Benders Cut Filtering for Affine Potential-Based Flow Problems with Robustness Scenarios and Topology Switching

Tim Donkiewicz* Oliver Gaul

Chair of Operations Research, RWTH Aachen University, Germany

Abstract

Many large-scale optimization problems decompose into a master problem and scenario subproblems, a structure that can be exploited by Benders decomposition. In Benders decomposition, each iteration may generate many cuts from scenario subproblems, and adding all of them as constraints then causes the master problem to grow rapidly. These are constraints that may need to be added to the master problem to guarantee optimality and feasibility of solutions, but we can avoid adding those constraints that are never violated. Adding fewer cuts per iteration can reduce the number of cuts added in total, but increase the number of iterations. In contrast, the cuts filtered for regular cut selection in mixed-integer programming solvers are optional and added exclusively to improve runtime behavior. We study *Benders cut filtering*: given the Benders cuts produced in an iteration, which subset should be added to the master problem? To our knowledge, few prior works have studied this question. We propose violation-based filtering (retaining the most-violated cuts), diversity-based filtering via k-medoids clustering on pairwise cosine distances (adding an original cut closest to the cluster centroid), and a hybrid that selects a most-violated cut per cluster. Each strategy can be augmented with an aggregated cut that retains discarded information. Computational experiments on 149 instances of an affine potential-based flow problem with topology switching and robustness scenarios—solved via Benders decomposition—show that all informed filtering strategies solve at least 125 instances (vs. 91 for the unfiltered baseline), reducing shifted geometric mean solve time by 55–57%. The hybrid strategy attains the best geometric mean (271.89 s vs. 629.34 s, a 57% reduction, $p < 0.001$). Our work demonstrates the effectiveness of Benders cut filtering techniques on a particularly suitable problem class and motivates further study on the generalization to other domains.

Keywords. Integer programming, Benders decomposition, cut filtering, potential-based flows, network optimization.

1 Introduction

Many problems in discrete optimization decompose into a master problem and one or more subproblems. Benders decomposition [5] exploits this structure by iterating between the master and subproblems, where each subproblem solution generates a constraint—commonly called a *Benders cut*—that is added to the master problem. Branch-and-Benders-cut extends this by incorporating Benders cut generation into branch-and-bound: subproblems are solved in a callback at each feasible master candidate, and the resulting cuts are added as lazy constraints. When the number of subproblems is large, each iteration during branch-and-Benders-cut can produce many such constraints, and adding all of them leads to rapid master problem growth and can lead to longer re-optimization times as solving progresses. Still, this is the standard approach in the literature: all generated Benders cuts are added as hard constraints to the master problem and enforced explicitly in every subsequent iteration.

*Corresponding author. donkiewicz@or.rwth-aachen.de

We study a problem that we call *Benders cut filtering*: given the pool of Benders cuts generated in an iteration, which subset should be added as constraints to the master problem? This can be a relevant consideration: for our problem instances, even a random selection of 5% of cuts reduces solve times substantially, confirming that constraint growth in the master problem can be the primary bottleneck. Prior work has filtered Benders cuts with machine-learning classifiers trained offline on labeled examples [19, 21]. However, to our knowledge, no prior work has studied filtering rules that do not require offline training on solved instances to determine which Benders cuts to add.

Outside the Benders context, the analogous question, selecting which cutting planes to add from a pool, is well studied in MIP branch-and-cut solvers [1, 3, 35]. Solvers manage cut pools and apply quality criteria such as violation, pairwise distance, or objective parallelism to decide which cuts to enforce (Section 2.5). In Benders decomposition, this pool-based management is not applicable directly: at least some Benders cuts have to be added as constraints, not as cuts optional to the formulation, to maintain correctness. We therefore adapt the principles of violation scoring and distance-based diversity to the Benders constraint setting.

A separate line of work, *Benders cut selection*, addresses a different problem not considered in this paper: choosing a strongest cut among multiple dual solutions of a single subproblem (Section 2.3).

We propose three filtering strategies (Section 4). *Violation-based* filtering retains the k most-violated cuts. *Diversity-based* filtering applies k-medoids clustering on pairwise cosine distances between cut coefficient vectors and selects one representative per cluster—an original cut nearest to the cluster centroid. *Hybrid* filtering clusters identically but selects a most-violated cut per cluster, combining violation with diversity. Random filtering is also considered as a baseline filtering technique. Each strategy can be augmented with an aggregated cut that retains at least some discarded information. The number of retained cuts is determined a priori or adaptively.

We demonstrate our Benders cut filtering on affine potential-based flow problems [28] with topology switching and robustness scenarios. This setting is well-suited: subproblems are solvable as systems of linear equations [13], producing many cuts cheaply, so that master constraint growth—rather than subproblem solving—is the bottleneck. Additionally, the subproblems are numerous and structurally similar, differing by at most two arcs in a graph with usually hundreds of arcs, which suggests that many cuts are similar.

Computational experiments on 149 instances show that all filtering strategies significantly reduce solve times ($p < 0.001$, Wilcoxon signed-rank test) compared to the unfiltered baseline (adding all cuts) common in the literature on Benders decomposition. Violation-based and diversity-based filtering solve 126 and 128 instances within the time limit (vs. 91 unfiltered), with 55–57% reductions in shifted geometric mean solve time. The hybrid strategy attains the best geometric mean (57% reduction) and solves 125 instances. Even random filtering solves 113, confirming that constraint growth is the bottleneck.

2 Related Work

2.1 Benders Decomposition and Cut Aggregation

Benders [5] introduced a decomposition for mixed-integer programs that iterates between a master problem over complicating variables and a subproblem that generates constraints (Benders cuts) for the master. Rahmaniani et al. [29] provide a comprehensive survey. For two-stage stochastic programs, the L-shaped method of Van Slyke and Wets [36] introduces a single auxiliary variable θ in the master that represents the expected recourse cost and, per iteration, adds one cut obtained by combining scenario duals with their probabilities to correct the master's estimate of the expected recourse cost. Birge and Louveaux [7] introduce the multi-cut

variant, which uses one auxiliary variable per scenario and adds one cut per scenario in each iteration; this can converge in far fewer iterations than the single-cut approach, at the cost of a larger master problem. Trukhanov et al. [34] dynamically adjust the aggregation level between single-cut and multi-cut by monitoring dual multipliers: scenario groups whose optimality cuts remain inactive (zero dual multiplier) for a threshold fraction of iterations are merged into a single aggregate. Their results demonstrate that an intermediate aggregation level can yield substantial computational savings. Notably, they also attempted to aggregate scenarios with similar cut gradients but found the pairwise comparison of cut coefficients too expensive. Wolf and Koberstein [38] use inactivity as an aggregation trigger at the *cut* rather than the *scenario* level: in multi-stage Benders decomposition (the nested L-shaped method), when a sufficient fraction of cuts generated in the same past iteration have been inactive for several consecutive iterations, those cuts are aggregated into a single consolidated cut and the originals are removed from the master. Song and Luedtke [32] formalize intermediate aggregation levels through an adaptive partition-based approach, proving that scenario groups with identical dual solutions can be merged without weakening the relaxation bound. Ramírez-Pico et al. [30] turn this into an adaptive procedure: scenarios are partitioned, one aggregated cut is generated per part, and the partition is iteratively refined using subproblem dual information.

2.2 Subproblem Selection and Partial Decomposition in Benders Decomposition

When the number of subproblems is very large, as in stochastic programming with many scenarios, adding all cuts leads to rapid master problem growth and long re-optimization times. Crainic et al. [10] develop partial Benders decomposition, which retains a subset of subproblems in the master problem; Donkiewicz [12] proposes an adaptive subproblem selection strategy that learns which subproblems to solve based on their historical contributions; Blanchot et al. [8] propose a batch variant that groups subproblems and selects which batches to solve in each iteration; Pauphilet and Wu [27] randomly retain a subset of continuous variables in the master, showing that even this simple partial decomposition performs well; and Bertsimas et al. [6] solve a sampled subset of scenario subproblems at each iteration and aggregate their dual solutions into a single deterministically valid cut, optionally clustering scenarios to further reduce the number of subproblems solved. These approaches reduce the number of subproblems solved or combine their dual information, which also leads to fewer added cuts, which are potentially weaker. Benders cut filtering, in contrast, solves all subproblems but selects *which of the resulting Benders cuts to add* to the master problem from the full pool of available constraints.

2.3 Cut Selection in Benders Decomposition

When the dual subproblem admits multiple solutions, the resulting cuts can differ in strength. A substantial literature studies how to pick a strong cut in this setting, starting with the Pareto-optimal cuts of Magnanti and Wong [25] and the practical variant of Papadakos [26]; follow-ups develop alternative selection criteria such as deepest, closest, or maximum-violation cuts [9, 15, 17, 20, 31]; see Kaltis and Saharidis [22] for a review. This line of work addresses a different problem than ours: choosing a strongest cut within a single subproblem rather than selecting across cuts from different subproblems.

2.4 Existing Cut-Filtering Approaches

Machine learning has been used to filter Benders cuts before adding them to the master. Jia and Shen [21] define a cut in the offline training phase as valuable if it is binding at an optimal master solution or yields sufficient bound improvement; a support vector machine trained on cut-coefficient and scenario features from solved instances classifies candidate cuts per iteration and

adds only those predicted valuable. Hasan and Kargarian [19] define usefulness by contribution to lower bound improvement; a classifier trained on solved security-constrained unit commitment instances discards cuts predicted non-useful, augmented by a regression model that predicts subproblem objectives for tighter cuts. Both methods require offline training on solved instances.

2.5 Cut Management in Integer Programming

In MIP branch-and-cut solvers, selecting which cutting planes to add from a candidate pool is critical for performance. Andreello et al. [3] identify efficacy (violation at the current LP relaxation point) and orthogonality (pairwise angle between cuts) as key quality metrics. The default cut selector of SCIP (through at least version 9) [1, 2] combines criteria such as violation, objective parallelism, and integrality support in a hybrid score. Turner et al. [35] develop adaptive cut selection that dynamically adjusts criteria weights during solving. Deza and Khalil [11] survey the growing literature on machine learning for cut selection, including reinforcement learning approaches [33].

We adopt two of these criteria directly: violation scoring and cosine distance as a pairwise diversity measure.

2.6 Positioning

Cut selection (Section 2.3) chooses among alternative dual solutions of a single subproblem; aggregation (Section 2.1) merges subproblem information into fewer, potentially weaker cuts; subproblem selection (Section 2.2) avoids cuts by solving fewer subproblems. None of these directly selects which cuts to add from the per-iteration pool, and all pose the risk of weakening or completely missing cuts. In the case of fast subproblem solves, we can afford to solve all subproblems, possibly missing fewer important cuts. ML-based filtering (Section 2.4) does select from this pool but requires offline training; moreover, the training labels—whether a cut is binding at the optimum or improves the bound—are end-state properties that need not identify cuts to facilitate fast progress during intermediate iterations. MIP cut management (Section 2.5) applies violation and diversity criteria to a candidate pool, but there cuts are optional strengthening inequalities; in Benders decomposition, cuts are constraints required for correctness, and at least one violated cut must be added per iteration to guarantee convergence.

We adapt these MIP criteria—violation scoring and cosine distance—to the Benders setting: violation-based filtering retains the k most-violated cuts, while diversity-based and hybrid filtering cluster cuts by cosine distance and select one representative per cluster. In mechanism, the adaptive partition of Song and Luedtke [32] is closest: they cluster *scenarios* by recourse behavior and produce one aggregated cut per group, which may be weaker than individual cuts [7]; we cluster the *cuts themselves* and select either a centroid-nearest original cut (*diversity*) or a most-violated cut per cluster (*hybrid*). In a parameter search, we determine the number of cuts to retain per iteration, either a fixed number k or an adaptive threshold based on the current gap between the incumbent and master objective values.

3 Problem Description

We solve problems following the potential-based flow framework, as discussed by Pfetsch et al. [28]. In these network flow problems, flows on arcs are determined by the potentials at their endpoints. Certain arcs can be switched on or off, decoupling the flow from the potential difference. The framework covers diverse network optimization problems, including electrical power systems (DC power flow), gas networks, and traffic assignment. We solve an extended version with adjustable demands, scenario-based robustness, and overloads. Each scenario corresponds to the failure of a set of arcs. Switching and demand decisions are made in a first

stage before observing which scenario occurs, while flows and potentials adapt per scenario in a second stage.

This two-stage structure admits a Benders decomposition [5]: switching and demand variables form the master problem, while each scenario yields a subproblem that checks flow and potential feasibility and computes overload costs under the given topology and outage.

For affine potential functions, the potential equation is linear. After contracting switchable arcs, the subproblem reduces to solving a fixed number of systems of linear equations, yielding both primal and dual solutions without solving an LP [13]. This significantly reduces subproblem solve time, but also reduces the number of available cuts per subproblem—often to one. Classical Benders cut selection, which chooses among multiple cuts from each subproblem, is therefore not applicable without losing the computational advantage of solving systems of linear equations. Filtering *across* subproblems, however, becomes particularly relevant.

The instances studied in this paper exhibit three properties that make cut filtering both beneficial and compatible with the decomposition:

1. **Many independent subproblems.** Each subproblem can yield a Benders cut, so as many cuts as scenarios can be generated per iteration, all from the same master solution.

2. **Many similar, possibly dominating cuts.** Contingencies that share affected regions produce cuts with similar coefficient vectors, so adding all cuts grows the master without bound improvement.

3. **Fast subproblem solves.** Solving subproblems as systems of linear equations is substantially faster than solving the corresponding LPs. The added runtime of solving a few more subproblems to re-add discarded cuts is negligible compared to the master re-optimization time saved by filtering.

4 Benders Cut Filtering Strategies

In each round of branch-and-Benders-cut, solving all scenario subproblems at the current master candidate produces a fresh candidate pool $\mathcal{P}$ of optimality and feasibility cuts, from which we select a subset $C \subseteq \mathcal{P}$ to add. We present three filtering strategies—violation-based scoring, diversity-based clustering, and a hybrid of the two—followed by a variant that retains information from discarded cuts via aggregation. Violation-based scoring focuses on the effectiveness of each individual cut, whereas diversity-based clustering avoids adding near-parallel cuts. All strategies prioritize feasibility cuts (Section 4.5): whenever the pool contains at least one violated feasibility cut and one optimality cut, at least one of each is added.

4.1 Violation-Based Filtering

Let $\bar{x}$ denote the current master solution. For an optimality cut $c_i = a_i^\top x \leq b_i$, $c_i \in \mathcal{P}$, the *violation* is $v(c_i, \bar{x}) := \max(0, a_i^\top \bar{x} - b_i)$, positive when the cut is violated. We use the absolute violation throughout, without normalizing by $\|a\|$: cuts from structurally similar subproblems (differing by at most two outaged arcs) have comparable coefficient magnitudes, so absolute and relative rankings agree in practice. Feasibility cuts are assigned a violation value larger than the largest violation of any optimality cut; this is a priority-encoding device, not a physical violation, ensuring feasibility cuts rank first whenever both types are present.

We consider two modes for determining how many cuts to retain. In the *fixed* mode, the $k \in \mathbb{N}$ cuts with the largest violation are retained. In the *adaptive* mode, cuts are added in order of non-increasing violation until the cumulative violation of the added cuts first exceeds the threshold $\rho \cdot (z_{\text{UB}} - z_{\text{MP}})$, where $z_{\text{UB}} \in \mathbb{R}$ is the incumbent objective value, $z_{\text{MP}} \in \mathbb{R}$ is the current master objective, and $\rho \geq 1$ is a buffer parameter. The threshold biases the selection toward more cuts when this gap is large and fewer cuts as the algorithm converges.

The adaptive mode also addresses a convergence concern: Each scenario $s \in \mathcal{S}$ has its own recourse variable θ_s in the master, and each Benders cut bounds a specific θ_s from below. Every time a new subset of cuts is added, another first-stage solution with identical switch-arc and demand assignment (z, b), but correspondingly adjusted θ_s variables, can be proposed by the master. This can happen until all cuts for this assignment are added, or until the objective value of the solution including the θ_s variables exceeds that of the incumbent. By requiring that the sum of added violations exceeds the current gap between z_{UB} and z_{MP}, we thus reduce the number of these identical assignments, at the cost of potentially adding more cuts overall.

4.2 Diversity-Based Filtering

Beyond scoring individual cuts, we filter cuts to ensure the selected subset contains cuts whose coefficient vectors are not near-parallel. We measure pairwise similarity by the cosine of the angle between the cut normals a_i, a_j (invariant to scaling of the coefficients),

$$\cos\theta(c_i, c_j) = \frac{a_i^\top a_j}{\|a_i\|\|a_j\|}, \tag{1}$$

and define the *cosine distance*

$$d_{\cos}(c_i, c_j) := 1 - \cos\theta(c_i, c_j) \in [0, 2].$$

Parallel coefficient vectors ($\cos\theta = 1$) yield $d_{\cos} = 0$, orthogonal vectors yield $d_{\cos} = 1$, and anti-parallel vectors ($\cos\theta = -1$) yield $d_{\cos} = 2$. Distinguishing anti-parallel vectors is appropriate here: the constraints $a^\top x \le b$ and $-a^\top x \le -b$ are not equivalent.

We apply k-medoids clustering [23] with cosine distance to partition the candidate cuts into $k \in \mathbb{N}$ groups, minimizing the sum of within-cluster distances. From each cluster $K \subseteq \mathcal{P}$ with members having coefficient vectors $\{a_j : c_j \in K\}$, we compute the centroid $\bar{a}_K := \frac{1}{|K|}\sum_{c_j \in K} a_j$ and select as representative a cluster member whose coefficient vector is closest to the centroid in Euclidean distance, $c_K^\star \in \arg\min_{c_j \in K} \|a_j - \bar{a}_K\|_2$. We set $k = \lceil \alpha \cdot |\mathcal{C}| \rceil$, where $|\mathcal{C}|$ is the number of scenarios and $\alpha \in \mathbb{R}$ is a proportion parameter.

4.3 Hybrid Filtering

The hybrid strategy combines violation-based filtering with diversity-based filtering: it applies k-medoids clustering as in Section 4.2 to partition cuts into groups with diverse coefficient directions, then selects a most-violated cut from each cluster as the representative. This ensures that the selected subset spans diverse constraint directions while retaining individually strongest cuts.

4.4 Cut Aggregation

Any filtering strategy discards non-selected cuts, losing information obtained at the cost of solving the corresponding subproblems. To retain this information, we augment each strategy with a supplementary aggregated cut: the non-selected *violated* cuts $R = \{c \in \mathcal{P} \setminus C \mid v(c, \bar{x}) > 0\}$ are combined into a single constraint using violation-normalized weights $w_c = v(c, \bar{x}) / \sum_{c' \in R} v(c')$. That way, for each θ_s variable, there is some constraint bounding it from below, either an original cut in C or the aggregate of discarded cuts in R. The resulting aggregate $\left(\sum_{c \in R} w_c\, a_c\right)^\top x \le \sum_{c \in R} w_c\, b_c$ weighs each discarded cut proportionally to its violation, yielding $k + 1$ cuts per iteration at negligible additional master cost. Note that one could in general also add currently non-violated Benders cuts, because they may become violated in future iterations. We denote aggregation-augmented variants by appending "+" to the strategy name. This implements the composition of selection and aggregation discussed in Section 2.6.

4.5 Correctness

Mathematically, filtering is admittable as long as at least one violated cut is added in each iteration at which the current master candidate is cut off: the master LP then changes and progress is made, and the algorithm converges in the usual way. A numerical issue specific to lazy constraint callback implementations arises: Commercial solvers like Gurobi [18] check a candidate master solution only against added lazy constraints and may accept a candidate where added cuts have small numerical violations (e.g., after solver-internal coefficient rescaling) while filtered-out cuts with larger violations would have rejected it. We call this a *cut error*: the returned objective is too optimistic because discarded cuts were never enforced.

To avoid such errors, all strategies prioritize feasibility cuts in the violation ranking and add at least one optimality cut per iteration whenever one exists in the pool. In preliminary experiments, violation-based filtering *without* this feasibility-first priority solved fewer instances than the unfiltered baseline; these runs exhibited cut errors that the above mechanism successfully mitigated.

5 Evaluation

5.1 Experimental Setup

All experiments run on machines with 2× Intel Xeon L5630 quad-core processors at 2.13 GHz (8 cores, 16 GB RAM). The implementation is in Julia using JuMP [24] with Gurobi 12.0.0 [18] as the MIP solver. A three-hour time limit is imposed per instance.

Statistical Methodology We aggregate solve times using the shifted geometric mean with shift $s = 10$: for times $t_1, \ldots, t_n \in \mathbb{R}$, the shifted geometric mean is $\left(\prod_{i=1}^{n}(t_i + s)\right)^{1/n} - s$. The shift reduces the influence of very small solve times; instances solved in under 50 s by all configurations are excluded from the aggregation to avoid these trivial cases dominating the measurements. Summary rows include only instances solved to optimality by all configurations compared.

To test whether solve time distributions differ between configurations, we use the Wilcoxon signed-rank test [37], a non-parametric two-sided test of the null hypothesis that the median paired difference in solve times is zero. In Tables 1 and 2, significance levels are denoted by * ($p < 0.05$), ** ($p < 0.01$), and *** ($p < 0.001$). In the per-instance table (Table 2), *TL* marks a time limit hit (optimality gap in parentheses), boldface marks the fastest configuration per instance, and † after the instance name indicates that all configurations solved the instance to optimality.

Instance Generation We generate test instances by extracting multiple subgraphs from benchmark power grid instances in the literature [4], projecting them into two dimensions using a spring layout [16], computing the convex hull of each subgraph, and connecting a determined number of connection points that are close to each other across subgraphs. This produces networks of varying size and topology that exhibit a structure similar to the original instances. We generated 240 candidate instances and discarded those that were infeasible (or not proven feasible by any setting within the time limit) for the underlying optimal power flow problem without switching, leaving 149 instances. The instances and code will be made available upon acceptance of the paper.

Configurations We evaluate Benders cut filtering strategies on these instances, solved via Benders decomposition with subproblems solved as systems of linear equations. The comparison covers six configurations:

1. **No filtering** (default): all generated cuts are added to the master problem in each iteration.
2. **Random**: k cuts are selected uniformly at random from all generated cuts.
3. **Violation**: the k most-violated cuts are added.
4. **Diversity**: k-medoids clustering on cosine distances (1); an original cut closest to the cluster centroid is added as representative.
5. **Hybrid**: k-medoids clustering on cosine distances; a most-violated cut per cluster is added.
6. **Hybrid+**: Hybrid, but with additional aggregation of all remaining non-selected cuts into a single supplementary constraint with violation-normalized weights (Section 4.4).

We tested "+" variants for all strategies but found no significant benefit except for a reduction in iteration count for hybrid+ (see below); we therefore report only hybrid+ alongside the base strategies. In preliminary experiments, we also tested DBSCAN-based clustering [14] and Jaccard distance on coefficient support. DBSCAN did not outperform k-medoids: its density-based clustering produced unevenly sized groups and required tuning two parameters (ε, minPts) rather than one. Jaccard distance was consistently outperformed by cosine distance, which incorporates coefficient magnitudes and directly measures the angle between cut normals.

In all filtering strategies, feasibility cuts are always prioritized (via violation) and at least one optimality cut is added per iteration; this feasibility-first variant avoids the cut errors described in Section 4.5. Additionally, they add $k = \lceil 0.05 \cdot |\mathcal{S}| \rceil$ cuts per iteration, where $\mathcal{S}$ is the set of scenarios. This proportion was determined via parameter search on 20 held-out validation instances (generated using the same methodology with a different seed, disjoint from the test set) over 21 settings: seven values each for three parameterization modes (fixed count k, fraction of subproblems $\alpha \cdot |\mathcal{S}|$, fraction of violated cuts per iteration $\beta \cdot n_{\text{viol}}$); see Figure 3 in the appendix. The subproblem fraction $\alpha = 0.05$ achieved the lowest shifted geometric mean ratio to the baseline (0.503, $p < 0.001$). The per-iteration violated-cut fraction provides an adaptive parameterization (since n_{viol} varies across iterations) but did not outperform the fixed subproblem fraction.

5.2 Results

Table 1 summarizes solve times across the six configurations on the $n = 87$ instances solved to optimality by all configurations. Figure 1 visualizes their time distribution, and Figure 2 provides a performance profile. Per-instance results are in Table 2 (appendix).

Table 1: Summary of runtime statistics across Benders cut filtering strategies on $n = 149$ instances. Aggregate solve times, iteration counts, and cuts per iteration cover only the $n = 87$ instances solved to optimality by all configurations. "Mean" denotes the arithmetic mean, "Geo. Mean" the shifted geometric mean with shift $s = 10$. "Ratio" denotes the ratio to `default`.

		Time (s)			Iterations		Cuts / Iteration	
	Solved	Sum	Mean	Geo. Mean	Geo. Mean	Ratio	Geo. Mean	Ratio
`default`	91	166753.30	1916.70	629.34	337	1.00×	105.8	1.00×
`random`	113	72127.58	829.05	392.70	714	2.12×	10.8	0.10×
`violation`	126	44477.11	511.23	273.92	472	1.40×	10.2	0.10×
`diversity`	128	43755.92	502.94	285.68	443	1.32×	9.9	0.09×
`hybrid`	125	41135.21	472.82	271.89	424	1.26×	9.8	0.09×
`hybrid+`	125	41737.71	479.74	275.18	410	1.22×	10.6	0.10×

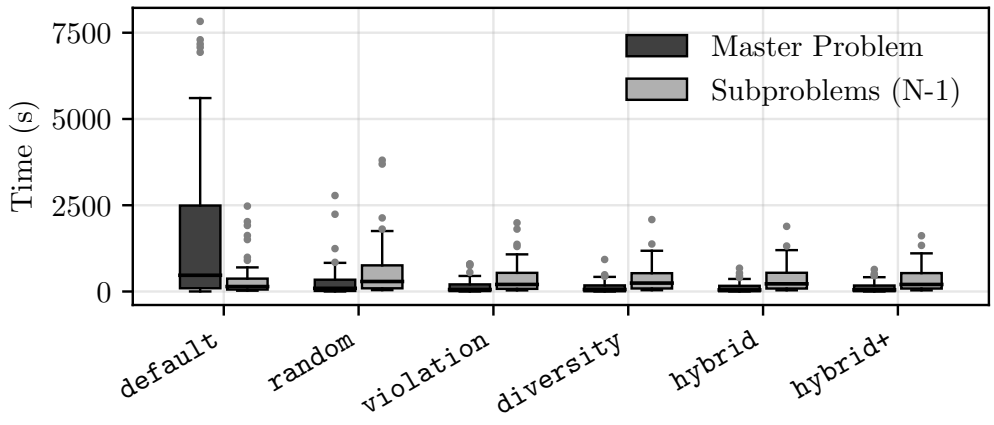

Figure 1: Distribution of solve times across Benders cut filtering strategies on $n = 87$ instances solved to optimality by all configurations. Each box spans the interquartile range; whiskers extend to 1.5× IQR; dots are outliers.

Filtering reduces the number of cuts added per iteration from 106 (default) to 10–11, around 10%, but increases the number of Benders iterations. An increase in iterations reflects a diminished overall cut strength per iteration. Hybrid requires the lowest increase, at 1.26× the baseline iterations, indicating that it yields the best overall set of cuts. However, even random requires only 2.12× the number of iterations. Despite adding only a tenth of the cuts per iteration, the resulting smaller master problems yield net solve-time reductions of 55–57%. As such, we can conclude that cut filtering as a technique is generally well-founded for these instances.

The hybrid strategy—k-medoids clustering followed by selecting a most-violated cut per cluster—achieves the best shifted geometric mean of 271.89 s, a 57% reduction from the baseline's 629.34 s ($p < 0.001$, Wilcoxon signed-rank test), and solves 125 of 149 instances. Despite adding the lowest number of cuts per iteration, around 9.8% of the geometric mean of `default`, the geometric mean of the number of iterations increases by only 26%. This indicates that the individual and overall cut strength is mostly preserved. Its "+" variant (275.18 s, 125 instances) has a similar solve time but requires fewer iterations (1.22× baseline vs. 1.26×), suggesting the aggregate marginally aids convergence without improving overall speed, likely due to the higher number of cuts.

Violation-based filtering solves 126 instances with a geometric mean of 273.92 s (56% reduction, $p < 0.001$). Despite its simplicity, requiring only to sort by violation, this approach closely matches the clustering-based strategies.

Diversity filtering (closest-to-centroid) solves the most instances (128) with a geometric mean of 285.68 s (55% reduction). Each representative is an original cut nearest to the mean coefficient vector of its cluster, yielding a geometrically diverse set of cuts.

Random filtering solves 113 instances—substantially more than the unfiltered baseline (91)—with a geometric mean of 392.70 s (38% reduction, $p < 0.001$). This confirms that master problem growth is the primary bottleneck: even uninformed filtering to at most 5% of subproblems accelerates convergence on most instances.

Summary All informed filtering strategies reduce geometric mean solve time by 55–57% ($p < 0.001$). Hybrid achieves the best geometric mean (271.89 s), while diversity solves the most instances (128). Diversity solves more instances than hybrid (128 vs. 125) but has a higher geometric mean (285.68 s vs. 271.89 s), reflecting a trade-off between robustness and speed.

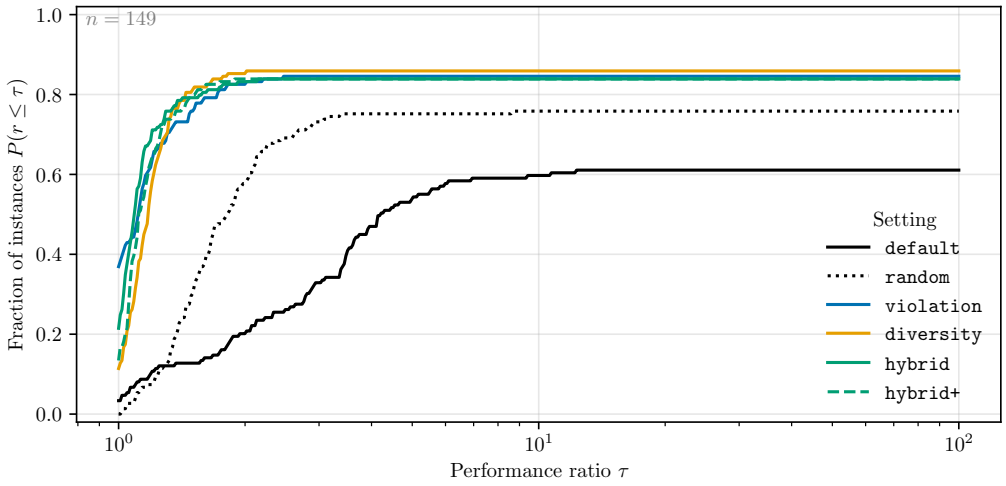

Figure 2: Performance profile of solve times across Benders cut filtering strategies ($n = 149$ instances).

6 Summary

We introduced Benders cut filtering: selecting which Benders cuts to add as constraints to the master problem, rather than adding all of them. Prior work has filtered Benders cuts with offline-trained classifiers [21, 19]; to our knowledge, this is the first study of training-free, geometric filtering strategies specifically applied in the Benders case. We proposed three strategies—violation-based, diversity-based via k-medoids clustering on cosine distances, and a hybrid that selects a most-violated cut per cluster—each optionally augmented with a violation-weighted aggregate of discarded cuts.

Computational experiments on 149 instances of an affine potential-based flow problem with topology switching and robustness scenarios show that all informed strategies reduce shifted geometric mean solve time by 55–57% ($p < 0.001$) and solve 125–128 instances within the time limit, compared to 91 for the unfiltered baseline. The hybrid strategy attains the best geometric mean (271.89 s, 57% reduction); the diversity strategy solves the most instances (128). Random filtering still solves 113, which confirms that master constraint growth is the bottleneck when subproblem solves are cheap. Aggregating discarded cuts does not consistently improve performance; only hybrid+ shows a modest reduction in iteration count.

Several directions remain open. We use cosine distance to measure pairwise similarity of cut coefficient vectors; incorporating objective-function parallelism—as done in MIP cut selectors [2]—could improve cut quality, particularly when cuts of similar direction differ in their alignment with the objective. Combining multiple criteria into a single hybrid score, as is standard for MIP cut selection, is a natural extension of the present single-criterion strategies. Adaptive strategies that adjust the number of retained cuts based on convergence progress may improve robustness. Additionally, the composition of selection, aggregation, or cut aging techniques could be explored further. Evaluation on other application domains (gas networks, traffic assignment) and comparison with solver-internal cut management (e.g., adding Benders cuts as pool cuts rather than hard constraints) would further clarify the scope of the approach.

Use of AI tools. Large language model tools were used for text polishing during the preparation of this manuscript and for assisting in generating scripts to produce the plots. All generated content was reviewed, verified, and edited by the authors.

References

[1] Tobias Achterberg. *Constraint integer programming*. PhD thesis, Technische Universität Berlin, 2007.

[2] Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi:10.1007/s12532-008-0001-1.

[3] Giovanni Andreello, Alberto Caprara, and Matteo Fischetti. Embedding cuts in a branch&cut framework: a computational study with $\{0, \frac{1}{2}\}$-cuts. *INFORMS Journal on Computing*, 19(2):229–238, 2007. doi:10.1287/ijoc.1050.0162.

[4] Sogol Babaeinejadsarookolaee, Adam Birchfield, Richard D. Christie, Carleton Coffrin, Christopher DeMarco, Ruisheng Diao, Michael Ferris, Stephane Fliscounakis, Scott Greene, Renke Huang, Cedric Josz, Roman Korab, Bernard Lesieutre, Jean Maeght, Terrence W. K. Mak, Daniel K. Molzahn, Thomas J. Overbye, Patrick Panciatici, Byungkwon Park, Jonathan Snodgrass, Ahmad Tbaileh, Pascal Van Hentenryck, and Ray Zimmerman. The power grid library for benchmarking AC optimal power flow algorithms. *arXiv preprint arXiv:1908.02788*, 2019. doi:10.48550/arXiv.1908.02788.

[5] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962. doi:10.1007/BF01386316.

[6] Dimitris Bertsimas, Ryan Cory-Wright, Jean Pauphilet, and Periklis Petridis. A stochastic Benders decomposition scheme for large-scale stochastic network design. *INFORMS Journal on Computing*, 37(5):1163–1181, 2025. doi:10.1287/ijoc.2023.0074.

[7] John R. Birge and François V Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392, 1988. doi:10.1016/0377-2217(88)90159-2.

[8] Xavier Blanchot, François Clautiaux, Boris Detienne, Aurélien Froger, and Manuel Ruiz. The Benders by batch algorithm: Design and stabilization of an enhanced algorithm to solve multicut Benders reformulation of two-stage stochastic programs. *European Journal of Operational Research*, 309(1):202–216, 2023. doi:10.1016/j.ejor.2023.01.004.

[9] René Brandenberg and Paul Stursberg. Refined cut selection for Benders decomposition: applied to network capacity expansion problems. *Mathematical Methods of Operations Research*, 94(3):383–412, 2021. doi:10.1007/s00186-021-00762-w.

[10] Teodor Gabriel Crainic, Mike Hewitt, Francesca Maggioni, and Walter Rei. Partial Benders decomposition: general methodology and application to stochastic network design. *Transportation Science*, 55(2):414–435, 2021. doi:10.1287/trsc.2020.1022.

[11] Antoine 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-23), Survey Track*, pages 6592–6600, 2023. doi:10.24963/ijcai.2023/740.

[12] Tim Donkiewicz. Adaptive subproblem selection in Benders decomposition for survivable network design problems. In Martin Aumüller and Irene Finocchi, editors, *24th International Symposium on Experimental Algorithms (SEA 2026)*, 2026. doi:10.48550/arXiv.2604.09031.

[13] Tim Donkiewicz and Oliver Gaul. Efficient subproblem solving in Benders decomposition for potential-based flow problems. Not available online yet, but will be added when the paper is published., 2026.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press, 1996. doi:10.5555/3001460.3001507.

[15] Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette. A note on the selection of Benders' cuts. *Mathematical Programming*, 124(1–2):175–182, 2010. doi:10.1007/s10107-010-0351-0.

[16] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. doi:10.1002/spe.4380211102.

[17] Lukas Glomb, Frauke Liers, and Florian Rösel. A novel Pareto-optimal cut selection strategy for Benders decomposition. *Mathematical Programming Computation*, 18(1):211–257, 2026. doi:10.1007/s12532-025-00291-1.

[18] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

[19] Fouad Hasan and Amin Kargarian. Accelerating L-shaped two-stage stochastic SCUC with learning integrated Benders decomposition. *arXiv preprint arXiv:2311.10835*, 2023. doi:10.48550/arXiv.2311.10835.

[20] Mojtaba Hosseini and John Turner. Deepest cuts for Benders decomposition. *Operations Research*, 73(5):2591–2609, 2025. doi:10.1287/opre.2021.0503.

[21] Huiwen Jia and Siqian Shen. Benders cut classification via support vector machines for solving two-stage stochastic programs. *INFORMS Journal on Optimization*, 3(3):278–297, 2021. doi:10.1287/ijoo.2020.0045.

[22] Timoleon Kaltis and Georgios K. D. Saharidis. Literature review on Benders cut selection and a multiple cut generation scheme. *INFOR: Information Systems and Operational Research*, 64(1):244–270, 2025. doi:10.1080/03155986.2025.2540205.

[23] Leonard Kaufman and Peter J. Rousseeuw. Clustering by means of medoids. In *Statistical Data Analysis Based on the L1-Norm and Related Methods*, pages 405–416. North-Holland, 1987.

[24] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15(3):581–589, 2023. doi:10.1007/s12532-023-00239-3.

[25] Thomas L. Magnanti and Richard T. Wong. Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29(3):464–484, 1981. doi:10.1287/opre.29.3.464.

[26] Nikolaos Papadakos. Practical enhancements to the Magnanti–Wong method. *Operations Research Letters*, 36(4):444–449, 2008. doi:10.1016/j.orl.2007.08.005.

[27] Jean Pauphilet and Yupeng Wu. The surprising performance of random partial Benders decomposition. *Optimization Online*, 2025. 32181.

[28] Marc E. Pfetsch, Martin Schmidt, Martin Skutella, and Johannes Thürauf. Potential-based flows – an overview, 2026.

[29] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017. doi:10.1016/j.ejor.2016.12.005.

[30] Cristian Ramírez-Pico, Ivana Ljubić, and Eduardo Moreno. Benders adaptive-cuts method for two-stage stochastic programs. *Transportation Science*, 57(5):1252–1275, 2023. doi:10.1287/trsc.2022.0073.

[31] Kiho Seo, Seulgi Joung, Chungmok Lee, and Sungsoo Park. A closest Benders cut selection scheme for accelerating the Benders decomposition algorithm. *INFORMS Journal on Computing*, 34(5):2804–2827, 2022. doi:10.1287/ijoc.2022.1207.

[32] Yongjia Song and James R. Luedtke. An adaptive partition-based approach for solving two-stage stochastic programs with fixed recourse. *SIAM Journal on Optimization*, 25(3):1344–1367, 2015. doi:10.1137/140967337.

[33] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2020.

[34] Svyatoslav Trukhanov, Lewis Ntaimo, and Andrew Schaefer. Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research*, 206(2):395–406, 2010. doi:10.1016/j.ejor.2010.02.025.

[35] Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive cut selection in mixed-integer linear programming. *Open Journal of Mathematical Optimization*, 4:1–28, 2023. doi:10.5802/ojmo.25.

[36] Richard M. Van Slyke and Roger Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969. doi:10.1137/0117061.

[37] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi:10.2307/3001968.

[38] Christian Wolf and Achim Koberstein. Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested L-shaped method. *European Journal of Operational Research*, 230(1):143–156, 2013. doi:10.1016/j.ejor.2013.04.033.

A Detailed Results

Table 2: Per-instance solve times (seconds), part 1 of 3. † = solved by all configurations. Statistics summary in Table 2 uses $n = 87$, solved by all configurations.

	default	random	violation	diversity	hybrid	hybrid+
92_2_tal1†	153.47	**99.80**	**79.54**	**115.02**	**93.93**	**98.92**
92_3_tal1†	45.33	61.57	**41.63**	51.15	**44.69**	48.19
94_tal1†	52.30	**49.14**	**47.29**	56.63	**52.06**	**50.75**
98_tal1†	38.74	57.58	51.14	56.03	49.61	52.85
100_tal1†	41.48	53.40	49.36	43.51	**40.94**	47.40
105_tal1†	35.80	49.72	48.16	53.52	45.98	48.07
110_tal1†	135.45	**82.76**	**76.95**	**75.00**	**75.39**	**82.66**
120_tal1†	59.45	92.74	79.60	85.31	60.86	62.57
120_2_tal1†	46.68	50.14	**44.10**	49.22	50.19	49.00
125_tal1†	205.47	224.81	**172.26**	**185.97**	**173.36**	**171.64**
126_tal1†	193.24	**108.93**	**105.35**	**93.94**	**119.54**	**114.88**
128_2_tal1†	29.06	47.57	45.94	51.11	51.25	50.41
138_tal1†	186.14	**133.85**	**107.44**	**138.13**	**115.55**	**118.82**
147_tal1†	584.98	724.35	**498.84**	**308.54**	**276.58**	**286.79**
148_tal1†	52.59	61.26	**49.36**	54.84	**51.17**	**52.58**
152_tal1†	81.67	**72.01**	**65.66**	**77.68**	**78.76**	82.76
155_tal1†	493.26	**298.77**	**212.73**	**251.17**	**220.35**	**240.21**
155_2_tal1†	508.42	**280.25**	**176.04**	**246.75**	**236.84**	**221.96**
156_tal1†	64.82	**60.01**	**53.24**	**55.70**	**58.57**	**56.09**
161_tal1†	130.66	**128.49**	159.95	**72.42**	**99.28**	**107.83**
162_tal1†	1482.13	**1387.53**	**491.26**	**410.68**	**420.52**	**437.69**
164_tal1†	44.07	59.13	44.36	46.58	47.77	49.12
167_tal1†	76.59	**57.43**	**55.97**	**70.96**	**63.44**	**64.02**
167_2_tal1†	155.25	**76.33**	**79.19**	**89.65**	**79.62**	**70.00**
170_tal1†	991.91	**428.78**	**283.17**	**370.08**	**320.10**	**364.20**
174_tal1†	166.27	**117.43**	**78.18**	**96.68**	**81.69**	**93.41**
183_tal1†	1164.38	**434.73**	**352.86**	**293.53**	**282.10**	**283.85**
185_tal1†	421.46	**420.86**	**257.90**	**250.55**	**227.47**	**237.63**
187_tal1†	388.94	**176.83**	**133.76**	**162.81**	**146.22**	**164.60**
191_tal1†	778.50	**543.60**	**362.56**	**351.87**	**306.36**	**278.76**
194_tal1†	2724.27	**1460.53**	**836.48**	**838.62**	**1071.73**	**760.78**
194_3_tal1†	1073.21	**526.67**	**377.00**	**382.41**	**378.75**	**367.19**
197_2_tal1†	1664.95	**1109.81**	**565.09**	**562.53**	**494.48**	**532.56**
197_3_tal1†	2224.30	**636.53**	**384.72**	**464.99**	**424.09**	**448.92**
205_tal1	140.82	–‡	**54.26**	**57.87**	**49.48**	**60.47**
207_tal1†	123.70	142.66	**106.20**	**112.18**	**112.20**	**105.18**
211_tal1†	4723.75	**1069.01**	**843.75**	**792.88**	**855.03**	**845.35**
215_tal1†	3511.71	**1250.68**	**808.16**	**795.47**	**782.70**	**776.69**
218_tal1†	284.59	**283.29**	**210.80**	**182.21**	**214.98**	**194.31**
234_tal1†	79.40	**74.02**	**70.73**	84.46	**78.58**	81.62
236_tal1†	790.85	**474.01**	**377.46**	**384.29**	**342.50**	**340.13**
237_tal1†	203.69	**139.88**	**101.94**	**120.84**	**98.98**	**105.39**
240_tal1†	10613.91	**1284.83**	**867.39**	**960.79**	**887.39**	**918.76**
241_tal1†	87.11	97.83	**73.72**	102.48	116.60	104.07
241_2_tal1†	733.78	**409.06**	**265.00**	**332.09**	**288.80**	**295.97**
242_tal1†	447.73	**333.66**	**163.09**	**272.29**	**211.07**	**190.92**
245_tal1†	2041.64	**1161.57**	**943.77**	**562.32**	**578.48**	**547.26**
252_tal1†	5200.35	**2616.03**	**1549.35**	**1693.78**	**1628.03**	**1622.80**
252_2_tal1†	492.37	**228.67**	**142.42**	**190.15**	**157.17**	**159.06**
255_tal1†	4994.45	**1201.31**	**910.11**	**1114.63**	**1085.83**	**1140.47**
⋮	⋮	⋮	⋮	⋮	⋮	⋮

– = not run

Table 3: Per-instance solve times (seconds), part 2 of 3.

	default	random	violation	diversity	hybrid	hybrid+
258_tal1†	428.87	**210.27**	**172.34**	**177.84**	**185.93**	**202.41**
262_tal1†	3716.93	**1013.17**	**747.25**	**816.98**	**850.33**	**749.67**
265_tal1†	339.04	**262.45**	**182.35**	**206.76**	**197.94**	**194.48**
269_tal1	(∞%)	**1678.54**	**802.60**	**1298.14**	**1242.68**	**1217.12**
272_tal1†	1118.66	**522.72**	**360.05**	**384.00**	**393.22**	**403.16**
273_tal1†	2012.70	**1046.13**	**773.94**	**537.09**	**563.45**	**514.06**
277_tal1†	35.33	97.70	**34.89**	41.11	47.79	36.04
287_2_tal1	(∞%)	**1594.75**	**1138.05**	**1082.88**	**875.09**	**919.81**
294_tal1†	1001.86	**553.17**	**407.21**	**539.92**	**334.36**	**298.78**
296_tal1	(∞%)	**4187.82**	**1787.91**	**1835.37**	**1883.65**	**1986.16**
299_tal1	(0.8%)	**3441.37**	**2705.15**	**3147.57**	**2095.27**	**3082.77**
303_tal1†	2122.25	**969.51**	**584.29**	**570.77**	**516.70**	**569.13**
307_tal1†	286.39	**244.89**	**166.08**	**177.40**	**163.26**	**175.98**
307_2_tal1†	7982.12	**6746.87**	**2693.59**	**3119.54**	**2664.39**	**2340.96**
310_tal1†	406.83	**278.73**	**254.34**	**332.69**	**289.10**	**327.91**
316_tal1†	2910.31	**1675.11**	**820.18**	**867.29**	**739.90**	**792.57**
323_tal1	(1.0%)	**754.25**	**344.52**	**568.28**	**561.18**	**529.18**
325_tal1†	4426.71	**3133.12**	**1535.02**	**1291.72**	**1350.46**	**1284.02**
330_tal1†	5616.16	**1830.80**	**1095.78**	**1295.56**	**1128.79**	**1148.98**
331_2_tal1†	3223.18	**1346.88**	**971.67**	**1114.58**	**885.08**	**1039.17**
335_tal1	(0.6%)	(∞%)	(0.6%)	(0.6%)	(0.6%)	(0.6%)
336_tal1†	5845.55	**2195.02**	**1461.72**	**1447.61**	**1321.60**	**1470.10**
337_tal1†	5161.23	**1333.99**	**2282.62**	**1088.95**	**930.31**	**1342.30**
341_tal1†	1577.17	**701.77**	**459.40**	**556.54**	**477.46**	**530.85**
343_tal1†	7319.81	**3087.70**	**1193.26**	**1245.10**	**1093.58**	**1059.51**
345_tal1†	9194.89	**1679.56**	**979.88**	**1066.94**	**1075.72**	**1047.13**
347_tal1	(∞%)	(∞%)	**7745.87**	**8500.19**	**8218.74**	**8824.06**
348_tal1†	1485.66	**609.22**	**361.93**	**431.94**	**417.30**	**425.87**
348_2_tal1	(∞%)	**10131.31**	**4749.67**	**3794.33**	**4021.12**	**4146.43**
349_tal1†	1975.06	**887.93**	**567.16**	**742.35**	**612.16**	**608.59**
349_2_tal1†	9341.39	**1858.41**	**879.59**	**1065.89**	**984.29**	**1193.05**
352_tal1	(0.0%)	**6652.27**	**4137.66**	**3602.61**	**3185.35**	**3342.92**
352_2_tal1†	557.92	**301.36**	**218.51**	**283.82**	**232.97**	**240.58**
358_tal1†	5412.90	**1636.21**	**1084.15**	**1170.30**	**1127.78**	**1129.02**
358_2_tal1	(∞%)	(∞%)	**6811.91**	**6168.39**	**4653.70**	**5811.00**
361_tal1	(0.0%)	**7765.31**	**4585.66**	**2848.28**	**2520.09**	**2806.56**
361_2_tal1†	274.87	852.92	**199.36**	**99.21**	**124.34**	**119.06**
371_tal1	(∞%)	(∞%)	**3862.46**	**3340.89**	**3426.42**	**4083.18**
371_2_tal1	(∞%)	**2438.76**	**1451.41**	**1507.75**	**1288.00**	**1363.80**
372_tal1†	5668.09	**1984.98**	**1608.46**	**1078.72**	**933.99**	**1003.02**
373_tal1†	3369.60	**1166.66**	**725.53**	**890.73**	**822.26**	**897.82**
377_tal1	(1.0%)	(∞%)	**5267.37**	**5342.40**	**9161.95**	**8502.99**
378_tal1	1175.71	**550.10**	–‡	**476.50**	–‡	**449.52**
381_2_tal1†	8131.65	**6126.74**	**2843.38**	**1950.43**	**1937.12**	**1933.48**
384_tal1†	230.19	**173.59**	**139.75**	**131.10**	**130.05**	**125.39**
386_tal1	(∞%)	(∞%)	**5238.17**	**6610.21**	**6292.98**	**5722.76**
394_tal1	(∞%)	**5062.83**	**2635.97**	**3136.26**	**2334.91**	**2920.36**
405_tal1	(0.3%)	(∞%)	**7130.81**	**6980.78**	**6094.57**	**6395.07**
416_tal1	(0.1%)	**3950.53**	**1513.11**	**1600.85**	**1627.18**	**1557.79**
418_tal1	(0.4%)	**7661.93**	**4217.11**	**4687.68**	**4330.22**	**4523.61**
⋮	⋮	⋮	⋮	⋮	⋮	⋮

– = not run

Table 4: Per-instance solve times (seconds), part 3 of 3.

	default	random	violation	diversity	hybrid	hybrid+
421_tal1	(∞%)	(∞%)	**7341.19**	**7809.27**	**6607.63**	**7668.48**
428_tal1	(0.3%)	**4008.27**	**3023.84**	**2043.28**	**2102.68**	**1991.36**
431_tal1†	702.30	**380.29**	**309.80**	**180.29**	**344.94**	**285.78**
434_tal1	(∞%)	**5239.96**	**3282.04**	**3647.77**	**3313.68**	**2769.09**
441_tal1†	3019.17	**1103.13**	**830.44**	**991.51**	**846.55**	**937.35**
458_tal1	(∞%)	(∞%)	**9676.16**	**7890.66**	**10539.37**	(∞%)
459_tal1	(0.1%)	**3966.61**	**2575.53**	**2368.07**	**2392.82**	**2432.41**
461_tal1†	4253.72	**2016.10**	**1224.29**	**1394.76**	**1416.73**	**1456.04**
462_tal1	(1.0%)	(∞%)	(∞%)	(∞%)	(1.0%)	(∞%)
466_tal1	(0.6%)	**5546.99**	**3237.35**	**3316.30**	**2955.37**	**3278.32**
473_tal1	(0.0%)	**10517.01**	**6197.43**	**6332.75**	**5526.00**	**6264.17**
486_tal1	(∞%)	(0.4%)	**7641.81**	**6572.60**	**6816.07**	**7060.37**
487_tal1	(∞%)	(∞%)	(0.1%)	(0.1%)	(0.1%)	(0.1%)
488_tal1†	1097.72	**749.04**	**650.27**	**627.10**	**616.45**	**544.85**
493_tal1	(0.5%)	**6075.30**	**4157.35**	**3999.02**	**3447.52**	**3539.54**
494_tal1†	221.76	238.94	**132.67**	**144.05**	**162.62**	**193.53**
498_tal1	(0.1%)	(∞%)	**6034.64**	**5241.29**	**3784.76**	**4396.09**
509_tal1	(0.5%)	(∞%)	(∞%)	(0.3%)	(0.4%)	(∞%)
511_tal1	(0.0%)	**5451.03**	**6136.12**	**4025.73**	**5198.58**	**5771.51**
512_tal1	(∞%)	**2801.74**	–‡	**2316.04**	**2413.71**	**2324.51**
518_tal1†	1929.09	**920.84**	**657.59**	**571.11**	**545.05**	**603.36**
525_tal1	2505.83	**838.06**	**824.72**	**920.58**	**898.10**	–‡
532_tal1	(0.8%)	(∞%)	(0.2%)	**9768.12**	(0.5%)	(0.4%)
540_tal1	(1.0%)	**6629.47**	**4290.00**	**4731.36**	**4006.97**	**4374.09**
555_tal1†	3494.14	**1177.96**	**931.21**	**837.01**	**816.47**	**820.37**
572_tal1	(0.0%)	(∞%)	(0.0%)	(0.0%)	(0.0%)	(0.0%)
583_tal1	(∞%)	(∞%)	**3613.46**	**4761.49**	(0.9%)	**4580.88**
585_tal1	(0.6%)	(∞%)	(0.2%)	(0.2%)	(0.6%)	(0.4%)
603_tal1	(0.9%)	(∞%)	(∞%)	(∞%)	(∞%)	(0.8%)
605_tal1†	37.82	73.81	**36.19**	72.79	77.17	68.27
613_2_tal1	(0.1%)	**5936.15**	**2642.92**	–‡	–‡	–‡
621_tal1	(∞%)	(∞%)	**8483.38**	**10069.87**	**9677.51**	**8933.56**
622_tal1	(∞%)	(∞%)	(∞%)	(0.3%)	(0.2%)	(0.3%)
629_tal1	(∞%)	(0.5%)	(0.5%)	(0.5%)	(0.4%)	(0.5%)
635_tal1	(0.1%)	(∞%)	(0.1%)	(0.1%)	(0.1%)	(0.1%)
644_tal1	7162.37	–‡	**1738.57**	**1858.73**	**1957.55**	**1830.37**
682_tal1	(0.6%)	(∞%)	(∞%)	(0.4%)	(∞%)	(∞%)
686_tal1	(∞%)	**10118.04**	**5280.22**	**5847.78**	**5127.56**	**5467.36**
699_tal1	10800.00	(∞%)	–‡	(∞%)	(∞%)	(∞%)
715_tal1	(0.6%)	–‡	(0.6%)	(0.6%)	(0.6%)	(0.6%)
715_2_tal1	(0.3%)	(0.3%)	(0.3%)	(0.3%)	(0.3%)	(0.3%)
719_tal1	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
720_tal1	(∞%)	**6711.55**	**7747.02**	**5272.84**	**5277.78**	**5297.45**
723_tal1	(∞%)	(∞%)	**10511.55**	**8368.94**	**7125.15**	**8414.85**
738_tal1	(0.5%)	(0.5%)	(0.5%)	(0.5%)	(0.5%)	(0.5%)
782_tal1	(∞%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)	(1.0%)
810_tal1	(0.8%)	(0.8%)	(0.8%)	(0.8%)	(0.8%)	(0.8%)
868_tal1	(0.4%)	(0.4%)	(∞%)	(0.3%)	(∞%)	(0.3%)
895_tal1	(0.2%)	(0.2%)	(0.2%)	(0.2%)	(0.2%)	(0.2%)
Solved	91	113	126	128	125	125
Solved by all	87/149	87/149	87/149	87/149	87/149	87/149
Sum	166753.30	**72127.58**	**44477.11**	**43755.92**	**41135.21**	**41737.71**
Mean	1916.70	**829.05**	**511.23**	**502.94**	**472.82**	**479.74**
Geo. Mean	629.34	**392.70**	**273.92**	**285.68**	**271.89**	**275.18**
Wilcoxon p	–	0.0000	0.0000	0.0000	0.0000	0.0000
Significance	(baseline)	***	***	***	***	***

–‡ = solver error – = not run

B Parameter Search

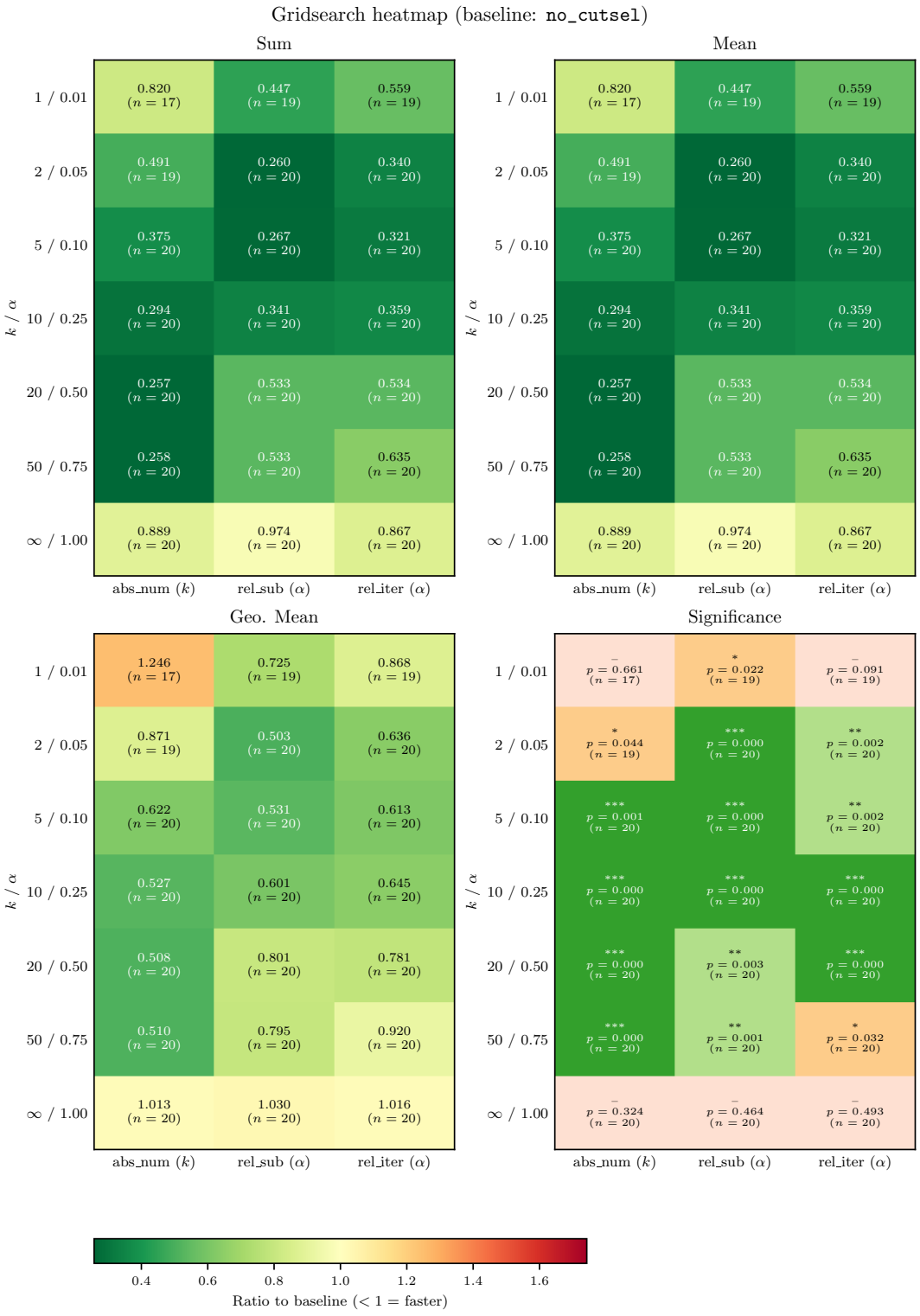

Figure 3: Parameter search on 20 validation instances. Each cell shows the ratio of shifted geometric mean solve time to the unfiltered baseline (lower is better). Three parameterizations are compared: fixed cut count k, subproblem fraction $\lceil \alpha \cdot |\mathcal{C}| \rceil$, and per-iteration violated-cut fraction $\lceil \alpha \cdot n_{\text{viol}} \rceil$. We select $\alpha = 0.05$ (subproblem fraction), which achieves the best geo mean ratio (0.503) with $p < 0.001$.