Adaptive Subproblem Selection in Benders Decomposition for Survivable Network Design Problems

Tim Donkiewicz*

Abstract

Scenario-based optimization problems can be solved via Benders decomposition, which separates first-stage (master problem) decisions from second-stage (subproblem) recourse actions and iteratively refines the master problem with Benders cuts. In conventional Benders decomposition, all subproblems are solved at each iteration. For problems with many scenarios, solving only a selected subset can reduce computation. We quantify the potential in selecting only those subproblems that yield cuts, and develop subproblem scoring and selection strategies. The proposed multi-criteria scoring methods combine historical subproblem performance metrics with problem-specific features, trained online via logistic regression to adapt to the changing likelihood of subproblem usefulness. Multiple stopping criteria balance exploration and exploitation: cut limits, proportional solve limits, and score thresholds. We evaluate our approach using a variant of the survivable network design problem as an example.

Computational experiments on 135 test instances demonstrate the potential and practical performance of subproblem selection. Analysis reveals that 52.1% of all subproblems solved are unnecessary (they contribute no cuts and occur outside cut-free rounds). An oracle with perfect foresight reduces total solve times by 34.4%. Random selection performs significantly worse than full enumeration, showing that naive strategies can degrade performance. Our best-scoring and selection method achieves statistically significant improvements in both runtime and primal-dual integrals. We show that informed subproblem selection can improve Benders decomposition, while highlighting challenges in developing reliable prediction models.

Keywords: Integer programming, Benders decomposition, subproblem selection, survivable network design, machine learning, operations research.

*Chair of Operations Research, RWTH Aachen University, Germany. E-mail: donkiewicz@or.rwth-aachen.de. ORCID: 0000-0002-5721-3563

1 Introduction

Benders decomposition [5] is a powerful technique for structured optimization problems, separating first-stage decisions from second-stage recourse problems. In an iterative manner, the master problem is solved to propose first-stage solutions. Then, subproblems corresponding to different scenarios are solved to check feasibility or optimality of the proposed solution. If any subproblem is infeasible or more expensive than expected by the master problem's estimation, Benders cuts are generated and added to the master problem to refine the solution space. This process repeats until the branch-and-bound algorithm terminates. Modern implementations of Benders decomposition apply Branch-and-Benders-cut [24], integrating Benders cuts as lazy constraints within a branch-and-bound framework for mixed-integer programming (see Figure 1).

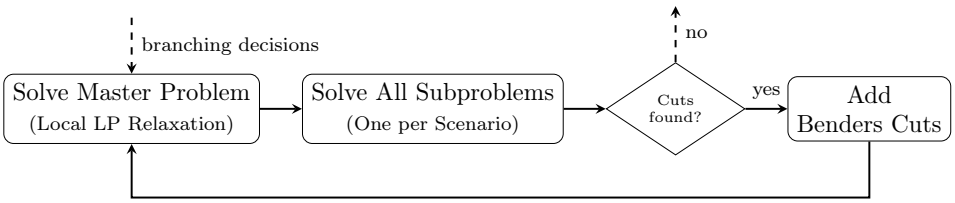

Figure 1: Basic Branch-and-Benders-cut framework.

However, when the number of scenarios grows large, solving all subproblems at each iteration becomes computationally prohibitive. Through computational experiments on survivable network design (SND) instances, we observe that 52.1% of subproblems solved under conventional Benders decomposition are unnecessary. This empirical observation motivates a data-driven computational approach to adaptive subproblem selection—we aim to dynamically learn which subproblems are most likely to contribute to the solving process by generating Benders cuts. To that end, we observe patterns in cut generation behavior, then prioritize subproblems accordingly, and stop once enough subproblems were solved (and at least one cut was generated).

We use SND as a testbed for developing adaptive subproblem selection strategies. SND problems exhibit characteristics that make them particularly suitable for this study: they naturally decompose into many subproblems (one per failure scenario), these subproblems vary substantially in their importance, and some scenarios can dominate others due to network topology. While our goal is not to advance the state-of-the-art for SND solving specifically, this problem class provides a suitable environment to demonstrate subproblem selection techniques.

To our knowledge, adaptive subproblem selection based on computational pattern learning has not been successfully demonstrated for Benders decomposition in prior work. We address this gap through four main contributions grounded in empirical analysis: First, we develop an oracle baseline

that quantifies the computational potential of perfect subproblem selection through empirical measurements. Second, we develop a multi-criteria scoring mechanism that combines observed historical performance metrics (cut generation frequency, cumulative contribution, recency) with computational features extracted from the current solution (failed edge capacity, flow, utilization, centrality). Third, we introduce an online learning approach using logistic regression that learns from computational observations during solving. We train incrementally on each subproblem outcome to predict scenario infeasibility and adapt to evolving cut generation patterns without requiring problem-specific tuning or offline training. The online training incurs a reasonable computational overhead—one update is comparable to solving a single subproblem—making it practical for real-time deployment. Fourth, we implement partial subproblem solving with multiple stopping criteria that balance computational effort between exploration and exploitation based on empirical performance indicators.

Our goal is twofold: demonstrate through computational experiments that dynamically learning from empirical observations can achieve statistically significant improvements over the baseline, and quantify via the oracle how much additional computational improvement could be attained with more sophisticated pattern recognition.

2 Related Work

Benders [5] first introduced the Benders decomposition algorithm for mixed-integer programming. Rahmaniani et al. [24] provide a comprehensive modern survey and an overview of Branch-and-Benders-cut. Benders decomposition has been applied to survivable network design problems [9, 16]. The SNDlib benchmark instances [19, 20] are widely used for evaluating network design algorithms.

2.1 Scenario Retention, Selection, and Partitioning

An approach to managing computational burden is to retain selected second-stage variables explicitly in the master problem rather than projecting them out through Benders cuts. Crainic et al. [13, 12] develop partial decomposition strategies for two-stage integer programs and stochastic multicommodity network design. Pauphilet et al. [23] show that random variable retention can achieve good performance. These methods fundamentally change the master problem structure.

Rather than modifying the master problem structure, scenario selection methods choose which subproblems to solve at each iteration. Rahmaniani et al. [24] note that solving only a subset of subproblems—especially early in the algorithm—can reduce computational burden, though such strategies remain underexplored for combinatorial optimization. Chen et al. [11] reformulate

the feasibility check for single-commodity survivable network design using an oracle based on minimum cut computations to identify violated scenarios to examine. Zhang et al. [29] propose optimization-based scenario reduction to approximate the recourse function while maintaining tight optimality gaps. Blanchot et al. [7] stop solving subproblems after the sum of the violation of obtained optimality cuts exceeds the current gap, often solving only 1% of subproblems. Mehamdi and Gendron [22] select pricing subproblems to solve in partial pricing in Branch-and-Price based on dual information. Celik and Toriello [10] develop a scenario-retention strategy that avoids resolving subproblems remaining feasible across iterations. Learning-based approaches have also emerged: Borozan et al. [8] use machine learning to predict which scenarios generate cuts binding in the final solution. These approaches adaptively select which subproblems to examine without modifying the problem structure.

2.2 Scoring Mechanisms for Algorithmic Decisions

The use of scoring functions to prioritize algorithmic decisions is common in mixed-integer programming [1, 2, 3], and multi-criteria scoring methods often outperform, e.g., pure violation-based approaches [18, 4]. Recent work has also explored machine learning for various selection decisions in MIP solvers, applying neural networks and reinforcement learning to cut selection, variable selection, and cut generation, learning to predict effectiveness from historical data and solution trajectories [25, 26, 27, 28, 30]. Li et al. [21] use machine learning to configure separator selection in branch-and-cut. Deza et al. [14] provide a comprehensive survey showing that while machine learning approaches demonstrate promise, they require substantial training data and careful feature engineering.

These principles from MIP solver design inform our multi-criteria scoring for subproblem selection, where we decide which feasibility checks to perform rather than selecting cuts from an existing pool.

2.3 Contributions in the Context of Related Works

Prior work on subproblem selection designs auxiliary optimization problems or heuristics to approximate subproblem objectives and identify scenarios to examine, often leveraging problem-specific structure. Our approach differs by scoring subproblems using multiple criteria and adaptively determining how many to solve without requiring problem reformulation. Through online learning from empirical observations during solving, we dynamically adapt prioritization to each instance and solution trajectory. We observe that for our test instances with feasibility cuts only, omitting even few cuts significantly degrades convergence, highlighting the challenge of effective subproblem selection.

Our multi-criteria scoring framework adapts principles from MIP solver decision-making (subsection 2.2) to decide which subproblems to solve before incurring costs, reducing computational effort by avoiding unnecessary solves entirely.

We do not aim to advance the state-of-the-art for survivable network design specifically, but rather use it as a representative problem class to develop and evaluate adaptive subproblem selection strategies applicable to Benders decomposition more broadly.

3 Problem Formulation

Consider a network $G = (V, E)$ with node set V and undirected edge set E. For each edge $\{i, j\} \in E$, we create two directed arcs: (i, j) and (j, i). Let A denote the set of all such directed arcs. For any node $v \in V$, we denote by $\delta^+(v)$ the set of arcs leaving v and by $\delta^-(v)$ the set of arcs entering v. We denote by $A(e)$ the two arcs corresponding to edge $e \in E$.

For each edge $e \in E$, capacity can be installed using a set of *capacity modules* M_e. Each module type $m \in M_e$ provides capacity $u_{e,m} > 0$ at installation cost $c_{e,m} \geq 0$. Edges may have preinstalled capacity (at zero cost). The number of installed modules m on each edge e is represented by $y_{e,m} \in \mathbb{N}$.

A set of demands D must be routed, where each demand $d \in D$ specifies a source node $o_d \in V$, destination node $t_d \in V$, and demand value $b_d > 0$. For convenience, define $b_{d,v} = b_d$ if $v = o_d$, $b_{d,v} = -b_d$ if $v = t_d$, and $b_{d,v} = 0$ otherwise.

We consider a set S of *failure scenarios*, where each scenario $s \in S$ is characterized by a subset $F_s \subseteq E$ of failed edges. We also define s_0 as the *base scenario* with $F_{s_0} = \emptyset$ (no failures), and let $S_0 = \{s_0\} \cup S$ denote all scenarios including the base case. The formulation supports arbitrary failure sets (single-edge, multi-edge, or scenario-specific combinations). In this paper, we focus on *single-edge failure scenarios*: $S = \{s_e : e \in E\}$ where $F_{s_e} = \{e\}$ (edge e fails), commonly known as N-1 reliability in telecommunications and power systems. The generalization to arbitrary failure sets is straightforward and does not affect the adaptive subproblem selection methodology.

This problem can be viewed as a two-stage integer program where, in each failure scenario, the demands must be routed. In the first stage, we install capacity modules on edges (integer decisions made before failures occur) and determine flow routing for the base scenario s_0. Then, in the second stage, for each scenario $s \in S$, we route demands on non-failed edges respecting installed capacities (recourse decisions after failures are revealed). The objective is to minimize total module installation costs.

Note that the second-stage decisions (flow routing) serve as recourse actions but have zero cost—we only consider routing feasibility, not routing

costs. The objective is purely to minimize first-stage installation costs, but the installed capacity must be sufficient to enable feasible routing under all failure scenarios.

3.1 Compact Formulation

Let $y_{e,m} \in \mathbb{N}$ be the number of modules of type m installed on link e and $f_{s,d,a} \in \mathbb{R}_+$ the flow of demand d on arc a in scenario s. We then consider the following compact formulation:

$$\min \quad \sum_{e \in E} \sum_{m \in M_e} c_{e,m} y_{e,m} \tag{1}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v)} f_{s,d,a} - \sum_{a \in \delta^-(v)} f_{s,d,a} = b_{d,v} \qquad \forall s \in S_0, d \in D, v \in V \tag{2}$$

$$\sum_{d \in D} \sum_{a \in A(e)} f_{s,d,a} \leq \sum_{m \in M_e} u_{e,m} y_{e,m} \qquad \forall s \in S_0, e \in E \setminus F_s \tag{3}$$

$$f_{s,d,a} = 0 \qquad \forall s \in S_0, d \in D, e \in F_s, a \in A(e) \tag{4}$$

$$y_{e,m} \in \mathbb{N} \qquad \forall e \in E, m \in M_e \tag{5}$$

$$f_{s,d,a} \geq 0 \qquad \forall s \in S_0, d \in D, a \in A \tag{6}$$

The capacity constraint (3) applies only to operational links $e \in E \setminus F_s$ in each scenario. For failed links $e \in F_s$, constraint (4) explicitly forces flow to zero on both arcs $a \in A(e)$ of the failed link, prohibiting any routing through failed components.

With $|S_0| = |E| + 1$ scenarios (base case plus all single-edge failures), we have $O(|S_0| \cdot |D| \cdot |A|)$ flow variables and $O(|S_0| \cdot (|D| \cdot |V| + |E|))$ constraints. When considering multi-edge failures, the number of scenarios grows exponentially, making the compact formulation impractical. For large networks, even solving the single-edge failure case becomes computationally intractable using conventional MIP solvers.

3.2 Benders Decomposition

Benders decomposition addresses the scalability issue by separating first-stage and second-stage decisions into a master problem and subproblems. Since we only consider feasibility in the second stage, we only use Benders *feasibility* cuts to iteratively refine the master problem.

Master Problem The Benders master problem includes first-stage capacity decisions and base case flow constraints:

$$\min \quad \sum_{e \in E} \sum_{m \in M_e} c_{e,m} y_{e,m} \tag{7}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v)} f_{s_0,d,a} - \sum_{a \in \delta^-(v)} f_{s_0,d,a} = b_{d,v} \qquad \forall d \in D, v \in V \tag{8}$$

$$\sum_{d \in D} \sum_{a \in A(e)} f_{s_0,d,a} \leq \sum_{m \in M_e} u_{e,m} y_{e,m} \qquad \forall e \in E \tag{9}$$

$$\text{Benders cuts (17)} \tag{10}$$

$$y_{e,m} \in \mathbb{N} \qquad \forall e \in E, m \in M_e \tag{11}$$

$$f_{s_0,d,a} \geq 0 \qquad \forall d \in D, a \in A \tag{12}$$

Constraints (8) and (9) enforce flow conservation and capacity constraints for the base case scenario s_0 with no failed edges ($F_{s_0} = \emptyset$), ensuring basic feasibility before considering failure scenarios.

Subproblem For each failure scenario $s \in S$ and candidate solution $\bar{y}$, the subproblem checks feasibility:

$$\min \quad 0 \tag{13}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v)} f_{s,d,a} - \sum_{a \in \delta^-(v)} f_{s,d,a} = b_{d,v} \qquad \forall d \in D, v \in V \tag{14}$$

$$\sum_{d \in D} \sum_{a \in A(e)} f_{s,d,a} \leq \sum_{m \in M_e} u_{e,m} \bar{y}_{e,m} \qquad \forall e \in E \setminus F_s \tag{15}$$

$$f_{s,d,a} = 0 \qquad \forall d \in D, e \in F_s, a \in A(e) \tag{16}$$

Constraints (14) enforce flow conservation (corresponding to (2) in the compact formulation), and constraints (15) enforce capacity limits on operational edges (corresponding to (3)). For failed edges $e \in F_s$, constraints (16) explicitly set flow to zero on both arcs, ensuring no routing through failed components.

We observe that this is a multicommodity flow problem with fixed capacities determined by the first-stage solution $\bar{y}$. It is solvable in polynomial time as a linear program with $O(|D| \cdot |A|)$ variables and $O(|D| \cdot |V| + |E|)$ constraints.

If the subproblem is feasible, the capacity $\bar{y}$ is sufficient for scenario s. If infeasible, we generate a Benders feasibility cut using the subproblem's dual solution.

Benders Feasibility Cuts When the subproblem for scenario s is infeasible, we obtain an unbounded ray (π^s, σ^s) of the dual problem, where $\pi^s_e \geq 0$ are the dual variables for capacity constraints (15) and $\sigma^s_{d,v}$ are the (unrestricted) dual variables for flow conservation constraints (14). The Benders feasibility cut is:

$$\sum_{e \notin F_s} \pi^s_e \sum_{m \in M_e} u_{e,m} y_{e,m} \geq - \sum_{d \in D} \sum_{v \in V} \sigma^s_{d,v} b_{d,v} \tag{17}$$

where we sum only over operational edges $e \notin F_s$ since failed edges have zero available capacity.

4 Adaptive Subproblem Selection

Our data-driven approach is based on two observations. Firstly, it is not required to solve all subproblems at each Benders iteration to make progress toward feasibility. Secondly, not all subproblems are equally likely to generate cuts. If we are able to predict which scenarios are unlikely to produce a cut, we can avoid solving those subproblems entirely, saving computation time.

We therefore propose Branch-and-Benders-cut with *adaptive subproblem selection*: prioritize which subproblems to solve first based on learned importance, and limit the number of subproblems solved per iteration, as illustrated in Figure 2. At each Benders iteration, we first extract features from the master solution (topology metrics like capacity, flow, and utilization). We then score all scenarios based on their predicted likelihood of generating cuts using a machine learning model. Scenarios are examined in decreasing score order, with stopping criteria determining when to terminate subproblem solving. If there exists any, at least one cut is added in each round. After each subproblem is solved, we train the regression model online using the observed outcome and updated historical performance (reliability, violation magnitude). Feasibility cuts from infeasible subproblems are added to the master, and the process repeats until convergence.

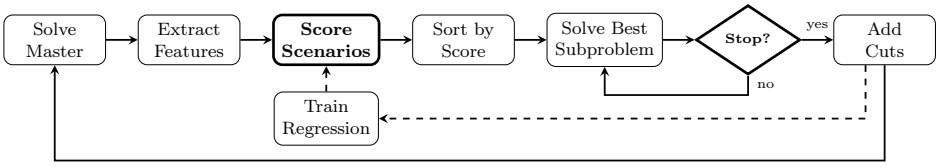

Figure 2: Adaptive subproblem selection pipeline in Benders decomposition. Scoring and subproblem selection step are highlighted.

In this section, we first describe an oracle that selects an optimal subproblem solving sequence to not miss any cuts. Then, we present our multi-criteria subproblem scoring mechanism in Section 4.2. Building upon the scoring system, we introduce stopping criteria for subproblem solving iterations in Section 4.3, aiming to approximate the oracle's behavior.

4.1 Oracle for Evaluating Prediction Quality

To assess scoring quality, we define a *cut prediction oracle* $\mathcal{O}^*$ with perfect foresight of all cuts generated until the algorithm converges. Let $\Pi = (P_1, P_2, \ldots, P_T)$ denote a subproblem selection sequence, where $P_t \subseteq S$ are the examined scenarios in iteration t. We define the set of cut-yielding

subproblems in iteration t as $V_t \subseteq S$. The *optimal subproblem selection* as given by $\mathcal{O}^*$ is $\Pi^* = (V_1, V_2, \ldots, V_t)$, the sequence consisting of exactly those subproblems yielding a cut. Note that a sequence $\Pi = (P_1, P_2, \ldots, P_T)$ with $P_t \cap V_t \subsetneq V_t$, i.e., missing some cut-yielding subproblems, may lead to lower overall runtime, but we focus on approximating the oracle's behavior.

Definition 1 (Cut Prediction Oracle Bound)**.** *Let τ_s^t denote the time to solve subproblem s at iteration t. Let $T(\Pi)$ denote the total subproblem solving time over all iterations for sequence Π. The cut prediction oracle achieves:*

$$T(\Pi^*) = \sum_{t=1}^{T} \sum_{s \in V_t} \tau_s^t \leq \sum_{t=1}^{T} \sum_{s \in S} \tau_s^t = T(\Pi_{all}). \tag{18}$$

Since identical cuts are added, the runtime behavior is equal otherwise, apart from effects of warmstarting.

In gerneral, this cut prediction oracle is not computable a priori, but rather serves as a benchmark. Our goal is to design a scoring mechanism that approximates the oracle's performance by prioritizing subproblems likely to generate cuts.

4.2 Subproblem Scoring

For each scenario $s \in S$, we maintain a score $\text{score}(s) \in [0, 1]$ that estimates its likelihood of generating a violated cut. We present a machine learning-based scoring approach that uses network topology features and historical performance metrics (Section 4.2.2).

4.2.1 Score Components

We utilize information derived from the master problem solution, in conjunction with the failure scenario and from historical scenario performance. Let $(\bar{y}, \bar{f})$ denote the current master solution where $\bar{y}_{e,m}$ is the number of modules of type m installed on link e, and $\bar{f}_{s_0,d,a}$ is the flow of demand d on arc a in the base case scenario s_0 (no failures). Let C_s denote the set of iterations where scenario s generated a cut, and I_s denote iterations where s was examined. Let $\nu_s^{t'}$ denote the violation magnitude of the Benders cut generated by scenario s at iteration $t' \in C_s$ (measured as the right-hand side value of the cut). Let betweenness centrality β_e be the fraction of shortest paths (in the network topology, without considering modules) between each demand pair that contain edge e. Using these definitions, we introduce eight score components as described in Table 1. Note that the violation magnitude $\nu_s^{t'}$ for feasibility cuts is not necessarily directly comparable, since the dual ray may scale arbitrarily.

Table 1: Score components for scenario prioritization.

Name	Definition	Description
Reliability	$\rho_s = \lvert C_s\rvert / \lvert I_s\rvert$	Fraction of times s produced a cut when solved
Total Share	$\tau_s = \lvert C_s\rvert / \sum_{s' \in S} \lvert C_{s'}\rvert$	Fraction of all cuts produced by s
Average Violation	$\bar{\nu}_s = \frac{1}{\lvert C_s\rvert} \sum_{t' \in C_s} \nu_s^{t'}$	Mean violation magnitude when s generated cuts
Staleness	$\zeta_s = t - \max\{t' : t' \in I_s\}$	Iterations since s was last examined
Failed Edge Capacity	$\kappa_e = \sum_{m \in M} u_{e,m} \bar{y}_{e,m}$	Total capacity on failed edge $e \in F_s$
Failed Edge Flow	$\psi_e = \sum_{d \in D} (\lvert \bar{f}_{s_0,d,a}\rvert + \lvert \bar{f}_{s_0,d,a'}\rvert)$	Base case flow on failed edge $e \in F_s$
Failed Edge Utilization	$\mu_e = \psi_e / \kappa_e$	Flow-to-capacity ratio on $e \in F_s$ (when $\kappa_e > 0$)
Betweenness Centrality	β_e	Topological centrality of edge $e \in F_s$ (computed once)

A straightforward approach would combine score components via weighted linear combination $R(s) = w_\rho \rho_s + w_\tau \tau_s + w_\zeta \zeta_s + \ldots$ with min-max normalization. However, determining appropriate weights is challenging: they vary across instances and solution phases. For problems with long solve times, static weights become problematic as the optimal balance shifts dynamically. This motivates the machine learning approach described next, which learns feature importance automatically.

4.2.2 Machine Learning-Based Scoring

Our regression-based approach (in the following, referred to as machine learning, or "ML") predicts scenario infeasibility using logistic regression trained online during the Benders decomposition process. We construct an 8-dimensional feature vector $\phi(s) \in \mathbb{R}^8$ for each scenario s combining the failed edge capacity κ_e, flow ψ_e, utilization μ_e, betweenness centrality β_e, and the historical components $\bar{\nu}_s$, ρ_s, τ_s, ζ_s described above. Features are normalized to zero mean and unit variance using Welford's online algorithm, with normalization parameters accumulated across the entire solution process (not reset during stabilization rounds).

Logistic Regression Model We use logistic regression for its low computational complexity during online training and superior interpretability compared to more sophisticated models like neural networks or ensemble methods.

Logistic regression models the probability of binary outcomes via a sigmoid transformation of a linear predictor. The model predicts infeasibility probability as:

$$p(s \text{ infeasible} \mid \phi(s); w, b) = \sigma(w^\top \phi(s) + b) \tag{19}$$

where $w \in \mathbb{R}^8$ is the weight vector, $b \in \mathbb{R}$ is the bias term, and $\sigma(z) = 1/(1+e^{-z})$ is the logistic function mapping $\mathbb{R} \to (0,1)$.

This formulation allows the model to learn non-uniform importance of features: features strongly predictive of infeasibility receive large absolute weights, while less informative features receive weights near zero. The sigmoid ensures output values are valid probabilities.

Online Training via Gradient Descent After solving the subproblem for scenario s at iteration t, we observe the outcome $z_s^t \in \{0,1\}$ where $z_s^t = 1$ if the subproblem was infeasible (generated a cut) and $z_s^t = 0$ if feasible. We update the model parameters to minimize the logistic loss:

$$\ell(w,b) = -\log p(z_s^t \mid \phi(s); w, b) \tag{20}$$

Applying gradient descent with L2 regularization yields the update rules:

$$w \leftarrow w - \alpha \left[(\hat{z}_s^t - z_s^t)\phi(s) + \lambda w\right] \tag{21}$$

$$b \leftarrow b - \alpha(\hat{z}_s^t - z_s^t) \tag{22}$$

where $\hat{z}_s^t = \sigma(w^\top \phi(s) + b)$ is the predicted probability, $\alpha > 0$ is the learning rate controlling step size, and $\lambda \geq 0$ is the regularization parameter preventing overfitting by penalizing large weights.

The gradient term $(\hat{z}_s^t - z_s^t)$ represents the prediction error: positive when the model overestimates infeasibility probability, negative when underestimating. The regularization term λw in (21) shrinks weights toward zero, favoring simpler models that generalize better.

4.3 Subproblem Selection

Table 2: Stopping criteria for partial subproblem solving and naming convention. One or more can be imposed, and a Benders iteration stops subproblem solving when any criterion is met. Stabilization rounds solve all subproblems to prevent selection bias and provide additional training data.

Name	Identifier	Description
Cut limit	k`C`	Stop if k cuts have been generated
Relative solve limit	p`P`	Stop after examining a proportion $p \in [0,1]$ of all scenarios, i.e., $\lceil p \cdot \lvert S \rvert \rceil$ subproblems
Consecutive misses	m`M`	Stop if no cuts found for m consecutive scenarios
Score limit	r`T`	Stop when the score of the next scenario falls below threshold r
Time limit	—	Stop if iteration time limit reached
Stabilization (solving all subproblems)		
Stabilization	n`S`	Every n iterations
Root node	n`R`	First n rounds at root

We solve subproblems in order of decreasing score. Using the stopping criteria introduced in Table 2, we aim to examine all cut-generating scenarios, and to stop once we are unlikely to find more cuts. Since limited examination may miss some cut-generating scenarios and, over time, introduce a selection bias, we periodically perform *stabilization rounds* where all scenarios are examined (disabling stopping criteria). This prevents systematic exclusion of subproblems and provides additional training data. Since initial training data is especially important, we always initialize the scores with at least one score initialization round, but up to n stabilization rounds at the root node (nR). At stabilization rounds, we reset the score components ρ_s, τ_s, $\bar{\nu}_s$ and ζ_s to avoid bias from outdated historical data.

5 Computational Experiments

5.1 Experimental Setup

We implement our Branch-and-Benders-cut framework with adaptive cut selection using Gurobi 12.0.3 [17], Julia 1.12.4 [6] with JuMP 1.12 [15], and lazy constraint callbacks. To improve efficiency, we maintain a single subproblem template adjusted for each scenario rather than constructing new subproblems from scratch. We reuse the Gurobi environment, allowing for LP warmstarting. We do not perform separation on fractional solutions as to isolate the impact of subproblem selection.

Table 3 presents the configuration parameters for all evaluated methods. We parameterize those configurations following an analysis of stopping criteria: For each (threshold, proportion, cut limit) individually, we examine how many cuts would have been generated under this setting. A distribution boxplot can be found in Figure 5 in Appendix A. Based on this analysis, we select more conservative (calling more subproblems, possibly solving more unrequired ones) parameters as well as more aggressive ones to evaluate the impact of stopping criteria. In particular, setting `ML-0.1T-0.6P-10C-20S-5R` uses all three main stopping criteria, and sets the limit at the upper quartile of required cuts per iteration (see Figure 5 in Appendix B). That way, we expect to capture most cut-generating scenarios while avoiding excessive unrequired subproblem solves.

Table 3: Configuration parameters for tested methods. ML methods use online logistic regression. See Table 2 for parameter descriptions.

Method	Selection	Threshold	Proportion	Cut Limit	Stabilization	Root Node Stabilization
Standard	All	—	—	—	—	—
`Random-0.5P-20S-5R`	Random	—	0.5	—	20	5 rounds
`ML-0.1T-0.6P-10C-20S-5R`	Threshold+Prop	0.1	0.6	10C	20	5 rounds
`ML-0.5P-20S-5R`	Proportion	—	0.5	—	20	5 rounds
`ML-0.5T-5C-20S-5R`	Threshold	0.5	—	5C	20	5 rounds
`ML-0.7P-20S-5R`	Proportion	—	0.7	—	20	5 rounds

Out of the original SNDlib instances [19, 20], only 11 solve within our time limit of three hours, which yields an insufficient sample for meaningful evaluation. We generate test instances from SNDlib benchmark networks by combining 2–5 subgraphs: (1) extract connected subgraphs via BFS from high-degree nodes using proportion $p \in [0.45, 0.65]$ to obtain $p \cdot |V|$ nodes per network; (2) merge subgraphs by connecting high-degree nodes; (3) generate N-1 failure scenarios ($|S| = |E|$); (4) filter scenarios that disconnect demand nodes. We generate 135 test instances with 16–229 scenarios, 8–57 nodes, 13–82 edges, and 18–857 demands. The online ML training overhead is negligible, comparable to solving a single subproblem per iteration. For stabilization frequency, we tested values of 5, 10, 20, 50, 100, and 200 iterations, with 20 providing the best balance for SND problems.

All experiments use machines with 2x Intel Xeon L5630 Quad Core processors at 2.13 GHz (8 cores, 16GB RAM). A three-hour time limit is imposed per instance. Statistical comparisons use the shifted geometric mean (shift of 10) to handle zero values.

The parameters for the regression model were determined using a grid search on a set of 14 instances, generated synthetically by the above procedure. We tried configurations with learning rates $\alpha \in \{0.02, 0.05, 0.075, 0.1, 0.15\}$ and regularization $\lambda \in \{0.05, 0.02, 0.01, 0.005, 0.001\}$. The selected parameters ($\alpha = 0.075$, $\lambda = 0.02$) provided the best runtime performance on the tuning set.

5.2 Results

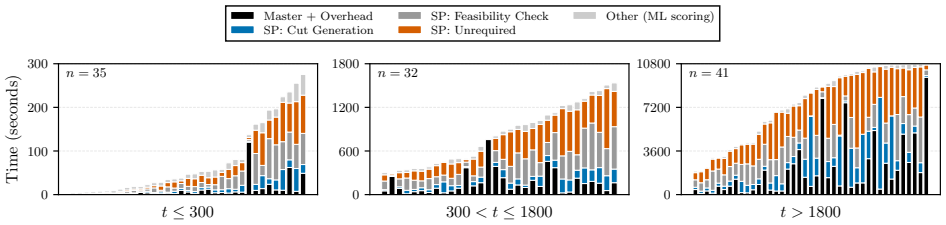

Figure 3: Time breakdown for. Master time (black), cut-generating subproblems (blue), and feasibility-proving subproblems in cut-free iterations (grey) represent necessary computation, unrequired subproblems (orange) represent wasted computation.

The potential for subproblem selection is established through the cut prediction oracle bound (see 1). As illustrated in Figure 3 (and described in detail in Table 5 in Appendix A), 34.4% of total solve time in standard Benders is spent on unrequired solving of subproblems. Since solving all subproblems that yield cuts does not always lead to the best runtime, this is not necessarily an upper bound on the achievable speedup. However, we

Table 4: Summary of runtime performance across all methods ($n = 79$ instances solved by all methods). Solved: instances solved to optimality; Sum/Mean/Sh. Geom. Mean†: computed over instances solved by all methods. Wilcoxon signed-rank test (one-sided): tests if each method is significantly better (faster) than Standard. Significance levels: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$, n.s. = not significant.

Method	Solved	Sum†	Mean†	Sh. Geom. Mean†	Wilcoxon p	Signif-icance
Standard	86	100941.2	1277.7	300.5	—	—
`Random-0.5P-20S-5R`	82	112118.3	1419.2	334.8	0.9996	n.s.
`ML-0.5T-5C-20S-5R`	86	86715.2	1097.7	270.8	0.0130	*
`ML-0.5P-20S-5R`	89	96097.3	1216.4	285.2	0.0136	*
`ML-0.7P-20S-5R`	88	100536.4	1272.6	288.3	0.2080	n.s.
`0.1T-0.6P-10C-20S-5R`	87	92977.6	1176.9	275.3	0.0081	**

consider it as a dimension for comparison.

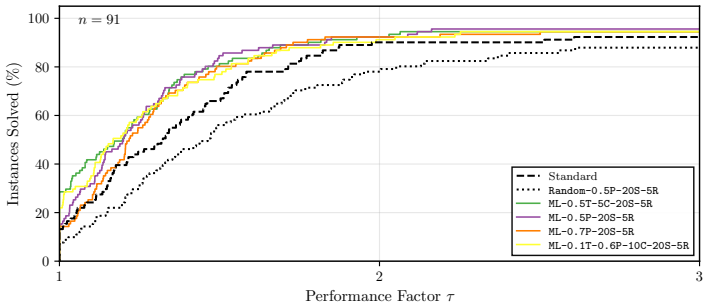

Figure 4: Performance profile on all instances solved by at least one method: Random selection performs worse than standard Benders, regression-based methods achieve modest gains.

Runtime Performance As illustrated in Figure 4 and detailed in Table 4, three out of four tested ML configurations achieve statistically significant speedups over standard Benders, the best being `ML-0.1T-0.6P-10C-20S-5R`. The reduction from 300.5 to 270.8 represents a speedup of 9.9%, as opposed to a "theoretical" maximum of 34.4% (see Table 5 in Appendix A). Since we aim for reducing unrequired subproblem solving times, the performance profile reduced to only those times shows a more pronounced improvement (see Figure 9 in Appendix F). The full results can be found in Table 7 in Appendix G. We observe that our scoring method indeed outperforms random selection, which is statistically significantly worse than standard Benders ($p = 0.9996$). This confirms that naive subproblem selection strategies can degrade performance, and that our proposed scoring mechanism is able to prioritize important subproblems.

Convergence Quality Beyond runtime performance, primal-dual integral analysis (detailed results in Table 8 in Appendix H) evaluates convergence quality. All four ML configurations achieve statistically significant improvements in convergence quality over standard Benders, with the best method `ML-0.5T-5C-20S-5R` reducing the shifted geometric mean from 1336.3 to 1145.9 billion (14.3% improvement). Random selection shows no significant difference from standard Benders ($p = 0.12$). These results demonstrate that ML-based subproblem selection not only reduces solve time but also improves the quality of the search by better exploring the solution space.

The ML model achieves an average accuracy of 84.4%, precision of 33.4%, recall of 70.9%, and F1-score of 45.2% (see Table 6 in Appendix B). The mediocre scores mean that we are "missing" cuts, which can lead to worse algorithm convergence that can also be seen in the runtime table (Table 7 in Appendix G).

The ML model's imperfect predictions can be attributed to two main factors: first, the features used may not fully capture the complex relationships around scenario violations (as can be observed by the high variance in feature importance, see Figure 6 in Appendix C) and second, the online training approach may not obtain sufficient training data for robust learning, especially when instances solve quickly.

To counteract the former, we also tested locality-based features (e.g., utilization in the n-hop neighborhood of failed edges) but they did not receive significant weights in our preliminary experiments. Addressing the latter, we tried pre-training on similar instances, but as Figure 8 in Appendix E shows, cut generation patterns evolve substantially during solving, and so do the weights of the ML model. Additionally, we implemented root node stabilization that serves the purpose of establishing a strong initial ML model and facilitating early primal heuristics. However, further research is needed to develop more reliable prediction models that can consistently deliver robust performance improvements across diverse instances.

6 Conclusions and Future Work

This paper investigated adaptive subproblem selection in Benders decomposition, motivated by the empirical observation that 52.1% of subproblems in our variant of the survivable network design problem unnecessarily solved: The rest either yield cuts (5.6%), or verify feasibility in cut-free iterations (42.4%). We developed a multi-criteria scoring mechanism combining historical performance metrics with topology-aware features, trained online via logistic regression. Multiple stopping criteria (score thresholds, proportional limits, cut limits) adaptively determine how many subproblems to solve per iteration. The online training overhead is negligible—comparable to solving a single subproblem—making the approach practical for real-time deployment.

Computational experiments on 135 survivable network design instances reveal that in standard Benders with full subproblem enumeration, 34.4% of total solve time is spent on unrequired subproblem solving. Our best runtime configuration (`ML-0.1T-0.6P-10C-20S-5R`) achieves 9.9% speedup over standard Benders (270.8s vs 300.5s shifted geometric mean, $p = 0.0081$), while the best convergence configuration (`ML-0.5T-5C-20S-5R`) reduces primal-dual integrals by 14.3% ($p < 0.001$). Three of four ML configurations achieve statistically significant runtime improvements, and all achieve statistically significant improvements in primal-dual integrals. While the scoring performance is modest (the regression model achieves only 33.4% precision, 70.9% recall, and 45.2% F1), these results establish that adaptive subproblem selection based on learned importance can improve Benders decomposition performance. Combining multiple stopping criteria appears to be a promising direction, as our best method employs three types simultaneously. Random selection degrades performance significantly ($p = 0.9996$), confirming that informed prioritization is necessary.

Future Research Directions Multiple future research directions can be identified: First, more sophisticated prediction models—such as graph neural networks—could better capture network topology and temporal evolution of cut generation patterns, potentially improving precision and allowing for preconditioning on a set of similar instances. Second, rather than using fixed thresholds, dynamic adaptation of stopping criteria during solving could adjust exploration-exploitation tradeoffs based on observed performance, also taking into account cut usefulness. Third, analyzing feature importance across different solving stages could reveal which metrics are most predictive early versus late in the algorithm, enabling stage-specific scoring. Finally, auxiliary (problem-specific) optimization techniques based on min-cut computations could provide tighter bounds on scenario criticality, complementing the learned scoring mechanism. These directions could help close the gap between current performance and the theoretical potential identified through our analysis of subproblem selection potential.

Acknowledgements

The author sincerely thanks Oliver Gaul for his help in previous work around adaptive subproblem selection and for insightful discussions. The author also thanks Alexander Helber and Marco Lübbecke for their helpful comments.

References

[1] Achterberg, T.: Constraint integer programming. Ph.D. thesis, Technische Universität Berlin (2007)

[2] Achterberg, T.: SCIP: solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)

[3] Andreello, G., Caprara, A., Fischetti, M.: 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)

[4] Avella, P., Mattia, S., Sassano, A.: Metric Inequalities and the Network Loading Problem. In: Discrete Optimization: The State of the Art, pp. 53–84. Elsevier (2004)

[5] Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische mathematik **4**(1), 238–252 (1962)

[6] Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM review **59**(1), 65–98 (2017)

[7] Blanchot, X., Clautiaux, F., Detienne, B., Froger, A., Ruiz, M.: 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)

[8] Borozan, S., Giannelos, S., Falugi, P., Moreira, A., Strbac, G., Zhang, T.: Machine learning-enhanced Benders decomposition approach for the multi-stage stochastic transmission expansion planning problem. Electric Power Systems Research **226**, 109956 (2024)

[9] Botton, Q., Fortz, B., Gouveia, L., Poss, M.: Benders decomposition for the hop-constrained survivable network design problem. INFORMS Journal on Computing **25**(1), 13–26 (2013)

[10] Çelik, Ş., Martin, L., Schrotenboer, A.H., Van Woensel, T.: Exact two-step Benders decomposition for the time window assignment traveling salesperson problem. Transportation Science **59**(2), 210–228 (2025)

[11] Chen, R.L., Cohn, A., Pinar, A.: An implicit optimization approach for survivable network design. In: 2011 IEEE Network Science Workshop. pp. 159–163. IEEE (2011)

[12] Crainic, T.G., Hewitt, M., Maggioni, F., Rei, W.: Partial Benders decomposition: general methodology and application to stochastic network design. Transportation Science **55**(2), 414–435 (2021)

[13] Crainic, T.G., Hewitt, M., Rei, W.: Partial decomposition strategies for two-stage stochastic integer programs. CIRRELT (Université de Montréal) **88** (2014)

[14] Deza, A., Khalil, E.B.: Machine learning for cutting planes in integer programming: A survey. arXiv preprint arXiv:2302.09166 (2023)

[15] Dunning, I., Huchette, J., Lubin, M.: JuMP: A modeling language for mathematical optimization. SIAM Review **59**(2), 295–320 (2017)

[16] Fortz, B., Poss, M.: An improved Benders decomposition applied to a multi-layer network design problem. Operations Research Letters **37**(5), 359–364 (2009)

[17] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), `https://www.gurobi.com`

[18] Hosseini, M., Turner, J.: Deepest Cuts for Benders Decomposition. Operations Research **73**(5), 2591–2609 (2024)

[19] Idzikowski, F., Orlowski, S., Raack, C., Woesner, H., Wolisz, A.: Dynamic routing at different layers in IP-over-WDM networks – Maximizing energy savings. Optical Switching and Networking, Special Issue on Green Communications (2011)

[20] Koster, A.M., Kutschka, M., Raack, C.: Towards Robust Network Design using Integer Linear Programming Techniques. In: Proceedings of the NGI 2010, Paris, France. Next Generation Internet, Paris, France (June 2010)

[21] Li, S., Ouyang, W., Paulus, M.B., Wu, C.: Learning to Configure Separators in Branch-and-Cut. In: Advances in Neural Information Processing Systems. vol. 37 (2023)

[22] Mehamdi, A.B., Lacroix, M., Martin, S.: Pricing filtering in Dantzig-Wolfe decomposition. Operations Research Letters **58**, 107207 (2025)

[23] Pauphilet, J., Wu, Y.: The surprising performance of random partial Benders decomposition. Optimization Online (2025), 32181

[24] Rahmaniani, R., Crainic, T.G., Gendreau, M., Rei, W.: The Benders decomposition algorithm: A literature review. European Journal of Operational Research **259**(3), 801–817 (2017)

[25] Scavuzzo, L., Aardal, K., Lodi, A., Yorke-Smith, N.: Machine learning augmented branch and bound for mixed integer linear programming. Mathematical Programming (2024)

[26] Tang, Y., Agrawal, S., Faenza, Y.: Reinforcement learning for integer programming: Learning to cut. In: International Conference on Machine Learning. pp. 9367–9376. PMLR (2020)

[27] Turner, M., Koch, T., Serrano, F., Winkler, M.: Adaptive cut selection in mixed-integer linear programming. Open Journal of Mathematical Optimization **4**, 1–25 (2023)

[28] Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., Wu, F.: Learning cut selection for mixed-integer linear programming via hierarchical sequence model. arXiv preprint arXiv:2302.00244 (2023)

[29] Zhang, W., Wang, K., Jacquillat, A., Wang, S.: Optimized scenario reduction: Solving large-scale stochastic programs with quality guarantees. INFORMS Journal on Computing **35**(4), 886–908 (2023)

[30] Zhang, X., Chen, L., Yang, Z., Zeng, Z.: Learning to select cutting planes in mixed-integer linear programming solving. Expert Systems with Applications **255**, 124654 (2025)

Appendix

A Subproblem Statistics

Table 5: Time and count breakdown for ML-train method across instances that completed within the time limit ($n = 86$). Unrequired subproblems are those that do not yield cuts in iterations where other subproblems do. Cut Generation shows subproblems that yielded cuts. Feasibility Proving shows subproblems in cut-free iterations (required to verify feasibility). Other includes ML scoring and callback overhead.

	Time			Subproblems	
Category	Time (s)	% Total	% SP	Count	% SP
Master Problem	87,626	25.8%	—	—	—
SP: Cut Generation	56,712	16.7%	23.5%	20,485	5.6%
SP: Feasibility Proving	68,367	20.2%	28.3%	155,251	42.4%
SP: Unrequired	**116,594**	**34.4%**	**48.2%**	**190,788**	**52.1%**
Other (ML, overhead)	9,786	2.9%	—	—	—
Total	339,086	100.0%	—	366,524	100.0%

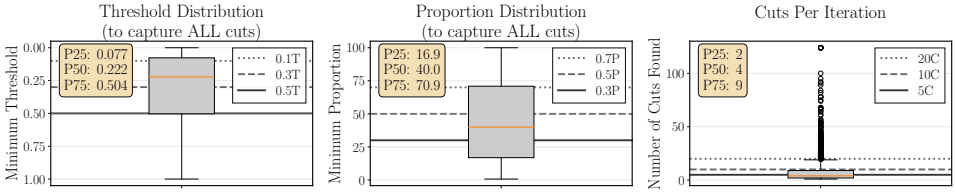

Figure 5: Distribution of minimum parameter values required to capture all cuts per iteration during ML training, with reference lines showing tested configuration values.

B Regression Training Performance Analysis

Table 6: Aggregated ML Model Classification Metrics (%) by Configuration. Values show geometric mean across all instances. Standard includes full enumeration, enabling slightly better performance due to more training data.

Configuration	Accuracy	Precision	Recall	F1-Score
Standard	86.0	34.3	71.0	46.1
`ML-0.5P-20S-5R`	83.8	32.8	70.4	44.6
`ML-0.5T-5C-20S-5R`	83.9	32.6	69.8	44.3
`ML-0.7P-20S-5R`	84.9	34.0	71.5	46.0
`ML-0.1T-0.6P-10C-20S-5R`	83.5	33.2	71.7	45.2
Average	84.4	33.4	70.9	45.2

C Regression Model Feature Importance

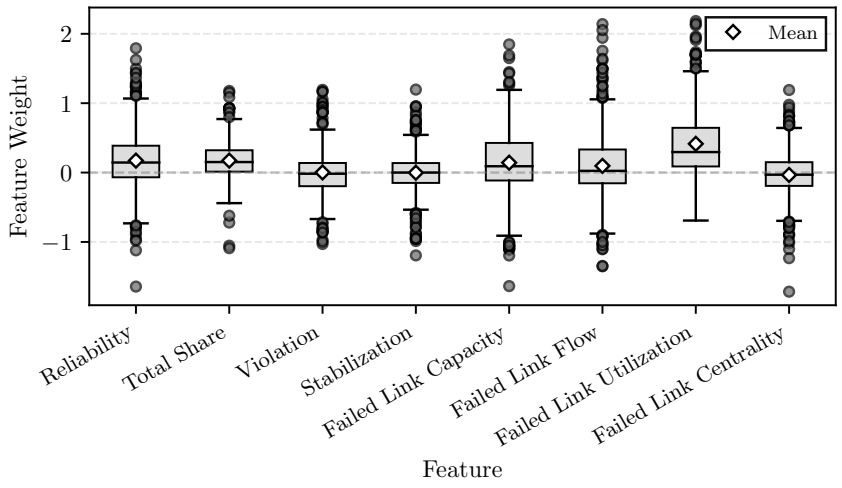

Figure 6: Distribution of learned feature weights across instances. Feature reference in Table 1. Box plots show median (black line), quartiles (box), and mean (white diamond).

D Subproblem Hit Quality Analysis

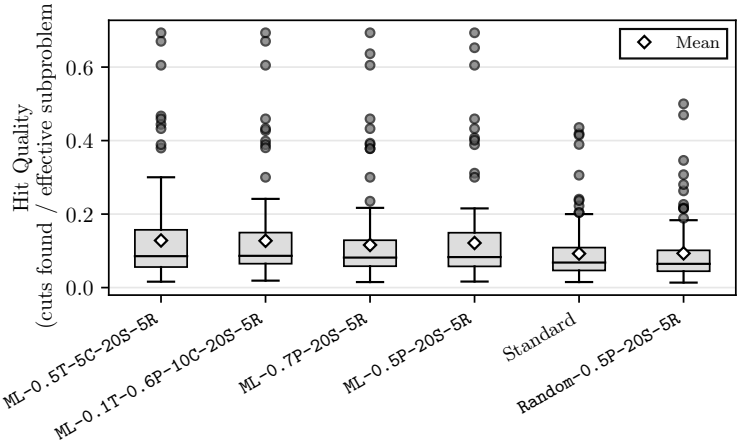

Figure 7: Distribution of hit quality (cuts found / subproblems examined) across methods. Higher values indicate more efficient subproblem selection. Box plots show median (black line), quartiles (box), and mean (white diamond). Outliers are shown as circles.

E Cut Yield Evolution

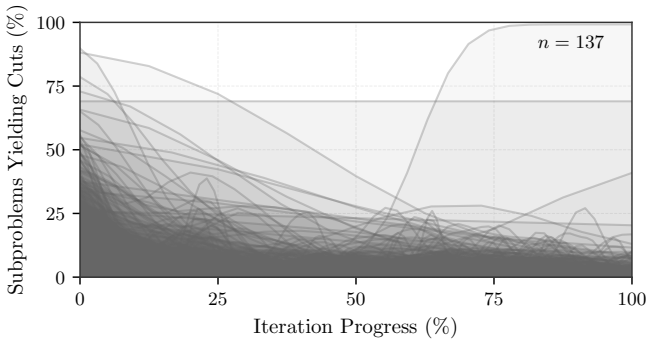

Figure 8: Cut yield evolution across iterations for full enumeration, showing declining patterns from 10–40% early on to 0–10% near convergence.

F Subproblem Performance Analysis

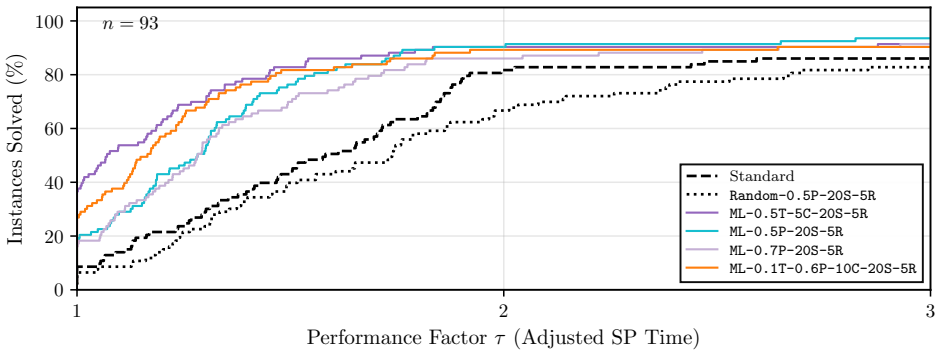

Figure 9: Performance profile based on adjusted subproblem time, which deducts the necessary subproblem time (Oracle baseline that solves all subproblems) from each method's total subproblem time. Shows computational effort spent on unrequired subproblems.

G Detailed Solve Times

Table 7: Solve times (seconds) for all instances. Timeouts: optimality gaps in brackets. Bold: methods faster than Standard (if Standard solved), or methods that solved (if Standard did not). Daggers (†): all methods optimal. Sum/Mean/Sh. Geom. Mean†: over instances solved by all. Wilcoxon test (one-sided): solved use time; unsolved use time-limit+gap%. Significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

	Baselines		ML-Based Methods			
Instance	Standard	Random-0.5P-20S-5R	0.5T-5C-20S-5R	0.5P-20S-5R	0.7P-20S-5R	0.1T-0.6P-10C-20S-5R
abilene†	6.9	8.5	**5.1**	**5.2**	**5.0**	**4.8**
atlanta†	56.2	89.2	**55.6**	68.5	**53.7**	**54.3**
di-yuan†	**735.7**	1743.3	765.6	961.8	814.9	1091.1
france†	248.7	**205.1**	**175.9**	**210.8**	**162.8**	**183.6**
geant†	614.3	1581.4	**337.5**	**428.9**	**359.2**	**370.7**
newyork†	841.4	1134.5	1103.0	983.9	**822.0**	1080.7
nobel-us†	2419.8	**2157.9**	**1418.7**	**1616.4**	**1797.9**	**1478.2**
nobel-germany†	40.9	**36.8**	**36.4**	**38.4**	43.0	**28.9**
pdh†	**103.8**	178.8	214.4	154.1	126.8	142.9
polska†	**6.5**	6.9	7.8	7.3	7.1	6.8
run†	4446.0	**2031.0**	**1612.6**	**1379.3**	**2022.1**	**1644.3**
instance_0001_s2_r26_seed901†	7.4	8.3	**7.3**	**6.7**		**6.1**
instance_0003_s2_r76_seed903†	2754.0	3537.4	4296.5	**2417.3**	2779.1	3716.2
instance_0004_s2_r33_seed904†	36.7	**26.0**	**25.3**	**31.1**	**26.5**	40.2
instance_0006_s2_r51_seed906†	510.0	560.5	554.8	**414.2**	**468.7**	513.5
instance_0007_s2_r43_seed907†	1196.8	1455.6	**1075.8**	**1075.9**	1269.4	**1055.0**
instance_0008_s2_r16_seed908	**(51.9%)**	(54.0%)	(56.9%)	(57.1%)	(53.4%)	(56.3%)
instance_0009_s2_r35_seed909†	12.6	17.7	13.1	**11.2**	**11.7**	**10.9**
instance_0010_s2_r42_seed910†	293.1	295.2	**190.5**	**262.0**	768.1	**254.8**
instance_0011_s3_r64_seed911†	681.8	738.0	**580.0**	**495.3**	**582.8**	**637.0**
instance_0012_s3_r78_seed912†	3871.1	4471.7	4282.4	**3339.7**	5164.7	**3770.7**
instance_0013_s3_r87_seed913	4774.8	(0.6%)	(0.2%)	**3621.0**	**4409.6**	5823.2
instance_0014_s3_r108_seed914	(53.1%)	(53.6%)	**(47.7%)**	**(52.5%)**	**(48.6%)**	(56.7%)
instance_0015_s3_r91_seed915†	959.3	**703.3**	**944.7**	**887.9**	**848.3**	963.0
instance_0017_s3_r84_seed917†	7273.0	**5546.6**	**5692.5**	**7248.8**	**5269.6**	**6855.1**
instance_0018_s3_r62_seed918	**(0.1%)**	(0.1%)	(0.2%)	(0.1%)	(0.2%)	(0.1%)
instance_0019_s3_r122_seed919	(∞%)	**(98.5%)**	**(97.8%)**	**(53.7%)**	**(54.3%)**	**(53.7%)**
instance_0020_s3_r101_seed920†	240.1	268.0	**137.3**	286.8	**237.6**	**235.9**
instance_0021_s4_r71_seed921†	211.6	247.7	**208.0**	242.1	259.0	230.1
instance_0022_s4_r83_seed922†	1365.4	**1065.4**	**879.1**	**984.0**	**1124.7**	**977.2**
instance_0023_s4_r97_seed923	(0.3%)	(0.7%)	**(0.3%)**	(0.3%)	**(0.1%)**	**(0.1%)**
instance_0025_s4_r93_seed925†	1573.4	2580.3	**1204.5**	2210.2	1685.9	1915.1
instance_0026_s4_r111_seed926	(13.4%)	(14.1%)	**(8.0%)**	**(10.6%)**	(17.4%)	**(3.2%)**
instance_0027_s4_r122_seed927	(22.4%)	(∞%)	(88.7%)	(46.0%)	**(18.7%)**	(∞%)
instance_0028_s4_r168_seed928	(91.5%)	**(21.6%)**	**(91.5%)**	**(21.4%)**	**(19.7%)**	(91.6%)
instance_0030_s4_r177_seed930	**(71.0%)**	(∞%)	(72.0%)	(72.0%)	(72.0%)	(72.0%)
instance_0031_s5_r87_seed931	(1.3%)	**10488.0**	**(0.5%)**	(8.6%)	**9117.6**	**10642.5**
instance_0032_s5_r88_seed932	8564.3	(0.1%)	(0.1%)	**7273.1**	**6418.6**	(0.1%)
instance_0033_s5_r104_seed933†	558.5	724.0	658.0	604.7	**501.3**	632.0
instance_0037_s5_r119_seed937	(3.2%)	(3.2%)	**(1.6%)**	**(0.2%)**	**(0.1%)**	**(0.1%)**
instance_0039_s5_r93_seed939†	2660.7	2985.9	**2351.8**	**2488.8**	**2270.7**	**2652.2**
instance_0040_s5_r118_seed940	(8.2%)	(13.3%)	(9.7%)	(8.8%)	(9.8%)	**(5.9%)**
instance_0041_s2_r29_seed941†	**11.7**	15.3	19.2	16.1	15.3	18.0
instance_0042_s2_r57_seed942†	2623.1	3410.1	**2756.5**	**2060.1**	**2666.7**	**2237.8**
instance_0043_s2_r16_seed943†	1.9	**1.9**	1.9	1.9	2.1	2.0
instance_0044_s2_r38_seed944	(0.2%)	(0.2%)	**(0.2%)**	**9959.8**	**(0.2%)**	**9656.2**
instance_0045_s2_r107_seed945	**(6.0%)**	(8.4%)	(6.0%)	(7.8%)	(7.0%)	(9.5%)
instance_0047_s2_r20_seed947†	5.1	**3.5**	7.1	5.8	**5.0**	7.2
instance_0048_s2_r82_seed948	(47.9%)	**(45.8%)**	**(4.1%)**	**(42.6%)**	**(47.5%)**	**(10.3%)**
instance_0049_s2_r42_seed949†	82.8	**59.5**	**32.9**	**44.9**	**40.5**	**36.5**
instance_0050_s2_r113_seed950	**(97.9%)**	(97.9%)	(97.9%)	(97.9%)	(97.9%)	(97.9%)
instance_0051_s3_r67_seed951†	**3568.2**	5811.0	3872.1	4076.4	3691.6	4487.0
instance_0052_s3_r62_seed952†	3068.9	**2422.7**	**2206.0**	**2765.8**	**1937.4**	**2106.3**
instance_0053_s3_r68_seed953†	384.0	871.4	**333.7**	1364.4	497.5	555.2
instance_0054_s3_r124_seed954	(23.1%)	**(15.4%)**	**(15.5%)**	**(19.4%)**	(50.2%)	**(4.2%)**
instance_0055_s3_r87_seed955†	3145.7	3027.1	3188.9	4113.0	3704.7	**2976.5**
instance_0056_s3_r134_seed956	(98.1%)	**(98.1%)**	**(98.0%)**	(98.1%)	(98.1%)	**(98.0%)**
instance_0057_s3_r108_seed957	—	(99.5%)	(99.5%)	(99.5%)	(99.5%)	(99.5%)
instance_0059_s3_r74_seed959†	8959.5	9412.8	**7684.7**	10216.9	9684.8	**7801.5**
instance_0060_s3_r43_seed960†	77.7	**69.7**	**45.9**	**44.5**	**38.3**	**29.8**
instance_0061_s4_r60_seed961†	64.7	**48.2**	67.0	72.8	86.2	**48.7**
instance_0062_s4_r101_seed962	(0.5%)	**(0.2%)**	**9909.2**	**8593.3**	**7744.2**	**(0.1%)**
instance_0063_s4_r133_seed963	(5.3%)	**(0.5%)**	**(0.5%)**	**(1.0%)**	**(1.3%)**	**(0.1%)**
instance_0064_s4_r129_seed964	(22.1%)	(100.0%)	(100.0%)	(100.0%)	**(21.8%)**	(100.0%)
instance_0066_s4_r126_seed966	(∞%)	(∞%)	**(74.5%)**	(∞%)	**(74.8%)**	**(74.5%)**
instance_0071_s5_r115_seed971	(4.7%)	**(2.2%)**	**(3.8%)**	**(1.9%)**	**(0.8%)**	**(2.6%)**
instance_0072_s5_r183_seed972	(3.3%)	**(1.9%)**	**(1.8%)**	**(0.2%)**		
instance_0075_s5_r142_seed975	(0.2%)	(5.4%)	**(0.1%)**	(0.6%)	(1.8%)	(0.8%)
instance_0076_s5_r122_seed976	(3.4%)	(9.5%)	**(0.6%)**	(12.5%)	**(2.2%)**	(10.2%)
instance_0081_s2_r160_seed981	(76.3%)	**(72.2%)**	**(73.0%)**	**(69.6%)**	**(67.3%)**	**(72.9%)**
instance_0083_s2_r76_seed983	(1.7%)	(7.8%)	**(0.1%)**	(16.1%)	(5.8%)	(8.3%)
instance_0084_s2_r38_seed984	**4829.9**	(0.1%)	(0.2%)	5518.6	(0.3%)	8062.0
instance_0085_s2_r87_seed985†	600.6	671.5	728.5	**563.1**	674.7	**557.3**
instance_0086_s2_r66_seed986†	1838.4	2670.9	**1710.8**	**1480.3**	**1511.3**	**1418.4**
instance_0087_s2_r86_seed987	(41.9%)	(45.5%)	**(29.0%)**	**(38.9%)**	**(39.3%)**	**(28.7%)**
instance_0088_s2_r55_seed988†	302.5	377.1	331.0	**200.7**	319.9	**199.9**
instance_0091_s3_r62_seed991†	890.6	960.3	1060.6	909.7	955.9	**860.4**
instance_0092_s3_r28_seed992†	**3.0**	3.6	3.7	3.7	4.4	4.2
instance_0093_s3_r90_seed993†	4849.0	**4714.0**	**2453.8**	**3245.9**	5371.7	5485.4
instance_0094_s3_r59_seed994†	560.7	**438.0**	**427.8**	**462.8**	**285.3**	**364.4**
instance_0095_s3_r55_seed995†	276.4	**240.2**	298.6	1409.2	288.8	288.0
instance_0098_s3_r94_seed998	7637.6	(2.9%)	9625.8	**6300.3**	8034.4	9498.9
instance_0099_s3_r102_seed999	**(31.6%)**	(33.0%)	(36.4%)	(36.4%)	(36.4%)	(36.4%)
instance_0104_s4_r152_seed1004	(28.5%)	(31.6%)	(34.7%)	(34.4%)	(30.3%)	**(28.5%)**
instance_0112_s5_r78_seed1012†	653.4	730.7	**481.4**	**627.8**	**600.1**	**451.3**
instance_0113_s5_r131_seed1013	(1.7%)	(5.6%)	**(1.0%)**	**(1.2%)**	(1.9%)	**(1.4%)**
instance_0115_s5_r150_seed1015	8807.6	10332.3	**8405.6**	**8528.3**	9563.1	(0.1%)
instance_0121_s2_r27_seed1021†	51.1	**43.6**	63.3	**32.8**	58.2	**49.9**
instance_0122_s2_r54_seed1022†	181.2	**164.0**	**143.3**	**152.7**	**151.7**	**145.2**
instance_0123_s2_r90_seed1023†	1021.8	1282.3	1049.3	1045.7	1092.8	**941.6**
instance_0124_s2_r27_seed1024†	**19.0**	32.9	21.4	20.0	26.3	19.2
instance_0125_s2_r100_seed1025†	1113.2	1238.1	**777.3**	**649.2**	**771.3**	**787.8**
instance_0127_s2_r36_seed1027†	35.3	37.8	**26.6**	41.8	**32.2**	**30.3**
instance_0128_s2_r27_seed1028†	13.9	**10.5**	15.2	15.1	16.2	**12.0**
instance_0129_s2_r65_seed1029†	786.4	1806.7	920.1	**705.2**	939.5	**783.7**
instance_0130_s2_r85_seed1030	(1.5%)	(80.1%)	**(0.5%)**	**(0.6%)**	(1.8%)	**(0.8%)**
instance_0131_s3_r96_seed1031	(0.1%)	(0.1%)	(0.1%)	**(0.1%)**	(0.1%)	(0.1%)
instance_0132_s3_r56_seed1032†	837.0	3010.3	**434.5**	**239.2**	**252.8**	1239.2
instance_0133_s3_r76_seed1033†	**35.4**	75.4	42.6	43.3	57.7	59.7
instance_0134_s3_r62_seed1034†	295.3	306.4	366.8	316.0	296.6	**265.8**
instance_0135_s3_r55_seed1035†	313.7	**257.9**	342.1	**308.1**	**267.2**	**233.6**
instance_0136_s3_r93_seed1036†	1248.0	1343.3	1333.6	**1034.4**	**1042.9**	**788.9**
instance_0137_s3_r97_seed1037†	6361.7	**6014.9**	**5440.0**	6191.0	7409.7	**6213.3**
instance_0139_s3_r144_seed1039	(34.8%)	**(17.1%)**	(49.3%)	(56.9%)	**(9.5%)**	**(30.2%)**
instance_0141_s4_r78_seed1041†	**7.0**	7.1	7.2	8.8	9.2	7.0
instance_0142_s4_r111_seed1042†	1814.3	2486.6	**1697.2**	**1240.8**	**1277.3**	**1642.4**
instance_0146_s4_r53_seed1046	3646.7	(1.2%)	**2080.7**	4237.9	9675.6	(0.5%)
instance_0151_s5_r76_seed1051†	1890.0	**1695.1**	**1011.5**	**1043.8**	**1448.8**	**1502.8**
instance_0153_s5_r123_seed1053	(11.0%)	(14.6%)	(12.1%)	(26.7%)	(33.6%)	**(9.2%)**
instance_0155_s5_r52_seed1055	—	—	(98.3%)	(98.3%)	(98.3%)	(98.3%)
instance_0160_s5_r139_seed1060	(9.7%)	**(4.6%)**	**(3.8%)**	**(2.9%)**	**(3.6%)**	**(3.4%)**
instance_0161_s2_r28_seed1061†	21.5	25.4	**20.4**	22.6	24.1	27.0
instance_0162_s2_r68_seed1062†	858.8	1147.3	**730.2**	876.9	979.2	880.5
instance_0164_s2_r71_seed1064†	160.9	169.2	**114.8**	**130.0**	**138.4**	**142.9**
instance_0166_s2_r43_seed1066†	**62.4**	80.0	71.8	66.4	81.1	75.6
instance_0167_s2_r50_seed1067†	522.7	**493.3**	**332.6**	**346.4**	528.9	**376.0**
instance_0169_s2_r85_seed1069†	3554.6	**3095.8**	4098.5	5726.2	7746.7	5621.0
instance_0169_s2_r47_seed1069†	230.0	262.8	245.0	**179.0**	**151.2**	**211.6**
instance_0170_s2_r64_seed1070	(0.4%)	(1.1%)	**8721.0**	**9421.3**	**10716.7**	**8293.7**
instance_0171_s2_r64_seed1071†	4766.6	9192.2	5179.5	4797.5	6276.2	**3826.1**
instance_0173_s3_r32_seed1073†	11.0	**10.9**	12.9	**10.4**	**10.9**	**9.1**
instance_0174_s3_r94_seed1074	8122.8	9465.9	**7483.8**	(0.1%)	**6583.5**	9118.3
instance_0175_s3_r70_seed1075	**(3.1%)**	(6.3%)	(4.3%)	(5.4%)	(4.3%)	(4.6%)
instance_0177_s3_r155_seed1077	(30.2%)	(85.2%)	**(25.6%)**	(85.4%)	(85.4%)	(85.2%)
instance_0179_s3_r100_seed1079	(4.4%)	**(1.5%)**	**(2.5%)**	**(3.3%)**	**(1.8%)**	**(2.1%)**
instance_0180_s3_r42_seed1080†	**17.1**	19.1	21.0	25.6	28.9	28.7
instance_0181_s4_r77_seed1081†	2092.8	**1204.1**	**1220.4**	**1536.9**	**2085.8**	**1474.3**
instance_0182_s4_r97_seed1082	(1.7%)	(32.9%)	**(0.9%)**	**(1.5%)**	**(0.3%)**	**(0.1%)**
instance_0183_s4_r113_seed1083	(1.0%)	(1.3%)	**4369.1**	**9448.8**	**(0.8%)**	**9842.2**
instance_0184_s4_r129_seed1084	(65.4%)	**(66.4%)**	**(55.7%)**	**(62.9%)**	**(64.8%)**	**(61.8%)**
instance_0191_s5_r94_seed1091†	57.6	**45.8**	**30.7**	**56.2**	**34.3**	**34.0**
instance_0192_s5_r150_seed1092	(99.0%)	—	**(29.5%)**	**(29.4%)**	**(29.4%)**	**(29.5%)**
instance_0193_s5_r138_seed1093	(∞%)	(∞%)	**(89.1%)**	**(88.7%)**	**(89.3%)**	**(89.1%)**
instance_0194_s5_r108_seed1094†	6624.3	**6288.6**	**4406.8**	**6269.1**	**5006.6**	**5079.5**
Solved	86	82	86	89	88	87
Solved by all†	79	79	79	79	79	79
Sum†	100941.2	112118.3	86715.2	96097.3	100536.4	92977.6
Mean†	1277.7	1419.2	1097.7	1216.4	1272.6	1176.9
Sh. Geom. Mean†	300.5	334.8	270.8	285.2	288.3	275.3
Wilcoxon p		0.9996	0.0130	0.0136	0.2080	0.0081
Significance		n.s.	*	*	n.s.	**

H Detailed Convergence Quality

Table 8: Primal-dual integral values (billions, $\int_0^T (\text{primal}(t) - \text{dual}(t))dt$, cost·seconds). Lower is better. Bold: better than Standard. Wilcoxon test (one-sided). Significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

	Baselines		ML-Based Methods			
Instance	Standard	Random-0.5P-20S-5R	0.5T-5C-20S-5R	0.5P-20S-5R	0.7P-20S-5R	0.1T-0.6P-10C-20S-5R
abilene	50.0	**40.0**	**50.0**	**50.0**	**50.0**	**40.0**
atlanta	257	505	317	269	377	317
di-yuan	10.1	10.3	10.1	**10.1**	20.1	20.1
france	1620	**1260**	**1050**	**1070**	**1120**	**1060**
geant	2420	5210	3370	4280	3590	3700
newyork	61.2	61.6	61.7	61.4	61.3	71.6
nobel-eu	1151	1251	1191	1161	1161	1171
nobel-germany	40.0	**40.0**	40.0	40.0	40.0	**40.0**
pdh	10.3	20.3	10.4	10.3	20.2	10.3
polska	10.0	20.0	20.0	20.0	20.0	20.0
sun	110	150	200	230	230	220
instance_0001_s2_e26_seed901	20.0	**20.0**	**20.0**	**20.0**	**20.0**	**20.0**
instance_0003_s2_e76_seed903	1040	1290	**670**	**660**	**660**	**670**
instance_0004_s2_e33_seed904	90.0	**50.0**	**40.0**	**40.0**	**40.0**	**40.0**
instance_0006_s2_e51_seed906	340	**300**	**270**	**280**	**290**	**270**
instance_0007_s2_e43_seed907	2090	5260	**1940**	2270	**1770**	2680
instance_0008_s2_e186_seed908	15540	**15100**	**15160**	**15130**	**15060**	**15200**
instance_0009_s2_e35_seed909	30.0	30.0	30.0	30.0	30.0	30.0
instance_0010_s2_e62_seed910	130	**120**	**120**	**120**	**120**	**120**
instance_0011_s3_e64_seed911	380	460	410	430	450	420
instance_0012_s3_e78_seed912	1820	**1650**	**1320**	**1370**	**1410**	**1400**
instance_0013_s3_e87_seed913	610	**600**	620	610	620	610
instance_0014_s3_e108_seed914	38702	**35792**	41271	47371	65469	56281
instance_0015_s3_e91_seed915	640	670	**570**	**630**	650	**640**
instance_0017_s3_e84_seed917	7380	10430	**7200**	**6890**	7820	**6980**
instance_0018_s3_e62_seed918	290	**280**	290	290	290	290
instance_0020_s3_e101_seed920	150	190	160	160	160	160
instance_0021_s4_e71_seed921	1650	**1410**	**1500**	1670	1880	1780
instance_0022_s4_e85_seed922	4040	5120	**3400**	**3880**	**3910**	**3740**
instance_0023_s4_e97_seed923	2280	**2181**	**1580**	**1620**	**1600**	**1560**
instance_0025_s4_e93_seed925	3820	4760	4940	5270	4230	5220
instance_0026_s4_e111_seed926	13522	14082	**11082**	14092	**12512**	13601
instance_0027_s4_e122_seed927	76867		90699	96654	98861	
instance_0028_s4_e88_seed928	111379	**103689**	120349	**111179**	115909	**107999**
instance_0030_s4_e177_seed930	107090		**104530**	**104740**	**104460**	**104400**
instance_0031_s5_e87_seed931	13372	**9722**	**6992**	**8102**	**8232**	**7712**
instance_0032_s5_e88_seed932	526	**486**	527	537	526	527
instance_0033_s5_e104_seed933	869	**819**	**650**	**829**	**849**	**769**
instance_0037_s5_e119_seed937	2406	2590	**2395**	2443	2533	**2361**
instance_0039_s5_e93_seed939	4369	**3889**	**2969**	**2699**	**2869**	**3109**
instance_0040_s5_e118_seed940	3255	3263	3404	3284	3275	3274
instance_0041_s2_e29_seed941	20.0	**10.0**	20.0	20.0	20.0	20.0
instance_0042_s2_e57_seed942	2100	2840	**1630**	**1760**	2200	**1610**
instance_0043_s2_e16_seed943	10.00	10.00	10.00	10.00	10.00	10.00
instance_0044_s2_e38_seed944	310	**290**	**270**	**270**	**280**	**280**
instance_0045_s2_e107_seed945	10330	12750	**9560**	11960	12540	13060
instance_0047_s2_e20_seed947	10.0	**10.0**	10.0	10.0	**10.0**	**10.0**
instance_0048_s2_e82_seed948	861	881	941	941	971	941
instance_0049_s2_e42_seed949	110	**80.0**	**70.0**	**70.0**	**70.0**	**70.0**
instance_0050_s2_e113_seed950	107997	**55218**	**107997**	**107997**	**107997**	**107997**
instance_0051_s3_e67_seed951	1431	**1381**	**1171**	**1141**	**1131**	**1151**
instance_0052_s3_e62_seed952	760	**650**	**680**	**670**	**670**	**670**
instance_0053_s3_e68_seed953	20.0	30.0	30.0	30.0	30.0	30.0
instance_0054_s3_e124_seed954	10790	19590	**8900**	**7830**	**7980**	**7920**
instance_0055_s3_e87_seed955	18060	29099	**13600**	**13700**	**11720**	**11310**
instance_0056_s3_e134_seed956	130329	**112589**	**108679**	**112719**	**108029**	**107999**
instance_0059_s3_e74_seed959	14260	**11970**	15500	**14140**	**14130**	14750
instance_0060_s3_e43_seed960	32.3	**31.9**	**31.9**	32.4	32.8	32.4
instance_0061_s4_e80_seed961	350	370	420	410	**330**	**280**
instance_0062_s4_e101_seed962	41658	**27029**	**17409**	**32378**	**22779**	**18429**
instance_0063_s4_e133_seed963	58637	57388	**52848**	**45438**	**45868**	**44888**
instance_0064_s4_e129_seed964	109379	114018	116088	112588	**94289**	110188
instance_0071_s5_e113_seed971	35099	47168	**32079**	40368	**25529**	45598
instance_0072_s5_e193_seed972	58790	**51821**	**48922**	**49012**		
instance_0075_s5_e142_seed975	62244	**50186**	**28537**	**40166**	**38066**	**35747**
instance_0076_s3_e22_seed976	14561	21101	**13110**	**12832**	**13272**	**12702**
instance_0081_s2_e160_seed981	46350	**42150**	**39640**	**43530**	**44010**	**39440**
instance_0083_s2_e76_seed983	22340	94267	31170	38010	35550	39790
instance_0084_s2_e38_seed984	50.0	**40.0**	50.0	**40.0**	**40.0**	**40.0**
instance_0085_s2_e83_seed985	150	150	160	160	160	160
instance_0086_s2_e95_seed986	4060	11760	4770	5710	6500	4730
instance_0087_s2_e86_seed987	73647	112043	**70107**	**62328**	**67328**	**72647**
instance_0088_s2_e55_seed988	360	**230**	**250**	**270**	**290**	**280**
instance_0091_s3_e62_seed991	190	200	210	210	210	210
instance_0092_s3_e26_seed992	20.0	**10.00**	20.0	20.0	20.0	20.0
instance_0093_s3_e90_seed993	7369	**6119**	**3040**	**4889**	**6769**	**4529**
instance_0094_s3_e69_seed994	390	**360**	**280**	**290**	**280**	**280**
instance_0095_s3_e55_seed995	40.0	**40.0**	50.0	50.0	**40.0**	50.0
instance_0098_s3_e94_seed998	4681	**3843**	4811	**4541**	**4591**	**4561**
instance_0099_s3_e102_seed999	118161	**114103**	**67746**	**69756**	**71684**	**67687**
instance_0104_s4_e152_seed1004	4746	5317	5167	5128	5148	5158
instance_0112_s5_e78_seed1012	330	**250**	**270**	**310**	**330**	**300**
instance_0113_s5_e131_seed1013	26668	30337	**16443**	**17083**	**17723**	**17062**
instance_0115_s5_e150_seed1015	2037	**1786**	**1802**	**1798**	**1799**	**1790**
instance_0121_s2_e27_seed1021	80.0	**70.0**	80.0	80.0	80.0	80.0
instance_0122_s2_e54_seed1022	460	620	550	560	540	560
instance_0123_s2_e90_seed1023	540	1150	760	900	840	770
instance_0124_s2_e27_seed1024	100.0	140	100.0	100.0	100.0	**90.0**
instance_0125_s2_e100_seed1025	4940	7770	**2690**	**2700**	**2580**	**2490**
instance_0127_s2_e34_seed1027	100.0	**90.0**	**90.0**	101	120	110
instance_0128_s2_e27_seed1028	20.0	**20.0**	**20.0**	20.0	**20.0**	**20.0**
instance_0129_s2_e85_seed1029	2628	**1269**	**1559**	**1429**	**1509**	**1499**
instance_0130_s2_e95_seed1030	74930	110020	**49200**	**72410**	91980	**52010**
instance_0131_s3_e96_seed1031	2260	**1880**	**2150**	**2120**	**2230**	**2150**
instance_0132_s3_e56_seed1032	160	**100**	110	110	110	110
instance_0133_s3_e76_seed1033	160	480	370	160	160	230
instance_0134_s3_e82_seed1034	80.0	**70.0**	90.1	90.0	90.0	90.0
instance_0135_s3_e85_seed1035	290	**220**	**200**	**220**	**220**	**220**
instance_0136_s3_e63_seed1036	486	**454**	**453**	**453**	**464**	**463**
instance_0137_s3_e97_seed1037	8521	10598	**7650**	**7732**	**7900**	**7680**
instance_0139_s3_e144_seed1039	18190	28090	29700	31370	31520	30350
instance_0141_s4_e76_seed1041	20.0	20.0	20.0	20.0	20.0	20.0
instance_0142_s4_e111_seed1042	1530	**1480**	**900**	**970**	**980**	**930**
instance_0146_s4_e53_seed1046	110	**100**	**60.0**	**60.0**	**60.0**	**60.0**
instance_0151_s5_e78_seed1051	3410	3690	3630	4220	5089	5389
instance_0153_s5_e123_seed1053	35990	46660	36640	44080	**35630**	46210
instance_0160_s5_e139_seed1060	21880	22750	**14100**	**14111**	**14401**	**13851**
instance_0161_s2_e28_seed1061	20.1	20.1	20.1	**20.1**	20.1	20.1
instance_0162_s2_e56_seed1062	170	180	**170**	**170**	**170**	**170**
instance_0164_s2_e71_seed1064	330	330	**310**	**260**	**250**	**260**
instance_0166_s2_e43_seed1066	66.2	71.8	77.7	76.7	**57.7**	70.4
instance_0167_s2_e50_seed1067	230	230	**180**	**180**	**180**	**180**
instance_0168_s2_e65_seed1068	21386	**12472**	**7336**	**9234**	**12273**	**8555**
instance_0169_s2_e67_seed1069	108	**106**	**103**	108	108	**103**
instance_0170_s2_e64_seed1070	4160	4510	**4070**	**4090**	**4100**	**4080**
instance_0171_s3_e74_seed1071	18840	**9854**	**3360**	**3372**	**14971**	**3380**
instance_0173_s3_e32_seed1073	20.0	**20.0**	50.0	40.0	50.0	40.0
instance_0174_s3_e94_seed1074	9580	11150	10680	**8031**	**8890**	**7070**
instance_0175_s3_e70_seed1075	230	**220**	240	240	240	230
instance_0177_s3_e108_seed1077	45690	52841	52200	**36801**	**38121**	**33871**
instance_0179_s3_e100_seed1079	8942	**7389**	**6299**	**6214**	**6264**	**6248**
instance_0180_s3_e42_seed1080	150	**140**	150	150	190	180
instance_0181_s4_e77_seed1081	6520	8990	**3200**	**3220**	**4940**	**4990**
instance_0182_s4_e97_seed1082	14734	**12227**	**7152**	**8243**	**8643**	**8163**
instance_0183_s4_e113_seed1083	14230	**13320**	**13420**	**11920**	**12480**	15869
instance_0184_s4_e129_seed1084	58872	**48052**	**24433**	**30634**	**31915**	**29114**
instance_0191_s5_e94_seed1091	340	**230**	**240**	**310**	**300**	**280**
instance_0192_s5_e190_seed1092	97242	—	**63757**	**63677**	**63707**	**63637**
instance_0194_s5_e108_seed1094	1225	**1114**	**993**	**1003**	**1014**	**1004**
Sum	1888056	1705096	1605588	1682330	1639649	1490068
Mean	14636	13533	12446	13041	12805	11733
Sh. Geo. Mean	1348.4	1273.8	1198.8	1235.0	1232.7	1151.7
Wilcoxon p		0.6052	0.0004	0.0006	0.0004	0.0006
Significance		n.s.	***	***	***	***