasible*, if (P^N) is.

The $x_i = a$-branch at a node N, $a \in \{0,1\}$, is the directed subtree rooted at the $x_i = a$-child of N. Without loss of generality, we assume every variable occurs at most once as a label of a node on any root-leaf path in T.

We are then interested in $\mathcal{T}(I)$, which for an infeasible instance I of problem (P) is the size of a shortest branch-and-bound proof; the *size* of a proof is the number of its nodes.

2.2 Boolean Formulas and the Exponential Time Hypothesis

A *literal* is a Boolean variable or its negation. A *clause* C is a disjunction of literals. A Boolean formula φ in *conjunctive normal form (CNF)* is a conjunction of clauses. Let $\mathcal{C}$ denote the set of clauses in φ. For a clause $C \in \mathcal{C}$ denote by $\mathcal{P}(C)$ the set of unnegated variables in C and by $\mathcal{N}(C)$ the set of negated variables in C. A CNF φ is a *k-CNF*, if every clause of φ contains at most k literals. Without loss of generality, every k-CNF contains at most $\binom{n}{k}2^k$ clauses.

For some results we will rely on the (strong) exponential time hypothesis formulated by Impagliazzo and Paturi [16] as a hardness assumption:

Let (k-)SAT denote the problem of deciding whether a (k-)CNF is satisfiable and

$$s_k \coloneqq \inf\{s \in \mathbb{R}_+ : k\text{-SAT can be decided in time } O(2^{sn})\}.$$

The *exponential time hypothesis (ETH)* states $s_3 > 0$ and the *strong exponential time hypothesis (SETH)* states $s_\infty \coloneqq \lim_{k\to\infty} s_k = 1$.

3 Hardness of Approximation

We first prove that the size $\mathcal{T}$ of a smallest branch-and-bound proof cannot be approximated within any exponential factor within subexponential time, unless ETH fails.

Consider $J_n = \{x \in [0,1]^{2n} : \sum_{i=1}^{2n} 2x_i = 2n+1\}$, a classical example by Jeroslow [17]. Note that $J_n \cap \mathbb{Z}^{2n}$ is empty, but J_n is not. It is easy to see that one needs to fix at least n variables before attaining LP-infeasibility. Thus, any proof of $J_n \cap \mathbb{Z}^{2n} = \varnothing$ must contain at least $2^{n+1} - 1$ nodes.

Furthermore, for a given CNF φ with variables $x_1, \dots, x_n$ and clauses $\mathcal{C}$, consider the following binary problem:

$$\sum_{x_i \in \mathcal{P}(C)} x_i + \sum_{x_i \in \mathcal{N}(C)} (1 - x_i) \geq \tfrac{1}{2} \quad \forall C \in \mathcal{C}, \qquad x_1, \dots, x_n \in \{0,1\}. \qquad (Q_\varphi)$$

Satisfying assignments of φ correspond to integer points in (Q_φ).

Given a set of instances $\mathcal{I}$ and $\alpha \geq 1$, an *α-approximation algorithm* for a function $f : \mathcal{I} \mapsto \mathbb{N}$ is an algorithm A which satisfies $A(I)/\alpha \leq f(I) \leq A(I) \cdot \alpha$ for all $I \in \mathcal{I}$. Our first result is

Theorem 1. *There is no polynomial time α-approximation algorithm for $\mathcal{T}$ with $\alpha < 2^{(\frac{1}{2}-\varepsilon)n}$ for any $\varepsilon > 0$, unless $P = NP$.*

The proof is analogous to that of the next result:

Theorem 2. *For any parameter $\lambda \in \mathbb{N}$ with $\lambda > 1$, there is no α-approximation algorithm for $\mathcal{T}$ with $\alpha < 2^{(\frac{1}{2} - \frac{2}{2\lambda+1})n}$ running in time $O(2^{\delta n})$ with $\delta < \frac{1}{1+2\lambda}$, unless SETH fails. Furthermore, for every $\varepsilon > 0$, there is some $\delta > 0$ such that there is no α-approximation algorithm for $\mathcal{T}$ with $\alpha < 2^{(\frac{1}{2}-\varepsilon)n}$ running in time $O(2^{\delta n})$, unless ETH fails.*

In particular, for $\lambda = 2$, we obtain that it is not possible to approximate $\mathcal{T}$ within a factor of $2^{\frac{1}{5}n}$ in time $2^{\delta n}$ with $\delta < \frac{1}{5}$, unless SETH fails. Moreover, Theorem 1 corresponds to $\lambda \to \infty$.

Proof. We prove the theorem by contraposition: Assume such an algorithm exists for some λ. We will show that we can then give a polynomial time algorithm for SAT. Let a CNF φ with variables $y_1, \dots, y_m$ and clauses $\mathcal{C}$ be given, and let

$\ell := \lambda m$. Consider the following binary program (J) with $n = 2\ell + m$ variables:

$$\sum_{i=1}^{2\ell} 2x_i = 2\ell + 1, \tag{Ja}$$

$$\sum_{y_i \in \mathcal{P}(C)} y_i + \sum_{y_i \in \mathcal{N}(C)} 1 - y_i \geq \tfrac{1}{2} \qquad \forall C \in \mathcal{C}, \tag{Jb}$$

$$x_i,\ y_j, \in \{0,1\} \qquad \forall i \in [2\ell],\ j \in [m].$$

Assume that φ has no satisfying assignment. Then exhaustively branching on all y_i yields infeasibility due to (Jb) and thus a proof of size at most $2^{m+1}-1$.

Conversely, assume there is an assignment satisfying φ and we are given a branch-and-bound proof T of (J). It is easy to see that we can obtain a proof T' for $J_\ell \cap \mathbb{Z}^{2\ell} = \varnothing$ with $|T'| \leq |T|$ from T and thus we must have $|T| \geq 2^{\ell+1} - 1$. Since $\ell \geq m$, we have

$$\frac{2^{\ell+1}-1}{2^{m+1}-1} \geq \frac{2^{\ell+1}}{2^{m+1}} = 2^{\ell-m} = 2^{(\lambda-1)m} = 2^{\frac{\lambda-1}{2\lambda+1}(2\ell+m)} = 2^{\frac{\lambda-1}{2\lambda+1}n}.$$

Therefore, approximating the size of a shortest branch-and-bound proof for (J) within factor $2^{\frac{\lambda-1}{2\lambda+1}n}$ decides whether φ is satisfiable. Further, an $O(2^{\delta n})$-time algorithm with $\delta < \frac{1}{2\lambda+1}$ for this task is an $O(2^{(2\lambda+1)\delta m})$-time algorithm for k-SAT for arbitrary k with $(2\lambda+1)\delta < 1$ and thus contradicts SETH.

Furthermore, choose λ with $\frac{2}{2\lambda+1} < \varepsilon$ for any fixed $\varepsilon > 0$. Then an $O(2^{\delta n})$-time algorithm approximating $\mathcal{T}$ within factor $2^{(\frac{1}{2}-\varepsilon)n}$, yields an $O(2^{\delta' m})$-time algorithm for 3-SAT for $\delta' := (2\lambda+1)\delta$. If such an algorithm exists for every $\delta > 0$ (and therefore every δ'), this violates ETH, which yields the second part of the statement. □

Remark 3. In the above proof, we can always branch on a fractional variable, i.e., Theorem 1 and 2 hold for LP-based branch-and-bound trees.

Remark 4. Note that for BPs with an exponential number of constraints, we can replace J_n by $P_n := \{x \in [0,1]^n : \sum_{i \in R} x_i + \sum_{i \notin R}(1-x_i) \geq \frac{1}{2}\ \forall R \subseteq [n]\}$, for which any branch-and-bound proof with variable disjunctions must be a complete binary tree of depth n. We can then strengthen the factor $\frac{1}{2} - \frac{2}{2\lambda+1}$ in the exponent of the approximation ratio in the first part of Theorem 2 to $1 - \frac{2}{\lambda+1}$ and the factor of $\frac{1}{2} - \varepsilon$ in the second part to $1 - \varepsilon$. Note P_n is separable in polynomial time: Given a point $\hat{x} \in [0,1]^n$, the set $\hat{R} := \{i \in [n] : \hat{x}_i \leq \frac{1}{2}\}$ minimizes the left hand side. Thus, $\hat{x} \in P_n$ if and only if $\hat{x}$ satisfies $\sum_{i \in \hat{R}} \hat{x}_i + \sum_{i \notin \hat{R}}(1-\hat{x}_i) \geq \frac{1}{2}$.

Cook et al. [10] showed that there is no cutting plane proof of $P_n \cap \mathbb{Z}^n = \varnothing$ of size at most $2^n/n$, even if we replace $\frac{1}{2}$ on the right hand side by 1. Dadush and Tiwari [11] showed that there also does not exist a branch-and-bound proof of size $2^n/n$ even if we allow branching on general disjunctions.

4 (Non-)Automatizability of Branch-and-Bound

In this section we sketch how results about automatizing finding shortest so-called treelike resolution refutations transfer to the case of branch-and-bound proofs.

Treelike resolution is a proof system which certifies the infeasibility of Boolean formulas in CNF. Treelike resolution can be seen as a branch-and-bound procedure, where we branch by fixing a variable to either 1 (true) or 0 (false) and prune only nodes in which the already fixed variables *falsify a clause*, i.e., we prune a node N, if there is a clause C with fixings $x = 0$ for all $x \in \mathcal{P}(C)$ and $x = 1$ for all $x \in \mathcal{N}(C)$ in the subproblem associated with N [19, Section 5.2]. A *treelike resolution proof* or *refutation* is a tree produced by this procedure.

Automatizability. It is easy to see that not all infeasible instances of an NP-complete problem can have proofs of size bounded by some polynomial, unless NP = coNP. Therefore the reason for the inability of polynomial-time algorithms to find shortest proofs might be the size of these proofs instead of the computational difficulty of finding them. To investigate this possibility, Bonet et al. [9] introduced the concept of *automatization*. A proof system is called *automatizable (quasi-automatizable)*, if given an infeasible instance I, an infeasibility proof in this system can be produced in time polynomial (quasipolynomial) in $|I|$ and L, where $|I|$ is the encoding length of I and L is the size of a shortest proof of I.

Beame and Pitassi [7] proved the positive result that branch-and-bound using variable disjunctions is quasi-automatizable for binary programs, for which we reproduce the proof for completeness:

Consider the following recursive procedure $R(S, n)$, which generates a branch-and-bound proof for a given binary program (P) with n variables, if there is a proof of size at most S. Among the $2\,n$ possible fixings of a variable in (P), find a fixing $x_i = a \in \{0, 1\}$, such that $R(S/2, n-1)$ succeeds on $(P \cap \{x_i = a\})$ by trial-and-error. This works if there exists a proof T of size at most S, because one of the branches at the root node of T has size at most $S/2$. Then apply $R(S, n-1)$ to $(P \cap \{x_i = 1 - a\})$, which succeeds if there exists a proof of size at most S and because $\mathcal{T}$ is non-increasing with respect to fixing variables. The resulting running time recursion $t(S, n) = 2\,n \cdot t(S/2, n-1) + t(S, n-1)$ solves to roughly $n^{\log S}$. We can then try $R(S, n)$ for increasing values of S until it succeeds. □

We now observe that the treelike resolution refutations of φ are the branch-and-bound proofs using variable disjunctions of (Q_φ). Indeed, in both cases a node is infeasible if and only if the already fixed variables falsify a clause.

Thus, we can replace "treelike resolution" by "branch-and-bound using variable disjunctions" in a result by Alekhnovich and Razborov [3], which has been strengthened by Eickmeyer et al. [14] to obtain: Branch-and-bound using variable disjunctions is not automatizable unless FPT = W[P]. Here, FPT $\subseteq$ W[P] may be seen as an analogue of the question whether NP $\subseteq$ P in the world of parameterized problems.

Notably, the recent breakthrough of Atserias and Müller [5], showing non-automatizability of general resolution under the weaker and optimal assumption $\mathrm{P} \neq \mathrm{NP}$, likely does not transfer to branch-and-bound using variable disjunctions (or treelike resolution). In fact, their approach – to show that it is NP-hard to distinguish between instances with a refutation of size bounded by some polynomial and instances with refutations of at least exponential size – likely does not transfer; otherwise, quasi-automatizability of branch-and-bound using variable disjunction would cause the exponential time hypothesis to fail [3, 5].

5 #P-Hardness of Exact Computation

We first review #P-hardness as introduced by Valiant [25, 24], which we will use to bound the hardness of finding a shortest branch-and-bound proof from below:

Definition 5. *A function $g\colon \{0,1\}^* \to \mathbb{N}$ is in #P if there exists a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ and a polynomial time Turing machine M such that for every $x \in \{0,1\}^*$ we have*

$$g(x) = |\{y \in \{0,1\}^{p(|x|)} : M \text{ accepts on input } (x,y)\}|.$$

So #P is the class of functions which count certificates of some NP-problem. For example, #SAT is the function that on input of a Boolean formula returns the number of satisfying assignments.

We will identify natural numbers with their binary representation and let FP^f denote the class of functions $g\colon \{0,1\}^* \to \mathbb{N}$ which are computable in polynomial time by a Turing machine with access to an oracle computing f. A function f is *#P-hard*, if $\#\mathrm{P} \subseteq \mathrm{FP}^f$. Because the Cook-Levin reduction preserves the number of certificates, #SAT is #P-complete, see, e.g., [4, Theorem 17.10].

Our main result is the following:

Theorem 6. *Computing $\mathcal{T}$ for binary programs is #P-hard.*

Our proof is based on a highly symmetrical problem with additional constraints on a subset of variables, ensuring that a Boolean formula φ is satisfied. Due to the symmetry without additional constraints, it is optimal to first branch on variables occuring in additional constraints. Then the number of surviving branches after fixing all the variables subject to additional constraints – i.e. the number of assignments satisfying φ – is roughly proportional to the length of a smallest branch-and-bound proof.

Let a formula φ in CNF with variables $x_1 \ldots, x_n$ be given. We then introduce additional variables $x_{n+1}, \ldots, x_{3n}, y_1, \ldots, y_{3n}, z_1, \ldots, z_{3n}$ and consider the

following BP (P_φ):

$$\sum_{x_i \in \mathcal{P}(C)} x_i + \sum_{x_i \in \mathcal{N}(C)} (1 - x_i) \geq \tfrac{1}{2} \qquad \forall C \in \mathcal{C}, \qquad (P_\varphi\text{a})$$

$$\sum_{i=1}^{3n} (y_i + z_i) \geq 3n + \tfrac{1}{2}, \qquad (P_\varphi\text{b})$$

$$y_i \leq \tfrac{3}{2} - x_i,\ z_i \leq \tfrac{1}{2} + x_i \qquad \forall i \in [3n], \qquad (P_\varphi\text{c})$$

$$(x, y, z) \in \{0,1\}^{3 \cdot 3n}.$$

In particular, none of the variables $x_{n+1}, \ldots, x_{3n}$ are contained in a constraint of type $(P_\varphi\text{a})$. Note that because of $(P_\varphi\text{b})$ and $(P_\varphi\text{c})$, this problem is always infeasible, even when ignoring the constraints $(P_\varphi\text{a})$. The following Lemmas 7 and 8 will give sufficiently tight bounds on $\#\mathrm{SAT}(\varphi)$ depending on $\mathcal{T}(P_\varphi)$ to see that knowledge of $\mathcal{T}(P_\varphi)$ suffices to compute $\#\mathrm{SAT}(\varphi)$ in polynomial time. This then proves Theorem 6.

Lemma 7. $\mathcal{T}(P_\varphi) \leq 6 \cdot 2^{2n} \cdot \#\mathrm{SAT}(\varphi) - 5 + 6(2^n - \#\mathrm{SAT}(\varphi))$

Proof. Construct a branch-and-bound proof T by choosing the branching variable in every node N as follows:

1. If N is a $x_i = 0$ child, choose z_i and if N is a $x_i = 1$-branch choose y_i,
2. otherwise choose an unbranched x_i with smallest index.

We say a node is at depth ℓ, if its distance from the root is ℓ. By construction, each node at even depth is labeled with some x_i, whereas nodes at odd depth are labeled with either some y_i or z_i.

Since we are proving an upper bound, we can assume that constraint $(P_\varphi\text{a})$ is only added at depth $2n$. Therefore, the problem never becomes infeasible after fixing a variable x_i. Thus, all nodes at odd depths are feasible. At even depths except for $2n$ and $6n$, exactly half of the nodes are feasible, namely the $y_i = 0$ and $z_i = 0$ nodes.

Together, this yields 2^{i+1} nodes at odd depth $2i+1$ for $i = 0, \ldots, n-1$ and $2 \cdot \#\mathrm{SAT}(\varphi)$ nodes at depth $2n+1$. For odd depth $2(n+i)+1$ for $i = 0, \ldots, 2n-1$, we have 2^{i+1} nodes for each solution of φ. Moreover, assigning to every node at odd depth the set consisting of its two children and itself, yields a partition of the node set without the root. This yields:

$$\begin{aligned}
\mathcal{T}(P_\varphi) &\leq 3 \cdot |\{\text{nodes labeled with } y \text{ or } z \text{ before level } 2n\}| \\
&\quad + 3 \cdot |\{\text{nodes labeled with } y \text{ or } z \text{ after level } 2n\}| + 1 \\
&= 3 \left(2 \sum_{i=0}^{n-1} 2^i + 2 \cdot \#\mathrm{SAT}(\varphi) \cdot \sum_{i=0}^{2n-1} 2^i \right) + 1 \\
&= 6(2^n - 1) + 6 \cdot \#\mathrm{SAT}(\varphi) \cdot (2^{2n} - 1) + 1,
\end{aligned}$$

which shows the claim. □

Lemma 8. $\mathcal{T}(P_\varphi) \geq 6 \cdot 2^{2n} \cdot \#\mathrm{SAT}(\varphi) - 5$.

Proof. Let $\mathcal{S}$ denote the set of points $(x, y, z) \in \{0,1\}^{3 \cdot 3n}$ such that

1. $x_1, \ldots, x_n$ satisfy φ,
2. $x_{n+1}, \ldots, x_{3n}$ can have any value and
3. $z_i = x_i$ and $y_i = 1 - x_i$ for all $i \in [3n]$ (i.e. y_i and z_i have the maximal binary value which is consistent with x_i with respect to $(P_\varphi \mathrm{c})$).

Evidently, $|\mathcal{S}| = \#\mathrm{SAT}(\varphi) \cdot 2^{2n}$.

Let T denote a smallest branch-and-bound proof of (P_φ) using variable disjunctions. We will show that every point $s \in \mathcal{S}$ must correspond to a unique leaf of T (Claim 2) and that the problems associated to these leaves must contain many variable fixings of y and z variables (Claim 1). This allows us to lower-bound the number of nodes of T labeled with these variables in terms of $|\mathcal{S}|$ (Claim 3). Together with the fact that T can be chosen exhibiting certain structure (Claim 4), this will allow us to isolate sufficiently many nodes of T for our desired bound.

We say that a 0/1-point $p = (x, y, z) \in \{0,1\}^{3 \cdot 3n}$ is *ruled out* by a leaf N of T, if p satisfies all variable fixings labeling the edges on the path from the root to N. Such a leaf N certifies that p is not feasible for (P_φ). Every $p \in \{0,1\}^{3 \cdot 3n}$ is ruled out by some leaf.

Claim 1. The subproblem $P_\varphi(N)$ associated to a leaf N ruling out $s = (x, y, z) \in \mathcal{S}$ must contain one of the variable fixings $y_i = 0$ or $z_i = 0$ for every $i = 1, \ldots, 3n$.

Proof of claim. Assume there is $i = 1, \ldots 3n$, such that $P_\varphi(N)$ neither contains $y_i = 0$ nor $z_i = 0$. Thus, if we define for $j = 1, \ldots, 3n$,

$$\hat{x}_j = x_j, \quad \hat{y}_j = \begin{cases} \frac{1}{2}, & \text{if } j = i \text{ and } x_j = 1, \\ 1, & \text{if } j = i \text{ and } x_j = 0, \\ y_j, & \text{otherwise,} \end{cases} \quad \hat{z}_j = \begin{cases} 1, & \text{if } j = i \text{ and } x_j = 1, \\ \frac{1}{2}, & \text{if } j = i \text{ and } x_j = 0, \\ z_j, & \text{otherwise,} \end{cases}$$

then $(\hat{x}, \hat{y}, \hat{z})$ is a feasible solution to $P_\varphi(N)$, contradicting infeasibility of N. ■

Claim 2. A leaf can rule out at most one point from $\mathcal{S}$.

Proof of claim. Assume $(\tilde{x}, \tilde{y}, \tilde{z})$, $(x', y', z') \in \mathcal{S}$ with $\tilde{x} \neq x'$ are both ruled out by a leaf N. (Note that differences in y and z imply differences in x.) Thus, there is $i \in [3n]$ such that $\tilde{x}_i \neq x'_i$ and the subproblem $P_\varphi(N)$ associated to N does not contain variable fixings on x_i, y_i or z_i. This is impossible by Claim 1. ■

Claim 3. There are at least $2(|\mathcal{S}| - 1)$ nodes in T labeled either with some y_i or z_i, which have a descendant ruling out a point from $\mathcal{S}$.

Proof of claim. Let X denote the set of inner nodes N in T for which both branches contain a node ruling out a point from $\mathcal{S}$. By Claim 2, we have $|X| = |\mathcal{S}| - 1$. For every $N \in X$, we will choose a set $\mathcal{H}_N$ of two descendants labeled with either y_i or z_i, which have a descendant ruling out a point from $\mathcal{S}$. To this end, we emphasize that both branches in T starting at N contain a leaf ruling out a point from S and consider the following cases:

1. If N is labeled by some x_i, then no node on the path from the root to N can be labeled with y_i or z_i, since otherwise not both the $x_i = 1$- and the $x_i = 0$-branch can contain a leaf ruling out a point from S (since only one is consistent with the earlier branching decision). Since both branches at N contain a leaf ruling out a a point from $\mathcal{S}$, both of these branches must contain a node labeled with either y_i or z_i by Claim 1. Choose H_N to be any two such nodes.
2. Otherwise, N is labeled with some y_i or z_i, say y_i. Once again, no node on the path from the root to N can be labeled with x_i or z_i. Furthermore, the $y_i = 1$-branch must contain a node ruling out a point from $\mathcal{S}$. Again by Claim 1, a node U in this branch must be labeled with z_i. Choose $\mathcal{H}_N = \{N, U\}$.

We claim that $\mathcal{H}_N \cap \mathcal{H}_M = \emptyset$ for distinct nodes N, $M \in X$. For the sake of contradiction assume $\mathcal{H}_N \cap \mathcal{H}_M \neq \emptyset$. By construction, the labels of N and M must be distinct elements of $\{x_i, y_i, z_i\}$ for some index i. Furthermore, since $\mathcal{H}_N$ and $\mathcal{H}_M$ contain only descendants of N and M, there must be a ancestor relationship between N and M in T, say N is a descendant of M. But then not both branches at N can contain a leaf ruling out a point from $\mathcal{S}$, as only for one of these branches the branching decision at N is consistent with the branching decision at M, contradicting our choice of X. Thus, the set $\mathcal{H} = \bigcup_{N \in X} \mathcal{H}_N$ is a collection of $2(|S|-1)$ nodes labeled with either y_i or z_i, which have a descendant ruling out a point from $\mathcal{S}$, as desired. ■

Claim 4. T can be chosen to satisfy the following properties:

1. Let be N be a node labeled with some y_i, then N either has an ancestor A labeled with x_i and is in the $x_i = 1$-branch at A or a descendant in the $y_i = 1$-branch at N is labeled with x_i. Analogously, if N is labeled with some z_i, then N either has an ancestor A labeled with x_i and is in the $x_i = 0$-branch at A or a descendant in the $z_i = 1$-branch at N is labeled with x_i.
2. If a variable y_i or z_i is branched after x_i, then it is branched immediately after, i.e., if there is a node N labeled with some y_i or z_i which is a descendant of some A labeled with x_i then N is A's child.

Proof of claim. 1. Assume there is a node N labeled with y_i that is neither in the $x_i = 1$ branch of an ancestor labeled with x_i nor has a descendant in the $y_i = 1$-branch labeled with x_i. We claim that the subtree S rooted at N can be replaced by the $y_i = 1$-branch S' at N as shown in Figure 1, removing the branching on y_i. Since S' is a subtree of S, every node of S' corresponds to a node of S in the obvious way. Let T' denote the tree obtained by cutting off S from T and attaching S' to T at N instead. It remains to show that all of the problems $P_\varphi(L')$ for a leaf L' of S' are infeasible, where $P_\varphi(L')$ is taken with respect to T'. Let L be the node corresponding to L' in S and $P_\varphi(L)$ be the associated subproblem (with respect to T). Then $P_\varphi(L)$ is $P_\varphi(L')$ strengthened by the variable fixing $y_i = 1$, but does not contain the variable fixing $x_i = 1$. It is obvious that if $P_\varphi(L')$ has a feasible solution, then it has a feasible solution

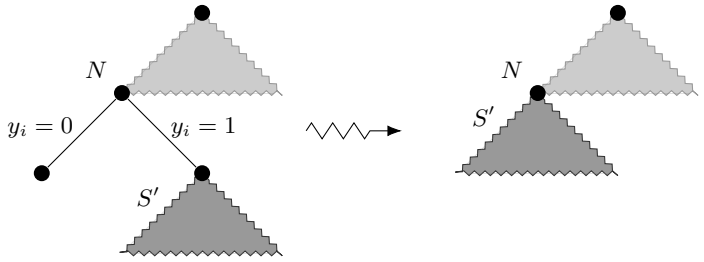

Fig. 1. The modifications to T to prove Part 1 of Claim 4. We see the case where N is labeled with y_i, the other case is analogous. The fixing of y_i does not contribute to the infeasibility of the leaves in the $y_i = 1$-branch and therefore can be removed.

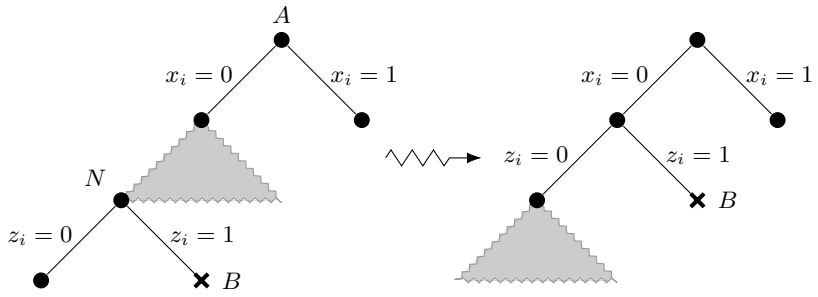

Fig. 2. The modifications to T to prove Part 2 of Claim 4. We see the case where N is labeled with z_i, the other case is analogous. The gray triangle is the $x_i = 0$-branch at A without the subtree rooted at N. The node B is infeasible because of the incompatible choices of z_i and x_i. The only leaf where fixed variables become free is B, thus all leaves remain infeasible.

with $y_i = 1$ (and $x_i = \frac{1}{2}$) and therefore the infeasibility of $P_\varphi(L)$ implies the infeasibility of $P_\varphi(L')$. The case where N is labeled with some z_i is analogous.

2. Assume a node N which is labeled with some y_i or z_i is a descendant, but not a child, of a node A labeled with x_i. Then by Part 1, either N is labeled with y_i and in the $x_i = 1$-branch at A, or N is labeled with z_i and in the $x_i = 0$-branch at A. In either case, we can find another branch-and-bound proof T' of the same size, in which the number of times a variable y_i or z_i is branched after, but not immediately after x_i is reduced by one, which suffices by induction. This can be achieved by reattaching certain subtrees of T as shown in Figure 2 and verifying that the leaves remain infeasible. ■

Finally, we assign to every node N labeled with y_i which has a descendant ruling out a point from $\mathcal{S}$ the following nodes:

1. If N's parent M is labeled with x_i:
   - Half of M and
   - N's infeasible child O, in which the branching decisions for x_i in M and for y_i in N contradict.

2. Otherwise, by Claim 4, there is a node M in the $y_i = 1$-branch at N which is labeled with x_i and we assign to N:
   - Half of M and
   - M's infeasible child O, in which the branching decisions for x_i in M and for y_i in N contradict.

We assign nodes to nodes labeled with some z_i with a descendant ruling out a point from $\mathcal{S}$ analogously. Note that in both cases, O cannot rule out a point from $\mathcal{S}$. We have assigned $\frac{3}{2}$ nodes to every N labeled with y_i or z_i, none of which are labeled with y_i or z_i and none of which can rule out a point from $\mathcal{S}$. Moreover, every node is assigned at most once: The only case in which a node could be assigned more than once appears if a node N labeled with x_i is assigned with $\frac{1}{2}$ to its two children and an ancestor each. Assume N is in the $y_i = 1$ branch of that ancestor. Then no node in its $x_i = 1$ branch rules out a node from $\mathcal{S}$ and N is not assigned to both of its children. All other cases are analogous.

In total, T contains at least

(a) $|\mathcal{S}|$ nodes ruling out a point from $\mathcal{S}$ by Claim 2,
(b) $2(|\mathcal{S}| - 1)$ nodes labeled with either y_i or z_i and have a descendant ruling out a point from $\mathcal{S}$ by Claim 3,
(c) and $\frac{3}{2} \cdot 2(|\mathcal{S}| - 1)$ nodes assigned to the nodes from (b).

Therefore T contains at least $6\,|\mathcal{S}| - 5 = 6 \cdot 2^{2n} \cdot \#\mathrm{SAT}(\varphi) - 5$ nodes. □

We note that Dey and Shah [13] independently from our work use a similar technique: They show exponential lower bounds on the length of branch-and-bound proofs for the lot-sizing problem by giving a large set $\mathcal{S}$ for which only a unique point can be ruled out by any leaf.

Finally, we can prove Theorem 6:

Proof (of Theorem 6). The intervals for $\mathcal{T}(P_\varphi)$ given by the bounds from Lemmas 7 and 8

$$[6 \cdot 2^{2n} \cdot \#\mathrm{SAT}(\varphi) - 5,\ 6 \cdot 2^{2n} \cdot \#\mathrm{SAT}(\varphi) - 5 + 6(2^n - \#\mathrm{SAT}(\varphi))]$$

are disjoint for different values of $\#\mathrm{SAT}(\varphi)$. Therefore, we can use $\mathcal{T}(P_\varphi)$ to infer $\#\mathrm{SAT}(\varphi)$. Since (P_φ) can be constructed in polynomial time and #SAT is #P-hard, $\mathcal{T}$ is #P-hard as well. □

If we use the left hand side of constraint $(P_\varphi\mathrm{b})$ as an objective function, the branch-and-bound proof constructed in Lemma 7 always branches on a variable which is fractional in the current unique LP-optimum. Thus, finding the length of a shortest proof which eliminates the current LP-optimum with each branching decision is also #P-hard.

Furthermore, the branching rule described in Lemma 7 is most-infeasible branching with the tie-breaking rule $x_1 > y_1 > z_1 > \cdots > x_{3n} > y_{3n} > z_{3n}$, because all fractional values of the unique optimum solution are $\frac{1}{2}$. Thus, if $\mathcal{M}(I)$ denotes the size of the branch-and-bound proof produced by variable branching using most-infeasible branching with the above tie-breaking rule, we have:

Corollary 9. *Computing $\mathcal{M}$ for binary programs is #P-hard.*

Using Theorem 6, it is not hard to derive hardness results similar to Theorems 1 and 2 for $\mathcal{M}$. To this end, note that the number of variables of (P_φ) is nine times the number of variables of φ.

6 Outlook

Firstly, we currently know $P^{\#SAT} \subseteq P^{\mathcal{T}} \subseteq$ PSPACE. It remains to determine the exact complexity of $\mathcal{T}$. Secondly, when trying to adapt Theorem 6 for the case of treelike resolution in the straightforward way, the analogous formula for (P_φ) suffers from exponential blow-up. It would be interesting to know if this can be remedied.

Acknowledgements. We thank three reviewers for their helpful comments that improved the presentation of the paper. This work was supported by the DFG, Project A01 in CRC/Transregio 154 (Project # 239904186).

References

1. Achterberg, T., Berthold, T.: Hybrid branching. In: van Hoeve, W., Hooker, J. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009. LNCS, vol. 5547, pp. 309–311. Springer (2009). https://doi.org/10.1007/978-3-642-01929-6_23
2. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33**(1), 42–54 (2005). https://doi.org/10.1016/j.orl.2004.04.002
3. Alekhnovich, M., Razborov, A.A.: Resolution is not automatizable unless W[P] is tractable. SIAM Journal on Computing **38**(4), 1347–1363 (2008). https://doi.org/10.1137/06066850X
4. Arora, S., Barak, B.: Computational complexity: a modern approach. Cambridge University Press (2009). https://doi.org/10.1017/CBO9780511804090
5. Atserias, A., Müller, M.: Automating resolution is NP-hard. Journal of the ACM **67**(5), 1–17 (2020). https://doi.org/10.1145/3409472
6. Basu, A., Conforti, M., Di Summa, M., Jiang, H.: Complexity of branch-and-bound and cutting planes in mixed-integer optimization – II. In: Singh, M., Williamson, D.P. (eds.) Integer Programming and Combinatorial Optimization. LNCS, vol. 12707, pp. 383–398. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73879-2_27
7. Beame, P., Pitassi, T.: Simplified and improved resolution lower bounds. In: Proceedings of 37th Conference on Foundations of Computer Science (FOCS). pp. 274–282. IEEE (1996). https://doi.org/10.1109/SFCS.1996.548486
8. Berthold, T., Salvagnin, D.: Cloud branching. In: Gomes, C., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, LNCS, vol. 7874, pp. 28–43. Springer (2013). https://doi.org/10.1007/978-3-642-38171-3_3

9. Bonet, M.L., Pitassi, T., Raz, R.: On interpolation and automatization for Frege systems. SIAM Journal on Computing **29**(6), 1939–1967 (2000). https://doi.org/10.1137/S0097539798353230
10. Cook, W., Coullard, C.R., Turán, G.: On the complexity of cutting-plane proofs. Discrete Applied Mathematics **18**(1), 25–38 (1987). https://doi.org/10.1016/0166-218X(87)90039-4
11. Dadush, D., Tiwari, S.: On the complexity of branching proofs. In: Proceedings of the 35th Computational Complexity Conference. Schloss Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.CCC.2020.34
12. Dey, S.S., Dubey, Y., Molinaro, M.: Branch-and-bound solves random binary IPs in polytime. In: Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 579–591. SIAM (2021). https://doi.org/10.1137/1.9781611976465.35
13. Dey, S.S., Shah, P.: Lower bound on size of branch-and-bound trees for solving lot-sizing problem. arXiv preprint arXiv:2112.03965 (2021)
14. Eickmeyer, K., Grohe, M., Grüber, M.: Approximation of natural W[P]-complete minimisation problems is hard. In: 2008 23rd Annual IEEE Conference on Computational Complexity. pp. 8–18. IEEE (2008). https://doi.org/10.1109/CCC.2008.24
15. Hendel, G., Anderson, D., Le Bodic, P., Pfetsch, M.E.: Estimating the size of branch-and-bound trees. INFORMS Journal on Computing (2021). https://doi.org/10.1287/ijoc.2021.1103, to appear
16. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. Journal of Computer and System Sciences **62**(2), 367–375 (2001). https://doi.org/10.1006/jcss.2000.1727
17. Jeroslow, R.G.: Trivial integer programs unsolvable by branch-and-bound. Mathematical Programming **6**(1), 105–109 (1974). https://doi.org/10.1007/BF01580225
18. Knuth, D.E.: Estimating the efficiency of backtrack programs. Mathematics of computation **29**(129), 122–136 (1975). https://doi.org/10.1090/S0025-5718-1975-0373371-6
19. Krajíček, J.: Proof complexity. Cambridge University Press (2019). https://doi.org/10.1017/9781108242066
20. Le Bodic, P., Nemhauser, G.: An abstract model for branching and its application to mixed integer programming. Mathematical Programming **166**(1), 369–405 (2017). https://doi.org/10.1007/s10107-016-1101-8
21. Sipser, M.: A complexity theoretic approach to randomness. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing. pp. 330–335 (1983). https://doi.org/10.1145/800061.808762
22. Stockmeyer, L.: On approximation algorithms for #P. SIAM Journal on Computing **14**(4), 849–861 (1985). https://doi.org/10.1137/0214060
23. Toda, S.: PP is as hard as the polynomial-time hierarchy. SIAM Journal on Computing **20**(5), 865–877 (1991). https://doi.org/10.1137/0220053
24. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science **8**(2), 189–201 (1979). https://doi.org/10.1016/0304-3975(79)90044-6
25. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing **8**(3), 410–421 (1979). https://doi.org/10.1137/0208032
26. Valiant, L.G., Vazirani, V.V.: NP is as easy as detecting unique solutions. In: Proceedings of the seventeenth annual ACM symposium on Theory of computing. pp. 458–463 (1985). https://doi.org/10.1016/0304-3975(86)90135-0