manuscript No.
(will be inserted by the editor)

Achieving Consistency with Cutting Planes

Danial Davarnia · Atefeh Rajabalizadeh · John Hooker

Received: date / Accepted: date

Abstract The primary role of cutting planes is to separate fractional solutions of the linear programming relaxation, which results in tighter bounds for pruning the search tree and reducing its size. Bounding, however, has an *indirect* impact on the size of the search tree. Cutting planes can also reduce backtracking by excluding inconsistent partial assignments that occur in the course of branching, which directly reduces the tree size. A partial assignment is inconsistent with a constraint set when it cannot be extended to a full feasible assignment. The constraint programming community has studied consistency extensively and used it as an effective tool for the reduction of backtracking. We extend this approach to integer programming by defining concepts of consistency that are useful in a branch-and-bound context. We present a theoretical framework for studying these concepts, their connection with the convex hull and their power to exclude infeasible partial assignments. We introduce a new class of cutting planes that target achieving consistency rather than improving dual bounds. Computational experiments on both synthetic and benchmark instances show that the new class of cutting planes can significantly outperform classical cutting planes, such as disjunctive cuts, by reducing the size of the search tree and the solution time. More broadly, we suggest that consistency concepts offer a new perspective on integer programming that can lead to a better understanding of what makes cutting planes work when used in branch-and-bound search.

Keywords Cutting planes · Branch-and-bound · Consistency · Integer programming · Constraint programming.

1 Introduction

Cutting planes have long been used in integer programming to accelerate branch-and-bound (B&B) search by excluding fractional solutions of the linear programming (LP) relaxation. This exclusion

D. Davarnia
Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA, USA
E-mail: davarnia@iastate.edu

A. Rajabalizadeh
Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA, USA
E-mail: alizadeh@iastate.edu

J. Hooker
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA,
E-mail: jh38@andrew.cmu.edu

improves the dual bound of the integer program, which is exploited through pruning rules to reduce the size of the search tree. The effectiveness of cutting planes is measured by the amount of the integrality gap that is closed by adding them to the model. This metric, however, is not representative of the amount of reduction in the size of the search tree. As a consequence, in many practical instances, a great deal of effort is invested into generating cutting planes that yield significant bound improvement at a node, but fail to have any effect on the tree size. This computational gap has been recognized [34] as one of the most important unresolved issues in regards to B&B algorithms.

Yet there is another way that cutting planes can reduce the B&B tree size: by eliminating backtracking directly. Backtracking is an expensive operation in B&B that allocates unnecessary time and memory due to creation of redundant nodes. Each node of the tree corresponds to a partial assignment to the problem variables, defined by the branches taken so far. That partial assignment is *consistent with* the constraint set if it can be extended to a feasible full assignment. If the partial assignment is inconsistent, the search must eventually backtrack to a node above the current one.

Cutting planes can help reduce backtracking by making it easier to recognize inconsistent partial assignments. They do so by helping to achieve *consistency for the constraint set as a whole*, a concept from the field of constraint programming (CP). A constraint set is *consistent* in the CP sense if any partial assignment that is inconsistent with it already violates some individual constraint in the set. Thus consistency is a much stronger condition on constraint sets than feasibility. Consistency maintenance allows one to identify inconsistent partial assignments early in the search process and thereby avoid backtracking. Since full consistency is rarely obtained due to the computational challenge it poses, various weaker forms of consistency have been defined that reduce backtracking to some extent. These weaker forms are generally achieved by adding certain implied constraints to the constraint set. We will generalize the CP concept of consistency so that cutting planes can achieve partial consistency in the context of integer programming.

The concept of consistency has been extensively studied in CP, where it is used as an effective tool to reduce backtracking [38]. To our knowledge, this concept has never been developed in the integer programming (IP) literature, even though consistency is closely related to backtracking in a B&B search, and even though cutting planes can achieve some types of consistency even when they do not separate fractional LP solutions. Consistency properties are key to the design of cutting planes that directly target reduction of the tree size, in contrast to the traditional role of cutting planes that targets bound improvement.

We therefore undertake an explicit study of the potential role of consistency concepts in IP, and in particular, how cutting planes can achieve consistency. This undertaking was initiated in [13], which defined a concept of *LP-consistency* for 0–1 programming problems by drawing an analogy with consistency in CP. It is shown in that paper that backtracking can be eliminated by checking the feasibility of LP relaxations for LP-consistent sets at each node of the search tree. The authors propose how to achieve a variant of LP-consistency with a specialized lift-and-project procedure. Yet it is unclear how this method can be used in practice.

The contributions of the present paper are as follows. We extend the work in [13] by providing a deeper analysis of consistency in IP that can point the way to practical application. We develop a theoretical framework to measure the power of LP-consistency and its connection to the conventional notions of convex hull and cutting planes. We design cutting plane algorithms that can achieve partial consistency and present computational experiments to show their impact on reducing the size of the B&B tree when compared to classical cutting plane methods. The new class of cutting planes separate inconsistent partial assignments, which is a fundamentally different task from that of the traditional cutting planes that separate infeasible fractional solutions. More broadly, we suggest that consistency concepts provide a novel perspective on IP that can lead to a better understanding of what makes cutting plane methods effective.

Due to the nature of the methodology developed in this paper as a bridge between CP and IP, the content of the paper is organized and presented in such a way that it is accessible to both IP and CP audience. As a result, there will be several illustrating examples and cross discussions from both perspectives within each section. The remainder of the paper is organized as follows. After a brief review of related work, we provide basic definitions and background in Section 3, where we define the relevant CP concepts and LP-consistency and summarize their connections with the resolution method of logical inference. In Section 4, we investigate the power of various forms of consistency as means for excluding infeasible partial assignments. We further discuss the connection between LP-consistency and classical IP notions such as convex hull.

At this point we introduce, in Section 5, a new concept of partial consistency and apply it in both CP and IP contexts. We propose a modified version of the reformulation and linearization technique (RLT) that achieves partial LP-consistency without generating resolvents. We define the rank of partial LP-consistency as a measure of its strength analogous to RLT ranks. As one of the key results of the paper, we show that partial LP-consistency of rank r can be obtained by applying substantially fewer than r levels of the RLT hierarchy. The section concludes by formulating a cut-generating LP (CGLP) that generates only the cutting planes that are needed to avoid inconsistent partial assignments in a branching context. It presents an algorithm that can significantly reduce the number of CGLPs needed to cut off an infeasible partial assignment using classical RLT. Section 6 reports preliminary computational experiments showing that the cuts that target consistency can substantially outperform the traditional disjunctive cuts in reducing the tree size and solution time. Concluding remarks are offered in Section 7.

2 Related Work

B&B methods have served for decades as the primary general-purpose approach to solving IP problems [29]. Various strategies have been developed to reduce the size of the B&B tree and thereby accelerate the search. Some studies focus on algorithmic choices. For instance, node selection strategies can reduce the number of nodes evaluated [31]. As reported in [2], branching strategies play a key role in determining the size of the branching tree, such as reliability branching (an extension of branching based on pseudocosts) initialized by strong branching, as well as branching based on conflict scores, inference scores, and pseudo reduced costs. Recently, machine learning algorithms have been applied to node selection and branching [7]. Other computational algorithms, such as parallel branching, have been proposed to improve the gap closure in a given amount of time [30]. As noted in [2], combinations of the above strategies are used in the state-of-the-art solvers to accelerate the solution process.

Other studies focus on tree structure and develop strategies to avoid exploring nodes that do not contain useful information. At the heart of such strategies lie *pruning rules* that target certain nodes of the tree whose subsequent nodes (included in the subtree rooted at these nodes) can be ignored altogether. Two main pruning rules have been proposed. The *primal-dual bound* rule prunes the nodes whose associated LP relaxation has an optimal value no better than the objective value of the incumbent integral solution. The *infeasibility* rule prunes nodes whose LP relaxation is infeasible.

The importance of these pruning rules gave rise to two of the most intensely studied topics in IP, cutting plane theory and heuristic algorithm design. Cutting planes help improve the dual bound by tightening the LP relaxation, whereas heuristic methods seek to find feasible solutions that provide a better primal bound [5, 6, 40]. While many studies in the IP literature focus on strengthening the primal-dual bound pruning rule, infeasibility rules have received less attention. Our paper aims to fill this gap by proposing an infeasibility pruning rule based on identifying inconsistent partial assignments.

Constraint programming has an extensive literature that deals with the infeasibility rule, surveyed in [38]. Classical CP solvers explore the solution space with a search tree similar to the B&B tree of IP, but without using an LP relaxation. As a result, there is no primal-dual bound pruning rule, leaving the infeasibility rule as the primary pruning strategy. For this reason, infeasibility pruning is intensively studied under the framework of consistency. Our paper adapts this approach to IP.

An alternative approach to dealing with infeasibility is conflict analysis, which has a long history in artificial intelligence and the satisfiability (SAT) community. It traces back at least to Gaschnig's (1977) concepts of backjumping and backchecking [15, 16] and has been generalized to other forms of dependency-directed backtracking, such as dynamic backtracking [17, 18] and conflict-driven clause learning in SAT solvers [8]. Learned clauses are themselves a special case of logic-based Benders cuts [24, 27, 28]. Conflict analysis is, in fact, a learning process. When branching encounters an infeasible partial assignment, conflict analysis identifies the specific variable assignments that create the conflict and generates a constraint that prevents these assignments in subsequent branching. In recent years, the idea has been extended to pseudo-Boolean optimization and integer programming as well [1, 9, 24, 26, 35, 39, 43]. Consistency maintenance differs fundamentally from conflict analysis because it does not analyze conflicts. Rather than *react* to infeasibility discovered in the course of a tree search, it *anticipates* and thereby avoids infeasibility by drawing out some of the latent implications of the constraint set. This requires an entirely different type of analysis.

The connection between consistency and resolution is studied in [24, 26] and elaborated in [13] as well as in the present paper. The relationship between resolution and cutting planes is first remarked in [11, 42] and further developed in [13, 20, 21, 22, 24, 26]. Further connections between cutting planes and CP are surveyed in [23, 25]. LP-consistency is introduced in [13] by reinterpreting consistency concepts in CP in terms of a problem relaxation and projection [27], and by replacing the relaxations used in CP with the LP relaxation. [13] also observes that cutting planes that achieve consistency can be stronger than facet-defining cuts, in the sense that they cut off a larger number of infeasible 0–1 partial solutions. [13] proves some basic properties of LP-consistency and proposes a modified lift-and-project method for LP-consistency maintenance. The present paper explores the relationship between consistency and IP more deeply and develops the much more efficient consistency maintenance method based on RLT and CGLPs that is mentioned above.

3 Definitions and Background.

In this section, we introduce basic terminology and background adapted from [13] that are required for later developments.

3.1 Consistency in constraint programming

The *domain* D_j of a variable x_j is the set of values that can be assigned to x_j. A *constraint* is an object that *contains* some set $\{x_1, \dots, x_k\}$ of variables, such that any given assignment of values to $(x_1, \dots, x_k)$ either *satisfies* or *violates* that constraint. Thus a constraint is satisfied or violated only when all of its variables have been assigned values. A *constraint set* $\mathcal{S}$ is a set of individual constraints and variable domains. An assignment to x satisfies a constraint set $\mathcal{S}$ when it satisfies all constraints in $\mathcal{S}$. A list of symbols defined hereafter appears in Table 3.1.

Let x_J be the tuple containing the variables in $\{x_j \mid j \in J\}$ for $J \subseteq N = \{1, \dots, n\}$. A *partial assignment* to x is a 0–1 assignment of values to x_J for some $J \subseteq N$. We can now define a consistent partial assignment and a consistent constraint set.

Table 3.1: List of symbols

x_J	tuple of variables x_j for $j \in J$
N	the set $\{1, \dots, n\}$
J_k	the set $\{1, \dots, k\}$
D_j	domain of x_j
D_J	cartesian product of D_j for $j \in J$
$\mathcal{S}_J$	set of constraints in $\mathcal{S}$ that contain only variables in x_J
S_{LP}	LP relaxation of 0–1 constraint set $\mathcal{S}$
$\bar{C}$	0–1 inequality that represents logical clause C
$\mathcal{S}_{\text{C}}$	set of clausal inequalities implied by individual constraints of $\mathcal{S}$
$C(J^+, J^-)$	logical clause containing literals x_j for $j \in J^+$ and $\neg x_j$ for $j \in J^-$
$\mathcal{C}^{(r)}$	set of clauses obtained after applying r rounds of resolution to $\mathcal{C} = \mathcal{C}^{(0)}$

Definition 3.1 Given a constraint set $\mathcal{S}$, a partial assignment $x_J = v_J$ is *consistent with* $\mathcal{S}$ if $\mathcal{S} \cup \{x_J = v_J\}$ is feasible. The constraint set $\mathcal{S}$ is *consistent* if every partial assignment to x that violates no constraint in $\mathcal{S}$ is consistent with $\mathcal{S}$.

In words, a partial assignment is consistent with $\mathcal{S}$ if it can be extended to a feasible solution of $\mathcal{S}$. Furthermore, $\mathcal{S}$ is consistent when every inconsistent partial assignment is explicitly ruled out by some individual constraint in the set. It follows from these definitions that a consistent constraint set $\mathcal{S}$ can be solved by branching without backtracking, regardless of the branching order. A given node of the search tree corresponds to a partial assigment $x_J = v_J$. If setting $x_j = v_j$ for $j \notin J$ violates no constraint in $\mathcal{S}$, we know that the partial assignment $x_{J\cup\{j\}} = v_{J\cup\{j\}}$, corresponding to a child node created from the current node, is consistent with $\mathcal{S}$, and the search will not backtrack to the current node.

Unfortunately, achieving full consistency is generally hard. For this reason, various weaker forms of consistency has been developed in CP with the goal of making it more practical. We give a brief introduction of these forms, the most widely used of which is domain consistency.

Definition 3.2 A constraint set $\mathcal{S}$ is *domain consistent* if the partial assignment $x_j = v_j$ is consistent with $\mathcal{S}$ for all $j \in N$ and all $v_j \in D_j$.

Achieving domain consistency tends to accelerate search because it reduces the size of variable domains, so that it is necessary to branch on fewer values.

Definition 3.3 A constraint set $\mathcal{S}$ is *k-consistent* if, given any $J \subset N$ with $|J| = k-1$ and any $j \notin J$, a partial assignment $x_J = v_J$ that violates no constraint in $\mathcal{S}$ can be extended to an assignment $x_{J\cup\{j\}} = v_{J\cup\{j\}}$ that violates no constraint in $\mathcal{S}$, where $v_j \in D_j$. Set $\mathcal{S}$ is *strongly k-consistent* if it is i-consistent for $i = 1, \dots, k$.

Strong k-consistency suffices to avoid backtracking if there is limited coupling of variables [14]. More relevant to our purposes is a weaker form of k-consistency, namely sequential k-consistency, which is defined with respect to the predetermined branching order $x_1, \dots, x_n$. Backtracking is avoided when $\mathcal{S}$ is sequentially k-consistent for $k = 1, \dots, n$. Let $J_k = \{1, \dots, k\}$.

Definition 3.4 A constraint set $\mathcal{S}$ is *sequentially k-consistent* if for every partial assignment $x_{J_{k-1}} = v_{J_{k-1}}$ that violates no constraint in $\mathcal{S}$, there is an extended assignment $x_{J_k} = v_{J_k}$ that violates no constraint in $\mathcal{S}$, where $v_k \in D_k$.

Example 3.1 Suppose that $\mathcal{S}$ is the 0–1 constraint set containing

$$x_1 + 2x_2 + 3x_3 \leq 3 \tag{3.1a}$$

$$2x_2 - x_3 \leq 1. \tag{3.1b}$$

Set $\mathcal{S}$ is not consistent because, for instance, the partial assignment $x_2 = 1$ violates no constraint in $\mathcal{S}$ but is inconsistent with $\mathcal{S}$. In addition, $\mathcal{S}$ is not 3-consistent because the partial assignment $(x_2, x_3) = (1, 1)$ violates no constraint in $\mathcal{S}$, but both of its extensions $(x_1, x_2, x_3) = (0, 1, 1)$ and $(x_1, x_2, x_3) = (1, 1, 1)$ violate (3.1a). However, $\mathcal{S}$ is sequentially k-consistent for all $k = 1, 2, 3$, implying that there is no backtracking for the branching order x_1, x_2, x_3.

3.2 Consistency and resolution

We will say that $\mathcal{S}$ is a *0–1 constraint set* if it has the form $\mathcal{S} = \{Ax \leq b,\ x \in \{0,1\}^n\}$. A general method for achieving consistency in 0–1 constraint sets is *resolution*, an inference method in propositional logic. We present the basics of this technique.

A logical *clause* has the form

$$C(J^+, J^-) = \Big(\bigvee_{j \in J^+} x_j \Big) \vee \Big(\bigvee_{j \in J^-} \neg x_j \Big)$$

where $J^+, J^- \subseteq N$ and $J^+ \cap J^- = \emptyset$. Terms of the form x_j and $\neg x_j$ are *literals*. Clause $C(J_1^+, J_1^-)$ *logically implies* (*absorbs*) $C(J_2^+, J_2^-)$ if and only if $J_1^+ \subseteq J_2^+$ and $J_1^- \subseteq J_2^-$. A clause $C(J^+, J^-)$ is *represented* by the 0–1 inequality

$$\sum_{j \in J^+} x_j + \sum_{j \in J^-} (1 - x_j) \geq 1.$$

We will denote by $\bar{C}$ the 0–1 inequality that represents clause C. Accordingly, given a clause set $\mathcal{C}$, we denote by $\bar{\mathcal{C}}$ the set of inequalities representing clauses in $\mathcal{C}$. A 0–1 inequality is *clausal* when it represents a clause.

Two clauses between which only one variable x_i changes sign have a *resolvent* that contains all the literals in either clause except x_i and $\neg x_i$. Thus given clauses $C(J_1^+ \cup \{i\}, J_1^-)$ and $C(J_2^+, J_2^- \cup \{i\})$, where $J_1^+ \cup J_2^+$ and $J_1^- \cup J_2^-$ are disjoint, their resolvent is $C(J_1^+ \cup J_2^+, J_1^- \cup J_2^-)$. For example, the resolvent of $x_1 \vee x_3$ and $\neg x_2 \vee \neg x_3$ is $x_1 \vee \neg x_2$.

The *resolution* algorithm applies resolution repeatedly until no further nonredundant resolvents can be obtained. Thus given clause set $\mathcal{C} = \mathcal{C}^{(0)}$, we let $\mathcal{C}^{(k)}$ be the clause set obtained by adding to $\mathcal{C}^{(k-1)}$ all its resolvents that are not absorbed by a clause in $\mathcal{C}^{(k-1)}$. The resolution algorithm generates $\mathcal{C}^{(0)}, \mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(k)}$ until $\mathcal{C}^{(k-1)} = \mathcal{C}^{(k)}$.

It is shown in [26] that $\mathcal{C}^{(k)}$ is consistent for some finite k. In other words, the resolution algorithm achieves consistency for any clause set. This procedure, however, is computationally expensive since the size of $\mathcal{C}^{(k)}$ can grow exponentially as k increases. To achieve the weaker and more practical forms of consistency defined above, resolution can be applied to a suitable subset of clauses. We refer the reader to [26] for details.

Resolution can also achieve consistency for a general 0–1 constraint set $\mathcal{S}$ [26]. Let the *implied clause set* of $\mathcal{S}$ be the set of clauses implied by *individual inequalities* in $\mathcal{S}$, and let $\mathcal{S}_C$ be the set of inequalities representing the implied clauses. A simple and efficient algorithm is given in [19, 26] for deriving the implied clauses. We can achieve consistency for $\mathcal{S}$ by applying the resolution algorithm to its implied clause set. We need only add to $\mathcal{S}$ the inequalities representing all clauses generated by the resolution algorithm.

Example 3.2 To achieve full consistency for Example 3.1, we first derive the implied clause set, which contains $\neg x_1 \vee \neg x_3$, $\neg x_2 \vee \neg x_3$, and $\neg x_2 \vee x_3$. The first two clauses are implied by (3.1a) and the third by (3.1b). One nonredundant resolvent can be generated from the implied clause set, namely $\neg x_2$. It is easy to verify that $\mathcal{S} \cup \{x_2 \leq 0\}$ is consistent.

3.3 LP-consistency in integer programming

The purpose of consistency is to help detect whether a partial assignment is infeasible at a node of the branching tree. If the constraint set $\mathcal{S}$ is consistent, it suffices to check whether the partial assignment is consistent with an appropriate *relaxation* of $\mathcal{S}$. In CP, the relaxation consists of the constraints in $\mathcal{S}$ that contain only variables in the partial assignment. While consistency with this relaxation is easy to check, the relaxation itself is weak, as it does not capture interactions between multiple constraints. A stronger relaxation is proposed in [13] that is natural to integer programming and yet easily checked. It is the LP relaxation, which gives rise to LP-consistency. Denote by $\mathcal{S}_{\text{LP}} = \{Ax \leq b,\ x \in [0,1]^n\}$ the LP relaxation of a 0–1 constraint set $\mathcal{S}$.

Definition 3.5 A 0–1 constraint set $\mathcal{S}$ is LP-consistent if any 0–1 partial assignment that is consistent with $\mathcal{S}_{\text{LP}}$ is consistent with $\mathcal{S}$.

In other words, $\mathcal{S}$ is LP-consistent if any partial assignment $x_J = v_J$ for which $S_{\text{LP}} \cup \{x_J = v_J\}$ is feasible is one for which $\mathcal{S} \cup \{x_J = v_J\}$ is feasible. LP-consistency can eliminate backtracking if we solve one or two LPs before each branch. Suppose, for example, that the branches taken so far impose the partial assignment $x_J = v_J$, and we wish to branch on x_j next. We can set $x_j = v_j$ if the LP problem $\mathcal{S}'_{\text{LP}} = \mathcal{S}_{\text{LP}} \cup \{x_{J\cup\{j\}} = v_{J\cup\{j\}}\}$ is feasible, where $v_j = 0$ or $v_j = 1$. Since $\mathcal{S}$ is LP-consistent, we know that $x_{J\cup\{j\}} = v_{J\cup\{j\}}$ in some feasible solution of $\mathcal{S}$, and no backtracking to the current node will occur. If $\mathcal{S}'_{\text{LP}}$ is infeasible for both $v_j = 0$ and $v_j = 1$, the 0–1 problem $\mathcal{S}$ is infeasible, and branching terminates.

Example 3.3 Let $\mathcal{S}^1 = \{x \in \{0,1\}^2,\ 2x_1 + 4x_2 \leq 5,\ 7x_1 - 2x_2 \leq 6\}$, illustrated in Fig. 3.1. Clearly, $\mathcal{S}^1$ is not LP-consistent, because the partial assignment $x_1 = 1$ is consistent with $\mathcal{S}^1_{\text{LP}}$ but not with $\mathcal{S}^1$. As a result, under the branching order x_1, x_2, a B&B process that sets $x_1 = 1$ will backtrack to set $x_1 = 0$. In contrast, the constraint set $\mathcal{S}^2 = \{x \in \{0,1\}^2,\ 7x_1 - 2x_2 \leq 6,\ x_1 + x_2 \leq 1\}$ is LP-consistent and avoids backtracking. Note, however, that $\mathcal{S}^2$ is not consistent in the traditional sense because $x_1 = 1$ violates no constraints but is inconsistent with $\mathcal{S}^2$. Thus LP-consistency does not imply traditional consistency but can nonetheless avoid backtracking.

Since LP-consistency is a weaker property compared to traditional consistency, achieving it is potentially cheaper. Sequential LP k-consistency is introduced in [13], in analogy with sequential k-consistency, to make consistency maintenance even more practical.

Definition 3.6 A constraint set $\mathcal{S}$ is sequentially LP k-consistent if for every 0–1 partial assignment $x_{J_{k-1}} = v_{J_{k-1}}$ that is consistent with $\mathcal{S}_{\text{LP}}$, there is an extended 0–1 assignment $x_{J_k} = v_{J_k}$ that is consistent with $\mathcal{S}_{\text{LP}}$.

Sequential LP k-consistency for $k = 1, \ldots, n$ avoids backtracking when the branching order is $x_1, \ldots, x_n$. In [13] the authors propose a lift-and-project technique to achieve sequential LP k-consistency. This technique, however, is not computationally efficient since it requires the addition of an exponential number of constraints to the original set as k increases. In this paper, we build upon this idea by introducing different forms of LP-consistency and investigating practical cutting plane algorithms to achieve them.

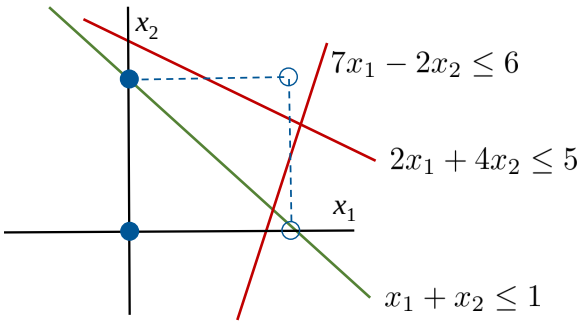

Fig. 3.1: Illustration of Examples 3.3 and 4.2.

4 Power of LP-consistency

A natural question in the consistency context is how powerful LP-consistency is, with respect to avoiding inconsistent partial assignments, when compared to classical consistency. The key to such a comparison is viewing resolvents as cutting planes. This perspective also helps us to understand that cutting planes have an important impact on increasing consistency in addition to their primary role of tightening relaxations. While the literature is mainly focused on the latter role of cutting planes, we aim to further explore the former role in this paper.

4.1 Resolution-based consistency in clause sets

It is well-known [11, 20, 42] that if a resolvent and its parents are converted to inequalities, the resolvent is a rank 1 Chvátal-Gomory (C–G) cut derived from the parents and $x \in [0,1]^n$. We begin the current section by generalizing this connection between rank 1 C-G inequalities and resolvents to a link between them and a particular type of resolution proof. In what follows, for the sake of uniformity of presentation, we refer to a rank r C–G inequality as one that contains lower rank classes, with r representing an upper bound for the derivation effort. For instance, a rank 1 C–G inequality contains the case where the inequality is obtained by aggregating the original constraints without performing the rounding step.

Given a clause set $\mathcal{C}$, we say that a clause C has a resolution proof if $C \in \mathcal{C}^{(k)}$ for some k. A resolution proof consists of a sequence of clauses $C_1, \ldots, C_m = C$ such that C_m is the resolvent of two previous clauses in the sequence, and each C_i for $i = 1, \ldots, m-1$ either belongs to $\mathcal{C}$ or is the resolvent of two earlier clauses in the sequence. Similarly, we say that a clause C_m has an *input proof* from $\mathcal{C}$ if it has a resolution proof in which at least one of the parents of each resolvent in the sequence belongs to $\mathcal{C}$ [10]. Every implied clause of $\mathcal{C}$ has a resolution proof, but not necessarily an input proof [36, 37]. The following relation between input proofs and C–G inequalities is proved in [21]. It is convenient to regard any inequality already in $\mathcal{S}$ as a rank 1 C-G inequality, and to regard any clause in $\mathcal{C}$ as having an input proof (of length 0) of itself.

Proposition 4.1 *A clause C has an input proof from clause set $\mathcal{C}$ if and only if $\bar{C}$ is a rank 1 C–G inequality for $\bar{\mathcal{C}}$.*

A clause obtained from a general resolution proof may, however, have a Chvátal rank greater than 1, as illustrated in the next example.

Example 4.1 Consider the set $\mathcal{C}$ of clauses on the left below:

$$\begin{array}{ll} C_1 : \neg x_1 \vee \neg x_2 \vee \neg x_3 & C_6 : \neg x_1 \vee \neg x_2, \text{ from } C_1, C_2 \\ C_2 : \neg x_1 \vee \neg x_2 \vee x_3 & C_7 : \neg x_1 \vee x_2, \text{ from } C_3, C_4 \\ C_3 : \neg x_1 \vee x_2 \vee \neg x_3 & C_8 : \neg x_1, \quad \text{from } C_6, C_7 \\ C_4 : \neg x_1 \vee x_2 \vee x_3 & C_9 : \neg x_2, \quad \text{from } C_5, C_6 \\ C_5 : x_1 \vee \neg x_2 & \end{array}$$

The implied clause $C_8 = \neg x_1$ has a resolution proof (but not an input proof) from $\mathcal{C}$ with the sequence C_1, C_2, C_3, C_4, C_6, C_7, C_8, where the parents of each resolvent are shown above. It can be verified that the inequality $x_1 \leq 0$ representing C_8 is a C–G inequality of rank 2. The implied clause $C_9 = \neg x_2$ has an input proof from $\mathcal{C}$ with the sequence C_1, C_2, C_6, C_5, C_9, and the inequality $x_2 \leq 0$ is a C–G inequality of rank 1, as prescribed by Proposition 4.1.

LP-consistency elucidates the relationship between resolution in general and input proofs in particular, because it is a necessary and sufficient condition for the two types of proof to coincide. To prove this result, we use the following proposition and its corollary, both proved in [13].

Proposition 4.2 *Given a 0–1 constraint set $\mathcal{S}$, a 0-1 partial assignment is inconsistent with $\mathcal{S}_{\text{LP}}$ if and only if it violates some rank 1 C-G clausal inequality for $\mathcal{S}_{\text{LP}}$.*

Corollary 4.1 *A 0–1 constraint set $\mathcal{S}$ is LP-consistent if and only if any clausal inequality implied by $\mathcal{S}$ is a rank 1 C–G inequality for $\mathcal{S}_{\text{LP}}$.*

The desired result ensues.

Corollary 4.2 *Given a clause set $\mathcal{C}$, $\bar{\mathcal{C}}$ is LP-consistent if and only if any clause that has a resolution proof from $\mathcal{C}$ also has an input proof from $\mathcal{C}$.*

Proof Since resolution is a complete inference method for clauses, a clause has a resolution proof from $\mathcal{C}$ if and only if it is implied by $\mathcal{C}$. It therefore follows from Corollary 4.1 that $\bar{\mathcal{C}}$ is LP-consistent if and only if any clause C that has a resolution proof from $\mathcal{C}$ is such that $\bar{C}$ is a rank 1 C–G inequality for $\bar{\mathcal{C}}$. But Proposition 4.1 states that $\bar{C}$ is a rank 1 C–G inequality for $\bar{\mathcal{C}}$ if and only if C has an input proof from $\mathcal{C}$. The proposition follows. □

The above result leads to a direct comparison between the power of LP-consistency and classical consistency for clause sets.

Corollary 4.3 *Let $\mathcal{D}$ be the set of clauses that have an input proof from clause set $\mathcal{C}$. Then $\bar{\mathcal{C}}$ is LP-consistent if and only if $\mathcal{D}$ is consistent.*

Corollary 4.3 shows a computational advantage for LP-consistency in identifying inconsistent partial solutions of $\mathcal{S}$. Inconsistent partial assignments that can be detected by generating all input proofs from $\mathcal{S}_{\text{C}}$ can be detected simply by checking whether they are consistent with the LP relaxation of $\mathcal{S}_{\text{C}}$.

At this point we can observe that cutting planes can help achieve LP-consistency. Corollary 4.2, for example, states that sufficiently many rank 1 C-G cuts achieve LP-consistency. Thus cutting planes can reduce backtracking by excluding inconsistent partial solutions as well as by cutting off fractional solutions of the LP relaxation.

4.2 Resolution-based consistency in 0–1 programming

In the previous section, we showed the relation between LP-consistency and consistency for clause sets. In this section, we focus on general 0-1 constraint sets. Consider a constraint set $\mathcal{S}$ and the set $\mathcal{S}_{\mathrm{C}}$ of inequalities representing the implied clause set of $\mathcal{S}$. We address the question of how LP-consistency for $\mathcal{S}$ compares to that of $\mathcal{S}_{\mathrm{C}}$. This is an important question, as $\mathcal{S}_{\mathrm{C}}$ is the natural gateway to studying consistency by means of resolution methods. We aim to understand how one can avoid the middle step of generating resolvents (which can be very expensive) by achieving LP-consistency.

We note first that we cannot check $\mathcal{S}$ for LP-consistency by checking $\mathcal{S}_{\mathrm{C}}$ for LP-consistency. The LP-consistency of neither implies that of the other, as shown by the next example.

Example 4.2 Consider the constraint set $\mathcal{S}^1$ of Example 3.3, which we noted is not LP-consistent. Yet $\mathcal{S}^1_C = \{x \in \{0,1\}^2,\ x_1 + x_2 \le 1,\ x_1 - x_2 \le 0\}$ is LP-consistent. Now suppose $\mathcal{S}^2$ is the set of 0–1 constraints on the left below, so that $\mathcal{S}^2_C$ contains the 0–1 constraints on the right.

$$\begin{array}{ll} x_1 + x_2 + x_3 \le 1 & x_1 + x_2 \le 1 \\ -x_1 - x_2 - x_3 + 2x_4 \le 0 & x_1 + x_3 \le 1 \\ & x_2 + x_3 \le 1 \\ & x_4 - x_1 - x_2 \le 0 \\ & x_4 - x_1 - x_3 \le 0 \\ & x_4 - x_2 - x_3 \le 0 \end{array}$$

$\mathcal{S}^2$ is LP-consistent, while $\mathcal{S}^2_C$ is not LP-consistent because the partial assignment $x_4 = 1$ is inconsistent with its LP relaxation.

While LP-consistency for $\mathcal{S}$ and for $\mathcal{S}_{\mathrm{C}}$ do not imply one another, consistency of $\mathcal{S}_{\mathrm{C}}$ implies LP-consistency of S. To establish this fact, we first show that the constraints of $\mathcal{S}_{\mathrm{C}}$ can be obtained as rank 1 C–G inequalities from $\mathcal{S}$.

Proposition 4.3 *Consider a 0–1 inequality $ax \le \beta$ that is satisfied by at least one point $x \in \{0,1\}^n$. Then any clausal inequality implied by $ax \le \beta$ is a rank 1 C–G inequality for $\{ax \le \beta,\ x \in [0,1]^n\}$.*

Proof Assume without loss of generality that $a_i \ge 0$ for all $i \in N$, since otherwise x_i can be replaced by $1 - x_i$. We also have $\beta \ge 0$, since at least one 0–1 point satisfies $ax \le \beta$. Let $C = C(I^+, I^-)$ be any clause implied by $ax \le \beta$ for some pair of disjoint subsets I^+ and I^- of N. Clause C is falsified by the partial assignment with components $x_i = 0$ for $i \in I^+$ and $x_i = 1$ for $i \in I^-$, a partial assignment that violates $ax \le \beta$ because $ax \le \beta$ implies C. As a result, we have $\sum_{i \in I^-} a_i > \beta$. Define $\sigma = \sum_{i \in I^-} a_i$, which is positive since $\beta \ge 0$. It suffices to show that the inequality

$$\bar{C} = -\sum_{i \in I^+} x_i + \sum_{i \in I^-} x_i \le |I^-| - 1, \tag{4.1}$$

is a rank 1 C–G inequality for $\{ax \le \beta,\ x \in [0,1]^n\}$. We show this by taking a weighted sum of $ax \le \beta$ with weight $1/\sigma$, inequalities $-x_i \le 0$ for $i \in N \setminus (I^- \cup I^+)$ with weights a_i/σ, inequalities $-x_i \le 0$ for $i \in I^+$ with weights $1 + a_i/\sigma$, and inequalities $x_i \le 1$ for $i \in I^-$ with weights $1 - a_i/\sigma$. This yields (4.1) with right-hand side $\le |I^-| - 1 + \beta/\sigma$. Since $0 \le \beta/\sigma < 1$, we can round down the right-hand side to obtain (4.1) as a rank 1 C–G inequality. □

It follows from Corollary 4.1 and Proposition 4.3 that if an inconsistent partial solution of $\mathcal{S}$ violates a constraint in $\mathcal{S}_{\mathrm{C}}$, then it is inconsistent with $\mathcal{S}_{\mathrm{LP}}$. This shows that an LP-consistent $\mathcal{S}$ has at least as much power as a consistent $\mathcal{S}_{\mathrm{C}}$ for excluding inconsistent partial assignments.

Corollary 4.4 *A constraint set $\mathcal{S}$ is LP-consistent if $\mathcal{S}_{\mathrm{C}}$ is consistent.* □

To complete the hierarchy of comparisons between different consistency concepts, we next provide the relation between LP-consistency for $\mathcal{S}$ and input proofs from $\mathcal{S}_{\mathrm{C}}$.

Proposition 4.4 *Consider a 0–1 constraint set $\mathcal{S}$. Assume that clause C_1^k is obtained through a sequence of $k \geq 0$ resolutions as shown in Fig. 4.1, where $\bar{C}_1^0, \bar{C}_2^0 \in \mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}$ implies $\bar{C}_2^i$ for $i \in \{1, \ldots, k-1\}$. Let $\widehat{\mathcal{S}}$ be the set obtained by adding $\bar{C}_2^i$ to $\mathcal{S}$ for $i = 0, \ldots, k-1$. Then any 0–1 partial assignment that violates C_1^k is inconsistent with $\widehat{\mathcal{S}}_{\mathrm{LP}}$.*

Proof We prove the result by induction on k. Let $\widehat{\mathcal{S}}^0 = \mathcal{S}$ and $\widehat{S}^k = \mathcal{S} \cup \{\bar{C}_2^0, \ldots, \bar{C}_2^{k-1}\}$ for $k \geq 1$. The claim follows for $k = 0$ from Proposition 4.2. As the induction hypothesis, we assume that $\bar{C}_1^{k-1}$ is a rank 1 C–G inequality for $\widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1}$, and any 0–1 partial solution that violates C_1^{k-1} is inconsistent with $\widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1}$. We wish to show that any partial solution that violates C_1^k is inconsistent with $\widehat{\mathcal{S}}_{\mathrm{LP}}^k = \widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1} \cup \{\bar{C}_2^{k-1}\}$. Let $C_1^{k-1} = C(I_1^+ \cup \{l\}, I_1^-)$ and $C_2^{k-1} = C(I_2^+, I_2^- \cup \{l\})$, where $I_1^- \cap I_2^+ = \emptyset$ and $I_2^- \cap I_1^+ = \emptyset$. The resolvent of these two clauses is $C_1^k = C(I_1^+ \cup I_2^+, I_1^- \cup I_2^-)$, with

$$\bar{C}_1^k = -\sum_{i \in I_1^+ \cup I_2^+} x_i + \sum_{i \in I_1^- \cup I_2^-} x_i \leq |I_1^- \cup I_2^-| - 1.$$

While aggregating rank 1 C–G inequalities does not necessarily produce a rank 1 C–G inequality in general, we next show that $\bar{C}_1^k$ has the special property that it can be obtained as a rank 1 C–G inequality for $\widehat{\mathcal{S}}_{\mathrm{LP}}^k$. Since $\bar{C}_1^{k-1}$ is a rank 1 C–G inequality for $\widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1}$ by induction hypothesis, there exists a weighted sum of constraints of $\widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1}$ that yields an inequality $\boldsymbol{\alpha x} \leq \beta$ that implies $\bar{C}_1^{k-1}$. Using an argument similar to that in the proof of Proposition 4.3 for $\bar{C}_1^{k-1}$, we can aggregate $\boldsymbol{\alpha x} \leq \beta$ with 0–1 bound constraints to obtain the following inequality:

$$-x_l - \sum_{i \in I_1^+} x_i + \sum_{i \in I_1^-} x_i \leq |I_1^-| - 1 + f$$

where $0 \leq f < 1$. In other words, the above inequality is obtained by aggregating constraints of $\widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1}$ because $\boldsymbol{\alpha x} \leq \beta$ is itself an aggregated inequality for $\widehat{\mathcal{S}}_{\mathrm{LP}}^{k-1}$. Taking a weighted sum with equal weights $\frac{1}{2}$ of this inequality with

$$\bar{C}_2^{k-1} = x_l - \sum_{i \in I_2^+} x_i + \sum_{i \in I_2^-} x_i \leq |I_2^-|$$

along with $x_i \leq 1$ for $i \in I_1^- \Delta I_2^-$, and $-x_i \leq 0$ for $i \in I_1^+ \Delta I_2^+$, where Δ represents the symmetric difference, we obtain

$$-\sum_{i \in I_1^+ \cup I_2^+} x_i + \sum_{i \in I_1^- \cup I_2^-} x_i \leq |I_1^- \cup I_2^-| + \frac{f-1}{2}.$$

We have that $-1 < \frac{1}{2}(f-1) < 0$ since $0 \leq f < 1$. Therefore, rounding down the right-hand-side value yields the desired rank 1 C–G inequality $\bar{C}_k^1$ for $\widehat{\mathcal{S}}_{\mathrm{LP}}^k$. Proposition 4.2 now implies that any 0–1 partial solution that violates C_1^k is inconsistent with $\widehat{\mathcal{S}}_{\mathrm{LP}}^k$, and we have the desired result. □

Since resolution is a specialized Chvátal procedure performed on the inequalities representing two clauses, Proposition 4.2 implies that any partial assignment that violates the resolvent is inconsistent with the constraint set that contains the inequalities representing the two parent clauses. However, Proposition 4.4 gives a stronger result by showing that only one parent needs to be explicitly included in the constraint set to achieve the consistency power of the resolvent. We illustrate this benefit through an example.

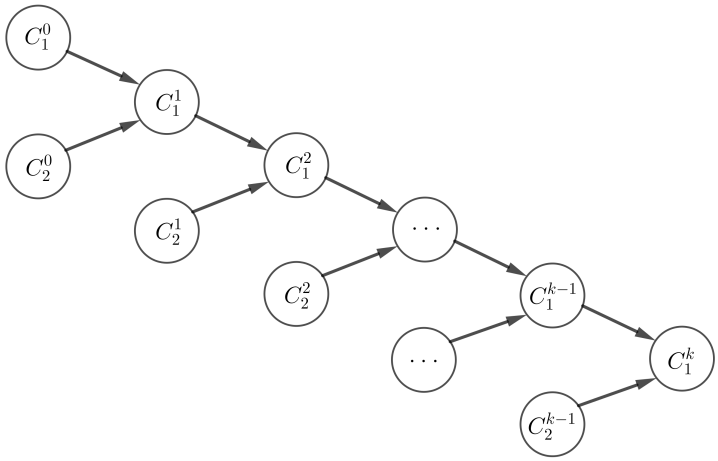

Fig. 4.1: Input proof sequence studied in Proposition 4.4

Example 4.3 Consider the 0–1 constraint set $\mathcal{S}^1$ of Example 3.3, which we saw earlier is not LP-consistent because the partial assignment $x_1 = 1$ is consistent with $\mathcal{S}_{\text{LP}}$ but inconsistent with $\mathcal{S}$. We can rule out $x_1 = 1$ by performing resolution on the clauses $\neg x_1 \vee \neg x_2$ and $\neg x_1 \vee x_2$ represented by $\mathcal{S}_{\text{C}}$ to obtain $\neg x_1$. However, due to Proposition 4.4, it is enough to add either one of the parents of the resolvent $\neg x_1$ to $\mathcal{S}$ to obtain a set $\widehat{\mathcal{S}}_{\text{LP}}$ with which the partial assignment $x_1 = 1$ is inconsistent. It is not necessary to add the other parent or the resolvent. Thus $\mathcal{S} \cup \{x_1 + x_2 \leq 1\}$ is LP-consistent, as is $\mathcal{S} \cup \{x_1 - x_2 \leq 0\}$. We can achieve the power of a resolution step without actually carrying out the resolution.

4.3 LP-consistency and the convex hull

In IP, an ideal formulation of a 0–1 problem is one that describes the convex hull of the feasible set. In CP, an ideal formulation is one that is consistent. It is shown in [13] that these two concepts do not coincide for general 0–1 programs. Yet [13] also shows that the convex hull formulation of a problem is necessarily LP-consistent. This relation implies that achieving LP-consistency could be cheaper than obtaining the convex hull, a fact that leads to a computational distinction between these two concepts. We provide theoretical evidence for this distinction in Section 5.2, which is corroborated by computational experiments in Section 6. A summary of some of the relations presented so far between various forms of consistency and the convex hull formulation is given in Figure 4.2.

We discussed in previous sections that LP-consistency is closely related to rank 1 C–G inequalities, which were originally introduced as a bound tightening technique that works toward a convex hull description. Based on this link, we further investigate the relation between LP-consistency and the convex hull. Two natural questions arise:

(i) Is a 0–1 constraint set with Chvátal rank 1 necessarily LP-consistent?
(ii) Does an LP-consistent constraint set necessarily have Chvátal rank 1?

The next example shows that neither question has a positive answer in general.

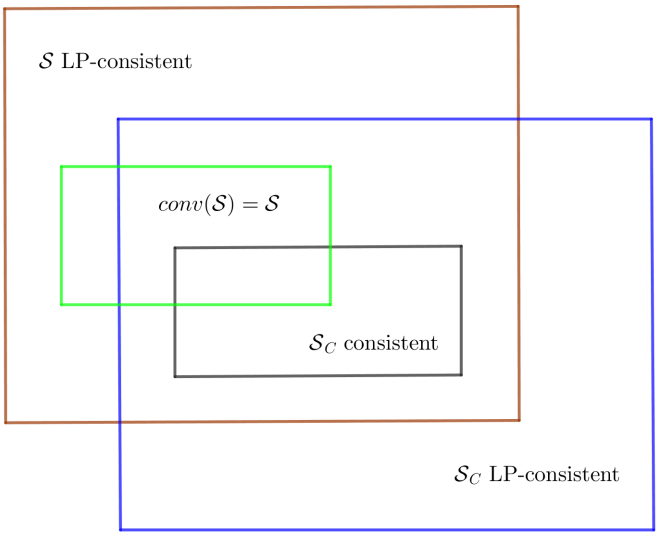

Fig. 4.2: Summary of the relations between variants of consistency and convex hull.

Example 4.4 Consider $\mathcal{S}^1 = \{x \in \{0,1\}^2,\ 4x_1 - 4x_2 \leq 3,\ x_1 + 4x_2 \leq 3\}$. It is clear that $\mathcal{S}^1$ is not LP-consistent, as the inconsistent partial assignment $x_1 = 1$ is consistent with $\mathcal{S}^1_{\text{LP}}$. However, the Chvátal rank of $\mathcal{S}^1$ is 1, as its convex hull is described by $x \in [0,1]^2$ and the rank 1 C–G inequalities $x_2 \leq 0$, $x_1 - x_2 \leq 0$.

Now consider $\mathcal{S}^2 = \{x \in \{0,1\}^n,\ \sum_{i=1}^n a_i x_i \leq b\}$, where $0 \leq a_i \leq b$ for all i. It is easy to verify that $\mathcal{S}^2$ is LP-consistent, as every inconsistent 0–1 partial assignment makes $\mathcal{S}^2_{\text{LP}}$ infeasible. However, it is well known that the Chvátal rank of a general 0–1 knapsack problem of the form $\mathcal{S}^2$ is not necessarily 1.

We show, however, that question (i) has an affirmative answer under a geometric condition for the convex hull polytope.

Proposition 4.5 *Consider a 0–1 rational constraint set $\mathcal{S}$, and let* $\text{conv}(\mathcal{S})$ *represent the convex hull of its feasible solutions. Assume that, for any 0–1 partial assignment $x_J = v_J$ that is inconsistent with $\mathcal{S}$, there exists a full-dimensional facet of* $\text{conv}(\mathcal{S})$ *whose defining inequality separates the face of the unit hypercube associated with $x_J = v_J$ from* $\text{conv}(\mathcal{S})$*. Then $\mathcal{S}$ is LP-consistent if its Chvátal rank is one.*

Proof Assume that constraints of $\mathcal{S}_{\text{LP}}$ are represented by a system $Ax \leq b$ that includes variable bounds $0 \leq x_j \leq 1$. Consider an inconsistent partial assignment $x_J = v_J$ of $\mathcal{S}$ for some $J \subseteq N$. We are given that there exists a full-dimensional facet of $\text{conv}(\mathcal{S})$ whose defining inequality separates the unit hypercube face associated with $x_J = v_J$ from $\text{conv}(\mathcal{S})$. Let $ax \leq \beta$ be the unique inequality (up to a scalar multiple) with integer coefficients and right-hand side that defines this full-dimensional facet. Because the Chvátal rank of $\mathcal{S}$ is one, we can obtain $ax \leq \beta$ as a rank 1 C–G inequality with weight vector $u \geq 0$ such that $uA = a$ and $\lfloor ub \rfloor = \beta$. Since $ax \leq \beta$ separates the face $x_J = v_J$ from $\text{conv}(\mathcal{S})$, we have that $a\bar{x} > \beta$ for any point $\bar{x}$ such that $\bar{x}_J = v_J$ and $\bar{x}_i \in \{0,1\}$ for all $i \in N \setminus J$. We therefore have $uA\bar{x} > ub$ for any such point, which implies that $uAx \leq ub$ separates the face $x_J = v_J$ from $\mathcal{S}_{\text{LP}}$. As a result, $\mathcal{S}_{\text{LP}} \cup \{x_J = v_J\}$ is infeasible. Since $x_J = v_J$ is an arbitrary inconsistent 0–1 partial assignment, it follows that $\mathcal{S}$ is LP-consistent. □

Various works in the literature study polyhedral properties of rank 1 Chvátal sets; see for instance [12] and references therein. Proposition 4.5 indicates that when facets of a rank-1 Chvátal set are suitably structured, the B&B procedure can advance without backtracking.

5 Partial Consistency

The previously introduced variants of consistency, k-consistency and sequential consistency are either too broad or too restrictive for typical B&B procedures. In particular, k-consistency is concerned with the consistency of partial assignments to all subsets of size k, whereas in branching search, only partial assignments defined by branches traversed in the tree need to be considered. On the other hand, sequential consistency considers only a fixed variable ordering, whereas in B&B the choice of the next branching variable is dynamically determined at each node. For these reasons, we introduce *partial consistency* as a variant of consistency that is better suited for implementation in B&B methods by occupying a middle ground between k-consistency and sequential consistency. We study this variant under both CP and IP frameworks.

5.1 Partial consistency in constraint programming

In this section, we develop the concept of partial consistency in CP as a generalization of the common variants in the literature.

Definition 5.1 Let $\mathcal{S}$ be a 0–1 constraint set, and consider a subset $I \subseteq N$. Then $\mathcal{S}$ has *partial consistency of rank r over I*, where $0 < r \leq n - |I|$, if for every partial assignment $x_I = v_I$ that violates no constraint in $\mathcal{S}$, and every $J \subseteq N \setminus I$ with $|J| = r$, there exists a value assignment $x_J = v_J$ such that the extended assignment $x_{I\cup J} = v_{I\cup J}$ violates no constraint in $\mathcal{S}$.

Partial consistency addresses practical shortcomings of previous variants because (i) the variable set I indexes the variables on which the search branches down to the current node of the search tree (in contrast to k-consistency, which considers all possible combinations of k variables), and (ii) the set of variables J, with respect to which consistency is computed, is independent of the subsequent branching order (in contrast to sequential consistency, which assumes a pre-fixed branching order). It is easy to verify that partial consistency of rank r eliminates backtracking for the next r levels of the search tree, regardless of the subsequent branching order. The next example illustrates the benefit of this property for reducing tree size.

Example 5.1 Consider $\mathcal{S} = \{\boldsymbol{x} \in \{0,1\}^{10},\ x_1 + x_2 \leq 1,\ x_1 - x_i \leq 0 \text{ for } i = 2,\ldots,10\}$. It follows from Definition 5.1 that $\mathcal{S}$ lacks partial consistency of rank 1 over $I = \{1\}$, because $x_1 = 1$ does not violate any constraint in $\mathcal{S}$, while both value assignments $(x_1, x_2) = (1, 0)$ and $(x_1, x_2) = (1, 1)$ violate some constraint in $\mathcal{S}$. Assume that the branching policy chooses x_1 as the first variable to branch on, and consider two follow-up branching scenarios. In the first scenario, the subsequent branching order is $x_2, \ldots, x_{10}$. The inconsistency of the partial assignment $x_1 = 1$ is detected immediately, since branching on both $x_2 = 0$ and $x_2 = 1$ violates some constraint in $\mathcal{S}$. This means that achieving sequential 2-consistency is sufficient to eliminate backtracking. In the second scenario, the branching order is $x_3, x_4, \ldots, x_{10}, x_2$. The inconsistency of the partial assignment $x_1 = 1$ will not be detected until the last branching level. This means that achieving sequential 10-consistency is required to eliminate backtracking, which leads to traversing 2^9 nodes only to trace back to the node with $x_1 = 1$ and prune it. This ordering requires a significant backtracking compared to the first one. One way to prevent this severe dependence on the subsequent branching order is to make the model partially consistent over $I = \{1\}$. To this end, we add the inequality representing resolvent $\neg x_1$ obtained from the resolution of clauses $\neg x_1 \vee \neg x_2$ and $\neg x_1 \vee x_2$, each of which is implied by a

Algorithm 1 Algorithm to achieve partial consistency of rank r for clause set $\mathcal{C}$

Input: A clause set $\mathcal{C}$, a subset $I \subseteq N$, and a positive number $0 < r \leq n - |I|$
Output: A set $\hat{\mathcal{C}}$ that has partial consistency of rank r over I
1: Initialize $\hat{\mathcal{C}} \leftarrow \mathcal{C}$, $t \leftarrow 1$
2: **while** $t \leq r$ **do**
3: Perform resolution on clauses in $\mathcal{C}$ to obtain all resolvents containing at most $r - t$ variables not in I.
4: Add the resolvents to $\mathcal{C}$ if they are not implied by its clauses
5: Add to $\hat{\mathcal{C}}$ all resolvents whose variables belong to I.
6: Update $t \leftarrow t + 1$
7: **end while**

constraint in $\mathcal{S}$. Now the node with assignment $x_1 = 1$ will be pruned no matter which branching order is executed next.

It follows from Definition 5.1 that if $\mathcal{S}$ has partial consistency of rank r_1 over I, it has partial consistency of rank r_2 over I for any $r_2 < r_1$. As the rank increases, partial consistency becomes closer to a full consistency, but achieving it becomes more expensive. For instance, consider $\mathcal{S} = \{\boldsymbol{x} \in \{0,1\}^3 \mid x_1 + x_2 + x_3 \leq 1,\ x_1 - x_2 - x_3 \leq 0\}$. The partial assignment $x_1 = 1$ is inconsistent with $\mathcal{S}$. It is easy to verify that $\mathcal{S}$ has partial consistency of rank 1 over $I = \{1\}$ but lacks partial consistency of rank 2 over I. The following result shows the relation between partial and full consistency.

Proposition 5.1 *If $\mathcal{S}$ has partial consistency of rank $r = n - |I|$ over I for every $I \subseteq N$, then $\mathcal{S}$ is consistent.*

Proof Assume to the contrary that $\mathcal{S}$ is not consistent. Then there exists a partial assignment $x_I = v_I$ for some $I \subseteq N$ that violates no constraints in $\mathcal{S}$ but for which $x_{I \cup J} = v_{I \cup J}$ is inconsistent with $\mathcal{S}$ for any assignment $x_J = v_J$ with $J = N \setminus I$. This contradicts the assumption that $\mathcal{S}$ has partial consistency of rank $n - |I|$ over I. □

From a CP perspective, partial consistency can be achieved for $\mathcal{S}$ as in Algorithm 1 by applying resolution to the clause set $\mathcal{C}$ represented by $\mathcal{S}_{\mathrm{C}}$.

Proposition 5.2 *Given a clause set $\mathcal{C}$, a subset $I \subseteq N$, and a positive number r with $0 \leq r \leq n - |I|$, Algorithm 1 achieves partial consistency of rank r over I.*

Proof Let $\mathcal{C}^{[r]}$ contain the clauses in $\mathcal{C}$ at the end of iteration r of Algorithm 1, so that all the clauses in $\mathcal{C}^{[r]}$ have rank at most r. It suffices to show that for any $I \in N$ and any r with $0 \leq r \leq n - |I|$, if $x_{I \cup J} = v_{I \cup J}$ violates some clause in $\mathcal{C}$ for any $v_J \in \{0,1\}^r$, then some clause in $\mathcal{C}^{[r]}$ is violated by $x_I = v_I$. We prove the claim by induction on r. The claim is trivially true for $r = 0$. For the induction hypothesis we assume that the claim holds for a given r, and we wish to show that it holds for $r+1$. Thus we wish to show for any given $\bar{I} \subseteq N$ that if $x_{\bar{I} \cup J \cup \{j\}} = v_{\bar{I} \cup J \cup \{j\}}$ violates some clause in $\mathcal{C}$ for any $v_{J \cup \{j\}} \in \{0,1\}^{r+1}$, then $\mathcal{C}^{[r+1]}$ contains a clause that is violated by $x_{\bar{I}} = v_{\bar{I}}$. We apply the induction hypothesis with $I = \bar{I} \cup \{j\}$. Thus if we set $v_j = 0$, there is a clause of the form $C_1 \vee x_j$ in $\mathcal{C}^{[r]}$ that $x_{\bar{I} \cup J \cup \{j\}} = v_{\bar{I} \cup J \cup \{j\}}$ violates, and if we set $v_j = 1$, there is a clause $C_2 \vee \neg x_j$ in $\mathcal{C}^{[r]}$ that $x_{\bar{I} \cup J \cup \{j\}} = v_{\bar{I} \cup J \cup \{j\}}$ violates. No variable changes sign between C_1 and C_2, since otherwise $x_{\bar{I}} = v_{\bar{I}}$ would not violate both C_1 and C_2. Thus the two clauses $C_1 \vee x_j$ and $C_2 \vee \neg x_j$ have a resolvent R that $x_{\bar{I}} = v_{\bar{I}}$ violates. Because these 2 clauses have at most 1 variable not in $\bar{I}$, their resolvent R belongs to $\mathcal{C}^{[r+1]}$, and the proof is complete. □

Since achieving higher ranks of partial consistency can significantly increase computational effort, as is common for other variants of consistency, it is practical to consider lower ranks to balance the cost of cut generation against the reduction in tree size. We note, in particular, that sequential consistency and k-consistency in CP can be viewed as rank 1 partial consistency.

Corollary 5.1 *A constraint set $\mathcal{S}$ has partial consistency of rank 1 over I if and only if it has sequential $(k+1)$-consistency with $k = |I|$ for all branching orders $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ where $\{i_1, \ldots, i_{k-1}\}$ is a permutation of elements in I, and $\{i_k, \ldots, i_n\}$ is a permutation of elements in $N \setminus I$.*

Corollary 5.2 *A constraint set $\mathcal{S}$ has k-consistency if and only if it has partial consistency of rank 1 over I for all $I \subseteq N$ such that $|I| \leq k-1$.*

5.2 Partial LP-consistency in integer programming

In this section, we adapt the concept of partial consistency to IP by making use of the LP relaxation.

Definition 5.2 Let $\mathcal{S}$ be a 0–1 constraint set, and consider a subset $I \subseteq N$. Then $\mathcal{S}$ has *partial LP-consistency of rank r over I*, where $0 < r \leq n - |I|$, if for every partial assignment $x_I = v_I$ that is consistent with $\mathcal{S}_{\text{LP}}$, and every $J \subseteq N \setminus I$ with $|J| = r$, there exists a 0–1 value assignment $x_J = v_J$ such that $x_{I \cup J} = v_{I \cup J}$ is consistent with $\mathcal{S}_{\text{LP}}$.

We noted in Section 3.3 that consistency implies LP-consistency, and that one can apply the resolution algorithm to the implied clause set to achieve consistency and thereby LP-consistency. The next example shows that such an approach is not viable for partial LP-consistency, because it may be necessary to generate resolvents of rank much higher than r to achieve rank r LP-consistency.

Example 5.2 Let $\mathcal{S}$ consist of the following inequalities along with $x \in \{0,1\}^n$:

$$-x_1 + x_2 + \sum_{i=3}^{n} x_i \leq 1, \quad 2x_2 - \sum_{i=3}^{n} i x_i \leq 0, \quad x_1 + x_2 \geq 1$$

for $n \geq 3$. Definition 5.2 implies that $\mathcal{S}$ lacks partial LP-consistency of rank 1 over $I = \{1\}$, because the partial assignment $x_1 = 0$ leads to infeasible LP relaxations for both $x_2 = 0$ and $x_2 = 1$. On the other hand, it is easy to verify that $\mathcal{S}$ has partial consistency of ranks $r = 1, 2, \ldots, n-2$ over $I = \{1\}$, but not of rank $r = n-1$. This means that the inconsistency of the partial assignment $x_1 = 0$ is detected only after considering all possible 0–1 assignments to variables $x_2, \ldots, x_n$. To achieve the partial LP-consistency of rank 1 over $I = \{1\}$ using resolution, we would need to perform $n-1$ iterations of Algorithm 1 on the implied clause set of $\mathcal{S}$ to obtain the desired resolvent represented by $-x_1 \leq -1$ that is violated by $x_1 = 0$.

Example 5.2 shows that achieving partial LP-consistency of a certain rank may require a much higher rank of partial consistency. This makes the achievement of partial LP-consistency cumbersome through traditional resolution-based techniques. We next describe techniques that can directly achieve partial LP-consistency by bridging consistency and cutting plane theory.

To this end, we use the core structure of the reformulation and linearization technique (RLT) of [41], but with a different objective. Consider a 0–1 constraint set of the form $\mathcal{S} = \{x \in \mathbb{Z}^n,\ Ax \leq b\}$, where $Ax \leq b$ includes bounds $0 \leq x_j \leq 1$. Given a partial assignment $x_I = v_I$ for some $I \subseteq N$, assume that there exists a nonempty $J \subseteq N \setminus I$ such that $\mathcal{S}_{\text{LP}} \cup \{x_{I \cup J} = v_{I \cup J}\}$ is infeasible for every 0–1 value assignments $x_J = v_J$. The existence of such a J would imply that $\mathcal{S}$ does not have partial LP-consistency of rank $|J|$ over I. We next present a variant of RLT applied to set $\mathcal{S}_{\text{LP}}$ that makes use of disjunctions formed by J to separate the partial assignment $x_I = v_I$, thereby achieving partial LP-consistency over I.

1. Generate the following nonlinear system:

$$(Ax - b) \prod_{j \in J_1} x_j \prod_{j \in J \setminus J_1} (1 - x_j) \leq 0, \quad \text{all } J_1 \subseteq J \tag{5.1}$$

2. Linearize (5.1) by replacing each x_i^2 with x_i, and $\prod_{k \in K} x_k$ with new variable y_K for each K with $|K| \geq 2$ that appears in the system.
3. Denote the resulting linear constraint set by $\mathcal{R}_J(\mathcal{S}_{\text{LP}})$, and its projection onto the space of variables x_I by $\mathcal{R}_J(\mathcal{S}_{\text{LP}})|_I$.

Proposition 5.3 *Consider a 0–1 constraint set of the form $\mathcal{S} = \{x \in \mathbb{Z}^n,\ Ax \leq b\}$, where $Ax \leq b$ includes bounds $0 \leq x_j \leq 1$. If partial assignment $x_I = v_I$ and a nonempty $J \subseteq N \setminus I$ are such that the linear system $\mathcal{S}_{\text{LP}} \cup \{x_{I \cup J} = v_{I \cup J}\}$ is infeasible for every 0–1 value assignment $x_J = v_J$, then $x_I = v_I$ is infeasible for $\mathcal{R}_J(\mathcal{S}_{\text{LP}})|_I$.*

Proof It follows from [41] that $\mathcal{R}_J(\mathcal{S}_{\text{LP}})$ is a lifted description of the convex hull of the union of the sets described by $\mathcal{S}_{\text{LP}} \cup \{x_J = v_J\}$ over all 0–1 value assignments $x_J = v_J$. We claim that $x_I = v_I$ is infeasible for $\mathcal{R}_J(\mathcal{S}_{\text{LP}})|_I$. Assume to the contrary. Then there exists a point $(x_I, x_J, x_K, y) = (v_I, \tilde{v}_J, \tilde{v}_K, \tilde{u})$ that satisfies $\mathcal{R}_J(\mathcal{S}_{\text{LP}})$, where $K = N \setminus (I \cup J)$. This point must be representable as a convex combination of points of the form $(v_I, v_J, \dot{v}_K, \dot{u})$ over all $v_J \in \{0,1\}^{|J|}$, since the components v_I are integral and cannot be represented as convex combination of other points. It follows from our assumption that such points do not exist, because $\mathcal{S}_{\text{LP}} \cup \{x_{I \cup J} = v_{I \cup J}\}$ is infeasible for all $v_J \in \{0,1\}^{|J|}$. This is a contradiction. □

Proposition 5.3 provides the basis for an algorithm that achieves partial LP-consistency.

Proposition 5.4 *Given a 0–1 constraint set $\mathcal{S}$, a subset $I \subseteq N$, and a positive number $0 < r \leq n - |I|$, Algorithm 2 achieves partial LP-consistency of rank r over I.*

Proof Consider any partial assignment $x_I = v_I$ for which $\mathcal{S}_{\text{LP}} \cup \{x_I = v_I\}$ is feasible while $\mathcal{S}_{\text{LP}} \cup \{x_{I \cup J} = v_{I \cup J}\}$ is infeasible for all $v_J \in \{0,1\}^r$ for some $J \subseteq N$ with $|J| = r$. It suffices to show that $\widehat{\mathcal{S}}_{\text{LP}} \cup \{x_I = v_I\}$ is infeasible. It follows from Proposition 5.3 that $x_I = v_J$ is infeasible for $\mathcal{R}_J(\mathcal{S}_{\text{LP}})|_I$. It is also clear that the projection of the feasible set of $\mathcal{R}(\mathcal{S}_{\text{LP}})|_I$ onto the space of variables included in $\mathcal{R}_J(\mathcal{S}_{\text{LP}})|_I$ is a subset of the feasible set of $\mathcal{R}_J(\mathcal{S}_{\text{LP}})|_I$ since $\mathcal{R}(\mathcal{S}_{\text{LP}}) \supseteq \mathcal{R}_J(\mathcal{S}_{\text{LP}})$. We conclude that $\widehat{\mathcal{S}}_{\text{LP}} \cup \{x_I = v_I\}$ is infeasible, as the constraints in $\mathcal{R}(\mathcal{S}_{\text{LP}})|_I$ are added to $\widehat{\mathcal{S}}_{\text{LP}}$. □

Although the method by which $\mathcal{R}_J(\mathcal{S}_{\text{LP}})$ is constructed is similar to well-known disjunctive programming techniques [4] for obtaining the convex hull of a disjunction, its implementation in Algorithm 2 can achieve a higher rank in LP-consistency compared with that of the RLT closure.

Proposition 5.5 *Consider a 0–1 constraint set $\mathcal{S}$ and a 0–1 partial assignment $x_I = v_I$ that is inconsistent with $\mathcal{S}$.*

Algorithm 2 Algorithm to achieve partial LP-consistency

Input: A constraint set $\mathcal{S}$, a subset $I \subseteq N$, and a positive number $0 < r \leq n - |I|$
Output: A set $\widehat{\mathcal{S}}$ that has partial LP-consistency of rank r over I

1: Initialize $\widehat{\mathcal{S}} = \mathcal{S}$
2: Generate the constraint set $\mathcal{R}(\mathcal{S}_{\text{LP}})$ that is the union of $\mathcal{R}_J(\mathcal{S}_{\text{LP}})$ over all $J \subseteq N$ with $|J| = r$.
3: Linearize the above system by replacing x_i^2 with x_i, and $\prod_{k \in K} x_k$ with new variable y_K for each $K \subseteq N$ that appears in the system. Denote the resulting linear constraint set by $\mathcal{R}(\mathcal{S}_{\text{LP}})$, and its projection onto the space of variables x_I by $\mathcal{R}(\mathcal{S}_{\text{LP}})|_I$.
4: Add to $\widehat{\mathcal{S}}$ the inequalities in $\mathcal{R}(\mathcal{S}_{\text{LP}})|_I$.

(i) If there exists $j \in N \setminus I$ such that $\mathcal{S}_{\text{LP}} \cup \{x_{I\cup\{j\}} = v_{I\cup\{j\}}\}$ is infeasible for $v_j \in \{0,1\}$, then $x_I = v_I$ is infeasible for the set $\widehat{\mathcal{S}}$ obtained by applying Algorithm 2 to $\mathcal{S}$ over I with rank $r = 1$.

(ii) Define

$$r_{\min} = \min_{J\subseteq N\setminus I} \left\{ |J| \;\middle|\; \mathcal{S}_{\text{LP}} \cup \{x_{I\cup J} = v_{I\cup J}\} \text{ is infeasible for all } v_J \in \{0,1\}^{|J|} \right\}$$

with a minimizer $J_{\min}$. Let J^* be composed of elements j of $J_{\min}$ such that $\mathcal{S}_{\text{LP}} \cup \{x_{I\cup\{j\}} = v_{I\cup\{j\}}\}$ is infeasible for exactly one 0–1 value assignment $v_j \in \{0,1\}$. Then, $x_I = v_I$ is infeasible for set $\widehat{\mathcal{S}}$ obtained by applying Algorithm 2 to $\mathcal{S}$ over I with rank $r = \max\{r_{\min} - |J^*|, 1\}$.

Proof Statement (i) follows from Proposition 5.3 by setting $J = \{j\}$. To show (ii), suppose to the contrary that there exists an assignment $x = \bar{x}$ with $\bar{x}_I = v_I$ that is feasible for the set $\mathcal{R}(\mathcal{S}_{\text{LP}})$ formed in step 3 of Algorithm 2. Since $r \geq 1$, for any $k \in J^*$ there exists $J_k \supseteq \{k\}$ with $|J_k| = r$. It is clear that the feasible set of $\mathcal{R}_{J_k}(\mathcal{S}_{\text{LP}}) \cup \{x_I = v_I\}$ lies on the hyperplane defined by $x_k = 1 - v_k$ because of the assumption that $\mathcal{S}_{\text{LP}} \cup \{x_{I\cup\{k\}} = v_{I\cup\{k\}}\}$ is infeasible. Since $\mathcal{R}(\mathcal{S}_{\text{LP}}) \supseteq \mathcal{R}_{J_k}(\mathcal{S}_{\text{LP}})$, we have that $\bar{x}_k = 1 - v_k$ for all $k \in J^*$. Consider now the set $\bar{J} = J_{\min} \setminus J^*$. There exists $\hat{J} \supseteq \bar{J}$ with $|\hat{J}| = r$. The set $\mathcal{R}_{\hat{J}}(\mathcal{S}_{\text{LP}})$ represents the lifted convex hull of the union of the feasible sets of $\mathcal{S}_{\text{LP}} \cup \{x_{\hat{J}} = v_{\hat{J}}\}$ over all $v_{\hat{J}} \in \{0,1\}^r$. The restriction of each constraint set $\mathcal{S}_{\text{LP}} \cup \{x_{\hat{J}} = v_{\hat{J}}\}$ to $x_I = v_I$ and $x_k = 1 - v_k$ is infeasible for $k \in J^*$, because of the assumption that $\mathcal{S}_{\text{LP}} \cup \{x_{I\cup J_{\min}} = v_{I\cup J_{\min}}\}$ is infeasible for every 0–1 value assignment $x_{J_{\min}} = v_{J_{\min}}$. As a result, $x = \bar{x}$ is infeasible for $\mathcal{R}(\mathcal{S}_{\text{LP}}) \supseteq \mathcal{R}_{\hat{J}}(\mathcal{S}_{\text{LP}}) \cup_{k\in J^*} \mathcal{R}_{J_k}(\mathcal{S}_{\text{LP}})$, contrary to assumption. □

It follows from part (ii) of Proposition 5.5 that applying Algorithm 2 for a certain rank r can achieve LP-consistency of ranks higher than r. The next example illustrates this fact.

Example 5.3 Consider a 0–1 constraint set S defined by

$$\begin{aligned} &2x_1 + 2x_4 \leq 3 \\ &2x_2 + 2x_4 \leq 3 \\ &-2x_1 - 2x_2 - 2x_3 + 2x_4 \leq 1 \\ &-2x_1 - 2x_2 + 2x_3 + 2x_4 \leq 3 \end{aligned}$$

It is clear that $x_4 = 1$ is inconsistent with $\mathcal{S}$. To apply Proposition 5.5(ii), we set $I = \{4\}$, and compute $r_{\min} = 3$ and $J_{\min} = \{1,2,3\}$, which implies that all 0–1 value assignments are infeasible at the face defined by $x_4 = 1$. Further, $J^* = \{1,2\}$ because $\mathcal{S}_{\text{LP}} \cup \{(x_1, x_4) = (v_1, 1)\}$ is feasible only for $v_1 = 0$ and $\mathcal{S}_{\text{LP}} \cup \{(x_2, x_4) = (v_2, 1)\}$ is feasible only for $v_2 = 0$, while $\mathcal{S}_{\text{LP}} \cup \{(x_3, x_4) = (v_3, 1)\}$ is feasible for $v_3 = 0, 1$. Proposition 5.5(ii) implies that $x_4 = 1$ is infeasible for the set $\hat{\mathcal{S}}$ obtained by applying Algorithm 2 to $\mathcal{S}$ over I with rank $r = 1$. Since $r_{\min} = 3$, the algorithm achieves partial LP-consistency of rank 3 by setting $r = 1$. As an alternative, assume that we follow the classical route to separate the face $x_4 = 1$ through derivation of the convex hull of the disjunctive unions of suitable restrictions of $\mathcal{S}_{\text{LP}}$. Such restrictions require all three variables x_1, x_2, x_3 to be fixed as $r_{\min} = 3$. To obtain the convex hull of unions of $\mathcal{S}_{\text{LP}} \cup \{(x_1, x_2, x_3) = (v_1, v_2, v_3)\}$ for 0–1 assignments to v_1, v_2, v_3, a classical RLT (or lift-and-project) of rank $r = 3$ must be applied to yield the convex hull.

Inspired by the computational advantage exemplified above, we next introduce a new role for cutting planes that leads to a substantially superior outcome compared to their traditional role of separating fractional solutions.

5.3 Consistency cuts

Generating all the inequalities necessary to achieve LP-consistency is computationally prohibitive, as they grow exponentially with the problem size. Yet not all inequalities are needed to reduce backtracking. A search tree node can be pruned if the corresponding partial assignment is inconsistent with a well-chosen subset of the inequalities in $\mathcal{R}(\mathcal{S}_{\text{LP}})|_I$. This motivates an approach that makes the implementation of Algorithm 2 more efficient. As common in classical cutting plane approaches, we use a cut-generating linear program (CGLP), to obtain inequalities that separate inconsistent partial assignments; see [3] for a detailed account on derivation of such CGLPs.

Proposition 5.6 *Consider a 0–1 constraint set $\mathcal{S}$, a subset $I \subseteq N$, and a rank $r \leq n-|I|$. Let $\mathcal{R}(\mathcal{S}_{\text{LP}}) = \{A_I x_I + A_J x_J + By \leq d\}$ be the lifted linear system generated in Algorithm 2. Consider the following CGLP*

$$
\begin{aligned}
w^*(v_I) = \max \quad & \alpha^\intercal A_I v_I - \alpha^\intercal d && \text{(5.2a)}\\
s.t. \quad & \alpha^\intercal A_J = 0 && \text{(5.2b)}\\
& \alpha^\intercal B = 0 && \text{(5.2c)}\\
& \alpha^\intercal 1 \leq 1 && \text{(5.2d)}\\
& \alpha \geq 0 && \text{(5.2e)}
\end{aligned}
$$

where variables α represent the dual weight vector associated with constraints of $\mathcal{R}(\mathcal{S}_{\text{LP}})$, and (5.2d) is a normalization constraint. Define $\hat{\mathcal{S}}$ to be the set obtained by adding to $\mathcal{S}$ constraints of the form $\bar{\alpha}^\intercal A_I x_I \leq \bar{\alpha}^\intercal d$, where $\bar{\alpha}$ is the optimal solution of (5.2a)–(5.2e), for all $v_I \in \{0,1\}^{|I|}$ that yield an optimal value $w^*(v_I) > 0$. Then $\hat{\mathcal{S}}$ has partial LP-consistency of rank r over I. □

Proposition 5.6 gives a more efficient way to prune nodes on the basis of partial consistency. At each node associated with the partial assignment $x_I = v_I$, the CGLP (5.2a)–(5.2e) is solved, and if its optimal value is positive, the corresponding cutting plane, which we refer to as a *consistency cut*, is added to the model. This cut makes the LP relaxation solved at the node infeasible, which allows the node to be pruned.

While the structure of this approach is similar to that of typical cutting plane methods, it has fundamental differences with both local and global cutting plane algorithms.

In local methods, the fixed variables at a node of the search tree are removed from the model, and cuts are generated in the space of the remaining unfixed variables with the goal of separating a fractional solution of the LP relaxation. Such cuts are locally valid only. In contrast, consistency cuts are globally valid and are generated in the space of fixed variables with the goal of separating the entire face defined by an inconsistent partial assignment.

Consistency cuts also differ from global cuts obtained from the classical RLT, as illustrated by the following example.

Example 5.4 Let the 0–1 constraint set $\mathcal{S}$ contain the inequalities

$$
\begin{aligned}
4x_1 + 4x_2 + 4x_3 &\leq 9\\
4x_1 + 4x_2 - 4x_3 &\leq 5\\
-4x_1 + 4x_2 + 4x_3 &\leq 5\\
-4x_1 + 4x_2 - 4x_3 &\leq 1
\end{aligned}
$$

Consider a node of the search tree associated with the partial assignment $x_2 = 1$, which is clearly inconsistent with $\mathcal{S}$. To formulate the CGLP of Proposition 5.6, consider $J = \{3\}$. It suffices to multiply constraints of $\mathcal{S}$ with x_3 and $1 - x_3$ and then linearize the lifted system according to Algorithm 2. Solving the CGLP yields the cutting plane $3x_2 \leq 2$, which leads to pruning the node.

In the classical RLT, the CGLP seeks to find facets of the convex hull of the union of the feasible sets of $\mathcal{S} \cup \{x_3 = 0\}$ and $\mathcal{S} \cup \{x_3 = 1\}$ that cut off an extreme point of the feasible region described by $\mathcal{S} \cup \{x_2 = 1\}$. There are two such facet-defining inequalities, $4x_1 + 4x_2 \leq 5$ and $-4x_1 + 4x_2 \leq 1$. Therefore, regardless of the extreme point chosen to be separated, the cut obtained from the CGLP is insufficient to cut off the entire face defined by $x_2 = 1$, or rather to make the LP relaxation at this face infeasible. In fact, at least two iterations of the CGLP, each separating an extreme point of the LP relaxation at the face $x_2 = 1$, are required to make LP relaxation infeasible and thereby to prune the node.

Example 5.4 can be extended in such a way that a large number of CGLPs would be required when using RLT to add cutting planes that make the LP relaxation infeasible, while the CGLP of Proposition 5.6 finds a separating cut in just one iteration.

6 Computational Results

In this section, we conduct preliminary computational experiments to assess the performance of consistency cutting planes in comparison with classical cutting planes. While we do not attempt to integrate consistency-based methods into a state-of-the-art IP solver at this stage of research, we demonstrate computational promise by showing how consistency cuts can reduce the size of the search tree and computation time relative to traditional separating cuts.

The results are obtained on a RedHat Enterprise Linux 7, 128 GB RAM, with 16 processors. The codes are written in Python v2.7 and IP models are solved with CPLEX v12.8.0. We perform our experiments on both synthetic and benchmark instances as presented in the next two subsections.

6.1 Synthetic instances

For our synthetic instances, we consider an integer program of general form to minimize structural bias that can impact the outcome of our comparison. In particular, we study 0–1 programs of the form $\max\{c^\intercal x : x \in \mathcal{S}\}$ where $\mathcal{S} = \{x \in \{0,1\}^n, Ax \leq b\}$. We consider different problem sizes for the number of variables n and number of constraints m as given in the first two columns of Table 6.1. These sizes, ranging from small to moderate, have been chosen so that the B&B algorithm terminates prior to running out of memory.

For each problem size, five instances are generated and reported in the third column of Table 6.1. The data for these instances are generated as follows. The objective function and constraints' coefficients are randomly generated from a discrete uniform distribution on $[-100, 100]$, resulting in coefficient matrices of high density. The right-hand-side values are randomly generated from a discrete uniform distribution on the interval $[0, \frac{1}{5}\sum \text{positive coefficients}]$. This data generation policy ensures that the optimization problem is feasible as the origin is a feasible solution. CPLEX is used at its default settings, except that cutting planes and presolve are turned off, and we fix the branching order to obtain a controlled experiment. Specifically, we branch on variables in the order they are indexed.

In these experiments, we investigate partial LP-consistency of rank 1, as higher ranks become exceedingly expensive to achieve, as common in typical cutting plane techniques. Columns 3–5 of Table 6.1 under "Consistency cut" show the result of applying the CGLP of Proposition 5.6 at each node of the B&B tree to obtain partial-LP consistency of rank 1. In particular, columns labeled "nodes", "Δ" and "time" show the B&B tree size at optimality after adding consistency cuts, the size reduction compared to the default CPLEX tree size without adding any cuts, and the total solution time for the B&B method including the cut-generation time. The next three columns under "RLT cut" show the result of applying the RLT of rank 1 at each node of the B&B tree to separate the fractional optimal solutions of the LP relaxation at each node. Symbol "–"

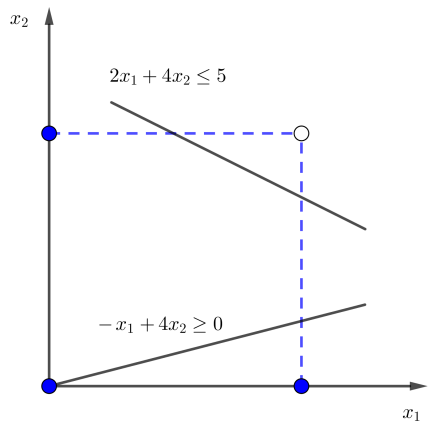

Fig. 6.1: The role of an optimality cut in creating an inconsistent partial assignment

indicates that the code returns an out of memory error before solving the problem to optimality. To be consistent with the consistency cuts, we limit the number of added cuts to one per each node. The disjunctions used to derive the CGLP of the RLT is the same as those used in the CGLP of the consistency cuts, i.e., they are formed with respect to the unfixed variables at each node of the B&B tree. As a result, the number of variables and constraints for both CGLPs are of the same order. In particular, if l is the number of unfixed variables, the number of variables in the CGLP for both problems is $2(m + 2n)l$, and the number of constraints for the RLT problem is nl and for the consistency problem is $nl + l$.

As is evident from these results, consistency cuts yield a substantial reduction in tree size and solution time as compared to the RLT cuts. It is also interesting to note the minimal effect of classical RLT cutting planes on the tree size in these instances. The performance gap widens as the problem size increases, and for the largest instances, the RLT method returns an out-of-memory error before obtaining the optimal solution. These results support the computational advantage of our cutting plane strategy as illustrated in Example 5.4.

To improve these results, the next columns use the same CGLP models for consistency and RLT, with a difference that the objective function is incorporated in the constraints of the original model. Let c_0 be the objective value of the best current integer solution, i.e., a lower bound for our problem. Then, at a node of the B&B tree, the constraint $c^\intercal x \geq c_0$ can be added to the constraint set with the goal of reducing candidate space for optimal solutions. Such an inequality is often regarded as *optimality cut* in the literature. Addition of the optimality cut can create new inconsistent partial assignments as illustrated in Figure 6.1, where we maximize $x_1 - 4x_2$ over $\mathcal{S} = \{x \in \{0,1\}^2, 2x_1 + 4x_2 \leq 5\}$. It is clear that the partial assignment $x_1 = 1$ is consistent with $\mathcal{S}$. However, adding the optimality cut $-x_1 + 4x_2 \geq 0$, corresponding to the origin as a feasible solution, makes $x_1 = 1$ inconsistent. Making use of the optimality cut and updating its right-hand-side value at each node of the B&B tree leads to a significant improvement in reducing the tree size. Columns under "Consistency + Opt cut" and "RLT + Opt cut" contain the result of applying the CGLP with the addition of the optimality cuts for consistency and RLT models, respectively. As indicated in these columns, optimality cuts reduce the size of the B&B tree as well as the solution time, whereas they rarely improve the performance of the RLT cuts.

Table 6.1: Evaluating consistency cuts for synthetic instances

m	n	#	Consistency cut			RLT cut			Consistency + Opt cut			RLT + Opt cut		
			nodes	Δ (%)	time (s)	nodes	Δ (%)	time (s)	nodes	Δ (%)	time (s)	nodes	Δ (%)	time (s)
30	30	1	666	82%	375.58	3729	0%	758.15	**344**	**91%**	**250.61**	3733	0%	773.23
		2	1125	72%	450.82	3658	8%	721.43	**371**	**91%**	**249.67**	3639	8%	733.34
		3	501	82%	251.14	2763	0%	512.39	**317**	**88%**	**201.91**	2766	0%	523.45
		4	219	89%	**147.84**	2085	0%	461.7	**204**	**90%**	149.21	2084	0%	466.96
		5	332	82%	176.62	1900	0%	387.94	**257**	**86%**	**157.28**	1900	0%	395.56
35	35	1	229	88%	290.64	1961	0%	761.96	**168**	**91%**	**227.3**	1988	0%	767.84
		2	1077	90%	1268.49	11241	0%	3823.5	**816**	**93%**	**1057.67**	11234	0%	3867.01
		3	1911	44%	1236.96	3392	1%	1057.2	**596**	**83%**	**678.75**	2477	27%	817.08
		4	319	90%	415.62	3265	2%	1288.59	**267**	**92%**	**371.47**	3261	2%	1298.74
		5	277	91%	339.34	3073	0%	992.48	**194**	**93%**	**276.15**	1719	41%	999.1
45	45	1	6676	78%	15229.93	34734	0%	29059.54	**3071**	**90%**	**10624**	31162	0%	24307.54
		2	4513	49%	6069.49	8939	0%	5715.35	**1047**	**88%**	**4016.22**	8921	0%	5749.21
		3	30422	13%	26140.12	34876	0%	24302.79	**23279**	**33%**	**21717.03**	34768	0%	24460.51
		4	13677	48%	14091.61	26591	0%	19405.75	**10545**	**60%**	**11606.9**	26605	0%	19619.94
		5	2512	82%	6808.15	13836	0%	10748.16	**897**	**94%**	**3416.67**	13834	0%	10825.77
50	40	1	676	91%	1480.96	7732	1%	5584.6	**552**	**93%**	**1345.31**	7733	1%	5671.79
		2	2647	90%	5622.66	25054	5%	16360.6	**1957**	**93%**	**4669.37**	25580	3%	17186.4
		3	1098	92%	2515.56	12988	0%	9252.69	**806**	**94%**	**2000.58**	12983	0%	9393.21
		4	2829	91%	5795.49	31646	0%	21040.4	**2190**	**93%**	**4977.14**	31635	0%	21386.51
		5	624	91%	1363.52	6919	1%	4645.85	**486**	**93%**	**1117.88**	6975	1%	4723.95
60	50	1	188857	45%	347286.43	–	–	–	**165753**	**52%**	**320449.04**	–	–	–
		2	6249	93%	38959.57	92866	0%	143895.79	**5315**	**94%**	**35467.4**	92855	0%	145203.91
		3	13691	95%	**78454.3**	–	–	–	**13599**	**95%**	79633.35	–	–	–
		4	18003	84%	76189.77	–	–	–	**7534**	**93%**	**45705.74**	–	–	–
		5	87746	85%	395934.76	–	–	–	**47478**	**92%**	**275751.36**	–	–	–

6.2 Benchmark instances

In this section, we present our computational experiments conducted on a few benchmark problem classes from MIPLIB. These results are given in Table 6.2, where the first four instances are taken from MIPLIB–2017 [32], and the rest are taken from MIPLIB 2.0 [33]. We chose instances that generated search trees that are large enough for a meaningful comparison, but small enough for the algorithms to run without a memory error. We again used a fixed branching order, based on the order in which variables appear in the input file. The first column of Table 6.2 indicates the name of the problem. The next two columns represent the number of variables and constraints of instances studied for each class.

For larger instances, applying the CGLP algorithm at each *layer* of the B&B tree can be extremely expensive; we use the term "layer" here to not only represent the depth of a node, but also the specific branching order. By choosing a subset of node layers, we can reduce the running time of the cut-generating algorithm. We record the layer numbers in which the CGLP is applied in set K as denoted in the fourth column of Table 6.2. Since the CGLP used for both cutting plane types is obtained from disjunctions associated with unfixed variables, we can reduce the size of the CGLP by choosing a subset of the disjunction candidates, thereby reducing the time spent to solve it. We record the layer numbers corresponding to disjunctions used for the CGLPs in set L as denoted in the fifth column of Table 6.2. In these two columns, "all" indicates that cut-generating algorithms are applied to all node layers, and all disjunctions corresponding to unfixed variables are used. It is intuitive to pick top layers of the B&B tree to be included in K, and choose variable indices at the bottom layers of the tree to be included in L, as illustrated in Figure 6.2. For instances with smaller than 100 variables, we set K as the top 50 variables and L as the bottom 25 variables in the branching order. For instances with larger than 100 variables, we shift K to include the second 50 variables at the top of the B&B tree to accommodate for the increase in the problem size.

Columns 6–8 of Table 6.2 under "Consistency cut" contain the result of applying consistency cuts, whereas columns 9–11 show the result of applying RLT cuts. Similarly to the synthetic problems, the consistency cuts achieve a substantial reduction in the B&B size in most of the

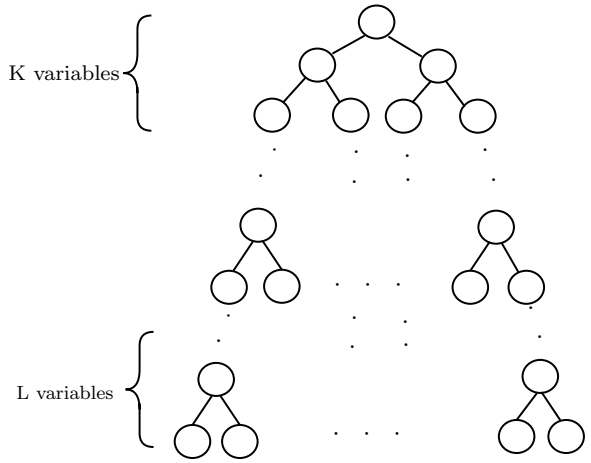

Fig. 6.2: The choice of K and L in applying the CGLP

Table 6.2: Evaluating consistency cuts for benchmark instances

Class	m	n	K	L	Consistency cut			RLT cut		
					nodes	Δ (%)	time (s)	nodes	Δ (%)	time (s)
`stein15inf`	37	15	all	all	**20**	**73%**	**1.64**	75	0%	3.37
`p0201`	133	201	50–100	176–201	**533**	**37%**	**514.82**	847	0%	518.77
`f2gap40400`	40	400	50-100	375–400	**780**	**9%**	**304.33**	861	0%	662.36
`mod008inf`	7	319	50-100	294–319	**65**	**99%**	**684.24**	57495	3%	35655.92
`bm23`	20	27	all	all	**38**	**79%**	**13.57**	178	0%	18.7
`p0033`	15	33	all	all	**321**	**98%**	**180.35**	22581	0%	4761.49
`p0040`	23	40	all	all	**30**	**40%**	31.10	50	0%	**26.53**
`pipex`	41	48	all	all	**547**	**28%**	1415.34	762	0%	**1361.64**
`sentoy`	30	60	all	all	**29**	**89%**	**79.92**	258	0%	151.6
`stein27`	118	27	all	all	**3900**	**5%**	**1715.20**	4099	0%	2241.77
`enigma`	42	100	1–50	75–100	**27960**	**62%**	**118.29**	40218	45%	423.2
`lseu`	28	89	1–50	64–89	**234450**	**5%**	**3096.16**	247795	0%	4195.62

instances of Table 6.2 compared to the RLT cuts. In all of these instances except `p0040` and `pipex`, the solution time for implementing the consistency cuts is also smaller than that of the RLT cuts.

7 Conclusion

Cutting planes have a crucial role in reducing the size of the branch-and-bound tree. In their traditional implementation, they improve the dual bounds by separating fractional optimal solutions of linear programming relaxations. Yet they also reduce backtracking in the search process by achieving different forms of consistency. In this paper, we investigate the latter role of cutting planes through the lens of a variant of consistency that is suitable for 0-1 integer programming. We introduce a class of cutting planes that is different from traditional cutting planes in both concept and effectiveness. In particular, these cutting planes target infeasible 0-1 partial solutions rather than fractional solutions. We develop methods to derive such inequalities and evaluate their power in comparison with the conventional cutting plane techniques.

Preliminary computational results show the potential of cutting planes in reducing the B&B size and solution time when their objective is separating inconsistent partial assignments rather

than the traditional one of separating fractional solutions of the LP relaxation. This outcome encourages the study of more efficient consistency cuts so as to make their implementation viable inside B&B, much as mixed integer Gomory cuts are faster to generate than RLT cuts even though they are often weaker. We believe that this unexplored role for cutting planes can contribute to our understanding of what makes cutting planes effective when used in branch-and-cut approaches.

References

1. Achterberg T (2007) Conflict analysis in mixed integer programming. Discrete Optimization 4:4–20
2. Achterberg T, Berthold T (2009) Hybrid branching. In: Van Hoeve WJ, Hooker JN (eds) CPAIOR Proceedings, Springer, pp 309–311
3. Balas E (1979) Disjunctive programming. Annals of Discrete Mathematics 5:3–51
4. Balas E (1985) Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. SIAM Journal on Algebraic and Discrete Methods 6:466–485
5. Balas E, Ceria S, Cornuéjols G (1993) A lift-and-project cutting plane algorithm for mixed 0-1 programs. Mathematical Programming 58:295–324
6. Balas E, Ceria S, Cornuéjols G, Natraj N (1996) Gomory cuts revisited. Operations Research Letters 19:1–9
7. Balcan MF, Dick T, Sandholm T, Vitercik E (2018) Learning to branch. arXiv 1803.10150
8. Beame P, Kautz H, Sabharwal A (2003) Understanding the power of clause learning. In: International Joint Conference on Artificial Intelligence (IJCAI 2003)
9. Chai D, Kuehlmann A (2005) A fast pseudo-boolean constraint solver. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24:305–317
10. Chang CL (1970) The unit proof and the input proof in theorem proving. Journal of the ACM 14:698–707
11. Cook W, Coullard CR, Turán G (1987) On the complexity of cutting plane proofs. Discrete Applied Mathematics 18:25–38
12. Cornuéjols G, Lee D (2018) On some polytopes contained in the 0,1 hypercube that have a small Chvátal rank. Mathematical Programming Series B 172:467–503
13. Davarnia D, Hooker JN (2019) Consistency for 0–1 programming. In: Rousseau LM, Stergiou K (eds) CPAIOR Proceedings, Springer, pp 225–240
14. Freuder EC (1982) A sufficient condition for backtrack-free search. Communications of the ACM 29:24–32
15. Gaschnig J (1977) A general backtrack algorithm that eliminates most redundant tests. In: Proceedings, 5th International Joint Conference on AI (IJCAI), p 457
16. Gaschnig J (1978) Experimental studies of backtrack vs. waltz-type vs. new algorithms for satisficing-assignment problems. In: Proceedings, 2nd National Conference of the Canadian Society for Computational Studies of Intelligence, pp 19–21
17. Ginsberg ML (1993) Dynamic backtracking. Journal of Artificial Intelligence Research 1:25–46
18. Ginsberg ML, McAllester DA (1994) GSAT and dynamic backtracking. In: Principles and Practice of Constraint Programming (CP 1994), Springer, Lecture Notes in Computer Science, vol 874, pp 216–225
19. Granot F, Hammer PL (1971) On the use of boolean functions in 0-1 programming. Methods of Operations Research 12:154–184
20. Hooker JN (1988) Generalized resolution and cutting planes. Annals of Operations Research 12:217–239
21. Hooker JN (1989) Input proofs and rank one cutting planes. ORSA Journal on Computing 1:137–145

22. Hooker JN (1992) Generalized resolution for 0-1 linear inequalities. Annals of Mathematics and Artificial Intelligence 6:271–286
23. Hooker JN (1997) Constraint satisfaction methods for generating valid cuts. In: Woodruff DL (ed) Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search, Kluwer, Dordrecht, pp 1–30
24. Hooker JN (2000) Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. Wiley, New York
25. Hooker JN (2002) Logic, optimization and constraint programming. INFORMS Journal on Computing 14:295–321
26. Hooker JN (2012) Integrated Methods for Optimization, 2nd ed. Springer
27. Hooker JN (2016) Projection, consistency, and George Boole. Constraints 21:59–76
28. Hooker JN, Ottosson G (2003) Logic-based Benders decomposition. Mathematical Programming 96:33–60
29. Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. Econometrica 28(3):497–520
30. Leoncini M, Montangero M, Valente P (2019) A parallel branch-and-bound algorithm to compute a tighter tardiness bound for preemptive global EDF. Real-Time Systems 55(2):349–386
31. Linderoth JT, Savelsbergh MWP (1999) A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing 11(2):173–187
32. MIPLIB (2017) The mixed integer programming library. URL `https://miplib.zib.de`
33. MIPLIB 2 (1996) The mixed integer programming library. URL `http://miplib2010.zib.de/miplib2/miplib2.html`
34. Morrison DR, Jacobson SH, Sauppe JJ, E CS (2016) Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization 19:79–102
35. Nieuwenhuis R (2014) The IntSat method for integer linear programming. In: O'Sullivan B (ed) Principles and Practice of Constraint Programming, Springer, pp 574–589
36. Quine WV (1952) The problem of simplifying truth functions. American Mathematical Monthly 59:521–531
37. Quine WV (1955) A way to simplify truth functions. American Mathematical Monthly 62:627–631
38. Rossi F, van Beek P, Walsh T (eds) (2006) Handbook of Constraint Programming. Elsevier
39. Sandholm T, Shields R (2006) Nogood learning for mixed integer programming. In: Workshop on Hybrid Methods and Branching Rules for Combinatorial Optimization, Montréal
40. Savelsbergh MWP (1994) Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing 6(4):445–454
41. Sherali HD, Adams WP (1990) A hierarchy of relaxations between the continuous and convex hull representations for zero–one programming problems. SIAM Journal on Discrete Mathematics 3:411–430
42. Williams HP (1987) Linear and integer programming applied to the propositional calculus. International Journal of Systems Research and Information Science 2:81–100
43. Witzig J, Berthold T, Heinz S (2019) A status report on conflict analysis in mixed integer nonlinear programming. In: Rousseau LM, Stergiou K (eds) CPAIOR Proceedings, Springer, pp 84–94