Strengthening Parametric Disjunctive Cuts

Shannon Kelley [a], Ted Ralphs [a], Aleksandr M. Kazachkov [b]
[a] Department of Industrial and Systems Engineering, Lehigh University
[b] Department of Industrial and Systems Engineering, University of Florida

July 30, 2025

Abstract

We study the performance of parametric disjunctive cuts for solving families of mixed-integer linear optimization problems (MILPs) with fixed dimensions. While these cuts have shown promise, their practical impact remains limited. We propose two subroutines to strengthen parametric disjunctive cuts. The first uses an incumbent solution to prune terms in the disjunction that cannot improve the primal bound. The second identifies when a cut fails to support the disjunctive hull and formulates a linear program to recover tight bounds. Computational results demonstrate that these enhancements yield tighter root relaxations and reduce overall solve times across a benchmark set of instances.

1 Introduction

We develop methods to strengthen the parametric disjunctive cut generation scheme proposed by Kelley et al. [20] for sequences of mixed-integer linear programs (MILPs) that share the same variables and number of constraints. Such fixed-structure problem families arise frequently in practice, particularly within online optimization [28, 23, 16] and decomposition-based approaches like bilevel optimization [27], multi-objective programming [29, 18], Benders decomposition [19], and branch-and-price [6, 15]. As a result, they underpin applications across energy management [28, 22], supply chain management [23, 16, 26], scheduling [6, 26], and revenue optimization [16].

Despite their strength, disjunctive cutting planes are often turned off in production solvers due to their high computational cost [8, 1]. The parametric disjunctive cut framework proposed by Kelley et al. [20] addresses this by amortizing the cost across a sequence of structurally similar MILPs. However, this speed comes at the cost of weakened cuts as instance perturbation increases. Understanding the tradeoff between cut strength and generation speed is critical, particularly because cut generation plays a central role in solver performance [8].

The shared structure of fixed-dimension MILP series makes them ideal candidates for *warm-starting*—reusing information from previously solved instances to accelerate future solves. Prior work has explored warm-starting through solution reuse [22], disjunction reuse [17, 21], solver parameter tuning [24], and branching priority transfer [16]. Of particular relevance to this work is the growing body of research on warm-starting the cut generation process. While much initial effort has focused on selecting from preexisting cuts [12], recent approaches aim to generate new ones. For instance, Becu et al. [7] and Chételat and Lodi [11] propose methods to predict parameters for tighter Gomory Mixed-Integer cuts, and Dragotto et al. [13] does analogously for disjunctive cuts. Distinct from these approaches, Kelley et al. [20] introduces a learning-free framework for parametric disjunctive cut generation applicable to any MILP family with shared dimensions and variable structure. We extend that framework by introducing two subroutines that strengthen the resulting cuts, investigating how to balance cut strength with generation speed.

We build on two key observations. First, when generating cuts from large disjunctions, many terms may be unnecessary because they cannot contain an optimal solution. Pruning these terms before cut generation can tighten the resulting cut and directs branching toward regions more likely to yield optimal solutions. Second, parametric disjunctive cuts do not always support the disjunctive hull. Kelley et al. [20] identify sufficient conditions—namely, fixed coefficient matrices and unchanged feasibility statuses of disjunctive terms and the cuts' generating bases—under which support is guaranteed. When these conditions are violated, the cut may be separated from the disjunctive hull, creating an opportunity for targeted strengthening.

We introduce two subroutines to strengthen parametric disjunctive cuts based on the observations above. First, to address unneeded terms, we use an incumbent solution to identify and discard disjunctive terms that provably cannot contain an optimal solution, resulting in potentially smaller disjunctions and tighter cuts. Second, to guarantee disjunctive hull support when sufficient conditions are not met, we solve a linear program (LP) for each retained term that tightens the cut against that term. The dual solutions of these programs form a Farkas certificate—a linear combination of the original constraints defining the cut—which, when passed to the parametric cut framework of Kelley et al. [20], ensures the resulting inequality supports the disjunctive hull.

Contributions. Our theoretical results show that disjunctions pruned using an incumbent solution still yield valid cuts for any optimal solution, allowing for smaller, more focused disjunctions without sacrificing correctness. We also prove that the dual solutions to our cut-tightening LPs are the unique Farkas certificates defining cuts that support individual disjunctive terms. When these certificates are used in the parametric framework of Kelley et al. [20], the resulting cuts are guaranteed to support the full disjunctive hull. Empirically, both strengthening routines, individually and in combination, can produce tighter root relaxations relative to and preserve most of the speed advantage of the original parametric framework. At the branch-and-cut level, the best among our strengthened variants consistently outperforms the baseline parameterization in solve time and relative time improvements. Overall, generating some form of disjunctive cut improves solve time more often than not across our test set.

Outline. The remainder of this paper is organized as follows. Section 2 introduces notation and summarizes relevant background needed to understand our cut generation framework. Section 3 presents the theoretical results that underpin our strengthening routines, including conditions for validity and disjunctive hull support. Section 4 details our computational experiments, highlighting the impact of our methods on both root relaxation tightness and overall solve time. Finally, Section 5 concludes with a discussion of the broader implications of our findings and directions for future work.

2 Notation and Preliminaries

We study methods for strengthening *parametric disjunctive inequalities* to further tighten the linear programming (LP) relaxation of a mixed-integer linear program (MILP) drawn from a structured family $\mathcal{K}$ of related problems. While our techniques apply as long as the set of integer-constrained variables is consistent across instances, we assume for simplicity that all instances are defined over the same variables and number of constraints.

Each instance $k \in \mathcal{K}$ takes the form:

$$\min_{x \in \mathcal{S}^k} \quad c^k x \tag{IP-k}$$

where

$$\mathcal{S}^k = \left\{ x \in \mathcal{P}^k : x_j \in \mathbb{Z} \text{ for all } j \in \mathcal{I} \right\} \quad \text{and} \quad \mathcal{P}^k = \left\{ x \in \mathbb{R}^n : A^k x \geq b^k \right\},$$

with $A^k \in \mathbb{Q}^{q \times n}$, $b^k \in \mathbb{Q}^q$, and $c^k \in \mathbb{Q}^{1 \times n}$. We assume the constraints encode $x \geq 0$ and all additional variable bounds. For any positive integer n, let $[n] := \{1, \ldots, n\}$, and denote the index set of integer variables by $\mathcal{I} \subseteq [n]$. For matrix A, we represent its i^{th} row as $A_{i\cdot}$ and its j^{th} column as $A_{\cdot j}$.

To strengthen the LP relaxation $\mathcal{P}^k$ without excluding any integer-feasible points from $\mathcal{S}^k$, we rely on *disjunctive cuts*, which are valid inequalities derived from valid disjunctions.

Definition 2.1. *A* ***valid inequality*** *for a set $\mathcal{S} \subseteq \mathbb{R}^n$ is a pair $(\alpha, \beta) \in \mathbb{R}^n \times \mathbb{R}$ such that*

$$\mathcal{S} \subseteq \{x \in \mathbb{R}^n : \alpha^\mathsf{T} x \geq \beta\}.$$

Definition 2.2. *A* ***valid disjunction*** *for a set $\mathcal{S} \subseteq \mathbb{R}^n$ is a collection $\mathcal{X} := \{\mathcal{X}^t\}_{t \in T}$, where T is an index set, $\mathcal{X}^t \subset \mathbb{R}^n$ for $t \in T$, and $\mathcal{S} \subseteq \mathcal{X}$.*

Throughout this work, we assume $\mathcal{X}$ subdivides dimensions corresponding to integer bound variables only. Each disjunctive term takes the form

$$\mathcal{X}^t := \left\{x \in \mathbb{R}^n : D^t x \geq D_0^t\right\},$$

where $D^t \in \mathbb{Q}^{q_t \times n}$ and $D_0^t \in \mathbb{Q}^{q_t}$.

We frequently reference the feasible region of an instance restricted to a disjunctive term:

$$A^{kt} := \begin{bmatrix} A^k \\ D^t \end{bmatrix}, \quad b^{kt} := \begin{bmatrix} b^k \\ D_0^t \end{bmatrix}, \quad \text{and} \quad \mathcal{Q}^{kt} := \mathcal{P}^k \cap \mathcal{X}^t = \left\{x \in \mathbb{R}^n : A^{kt} x \geq b^{kt}\right\}.$$

Additionally, we denote the subset of disjunctive terms that are feasible for IP-$k$as $\{\mathcal{X}^t\}_{t \in T^k_{\text{feas}}} := \{t \in T : \mathcal{Q}^{kt} \neq \emptyset\}$.

We also refer to constraint sets that are active at basic solutions. Let $\mathcal{A}^t \subseteq [q + q_t]$ denote such a subset of n constraints of $\mathcal{Q}^{kt}$. We call this an *active constraint set.* Its feasibility parallels that of a basis—i.e., it corresponds to a basic solution that belongs to $\mathcal{Q}^{kt}$.

Definition 2.3. *An active constraint set $\mathcal{A}^t$ is* ***feasible*** *for $\mathcal{Q}^{kt}$ if the associated basic solution*

$$(A^{kt}_{\mathcal{A}^t \cdot})^{-1} b^{kt}_{\mathcal{A}^t}$$

lies in $\mathcal{Q}^{kt}$.

Although feasibility is typically discussed in terms of the basis, each basis has a unique active constraint set. Since our theoretical results are more naturally expressed in terms of active constraint sets—yet occasionally require referencing feasibility—we will refer to $\mathcal{A}^t$ as a *basis* when context allows.

We refer to the collection of feasible regions defined by all terms of a disjunction as the *disjunctive hull*, defined as

$$\mathcal{P}_D^k := \operatorname{cl} \operatorname{conv} \left(\bigcup_{t \in T} \mathcal{Q}^{kt} \right).$$

It is with respect to this region that we aim to strengthen the cuts produced by the parametric disjunctive cut framework of Kelley et al. [20]. Before presenting our methodology, we review key concepts and definitions that will serve as a foundation for the remainder of the paper.

2.1 Disjunctive Cut Generation

Valid inequalities for the disjunctive hull $\mathcal{P}_D^k$ can be generated by solving a linear program — typically a *Cut Generating LP (CGLP)* [3, 9] or a *Point-Ray LP (PRLP)* [25, 4]. Regardless of the formulation, the solution defines an inequality $(\alpha, \beta) \in \mathbb{R}^n \times \mathbb{R}$ that satisfies:

$$\left.\begin{array}{l} \alpha^\mathsf{T} = v^t A^{kt} \\ \beta \leq v^t b^{kt} \\ v^t \in \mathbb{R}^{1 \times (q + q_t)}_{\geq 0} \end{array}\right\} \text{ for all } t \in T. \tag{FL}$$

By *Farkas' Lemma* [14], this condition is equivalent to the inequality (α, β) being valid for $\mathcal{P}_D^k$. We refer to $\{v^t\}_{t \in T}$ as the *Farkas multipliers* or *Farkas certificate* for (α, β) relative to the disjunction $\mathcal{X}$ and instance IP-k.

The method used to compute $\{v^t\}_{t\in T}$ depends on the LP formulation. In the CGLP, condition (FL) is included directly in the formulation, making $\{v^t\}_{t\in T}$ part of the solution. In contrast, the PRLP reduces the dimensionality of the problem by eliminating the Farkas certificate variables [25], allowing for larger disjunctions and, consequently, stronger cuts [4]. For this reason, we adopt the PRLP when generating disjunctive cuts from an LP. The corresponding Farkas certificate is then reconstructed post hoc, as described by Balas and Kazachkov [5, Lemma 3]. In practice, each term v^t is computed from $\mathcal{A}^t$ and $\mathcal{Q}^{kt}$, selected to represent the closest feasible basis to the cut (α, β) for term t. If $\mathcal{Q}^{kt}$ is infeasible and no such basis exists, Kelley et al. [20] assign $v^t = 0$ by default.

Throughout this work, we require a consistent framework for referring to instance–disjunction pairs and the bases from which Farkas certificates are derived. As any disjunctive cut—and its associated Farkas certificate—computed via an LP must satisfy (FL), we use this condition to formalize when a certificate is said to originate from a particular instance–disjunction pair.

Definition 2.4. *A MILP instance IP-$k$and disjunction $\{\mathcal{X}^t\}_{t\in T}$* ***induce*** *the Farkas certificate v^t if $v^{t_1}A^{kt_1} = v^{t_2}A^{kt_2}$ for all $(t_1, t_2) \in \{\mathcal{X}^t\}_{t\in T^k_{feas}} \times \{\mathcal{X}^t\}_{t\in T^k_{feas}}$.*

To identify the basis used to construct the Farkas coefficients for a given feasible disjunctive term $\mathcal{Q}^{kt}$, we associate v^t with $\mathcal{A}^t$, an active constraint set of $\mathcal{Q}^{kt}$, which contains the indices of the constraints whose linear combination produces the cut coefficients α.

Definition 2.5. *The active constraint set $\mathcal{A}^t$* ***determines*** *the Farkas certificate v^t if $\{i \in [q + q_t] : v^t_i > 0\} \subseteq \mathcal{A}^t$.*

2.2 Disjunctive Cut Parameterization

Although the generation of valid disjunctive cuts is conceptually straightforward, it remains computationally expensive in practice [4, 10]. As a result, such cuts are frequently disabled in commercial MILP solvers [8, 1]. This motivates the use of *parametric disjunctive cut generation* [20], which amortizes the cost of cut construction across a family of related MILPs.

Given a Farkas certificate $\{v^t\}_{t\in T}$ satisfying (FL) with respect to disjunction $\{\mathcal{X}^t\}_{t\in T}$ and instance IP-k, the framework of Kelley et al. [20] enables subsequent instances IP-ℓ to generate valid inequalities for $\mathcal{P}^\ell_D$ without solving an additional linear program. For clarity, we use the indices k and ℓ to denote, respectively, the instance that induces the Farkas certificate and the instance to which it is applied.

Theorem 2.6. *Let $\ell \in \mathcal{K}$ be the index of an arbitrary instance in a series. Then given a disjunction $\{\mathcal{X}^t\}_{t\in T}$ and a set $\{v^t\}_{t\in T}$ of nonnegative Farkas multipliers associated with each term of this disjunction, we have that $\alpha^\mathsf{T} x \geq \beta$ for all $x \in \mathcal{P}^\ell_D$, where $\alpha_j := \max_{t\in T}\{v^t A^{\ell t}_{\cdot j}\}$ and $\beta := \min_{t\in T}\{v^t b^{\ell t}\}$ for all $j \in [n]$.*

2.3 Sufficient Conditions for Disjunctive Hull Supporting Cuts

A natural question arises when generating a parametric disjunctive cut (α, β) for a disjunction $\{\mathcal{X}^t\}_{t\in T}$ and instance IP-k: does the cut support the disjunctive hull $\mathcal{P}^k_D$?

Definition 2.7. *The inequality (α, β) valid for a polyhedron $\mathcal{P}$* ***supports*** *$\mathcal{P}$ if there exists $x \in \mathcal{P}$ such that $\alpha^\mathsf{T} x = \beta$.*

Unlike many disjunctive cuts generated from solving a LP, parametric disjunctive cuts generated using Theorem 2.6 do not always support the disjunctive hull. However, Kelley et al. [20] identify sufficient conditions under which support is guaranteed: the coefficient matrix remains fixed, no terms of $\{\mathcal{X}^t\}_{t\in T}$ infeasible for IP-$k$become feasible for IP-ℓ, and no basis $\mathcal{A}^t$ becomes infeasible for $\mathcal{Q}^{\ell t}$.

Lemma 2.8. *Let $k, \ell \in \mathcal{K}$. Let $\{\mathcal{X}^t\}_{t\in T}$ be a disjunction, $\{v^t\}_{t\in T}$ be farkas multipliers induced by IP-$k$and $\{\mathcal{X}^t\}_{t\in T}$, and $\{\mathcal{A}^t\}_{t\in T}$ be a collection of constraint indices determining $\{v^t\}_{t\in T}$. If*

1) IP-ℓ and $\{\mathcal{X}^t\}_{t\in T}$ induce $\{v^t\}_{t\in T}$

2) and $\mathcal{A}^t$ is feasible for $\mathcal{Q}^{\ell t}$ for all $t \in T$,

then $(\alpha, \beta) =$ Theorem 2.6$(\ell, \{\mathcal{X}^t\}_{t\in T}, \{v^t\}_{t\in T})$ supports $\operatorname{cl}\operatorname{conv}(\cup_{t\in T}\mathcal{Q}^{\ell t})$.

The cut produced by Theorem 2.6 is guaranteed to support the disjunctive hull $\mathcal{P}_D^\ell$ when the above conditions hold. Condition 1 can fail in two ways: either the coefficient matrix changes, or a disjunctive term that was infeasible for IP-$k$becomes feasible for IP-ℓ. Condition 2 fails when a basis $\mathcal{A}^t$ becomes infeasible for some $\mathcal{Q}^{\ell t}$. In practice, the latter can often be avoided by removing infeasible terms from the disjunction prior to parameterization, reducing the risk to cases where a previously feasible basis becomes infeasible for a still-feasible term. In any of these scenarios, Theorem 2.6 may fail to produce a cut that supports $\mathcal{P}_D^\ell$. In Section 3.2, we present a method for recovering disjunctive hull support when this occurs.

3 Theoretical Results

Strengthening parametric disjunctive cuts can be achieved in two ways: by reducing the size of the disjunction or by tightening the cuts to ensure they support it. Section 3.1 and Section 3.2 detail each approach, respectively.

3.1 Prune the Disjunction with an Incumbent Solution

We can use the primal bound of a known solution to fathom disjunctive terms where better solutions could not possibly exist. Therefore even though the cut generated against this subset of the disjunctive hull is not valid for all integer feasible solutions, it is still valid for the optimal ones, as demonstrated in Figure 1.

Theorem 3.1. *Let $\ell \in \mathcal{K}$, $\{\mathcal{X}^t\}_{t\in T}$ be a disjunction, and $\{v^t\}_{t\in T}$ be Farkas multipliers. Let $x' \in \mathcal{S}^\ell$, $z^{\ell t} = \min_{x\in\mathcal{Q}^{\ell t}}\{c^\ell x\}$ for all $t \in T$, and $T' \subseteq T$. If $T' = \{t \in T : z^{\ell t} \le c^\ell x'\}$, then $(\alpha, \beta) =$ Theorem 2.6$(\ell, \{\mathcal{X}^t\}'_{t\in T_{T'}}, \{v^t\}_{t\in T_{T'}})$ is a valid inequality for all $x^* \in \arg\min_{x\in\mathcal{S}^\ell}\{c^\ell x\}$.*

Proof. Let $x^* \in \arg\min_{x\in\mathcal{S}^\ell}\{c^\ell x\}$. Since $\mathcal{S}^\ell \subseteq \{\mathcal{X}^t\}_{t\in T}$, there exists $t \in T$ such that $x^* \in \mathcal{Q}^{\ell t}$. Suppose, for the sake of contradiction, that $x^* \in \mathcal{Q}^{\ell t}$ for some $t \in T \setminus T'$. Because $x^* \in \mathcal{Q}^{\ell t}$, we have $c^\ell x^* \ge z^{\ell t}$. Furthermore, since $t \in T \setminus T'$, it follows that $z^{\ell t} > c^\ell x'$. However, by definition of $x^* \in \arg\min_{x\in\mathcal{S}^\ell}\{c^\ell x\}$, we must have $c^\ell x' \le c^\ell x^*$. This results in a contradiction:

$$c^\ell x^* \ge z^{\ell t} > c^\ell x' \le c^\ell x^*.$$

Therefore, $t \in T'$, and we conclude that $x^* \in \operatorname{cl}\operatorname{conv}\bigcup_{t\in T'}\mathcal{Q}^{\ell t}$. By Theorem 2.6, the cut (α, β) is valid for $\operatorname{cl}\operatorname{conv}\bigcup_{t\in T'}\mathcal{Q}^{\ell t}$, and thus valid for x^*. □

Pruning these terms before cut generation can tighten the resulting cut and directs branching toward regions that yield optimal solutions. The idea here is that by preventing branching decisions early in the tree, this could have a dramatic impact on performance given the exponential nature of the search space.

3.2 A Two Step Approach to Generate Disjunctive Hull Supporting Cuts

To strengthen parametric disjunctive cuts and ensure they support the disjunctive hull, we must address the three scenarios identified by Kelley et al. [20] in which Theorem 2.6 may fail to produce a supporting cut: (1) the coefficient matrix changes, (2) a disjunctive term infeasible for IP-$k$becomes feasible for IP-ℓ, and (3) a basis $\mathcal{A}^t$ becomes infeasible for $\mathcal{Q}^{\ell t}$. This subsection proceeds in two parts. First, we *reparameterize* by computing a new Farkas certificate for each relevant disjunctive term. Then, we *regenerate* the parametric disjunctive cut using the updated certificates to ensure the resulting inequality supports the disjunctive hull. Figures 2 to 4 illustrate these steps in each of the three failure cases

How to tighten parametric disjunctive cuts:

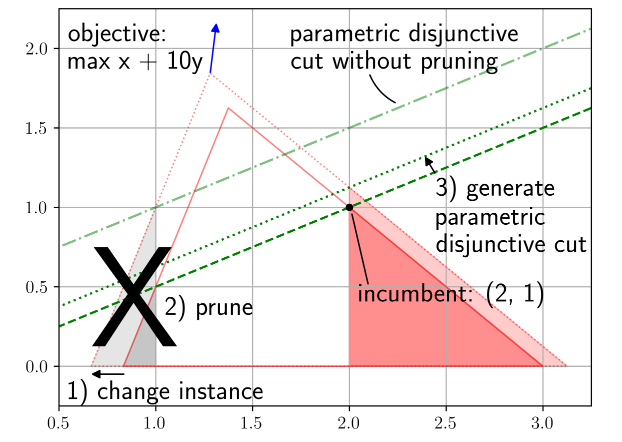

Figure 1: Pruning the disjunction with a known solution

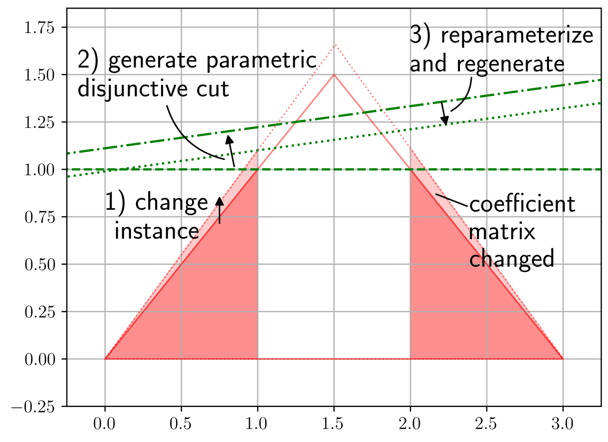

Figure 2: Restoring disjunctive hull support under matrix perturbation

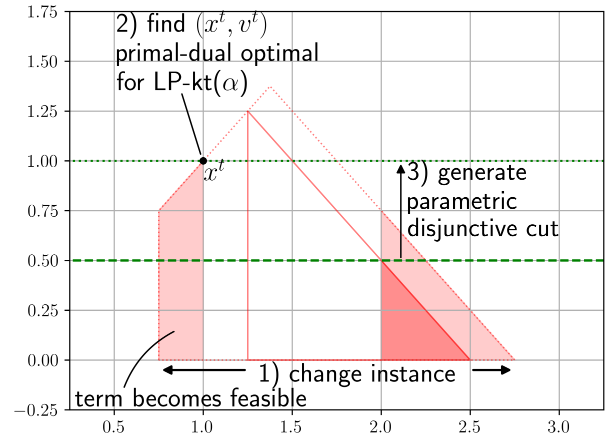

Figure 3: Finding a Farkas certificate, v^t, for perturbation-induced feasible terms

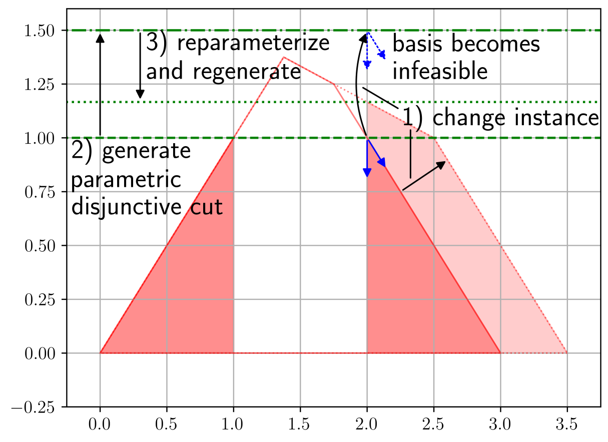

Figure 4: Restoring disjunctive hull support for perturbation-induced infeasible bases

3.2.1 Reparameterize to Find Supporting Certificates

We refer to the process of obtaining a supporting cut and its corresponding *supporting certificate*—the Farkas certificate that generates the supporting cut—as *reparameterization.* As shown in Lemma 3.2, this involves solving the primal-dual pair LP-$\ell t(\alpha)$ and Dual-$\ell t(\alpha)$

$$\min_{x \in \mathcal{Q}^{\ell t}} \quad \alpha^\intercal x \qquad \text{(LP-}\ell t(\alpha)) \qquad\qquad \max_{v \in \mathcal{D}^{\ell t}(\alpha)} \quad v b^{\ell t} \qquad \text{(Dual-}\ell t(\alpha))$$

where $\mathcal{D}^{\ell t}(\alpha) = \{v \in \mathbb{R}^{1 \times (q+q_t)} : vA^{\ell t} = \alpha,\ v \geq 0\}$.

Lemma 3.2. *Let $\ell \in \mathcal{K}$, $t \in T$, and $\alpha \in \mathbb{R}^n$. If $\mathcal{Q}^{\ell t}$ is bound and nonempty, then $(\alpha, \alpha^\intercal \bar{x})$ such that $\bar{x} \in \arg\min_{x \in \mathcal{Q}^{\ell t}}\{\alpha^\intercal x\}$ is a supporting inequality for $\mathcal{Q}^{kt}$ with supporting certificate $\bar{v}^t \in \arg\max_{v^t \in \mathcal{D}^{\ell t}(\alpha)}\{v^t b^{\ell t}\}$.*

Proof. Let $\bar{x} \in \arg\min_{x \in \mathcal{Q}^{\ell t}}\{\alpha^\intercal x\}$ and $\bar{v}^t \in \arg\max_{v^t \in \mathcal{D}^{\ell t}(\alpha)}\{v^t b^{\ell t}\}$. Since $\bar{x}$ optimal for LP-$\ell t(\alpha)$, we have that $\alpha^\intercal x \geq \alpha^\intercal \bar{x}$ for all $x \in \mathcal{Q}^{\ell t}$, which means that $\alpha, \alpha^\intercal \bar{x}$ supports $\mathcal{Q}^{kt}$. Since $\bar{v}^t$ is feasible for Dual-$\ell t(\alpha)$, we have that $\bar{v}^t A^{\ell t} = \alpha$. By stong duality, $\bar{v}^t b^{\ell t} = \alpha^\intercal \bar{x}$, which means that $\bar{v}^t$ is a certificate for the cut $(\alpha, \alpha^\intercal \bar{x})$ and is thus supporting. $\square$

Further, provided there is no constraint in $\mathcal{Q}^{\ell t}$ parallel to α, the only supporting cut with coefficients α and a non-negative Farkas certificate is the one derived from the optimal solutions to the primal–dual pair LP-$\ell t(\alpha)$ and Dual-$\ell t(\alpha)$. This is essential, since Theorem 2.6 guarantees the validity of parametric disjunctive cuts only when the Farkas certificate is non-negative.

Corollary 3.3. *Let $\ell \in \mathcal{K}$, $t \in T$, and $\alpha \in \mathbb{R}^n$. If $\mathcal{Q}^{\ell t}$ is bound and nonempty and $\alpha \neq A^{\ell t}_{i\cdot}$ for all $i \in [q + q_t]$, then there exists unique $\bar{x} \in \mathcal{Q}^{\ell t}$ and unique $\bar{v}^t \in \mathcal{D}^{\ell t}(\alpha)$ such that $\alpha^\intercal \bar{x} = \bar{v}^t b^{kt}$.*

Proof. Lemma 3.2 provides existence of $\bar{x}$ and $\bar{v}^t$. Since there does not exist $i \in [q + q_t]$ such that $\alpha = A^{kt}_{i\cdot}$, pivoting from $\bar{x}$ to another $x \in \mathcal{Q}^{kt}$ would violate dual feasibility. Therefore, $\bar{x} \in \mathcal{Q}^{kt}$ and $\bar{y} \in \mathcal{D}^{\ell t}(\alpha)$ such that $\alpha^\intercal \bar{x} = \bar{y}^\intercal b^{kt}$ are unique. □

A couple of observations are worth noting here. First, because the Farkas certificate that generates a supporting cut is almost always unique for given cut coefficients and disjunctive term, solving LP-$\ell t(\alpha)$ is unavoidable. However, in practice, we can often expedite this step by storing the basis that originally determined the certificate for each term. As a result, re-optimizing after small perturbations is typically fast. Second, this procedure is agnostic to which specific hypothesis of Lemma 2.8 is violated. Its sole objective is to recover a certificate for given cut coefficients and disjunctive term such that the resulting inequality supports that term. This is precisely the condition that, as we show next, is required for Theorem 2.6 to produce parametric disjunctive cuts that support the disjunctive hull.

3.2.2 Regenerate to Yield Supporting Parametric Disjunctive Cuts

We leverage Lemma 3.2 to construct a Farkas certificate that satisfies the conditions of Lemma 2.8, ensuring that Theorem 2.6 produces a cut supporting the disjunctive hull. This process is detailed in Algorithm 1.

Algorithm 1 generateStrongParameterizedDisjunctiveCut

```
Require: $k, \ell, \{\mathcal{X}^t\}_{t\in T}, \{v^t\}_{t\in T}, \{\mathcal{A}^t\}_{t\in T}$
Ensure: IP-$k$and $\{\mathcal{X}^t\}_{t\in T}$ induce $\{v^t\}_{t\in T}$
Ensure: $\{\mathcal{A}^t\}_{t\in T}$ determines $\{v^t\}_{t\in T}$.
Ensure: There exists $t \in T$ such that $\mathcal{A}^t$ is feasible for $\mathcal{Q}^{\ell t}$ and $v^t \neq 0$.
1: $T_f = \{t \in T : v^t \neq 0, (A^{\ell t}_{\mathcal{A}^t\cdot})^{-1} b^{\ell t}_{\mathcal{A}^t} \in \mathcal{Q}^{\ell t}\}$    ▷ Term has certificate and basis still feasible
2: $\{\bar{v}^t\}_{t\in T} \leftarrow \{v^t\}_{t\in T}$
3: $(\alpha, \beta) \leftarrow$ Theorem 2.6(IP-$\ell$, $\{\mathcal{X}^t\}_{t\in T_f}, \{v^t\}_{t\in T_f}$)    ▷ Find cut coefficients $\alpha$
4: for all $t \in T^\ell_{\text{feas}}$ such that $A^{kt} \neq A^{\ell t}$ or $t \notin T_f$ do    ▷ Terms where support could be lost
5:     $\bar{v}^t \leftarrow \arg\max_{y \in \mathcal{D}^{\ell t}(\alpha)} \{y^\intercal b^{\ell t}\}$    ▷ Find supporting certificate for $\alpha$
6: end for
7: $(\alpha, \bar{\beta}) \leftarrow$ Theorem 2.6(IP-$\ell$, $\{\mathcal{X}^t\}_{t\in T^\ell_{\text{feas}}}, \{\bar{v}^t\}_{t\in T^\ell_{\text{feas}}}$)    ▷ Regenerate with supporting certificate
8: return $(\alpha, \bar{\beta})$
```

We can unpack Algorithm 1 as follows. In Step 3, we apply Theorem 2.6 to generate an initial inequality valid for the disjunctive hull. Then, in Step 5, we solve the dual of LP-$\ell t(\alpha)$ to obtain a supporting certificate for each term $t \in T$ where disjunctive hull support might be lost, corresponding to the scenarios illustrated in Figures 2 to 4. By Lemma 3.2, each certificate is guaranteed to support its corresponding disjunctive term. Finally, in Step 7, with the conditions of Lemma 2.8 satisfied, we reapply Theorem 2.6 to regenerate a cut with coefficients α that supports the disjunctive hull, as shown in Theorem 3.4.

Theorem 3.4. *Let $k, \ell \in \mathcal{K}$. Let $\{\mathcal{X}^t\}_{t\in T}$ be a disjunction, $\{v^t\}_{t\in T}$ be farkas multipliers induced by IP-$k$and $\{\mathcal{X}^t\}_{t\in T}$, and $\{\mathcal{A}^t\}_{t\in T}$ be a collection of constraint indices determining $\{v^t\}_{t\in T}$. If there exists*

$t \in T$ *such that* $\mathcal{A}^t$ *is feasible for* $\mathcal{Q}^{\ell t}$ *and* $v^t > 0$*, then* $\alpha, \bar{\beta}$ = *Algorithm 1(*$k, \ell, \{\mathcal{X}^t\}_{t\in T}, \{v^t\}_{t\in T}, \mathcal{A}^t$*) supports* $\operatorname{cl}\operatorname{conv}(\cup_{t\in T}\mathcal{Q}^{\ell t})$*.*

Proof. Define T_f as in Step 1, which is nonempty due to the existence of $t \in T$ such that $\mathcal{A}^t$ is feasible for $\mathcal{Q}^{\ell t}$ and $v^t > 0$. As prescribed in Step 2, let $\{\bar{v}^t\}_{t\in T} = \{v^t\}_{t\in T}$, and, additionally, let $\{\bar{\mathcal{A}}^t\}_{t\in T} = \{\mathcal{A}^t\}_{t\in T}$. By Lemma 2.8, Step 3 provides $\alpha = \bar{v}^t A^{\ell t}$ for all $t \in T_c$. Consider $t \in T^{\ell}_{\text{feas}} \setminus T_c$. Update $\bar{\mathcal{A}}^t$ such that $A^{\ell t}_{\bar{\mathcal{A}}^t}.\bar{x}^t = b^{\ell t}_{\bar{\mathcal{A}}^t}$ for $\bar{x}^t \in \arg\max_{x\in\mathcal{Q}^{\ell t}}\{\alpha^\mathsf{T} x\}$. Lemma 3.2 proves in Step 5 that $\bar{v}^t \in \mathbb{R}^{1\times n}_{\geq 0}$ and $\alpha = \bar{v}^t A^{\ell t}$. Since $\bar{x}^t$ and $\bar{v}^t$ are dual to each other, we have that $\{i \in [q+q_t] : \bar{v}^t_i > 0\} \subseteq \bar{\mathcal{A}}^t$. Therefore, we have that IP-ℓ and $\{\mathcal{X}^t\}_{t\in T^{\ell}_{\text{feas}}}$ induce $\{\bar{v}^t\}_{t\in T^{\ell}_{\text{feas}}}$, $\{\bar{\mathcal{A}}^t\}_{t\in T}$ determines $\{\bar{v}^t\}_{t\in T^{\ell}_{\text{feas}}}$ and, for all $t \in T^{\ell}_{\text{feas}}$, $\bar{\mathcal{A}}^t$ is feasible for $\mathcal{Q}^{\ell t}$. thus, Lemma 2.8 provides $(\alpha, \bar{\beta})$ in Step 7 supports $\operatorname{cl}\operatorname{conv}(\cup_{t\in T^{\ell}_{\text{feas}}}\mathcal{Q}^{\ell t}) = \operatorname{cl}\operatorname{conv}(\cup_{t\in T}\mathcal{Q}^{\ell t})$. □

The key insight of Theorem 3.4 is that we can input *any* Farkas certificate from a previously generated valid disjunctive inequality and obtain a cut that is guaranteed to support the disjunctive hull. Because the resulting cuts are guaranteed to be as tight as possible, this framework has the potential to significantly improve the performance of MILP solvers, provided that cut generation remains computationally efficient. Moreover, if these strengthened cuts fail to improve the root relaxation, it indicates a limitation in the choice of parameters rather than in the cut generation procedure itself, offering a clear direction for further research.

We demonstrate the effectiveness of Algorithm 1 in Example 3.5, where the algorithm is applied to a case with a perturbed coefficient matrix. The resulting cut continues to support the disjunctive hull, showing that the approach is robust to such changes. Since the same procedure applies to cases involving newly feasible terms (Figure 3) and infeasible bases (Figure 4), this example highlights how Algorithm 1 can restore disjunctive hull support in general.

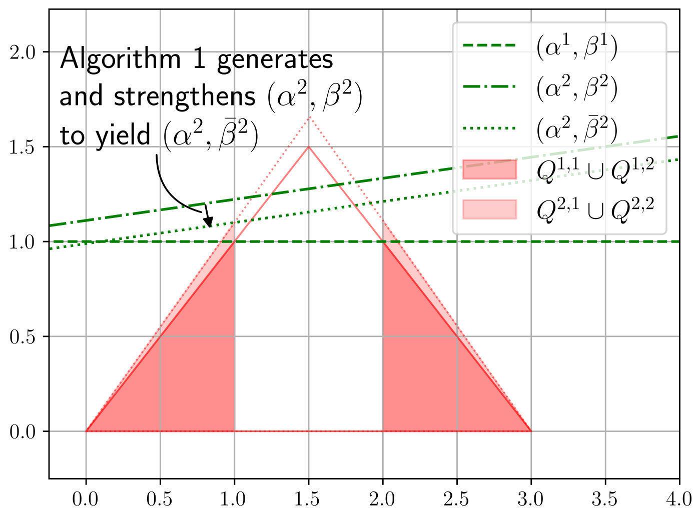

Figure 5: Algorithm 1 ensures parametric disjunctive cuts support the disjunctive hull when $A^1 \neq A^2$.

$$\begin{aligned}
\min_{x\in\mathbb{R}^2} \quad & && -x_2 \\
& x_1 && -x_2 \geq 0 \\
& -x_1 && -x_2 \geq -3 \\
& x_1 && \geq 0 \\
& && x_2 \geq 0 \\
& x_1, && x_2 \in \mathbb{Z}
\end{aligned} \qquad \text{(IP-1)}$$

$$\begin{aligned}
\min_{x\in\mathbb{R}^2} \quad & && -x_2 \\
& 1.1x_1 && -x_2 \geq 0 \\
& -x_1 && -0.9x_2 \geq -3 \\
& x_1 && \geq 0 \\
& && x_2 \geq 0 \\
& x_1, && x_2 \in \mathbb{Z}
\end{aligned} \qquad \text{(IP-2)}$$

Example 3.5. *Let* $\{v^1, v^2\} = \{[1,0,0,0,1], [0,1,0,0,1]\}$ *denote the Farkas certificate induced by IP-*1 *and the disjunction* $\{\mathcal{X}^1, \mathcal{X}^2\} = \{\{x \in \mathbb{R}^2 : x_1 \leq 1\}, \{x \in \mathbb{R}^2 : x_1 \geq 2\}\}$ *from the cut* $(\alpha^1, \beta^1) = ([0,-1], -1)$*, i.e.,* $-x_2 \geq -1$*. To generate a parametric disjunctive cut supporting the disjunctive hull of IP-*2 *and* $\{\mathcal{X}^1, \mathcal{X}^2\}$*, Algorithm 1 proceeds as follows:*

Step 1: Identify disjunctive terms with active constraint sets that are still feasible for IP-2 and have nonzero Farkas multipliers.

$$T_f = \{1, 2\}$$

Step 2: Default the new Farkas certificate $\{\bar{v}^t\}_{t\in T}$ to the values of the old one $\{v^t\}_{t\in T}$.

$$\{\bar{v}^t\}_{t\in T} = \{[1, 0, 0, 0, 1], [0, 1, 0, 0, 1]\}$$

Step 3: Generate an initial valid parametric disjunctive cut for IP-2 and $\{\mathcal{X}^1, \mathcal{X}^2\}$.

$$(\alpha^2, \beta^2) = \textit{Theorem } 2.6(2, \{\mathcal{X}^1, \mathcal{X}^2\}, \{v^1, v^2\}) = ([1, -9], -10)$$

Step 5: To restore disjunctive hull support, recalculate the Farkas certificates for disjunctive terms where it could be lost. For coefficient matrix perturbations, this can include all terms.

$$\bar{v}^1 = \underset{v\in\mathcal{D}^{2,1}(\alpha^2)}{\arg\max} \{vb^{2,1}\} = [.9, 0, 0, 0, .89] \quad \textit{and} \quad \bar{v}^2 = \underset{v\in\mathcal{D}^{2,2}(\alpha^2)}{\arg\max} \{vb^{2,2}\} = [0, 1, 0, 0, 1.1]$$

Step 7: Regenerate the parametric disjunctive cut with the new Farkas certificate.

$$(\alpha^2, \bar{\beta}^2) = \textit{Theorem } 2.6(2, \{\mathcal{X}^1, \mathcal{X}^2\}, \{\bar{v}^1, \bar{v}^2\}) = ([1, -9], -8.9)$$

Since IP-2 and $\mathcal{X}^1, \mathcal{X}^2$ induce and feasible bases determine the Farkas multipliers $\{\bar{v}^1, \bar{v}^2\}$, the resulting cut $x_1 - 9x_2 \geq -8.9$, corresponding to $(\alpha^2, \bar{\beta}^2) = ([1, -9], -8.9)$, supports the disjunctive hull, as shown in Figure 5.

4 Computational Results

The goal of our experiments is to assess how strengthening parametric disjunctive cuts affects performance when solving a series of related MILPs. This section is divided into four parts: the experimental setup (Section 4.1); two motivating examples (Section 4.2); and analyses of the impact on root node processing (Section 4.3) and overall solver performance (Section 4.4). Together, these results highlight the effectiveness of the proposed strengthening methods and their implications for MILP solving.

4.1 Computational Setup

Prior to experimentation, we generated series of randomly perturbed MILP instances based on MIPLIB 2017 and implemented three methods for strengthening parametric disjunctive cuts. These methods were evaluated against a baseline using standard MILP solvers on a server cluster, producing the data analyzed in the sections that follow. This setup provides a controlled and reproducible environment for assessing the impact of disjunctive cuts on solver performance. The **Bold**-font terms introduced below define column headers used throughout Sections 4.2 to 4.4.

4.1.1 Generate Collection of Randomly Perturbed Instances from MIPLIB 2017

We adopt the test instance generation approach from Kelley et al. [20, Section 4.1.1]. Starting with a set of **Base** instances $\mathcal{B}$, which are presolved MIPLIB 2017 problems taking at least 10 seconds to solve and containing no more than 5,000 variables and 5,000 constraints, we construct a collection of sets of **Test** instances indexed by $\mathcal{T}$. Each test set is identified by a tuple $(k, u, \theta) \in \mathcal{B} \times \{A, b, c\} \times \{.5, 2\}$, where k indexes the base instance, u the **Element** perturbed, and θ the perturbation **Degree**. Each instance in a test set is generated by applying Kelley et al. [20, Algorithm 3] to u^k, replacing the selected component with a new version rotated by an angle θ relative to the original. For each (k, u, θ), we run Kelley et al. [20, Algorithm 3] iteratively, bound by the following criteria: reaching 1,000 attempts, exceeding 4 hours, or successfully producing 10 test instances. As a result, the number of test instances per base instance may vary.

4.1.2 Three Strengthening Approaches Are Compared to Three Defaults

Our experiments evaluate strategies for strengthening parametric disjunctive cuts. Let d represent the number of disjunctive **Terms** in the Branch-and-Bound queue when a disjunction is constructed from the leaves of the solve's Branch-and-Bound tree. For each $d \in \{4, 64\}$ and test set $(k, u, \theta) \in \mathcal{T}$, we assess six disjunctive cut generation approaches unless stated otherwise.

The first three methods serve as baselines. **Default** follows Kelley et al. [20, Algorithm 4], solving the series $\mathcal{T}_{ku\theta}$ using default solver settings without disjunctive cuts. **CD,CC** corresponds to Kelley et al. [20, Algorithm 5], where a new disjunction is generated for each test instance and used to compute disjunctive cuts from an LP. **PD,PC** is based on Kelley et al. [20, Algorithm 7], where a previously generated disjunction is reused to produce parametric disjunctive cuts as described in Theorem 2.6.

Algorithm 2 Strengthened Parametric Disjunction, Parametric Cuts

```
Require: $d, k, u, \theta, \mathcal{T}$, prune, support
 1: $\{\mathcal{X}^t\}_{t\in T} \leftarrow$ Branch-and-Bound (IP-$k$, $d$)    ▷ Generate initial disjunction
 2: $\Pi^k \leftarrow$ Balas and Kazachkov [4, Algorithm 1](IP-$k$, $\{\mathcal{X}^t\}_{t\in T^k_{\text{feas}}}$)    ▷ Get initial cuts from PRLPs
 3: $\mathcal{V} \leftarrow \emptyset$, $\mathcal{A}_{\text{all}} \leftarrow \emptyset$
 4: for all $(\alpha, \beta) \in \Pi^k$ do.    ▷ For each cut
 5:     $\{v^t\}_{t\in T} \leftarrow \{0_{q+q_t}\}_{t\in T}$, $\{\mathcal{A}^t\}_{t\in T} \leftarrow \{\emptyset\}_{t\in T}$
 6:     for all $t \in \{\mathcal{X}^t\}_{t\in T^k_{\text{feas}}}$ do    ▷ For each feasible term
 7:         $\mathcal{A}^t \leftarrow \mathcal{A}$ such that $A^{\ell t}_{\mathcal{A}\cdot}\bar{x}^t = b^{\ell t}_{\mathcal{A}}$ and $\bar{x}^t \in \arg\max_{x\in\mathcal{Q}^{\ell t}}\{\alpha^{\mathsf{T}} x\}$    ▷ Get term's optimal basis
 8:         $v^t \leftarrow \arg\max_{v\in\mathcal{D}^{kt}(\alpha)}\{vb^{kt}\}$    ▷ Calculate its certificate
 9:     end for
10:     $\mathcal{V} \leftarrow \mathcal{V} \cup \{\{v^t\}_{t\in T}\}$, $\mathcal{A}_{\text{all}} \leftarrow \mathcal{A}_{\text{all}} \cup \{\{\mathcal{A}^t\}_{t\in T}\}$
11: end for
12: $S \leftarrow$ Branch-and-Cut (IP-$k$, $\Pi^k$)    ▷ Initialize the solution pool
13: for all $\ell \in \mathcal{T}_{ku\theta}$ do    ▷ For each test instance
14:     $\Pi^\ell \leftarrow \emptyset$, $T_\ell \leftarrow T^\ell_{\text{feas}}$
15:     if prune then    ▷ If pruning is enabled
16:         $z \leftarrow \min_{x\in\mathcal{S}^k\cap S}\{c^\ell x\}$    ▷ Get primal bound of $S$
17:         $T_\ell = \{t \in T : \min_{x\in\mathcal{Q}^{\ell t}}\{c^\ell x\}\} \leq z\}$    ▷ Prune disjunction with it
18:     end if
19:     for $i \leftarrow 1$ to $|\mathcal{V}|$ do    ▷ For each certificate/active set pair
20:         $\{v^t\}_{t\in T} \leftarrow \mathcal{V}[i]$, $\{\mathcal{A}^t\}_{t\in T} \leftarrow \mathcal{A}_{\text{all}}[i]$, $(\alpha, \beta) \leftarrow (0_n, 0)$
21:         if support then    ▷ If disjunctive hull support desired
22:             $(\alpha, \beta) \leftarrow$ Algorithm 1$(k, \ell, \{\mathcal{X}^t\}_{t\in T_\ell}, \{v^t\}_{t\in T_\ell}, \{\mathcal{A}^t\}_{t\in T_\ell})$    ▷ Generate supporting cut
23:         else
24:             $(\alpha, \beta) \leftarrow$ Theorem 2.6$(\ell, \{\mathcal{X}^t\}_{t\in T_\ell}, \{v^t\}_{t\in T_\ell})$
25:         end if    ▷ Otherwise
26:         $\Pi^\ell \leftarrow \Pi^\ell \cup \{(\alpha, \beta)\}$    ▷ Generate valid cut
27:     end for
28:     $S \leftarrow S \cup$ Branch-and-Cut (IP-$\ell$, $\Pi^\ell$)    ▷ Solve with strengthened cuts
29: end for
```

The remaining three are our proposed strengthening variants, all built on PD,PC. Each is defined by specific settings of the boolean flags `prune` and `support` in Algorithm 2. **Prune** sets `prune` to true and `support` to false, pruning the disjunction using a primal bound before generating cuts, as formalized in Theorem 3.1. When `prune` is false and `support` is true, **Support** calls Algorithm 1 to generate cuts that support the disjunctive hull. **P&S** enables both flags, combining pruning with the generation of hull-supporting parametric disjunctive cuts. Support and P&S are omitted when the objective is perturbed. This case satisfies Lemma 2.8, guaranteeing that parametric disjunctive cuts

Table 1: Average percent integrality gap closed by disjunctive cuts alone and implied by their generating disjunctions, grouped by degree of perturbation, size of disjunction, and element perturbed for test sets generated from `bm23.mps`.

Degree	Terms	Element	CD	PD	CD,CC	P&S	Prune	Support	PD,PC
0.5	4	matrix	14.78	14.78	14.78	14.63	14.62	14.63	14.62
		objective	14.63	14.63	14.63	–	14.63	–	14.63
		rhs	13.85	14.60	13.85	14.22	14.22	14.22	14.22
	64	matrix	71.00	71.35	69.05	67.55	67.05	67.23	66.46
		objective	68.83	71.48	66.74	–	69.37	–	69.37
		rhs	69.03	70.93	66.91	68.17	68.15	68.17	64.77
2	4	matrix	13.38	14.33	13.22	13.00	12.90	13.00	12.90
		objective	14.79	14.79	14.74	–	14.76	–	14.76
		rhs	11.77	13.27	11.61	12.16	12.16	12.16	12.16
	64	matrix	70.44	71.38	67.68	54.79	43.18	49.42	28.17
		objective	66.90	70.41	64.82	–	67.60	–	67.60
		rhs	69.49	63.82	66.88	60.35	52.45	60.35	44.03

from Theorem 2.6 already support the disjunctive hull and rendering Algorithm 1 to have no effect.

Algorithm 2 relies on the subroutines `Branch-and-Cut` and `Branch-and-Bound`. `Branch-and-Cut` solves a MILP using a standard solver (e.g., CBC or Gurobi) with default settings unless otherwise specified. It takes as input the instance IP-k and a pool of cutting planes $\Pi \subseteq \mathbb{R}^{n+1}$. Its output is a set of feasible solutions $S \subseteq \mathcal{S}^k$. `Branch-and-Bound` performs a similar solve but disables cut generation and does not take a cutting-plane pool as input. It returns the terminal disjunction (including all fathomed terms) and is configured according to Balas and Kazachkov [4, Appendix C].

4.1.3 Experiments Assess Disjunctive Cuts' Impact on Gurobi

Our experiments relied on the following software and hardware. We relied on Gurobi 10 as our `Branch-and-Cut` solver, writing a callback to add our cuts at the root node, and CBC 2.10 for `Branch-and-Bound`. We used the C++ library VPC [2] as the implementation for Balas and Kazachkov [4, Algorithm 1]. The experiments are conducted on a server cluster with 16 AMD Opteron Processor 6128's; 2 CPUs and up to 15GB of RAM were dedicated to each experiment. We allowed 3600 seconds each for generating disjunctive cuts as well as for solving each MILP instance.

4.2 High Performing Subset of Experiments

The experiments in Sections 4.3 and 4.4 are designed to investigate two key patterns observed in selected MILP series. First, strengthening parametric disjunctive cut generation can tighten root relaxations with minimal overhead in cut generation time (Tables 1 to 3). Second, it can lead to substantial reductions in overall solve time (Table 4). These findings are examined in detail in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 Strengthening Can Efficiently Tighten Root Relaxations

Tables 1 to 3 are designed to illustrate this first pattern. For all three tables, columns 1 through 3 designate their primary keys: degree of perturbation, the number of disjunctive terms, and the element perturbed, respectively.

Table 1 reports the percentage of integrality gap closed across 20 random perturbations of `bm23.mps`. CD reflects the gap closed by dual bound of the disjunction used by CD,CC, while PD reflects the gap closed by the shared disjunction used by P&S, Prune, Support, and PD,PC. The remaining columns show the percentage of gap closed by the LP relaxation after adding disjunctive cuts from each corresponding method.

The patterns observed in Table 1 are broadly consistent with expectations. Since CD,CC generates cuts valid for the disjunction used in CD, and P&S, Prune, Support, and PD,PC all generate cuts valid for the disjunction used in PD, the dual bounds provided by CD and PD serve as upper bounds on the LP relaxation objective values after cuts are added. As a result, the percentage of integrality gap closed by CD upper bounds that of CD,CC, and similarly, the gap closed by PD upper bounds those of P&S, Prune, Support, and PD,PC.

Since CD,CC generates Farkas certificates optimized for cut depth, while P&S, Prune, Support, and PD,PC reuse previously computed certificates, CD,CC would generally be expected to close more of the integrality gap. This trend is reflected in Table 1. Exceptions arise when PD closes more gap than CD, which is can happen for smaller disjunctions or low degrees of perturbation. In such cases, P&S, Prune, Support, and PD,PC benefit from generating cuts against a tighter relaxation without suffering the degradation that can affect them relative to CD,CC under larger disjunctions or perturbations.

As specified in Algorithm 2, P&S strengthens the cuts generated by both Prune and Support, which in turn strengthen those generated by PD,PC. Accordingly, the gap closed by P&S is always at least as large as those of Prune and Support, which themselves are no smaller than that of PD,PC. Consistent with findings in Kelley et al. [20], gap closure tends to increase with disjunction size, since larger disjunctive hulls yield tighter relaxations. In contrast, it tends to decrease with higher perturbation levels, as LP relaxation solutions may shift to positions where the cuts, although still valid, are weakened.

Figures 2 to 4 show that for perturbations to the constraint matrix or right-hand side, PD,PC may produce cuts that do not support the disjunctive hull. In such cases, stronger cuts can be obtained by pruning disjunction terms or computing a supporting certificate before applying Theorem 2.6. This effect is particularly pronounced at higher perturbation levels or with larger disjunctions, as also reflected in Table 1, since both settings introduce greater potential for cut weakening and, correspondingly, greater opportunity for strengthening.

One intuitively appealing pattern that does not appear, however, is that CD might be expected to consistently upper bound PD. Since CD represents disjunctions tailored to each test instance and PD uses a shared disjunction across the test set, it is natural to expect the former to dominate. Yet this is not consistently the case in Table 1. This discrepancy is not entirely surprising, as disjunctions are derived from partial Branch-and-Bound trees influenced by solver heuristics and settings, and the degree of perturbation may not be large enough to offset this variability for `bm23.mps`.

Table 2 reports the percentage of integrality gap closed by the LP relaxation after root node processing for 20 random perturbations of `bm23.mps`. These values reflect the effect of all cuts generated at the root node, including the disjunctive cuts.

The general trends in Table 2 mirror those in Table 1. As before, CD,CC provides the strongest performance, followed by P&S, then Prune and Support, with PD,PC typically trailing. The integrality gap closed increases with disjunction size and decreases with the level of perturbation. Additionally, the benefit of the strengthening variants P&S, Prune, and Support over PD,PC tends to grow as disjunctions get larger and perturbations more severe. That said, because the interaction between cuts is complex and not fully understood, it is possible for a method with stronger individual cuts to result in a smaller overall gap closure after root node processing, especially when those cuts alone recover less of the dual bound.

New patterns also emerge. One might expect Default to consistently close less of the integrality gap, as it lacks access to disjunctive cuts. This generally holds true, but exceptions occur when disjunctive cuts contribute little to the dual bound on their own. As shown in Balas and Kazachkov [4], in such cases, default cuts can often tighten the same parts of the feasible region, explaining the comparable performance. The converse is particularly important. As shown in Table 2, when

Table 2: Average percent integrality gap closed after root node processing, grouped by degree of perturbation, size of disjunction, and element perturbed for test sets generated from `bm23.mps`.

Degree	Terms	Element	CD,CC	P&S	Prune	Support	PD,PC	Default
0.5	4	matrix	42.63	40.35	40.45	40.62	40.48	42.16
		objective	39.18	–	41.93	–	40.89	39.23
		rhs	43.19	41.87	41.80	41.49	42.00	43.87
	64	matrix	70.30	69.32	69.01	69.06	68.57	42.13
		objective	68.40	–	70.52	–	70.52	39.38
		rhs	68.69	69.41	69.39	69.44	68.08	42.63
2	4	matrix	39.92	40.75	40.91	40.10	41.11	38.71
		objective	40.16	–	39.64	–	39.54	40.02
		rhs	42.92	42.49	41.60	42.49	41.60	41.16
	64	matrix	69.56	59.96	52.73	56.42	48.44	38.82
		objective	67.09	–	69.25	–	69.28	40.01
		rhs	69.34	62.27	58.32	62.25	56.53	41.16

disjunctions are large enough, disjunctive cuts can tighten the root relaxation in ways that Gurobi's default cuts cannot. This same pattern holds when comparing strengthened parametric disjunctive cuts to their default counterparts, underscoring the added value of our proposed methods.

Table 3: Time to process root node, grouped by degree of perturbation, size of disjunction, and element perturbed for test sets generated from `bm23.mps`.

Degree	Terms	Element	CD,CC	P&S	Prune	Support	PD,PC	Default
0.5	4	matrix	0.227	0.122	0.112	0.103	0.113	0.134
		objective	0.200	–	0.112	–	0.122	0.105
		rhs	0.213	0.113	0.120	0.122	0.124	0.162
	64	matrix	1.629	0.253	0.176	0.405	0.176	0.131
		objective	1.661	–	0.174	–	0.173	0.103
		rhs	1.629	0.187	0.172	0.214	0.174	0.127
2	4	matrix	0.219	0.145	0.125	0.143	0.126	0.102
		objective	0.209	–	0.120	–	0.119	0.124
		rhs	0.227	0.134	0.119	0.135	0.118	0.105
	64	matrix	1.702	0.347	0.202	0.548	0.214	0.104
		objective	1.633	–	0.176	–	0.176	0.118
		rhs	1.589	0.251	0.195	0.364	0.198	0.106

Table 3 reports the root node processing time for 20 random perturbations of `bm23.mps`. The observed trends largely align with expectations. As shown in Kelley et al. [20], PD,PC significantly reduces root node time compared to CD,CC by generating disjunctive cuts directly from Farkas certificates, avoiding the need to solve an LP. However, this still incurs some cost, so Default generally provides a lower bound on PD,PC. Similarly, P&S, Prune, and Support tend to fall between CD,CC and Default. While P&S and Support methods do solve LPs like CD,CC, they benefit from warm starts and, in this case, smaller formulations, preserving a time advantage.

We also observe that enabling pruning usually reduces processing time. For example, P&S is typically faster than Support, and Prune is often faster than PD,PC. This is expected, as pruning decreases the number of disjunctive terms and therefore the number of calculations during disjunctive cut generation.

Minor deviations from these patterns are not surprising. Because the interactions between disjunctive and default cuts are not fully known, a slightly more expensive disjunctive cut procedure can occasionally reduce overall processing time by decreasing the need for default cuts later at the root.

The main takeaway is that our strengthening methods preserve much of the time advantage of parametric disjunctive cuts over CD,CC. Combined with the gap-closing results in Table 2, they often deliver tighter relaxations with minimal additional time, confirming the existence of the first pattern we set out to explore.

4.2.2 Strengthening Can Significantly Reduce Solve Time

Table 4 is structured as follows. The first three columns mirror those in Section 4.2.1, indicating the degree of perturbation, number of disjunctive terms, and element perturbed. The fourth column identifies the base instance from which each test series was generated. The next four columns report overall solve times (in seconds) for the corresponding cut generator. The final column gives the sample size, representing the number of instances in each test series.

Table 4: Solve time (in seconds) for a subset of series with significant solve time improvement when strengthening parametric disjunctive cuts with Algorithm 1.

Degree	Terms	Element	Instance	Default	CD,CC	PD,PC	Support	Count
0.5	4	matrix	danoint	1481	1597	1349	657.5	3
		rhs	gen-ip002	1380	1183	1319	695.5	2
	64	matrix	k16x240b	314.8	243.4	245.7	173.6	3
		rhs	timtab1	244.0	251.3	219.0	172.2	4
2.0	4	matrix	timtab1	56.18	55.78	44.02	28.52	2
		rhs	gen-ip036	295.7	250.5	265.3	193.2	4
	64	matrix	gen-ip021	384.3	440.0	359.6	204.9	4
		rhs	icir97_tension	1028	1343	1700	953.8	3

Table 4 varies the degree of perturbation, number of disjunctive terms, perturbed element, and base instance to demonstrate that strengthening-induced solve time improvements persist across a wide range of settings. As shown in Table 1 and later in Table 7, the strength of parametric disjunctive cuts often increases monotonically with both disjunction size and perturbation degree. However, because the relationship between cut strength and solve time remains only partially understood, we do not expect a direct correlation. In some cases, smaller disjunctions or larger perturbations yield greater reductions in solve time. This does not diminish the main takeaway of the table: strengthening parametric disjunctive cuts can substantially reduce solve time relative to Default, CD,CC, and PD,PC across a variety of problem characteristics. In some cases, it even determines whether disjunctive cuts help at all.

4.3 Root Node Performance

Building on the findings of Section 4.2.1, which showed that strengthened parametric disjunctive cuts can tighten the root relaxation without increasing generation time, we now evaluate how broadly this effect holds across a larger set of instances. We restrict attention to test cases where Calc Disj, Calc Cuts completes within the one-hour time limit and where Lemma 2.8 does not hold, allowing the possibility for Algorithm 1 to take effect. Due to an insufficient number of cases with small disjunctions or low perturbation degrees in which pruning took effect, we exclude Prune and P&S results and focus on the Support strengthening method. Table 5 reports the number of qualifying base and test instances for each perturbed element.

Table 5: The number of base and test instances where VPC generation succeeds for all combinations of possible degree of perturbation and terms.

Element	Base Instances	Test Instances
matrix	45	106
rhs	24	49

Table 6: Counts of infeasible nodes that become feasible under perturbation, feasible nodes with supporting bases that become infeasible, and the percentage of cuts with modified coefficients, grouped by perturbed element, degree of perturbation, and number of disjunctive terms, for test instances included in Table 5.

Element	Degree	Terms	Term Δ	Basis Δ	Cut Δ (%)
matrix	0.5	4	0.00	2.27	92.7
		64	1.11	20.9	97.9
	2.0	4	0.02	2.93	99.6
		64	7.71	25.0	99.9
rhs	0.5	4	0.00	3.67	0.00
		64	1.12	40.6	0.00
	2.0	4	0.00	3.67	0.00
		64	2.14	37.0	0.00

4.3.1 Strengthening Closes More Root Optimality Gap

We begin our root node analysis of strengthened parametric disjunctive cuts by evaluating their potential to close additional root optimality gap relative to PD,PC. In Table 6, we quantify this potential by tracking how often, under perturbation, infeasible disjunctive terms become feasible, supporting bases for feasible terms become infeasible, and the coefficients of parametric disjunctive cuts change. These conditions correspond precisely to cases where Lemma 2.8 does not hold, i.e. situations in which tightening the cuts may be possible.

Table 6 reports the changes in status for disjunctions and parametric disjunctive cuts under perturbation. The first three columns separate test instances by the element perturbed, degree of perturbation, and disjunctive terms, respectively. Column 4 gives the number of disjunctive terms on average that go from infeasible to feasible for the given subset of test instances as illustrated by Figure 3. Column 5 gives the average number of feasible terms with supporting bases that become infeasible for the given subset of test instances as illustrated by Figure 4. Column 6 gives the average percentage of parametric disjunctive cuts whose coefficients change for the given subset of test instances as illustrated by Figure 2.

The patterns in Table 6 align with expectations. The frequency of infeasible disjunctive terms becoming feasible, supporting bases for feasible terms becoming infeasible, and changes in the coefficients of parametric disjunctive cuts almost always increase with greater perturbation and larger disjunctions. This is consistent with the intuition that more perturbation and more terms lead to a higher likelihood of status changes in nodes and bases. Similarly, greater perturbation increases the number of entries in the coefficient matrix that change, and larger disjunctions raise the chance that such changes affect the resulting cuts from Theorem 2.6. As expected, we observe no changes in cut coefficients for right-hand side perturbations, since only matrix perturbations can alter coefficients under the Support method. We use these result to help explain the patterns we see, or lack thereof, in Support's ability to close additional root optimality gap relative to PD,PC as shown in Table 7.

Table 7 reports the percentage of integrality gap closed across our collection of test sets. Column

Table 7: Average percent integrality gap closed by disjunctive cuts and implied by disjunctions, grouped by degree of perturbation, size of disjunction, and element perturbed for test sets generated from MIPLIB 2017.

Degree	Terms	Element	CD	PD	CD,CC	Support	PD,PC
0.5	4	matrix	5.96	2.67	3.38	1.90	1.90
		rhs	8.58	3.69	3.32	2.23	2.23
	64	matrix	12.61	7.88	5.94	3.51	3.31
		rhs	17.66	9.99	11.16	6.47	5.21
2.0	4	matrix	8.33	2.53	4.30	1.10	1.10
		rhs	18.63	3.00	3.14	0.48	0.48
	64	matrix	16.26	5.75	7.43	1.18	0.44
		rhs	27.38	7.31	9.13	1.16	1.08

definitions match those in Table 1, with the final three columns showing the gap closed after adding disjunctive cuts from each corresponding method. As before, CD and PD represent the gap closed by the dual bounds of their respective disjunctions.

The patterns observed in Table 7 are generally consistent with earlier findings. CD,CC continues to outperform Support, which in turn outperforms PD,PC. Gap closure increases with disjunction size and tends to decrease with higher degrees of perturbation. Strengthened cuts yield greater benefits as either disjunction size or perturbation level increases, as previously seen. Notably, CD closes more gap at higher perturbation levels, helping to explain why CD,CC also improves under these conditions—a departure from the typical pattern but consistent with the underlying data.

One might expect disjunctive methods to close more of the integrality gap overall, but their performance depends heavily on where the cuts exert their strength. As shown in Balas and Kazachkov [4], disjunctive cuts—such as those in CD,CC, Support, and PD,PC —often tighten regions of the LP relaxation that lie far from the LP relaxation solution. This limitation is amplified in parametric methods, where cuts may be generated from bases no longer near the LP solution. Matrix perturbations can further shift the direction of the cuts, compounding this effect and diminishing the practical strength of the resulting cuts.

Additional insight helps explain why Support often appears to offer limited improvement over PD,PC. If the supporting term for a parametric disjunctive cut does not change, then strengthening has no effect, leaving Support equivalent to PD,PC. Further, in the case of 4-term disjunctions, Support shows no gain over PD,PC, likely due to the small number of infeasible terms that become feasible, as seen in Table 6. This matters because PD,PC assigns a zero vector as the default certificate for infeasible terms. When these terms become feasible, the corresponding change in cut direction and bound can be substantial. Additionally,

Still, this does not mean that disjunctive methods, and Support in particular, lack value. While they may not refine the LP relaxation near its optimal solution, as we will see next, the gap they do close often arises from regions that Gurobi's default cuts do not address. In Section 4.4, it may be this orthogonality that plays a key role in overall solver performance improvements.

Table 8 reports the percentage of integrality gap closed in the LP relaxation after root node processing of our collection of test sets, analogous to Table 2. Again, these values reflect the effect of all cuts generated at the root node, including the disjunctive cuts.

Table 8 shows some trends consistent with those in Table 7. On average, CD,CC performs best and Default worst, and gap closure increases with disjunction size. The benefit of strengthened variants grows under increased perturbation and terms. Improved effectiveness of disjunctions at higher perturbation again leads to greater gap closure across all disjunctive cut generators.

This time, however, the ordering amongst disjunctive cut generators becomes less clear. Default

Table 8: Average percent integrality gap closed after root node processing, grouped by degree of perturbation, size of disjunction, and element perturbed for test sets generated from MIPLIB 2017.

Degree	Terms	Element	CD,CC	Support	PD,PC	Default
0.5	4	matrix	73.26	73.25	73.62	73.34
		rhs	51.76	48.47	49.65	50.20
	64	matrix	73.58	73.59	73.40	73.42
		rhs	53.47	53.02	52.69	49.72
2	4	matrix	80.24	80.39	80.41	80.40
		rhs	52.80	52.45	52.89	53.77
	64	matrix	80.61	80.24	79.39	80.12
		rhs	56.35	53.47	53.17	53.91

Table 9: Time to process root node, grouped by degree of perturbation, size of disjunction, and element perturbed for test sets generated from MIPLIB 2017.

Degree	Terms	Element	CD,CC	Support	PD,PC	Default
0.5	4	matrix	1.39	0.480	0.435	0.314
		rhs	0.690	0.297	0.300	0.275
	64	matrix	11.9	2.14	1.06	0.336
		rhs	6.43	0.744	0.549	0.313
2	4	matrix	2.45	0.377	0.373	0.296
		rhs	0.794	0.248	0.234	0.239
	64	matrix	9.56	1.70	1.21	0.288
		rhs	4.36	0.546	0.515	0.234

cut generation variability can obscure the advantage of stronger cuts, particularly when said cuts are not that much stronger.

Notably, for large disjunctions, Support tightens the root relaxation in ways that both Default and PD,PC cannot, reinforcing the value of the proposed method in general.

4.3.2 Strengthening Maintains Parametric Disjunctive Cut Generation Time

Table 9 shows root node processing times. The trends align with earlier findings in Table 3: PD,PC is significantly faster than CD,CC, while Default typically offers a lower bound. Support falls in between CD,CC and PD,PC, leaning more towards the latter due to the availability of warm starts when it solves LPs. As before, the processing times of all disjunctive methods grows with the number of disjunctive terms. These results indicate in general that strengthened parametric disjunctive cuts retain most of the time advantage over CD,CC while frequently producing tighter root relaxations than PD,PC, completing the first pattern we aimed to validate for collection of test sets generated from MIPLIB 2017.

4.4 Branch-and-Cut Performance

This subsection presents the second main finding of our computational study: strengthening parametric disjunctive cuts can substantially reduce overall solve effort, even compared to both default disjunctive cut generators. We demonstrate this by evaluating the performance of strengthened versus default parametric disjunctive cuts on test instances with solve times between 10 seconds and

Table 10: Average node wins by disjunctive cut generator and degree of perturbation.

Degree	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
0.5	14.88	13.41	33.90	38.05	38.54	53.17	59.27	77	410
2.0	12.00	17.00	37.00	42.00	34.67	52.33	60.67	64	300

Table 11: Average run time wins by disjunctive cut generator and degree of perturbation.

Degree	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
0.5	23.66	20.73	26.34	40.24	37.80	55.37	61.22	77	410
2.0	21.00	21.33	28.00	39.00	40.00	54.67	59.33	64	300

1 hour. Results are grouped by degree of perturbation (Section 4.4.1), number of disjunctive terms (Section 4.4.2), perturbed element (Section 4.4.3), and default solve time bracket (Section 4.4.4).

For each grouping, we compare branch-and-bound node count and solve time. Tables report the average percentage of wins, where a win is defined as achieving at least a 10% improvement. Column 2 specifices the percent of time Default wins against all other methods, and column 3 does analogously for Support. The next five columns then show the percentage of wins over Default for the corresponding generators. These columns include two aggregate categories: **Any D**, which records wins by any disjunctive cut generator, and **Any P**, which aggregates wins across the parametric generators. We also list the number of base and test instances per group.

Tables 10 to 17 summarize these results, offering a comprehensive view of generator performance and underscoring the value of strengthening across a variety of settings.

4.4.1 Disjunctive Cut Wins by Degree of Perturbation

Tables 10 and 11 reports the win percentages of our disjunctive cut generators across different degrees of perturbation, with sample sizes noted at the end.

We observe the following trends for branch-and-bound nodes across perturbation levels, mostly aligning with expectations. Support improves relative to PD,PC as perturbation increases, since greater perturbation creates more opportunity for strengthening. Any P wins less often under higher perturbation, likely because even tight cuts may lie farther from the LP relaxation solution. PD,PC performs significantly worse with more perturbation, consistent with prior findings that default parametric disjunctive cuts can refine suboptimal bases and often fail to support the disjunctive hull. These effects help justify the higher cost of CD,CC, which becomes more advantageous as perturbation increases.

One result deviates from our expectations: we anticipated that Support would degrade with more perturbation—albeit less severely than PD,PC —but it does not. This is plausible, as the 2-degree disjunctions appear stronger than those at 0.5 degrees, as shown in Table 7.

We find no consistent trend emerges in run time when breaking down by degree of perturbation, but this is unsurprising given the unpredictable relationship between disjunctive cuts and simultaneous improvements in node count and solve time [4].

4.4.2 Disjunctive Cut Wins by Number of Disjunctive Terms

Tables 12 and 13 reports the win percentages of our disjunctive cut generators across different sizes of disjunction, with sample sizes noted at the end.

Table 12: Average node wins by disjunctive cut generator and disjunctive terms.

Terms	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
4	13.80	15.01	35.59	39.71	37.77	53.75	59.81	96	413
64	13.47	14.81	34.68	39.73	35.69	51.52	59.93	70	297

Table 13: Average run time wins by disjunctive cut generator and disjunctive terms.

Terms	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
4	17.92	22.76	29.78	42.62	39.47	58.35	64.65	96	413
64	28.96	18.52	23.23	35.69	37.71	50.51	54.55	70	297

Table 14: Average node wins by disjunctive cut generator and element perturbed.

Element	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
matrix	13.91	15.0	35.87	38.91	36.3	52.39	60.0	89	460
rhs	13.20	14.8	34.00	41.20	38.0	53.60	59.6	59	250

When examining branch-and-bound nodes, we see that Support's advantage over PD,PC increases with more disjunctive terms—an expected result, as additional terms introduce more opportunities for weakening under perturbation (Table 6). While one might anticipate that more terms and stronger cuts (CD,CC and Support) would reduce node counts, that pattern does not appear here. This is consistent with the behavior of cuts that close only a small portion of the integrality gap, as shown in Table 7 for our MIPLIB 2017 instances. In such cases, adding cuts can actually increase the number of nodes processed; see **?**] for further discussion.

When examining run time, the expected patterns emerge. Disjunctive methods become less effective as the number of terms increases, due to the growing cost of cut generation. This aligns with the findings of Kelley et al. [20] for parametric disjunctive cuts and Balas and Kazachkov [4] for CD,CC. Given these relationships, it is unsurprising that CD,CC shows a clear separation from the parametric methods in run time, as its cuts are significantly more expensive to generate, again consistent with prior work [20]. As a result, PD,PC degrades the least among the disjunctive methods, while Default improves in relative performance.

4.4.3 Disjunctive Cut Wins by Perturbed Element

Tables 14 and 15 reports the win percentages of our disjunctive cut generators across different elements perturbed, with sample sizes noted at the end.

When examining branch-and-bound nodes, we find that Support consistently outperforms PD,PC, as expected given its stronger cuts. For both methods, rhs exhibits greater root node strength than matrix on average (Table 8), which helps explain its relative performance. One might still expect matrix to outperform rhs due to greater potential for strengthening, especially deeper in the tree. However, even if such strengthening occurs, the non-monotonic behavior observed by **?**] could still explain why this advantage does not translate into fewer nodes.

When examining run time, our results largely align with prior literature. PD,PC degrades more than Support, consistent with Table 6, which shows substantial room for strengthening in both matrix

Table 15: Average run time wins by disjunctive cut generator and element perturbed.

Element	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
matrix	24.78	22.39	25.43	40.00	36.52	54.57	59.78	89	460
rhs	18.40	18.40	30.00	39.20	42.80	56.00	61.60	59	250

Table 16: Average node wins by disjunctive cut generator and default instance run time bracket.

Bracket	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
short	14.43	15.12	32.99	38.83	39.18	53.95	60.14	74	291
medium	14.71	15.07	35.66	39.34	33.82	51.47	59.56	60	272
long	10.20	14.29	38.78	42.18	38.10	53.06	59.86	36	147

Table 17: Average run time wins by disjunctive cut generator and default instance run time bracket.

Bracket	Wins vs. All		Wins vs. Default					Count	
	Default	Support	CD,CC	Support	PD,PC	Any P	Any D	Base	Test
short	32.65	20.27	14.09	36.43	35.40	51.20	53.95	74	291
medium	17.65	22.79	31.25	39.34	41.18	56.99	63.60	60	272
long	11.56	19.05	44.90	46.94	40.82	59.18	67.35	36	147

and rhs perturbations. Table 8 further indicates that disjunctive cuts have a greater average effect on tightening the root relaxation for rhs than for matrix. Given that such cuts are typically dense, they may remain active deeper in the tree and slow down node processing more [4]. This could explain why Support improves over PD,PC for rhs perturbations in terms of nodes processed but loses that advantage when considering run time, while this does not happen for matrix perturbations. Similarly, CD,CC wins less often on run time than on nodes possibly due to its high generation cost and strong cuts, whereas PD,PC, with its weaker and faster cuts, does not degrade as much.

4.4.4 Disjunctive Cuts Are More Advantageous for Longer Solve Times

Tables 16 and 17 reports win percentages for each disjunctive cut generator, grouped by *solve time bracket*, with sample sizes shown at the end. **Short** includes test instances that Default solves in 10–60 seconds, **medium** covers 60–600 seconds, and **long** includes those requiring more than 600 seconds.

When examining nodes processed by solve time bracket, we observe that longer runs increasingly favor Support and CD,CC. This is expected, as both methods produce relatively stronger cuts, and pruning near the root has greater impact in larger branch-and-bound trees given the size of subtrees avoided. The growing advantage of Support with longer solve times aligns with Balas and Kazachkov [4], which observed a similar trend for Calc Disj, Calc Cuts. PD,PC fails to exhibit the same monotonic improvement across brackets, and a key distinction may lie in the fact that both Support and CD,CC generate cuts that support the disjunctive hull. Without disjunctive hull support, PD,PC may not be able to prune as aggressively near the root where subtrees grow with processing time.

When comparing run time win percentages across brackets, we observe that short runs follow expected patterns: disjunctive methods win less often on time than on nodes, likely due to cut density slowing LP solves [4]. Surprisingly, the opposite occurs in long runs, where run time wins exceed node wins. One explanation may be that disjunctive hull-supporting cuts enable the solver to skip more cut generation later, offsetting the cost of denser LPs. Alternatively, we could have shallower trees, which lead to more effective parallelization and improvements in overall solve time despite increased node counts.

4.4.5 Support Is Regularly the Top Performing Disjunctive Cut Generator

Some patterns hold consistently across all groupings and metrics. Across every table, disjunctive cuts (Any D) achieve substantial win rates in both nodes processed and run time, consistent with their known strength. The methods also exhibit orthogonality: a given disjunctive approach often outperforms Default on instances where others do not. Notably, Support outperforms all other methods 15–20% of the time. These findings underscore the practical value of using a diverse set of disjunctive cut generators, including Support, as they complement one another and collectively enhance performance. They also reinforce this paper's second main claim: that strengthening parametric disjunctive cuts can significantly improve overall branch-and-cut effectiveness.

5 Conclusion

Theoretically, our results demonstrate that pruning disjunctions using an incumbent solution yields smaller, more targeted disjunctions without compromising validity, as the resulting cuts remain correct for any optimal solution. We also show that the dual solutions to our cut-tightening LPs uniquely certify support for individual disjunctive terms, and when embedded in the parametric framework of Kelley et al. [20], produce cuts that provably support the full disjunctive hull. Empirically, both strengthening routines, used separately or together, tighten the root relaxation while preserving much of the original framework's speed advantage. At the branch-and-cut level, our best strengthened variant consistently outperforms the baseline in solve time. More broadly, disjunctive cuts improve solve time in the majority of cases across our test set.

These results come with important caveats. It remains an open question both when disjunctive cuts improve MILP performance and which generator will be most effective for a given instance. Prior work has shown that disjunctive cuts tend to reduce the number of nodes more often than overall run time [4, 20], likely because their density increases the cost of node processing. Yet, for our longest solves, we observe the opposite: disjunctive cuts improve run time more often than node count, a result that warrants further investigation. In particular, for matrix perturbations, it may be beneficial to explore generating the supporting certificate from the initial (non-parametric) cut rather than the parameterized form. This would ensure the parametric cut remains parallel to the original and could potentially yield further gains in solver performance.

Several natural extensions emerge from this work. First, given the growing use of learning-based methods to improve MILP performance, a promising direction would be to train a neural network to predict which disjunctive cut generators to apply and under what conditions. Second, since our approach implicitly warm-starts a MILP using a disjunction, it would be valuable to compare its performance with that of explicitly warm-starting the MILP using the same disjunction, to better understand the trade-offs between the two strategies. Finally, our strengthening routine is not limited to disjunctive cuts. Because it can accept arbitrary cut coefficients, it may be worth exploring its use in parameterizing other expensive cuts across related MILPs, such as interdiction cuts in bilevel optimization, using Algorithm 1.

Our results carry several key implications. Theoretically, we provide guarantees for maximizing the strength of parametric disjunctive cuts. Computationally, our findings offer valuable guidance for warm-starting the dual side of a MILP solve, particularly the pool of cutting planes, which is often both neglected in the literature and a frequent bottleneck in practice. From an applied perspective, this opens the door to significantly improving the solve times of real-time and decomposed MILPs, which are increasingly important as problem size and speed demands grow. Overall, this work adds to the growing body of evidence that solver performance can be improved not only by exploiting single-instance structure, but also by leveraging structural patterns across sequences of related problems.

References

[1] COIN-OR Branch and Cut. https://github.com/coin-or/Cbc.

[2] V-polyhedral Disjunctive Cuts. https://github.com/spkelle2/vpc.

[3] Egon Balas. Disjunctive programming. *Ann. Discrete Math.*, 5:3–51, 1979. URL https://www.sciencedirect.com/science/article/pii/S016750600870342X.

[4] Egon Balas and Aleksandr M. Kazachkov. $\mathcal{V}$-polyhedral disjunctive cuts, 2022. URL https://arxiv.org/abs/2207.13619.

[5] Egon Balas and Aleksandr M. Kazachkov. Monoidal strengthening of $\mathcal{V}$-polyhedral disjunctive cuts, 2023. URL https://optimization-online.org/2023/02/monoidal-strengthening-of-simple-v-polyhedral-disjunctive-cuts/.

[6] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: column generation for solving huge integer programs. *Oper. Res.*, 46(3):316–329, 1998. ISSN 0030-364X,1526-5463. doi: 10.1287/opre.46.3.316. URL https://doi.org/10.1287/opre.46.3.316.

[7] Berkay Becu, Santanu S. Dey, Feng Qiu, and Alinson S. Xavier. Approximating the gomory mixed-integer cut closure using historical data, 2024. URL https://arxiv.org/abs/2411.15090.

[8] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—a look back from the other side of the tipping point. volume 149, pages 37–41. 2007. doi: 10.1007/s10479-006-0091-y. URL https://doi.org/10.1007/s10479-006-0091-y. History of integer programming: distinguished personal notes and reminisences.

[9] Binyuan Chen, Simge Küçükyavuz, and Suvrajeet Sen. Finite disjunctive programming characterizations for general mixed-integer linear programs. *Oper. Res.*, 59(1):202–210, 2011. URL https://doi.org/10.1287/opre.1100.0882.

[10] Binyuan Chen, Simge Küçükyavuz, and Suvrajeet Sen. A computational study of the cutting plane tree algorithm for general mixed-integer linear programs. *Oper. Res. Lett.*, 40(1):15–19, 2012. URL https://doi.org/10.1016/j.orl.2011.10.009.

[11] Didier Chételat and Andrea Lodi. Continuous cutting plane algorithms in integer programming. *Operations Research Letters*, 51(4):439–445, 2023. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2023.06.004. URL https://www.sciencedirect.com/science/article/pii/S0167637723001074.

[12] Arnaud Deza and Elias B. Khalil. Machine learning for cutting planes in integer programming: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI-2023, page 6592–6600. International Joint Conferences on Artificial Intelligence Organization, August 2023. doi: 10.24963/ijcai.2023/739. URL http://dx.doi.org/10.24963/IJCAI.2023/739.

[13] Gabriele Dragotto, Stefan Clarke, Jaime Fernández Fisac, and Bartolomeo Stellato. Differentiable cutting-plane layers for mixed-integer linear optimization. *arXiv preprint arXiv:2311.03350*, November 2023. URL https://arxiv.org/abs/2311.03350.

[14] Julius Farkas. Theorie der einfachen Ungleichungen. *J. Reine Angew. Math.*, 124:1–27, 1902. URL https://doi.org/10.1515/crll.1902.124.1.

[15] M.V. Galati and Ted K. Ralphs. Decomposition in integer programming. Technical Report 04T-019, Lehigh University Industrial and Systems Engineering Department, 2005.

[16] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15554–15566, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/d14c2267d848abeb81fd590f371d39bd-Abstract.html`.

[17] M. Güzelsoy. *Dual Methods in Mixed Integer Linear Programming.* PhD, Lehigh University, 2009. URL `http://coral.ie.lehigh.edu/~ted/files/papers/MenalGuzelsoyDissertation09.pdf`.

[18] Yacov Y. Haimes, Leon S. Lasdon, and David A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Trans. Systems Man Cybernet.*, SMC-1:296–297, 1971. ISSN 0018-9472.

[19] A. Hassanzadeh and T.K. Ralphs. A Generalization of Benders' Algorithm for Two-Stage Stochastic Optimization Problems with Mixed Integer Recourse. Technical Report 14T-005, COR@L Laboratory, Lehigh University, 2014. URL `http://coral.ie.lehigh.edu/~ted/files/papers/SMILPGenBenders14.pdf`.

[20] Shannon Kelley, Aleksandr Kazachkov, and Ted Ralphs. Warm starting of mixed integer linear optimization problems via parametric disjunctive cuts. 2025.

[21] Gonzalo Muñoz, Joseph Paat, and Álinson S. Xavier. Compressing branch-and-bound trees. In *Integer programming and combinatorial optimization*, volume 13904 of *Lecture Notes in Comput. Sci.*, pages 348–362. Springer, Cham, 2023. ISBN 978-3-031-32725-4; 978-3-031-32726-1. doi: 10.1007/978-3-031-32726-1_25. URL `https://doi.org/10.1007/978-3-031-32726-1_25`.

[22] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *CoRR*, abs/2012.13349, 2020. URL `https://arxiv.org/abs/2012.13349`.

[23] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf`.

[24] Krunal Patel. Progressively strengthening and tuning mip solvers for reoptimization, 2023. URL `https://arxiv.org/pdf/2308.08986.pdf`.

[25] Michael Perregaard and Egon Balas. Generating cuts from multiple-term disjunctions. In *Integer programming and combinatorial optimization (Utrecht, 2001)*, volume 2081 of *Lecture Notes in Comput. Sci.*, pages 348–360. Springer, Berlin, 2001. ISBN 3-540-42225-0. doi: 10.1007/3-540-45535-3_27. URL `https://doi.org/10.1007/3-540-45535-3_27`.

[26] R. Raman and I.E. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers & Chemical Engineering*, 18(7):563–578, 1994. ISSN 0098-1354. doi: https://doi.org/10.1016/0098-1354(93)E0010-7. URL `https://www.sciencedirect.com/science/article/pii/0098135493E00107`. An International Journal of Computer Applications in Chemical Engineering.

[27] Sahar Tahernejad, Ted K. Ralphs, and Scott T. DeNegre. A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. *Math. Program. Comput.*, 12(4):529–568, 2020. ISSN 1867-2949,1867-2957. doi: 10.1007/s12532-020-00183-6. URL `https://doi.org/10.1007/s12532-020-00183-6`.

[28] Álinson S. Xavier, Feng Qiu, and Shabbir Ahmed. Learning to solve large-scale security-constrained unit commitment problems. *INFORMS J. Comput.*, 33(2):739–756, 2021. ISSN 1091-9856,1526-5528. doi: 10.1287/ijoc.2020.0976. URL `https://doi.org/10.1287/ijoc.2020.0976`.

[29] L. Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, 8(1):59–60, 1963. doi: 10.1109/TAC.1963.1105511.

A Additional Tables and Figures

Table 18: Solve time (in seconds) for a subset of series with significant solve time improvement when tightening the disjunction prior to parametric disjunctive cut generation.

Degree	Terms	Element	Base Instance	Default	Calc Disj Calc Cuts	Param Disj Param Cuts	Tighten Disj	Series Length
1	4	matrix	neos-860300	129	97.9	70.5	40.6	10
	4	objective	ran13x13	35.5	15.0	17.0	13.1	10
4	4	rhs	neos-3046615-murg	232	169	200	151	2
	64	rhs	pg5_34	3.15	71.7	2.06	1.94	8
16	4	matrix	n7-3	8.89	30.5	13.2	5.29	2
	64	objective	misc07	83.7	47.2	41.4	28.7	10

Table 19: Solve time (in seconds) for a subset of series with significant solve time improvement when tightening parametric disjunctive cuts for terms where the coefficient matrix was perturbed.

Degree	Terms	Element	Base Instance	Default	Calc Disj Calc Cuts	Param Disj Param Cuts	Matrix	Series Length
1	4	matrix	neos-860300	129	97.9	70.5	50.0	10
	64	matrix	ran13x13	22.6	47.2	20.4	15.9	10
4	4	matrix	neos18	5.27	11.8	4.61	3.31	9
	64	matrix	g200x740	26.8	62.4	33.2	20.6	2
16	4	matrix	n7-3	8.89	30.5	13.2	5.34	2
	64	matrix	neos-911970	29.3	26.2	26.8	19.6	6

Table 20: Solve time (in seconds) for a subset of series with significant solve time improvement when tightening parametric disjunctive cuts for infeasible terms that become feasible after perturbation.

Degree	Terms	Element	Base Instance	Default	Calc Disj Calc Cuts	Param Disj Param Cuts	Term	Series Length
1	4	matrix	neos-860300	129	97.9	70.5	39.3	10
	64	rhs	neos18	13.7	639	16.6	9.64	2
4	4	matrix	rout	10.1	8.25	10.5	6.19	10
	64	rhs	pg5_34	3.15	71.7	2.06	1.98	8
16	4	rhs	mas76	47.0	63.6	44.4	38.4	4
	64	matrix	neos-911970	29.3	26.2	26.8	19.4	6

Table 21: Solve time (in seconds) for a subset of series with significant solve time improvement when tightening parametric disjunctive cuts for feasible bases that become infeasible after perturbation.

Degree	Terms	Element	Base Instance	Default	Calc Disj Calc Cuts	Param Disj Param Cuts	Basis	Series Length
1	4	matrix	neos-860300	129	97.9	70.5	38.0	10
	64	rhs	neos18	13.7	639	16.6	9.83	2
4	4	rhs	neos-1445743	43.9	2707	36.1	24.3	4
	64	matrix	ran13x13	54.5	66.4	77.4	42.1	10
16	4	matrix	neos-911970	19.3	18.7	17.7	10.6	5
	64	matrix	neos-911970	29.3	26.2	26.8	16.1	6

Table 22: Solve time (in seconds) for a subset of series with significant solve time improvement when enabling all tightening methods.

Degree	Terms	Element	Base Instance	Default	Calc Disj Calc Cuts	Param Disj Param Cuts	All	Series Length
1	4	matrix	neos-860300	129.2	97.9	70.5	56.3	10
	64	matrix	mas76	17.7	39.0	13.8	9.94	4
4	4	rhs	blp-ir98	12.7	143.3	8.67	7.34	2
	64	matrix	ran13x13	54.5	66.4	77.4	46.0	10
16	4	objective	neos-911970	102.5	69.2	139.2	41.9	2
	64	objective	misc07	83.7	47.2	41.4	35.8	10