A Approximation Bounds for Graph Inspection Planning

In this section, we leverage the connection between GIP and Group-ST to establish approximation upper and lower bounds for GIP. For upper bound, Mizutani et al. [41] Thm. 4 stated a straight forward $O(k)$-approximation algorithm. To the best of our knowledge, approximation lower bounds have not previously been established for GIP.[5] We begin by restating the following approximation bounds for Group-ST. Below, k denotes the number of groups, and $n = |V|$.

Theorem 1 (Lower approximation bounds for Group-ST [28]). *Group-ST is hard*[6] *to approximate by a factor of* $\Omega(\log^{2-\epsilon} k)$*, for any constant* $\epsilon > 0$*. For Theorem 3 we assume the same computational model.*

Theorem 2 (Upper approximation bounds for Group-ST [8, 25]). *There exists a randomized polynomial time* $O(\log n \cdot \log^2 k)$*-approximation algorithm for Group-ST.*

We recall that Group-ST was formally defined in Def. 4. Although the problem is stated in terms of finding a tree, an equivalent formulation may allow any connected subgraph, since an optimal solution can always be taken to be a tree.

To derive analogous approximation bounds for GIP, we rely on the following two observations:

O1 Any feasible solution to GIP induces a valid solution to Group-ST, since a GIP tour is, by definition, a connected subgraph that covers all groups.

O2 Any feasible solution to Group-ST can be transformed into a feasible solution to GIP by traversing the solution tree in a depth-first manner and returning along each edge. This transformation yields a GIP tour whose cost is at most twice the cost of the original Group-ST solution.

The following theorem provides a lower approximation bound for GIP.

Theorem 3 (Lower approximation bound for GIP). *GIP is hard*[6] *to approximate within a factor of* $\Omega(\log^{2-\epsilon} k)$ *for all constant* $\epsilon > 0$*.*

Proof. Assume by contradiction we are given an α-approximation oracle $\mathcal{O}$ for GIP, such that $\alpha < \Omega(\log^{2-\epsilon} k)$. Given a Group-ST instance, we consider the corresponding GIP and apply $\mathcal{O}$ to obtain a tour $\tau_{\mathcal{O}}$ that α-approximates τ^*, the optimal GIP tour.

Let T^* be an optimal Group-ST solution. By traversing T^* back and forth and following Observation **O2**, we can construct a closed tour for GIP, T_p^*, such that $|T_p^*| \leq 2 \cdot |T^*|$. As T_p^* is a covering-tour for GIP and τ^* is the optimal tour we have that

$$|\tau^*| \leq |T_p^*|.$$

[5] The discussion below considers the *undirected* version of GIP.

[6] Halperin and Krauthgamer [28] rely on the assumption that NP has no quasi-polynomial Las Vegas algorithms. This is slightly stronger assumption than P $\neq$ NP but widely accepted as plausible.

Using the assumption that $\mathcal{O}$ is an α-approximation algorithm, we have that

$$|\tau_{\mathcal{O}}| \leq \alpha \cdot |\tau^*| \leq 2\alpha \cdot |T^*|.$$

This solution $\tau_{\mathcal{O}}$ forms a covering and connected edge subset, and hence is a solution to the Group-ST problem by Observation **O1** (and may be tightened to a tree by the removal of cycle-closing edge). Hence we found a solution $\tau_{\mathcal{O}}$ to the Group-ST problem that is $2 \cdot \alpha < \Omega(\log^{2-\epsilon} k)$ approximating the optimal Group-ST solution, in contradiction to Thm 1. Hence no such approximation algorithm exists for GIP under the computational model of Halperin and Krauthgamer [28] □

The following theorem, describing upper approximation bounds for GIP, improves the previous $O(k)$ approximation result provided by Mizutani et al. [41].

Theorem 4 (Upper approximation bounds for GIP). *There exists an efficient $O(\log n \cdot \log^2 k)$-approximation algorithm for GIP.*

Proof. Given a GIP instance with optimal solution T^*, we apply the approximation algorithm $A_{\text{Group-ST}}$ provided by Thm. 2, yielding a solution T_A to the Group-ST problem for which $|T_A| \leq O(\log n \cdot \log^2 k) \cdot |T^*|$. We build a tour τ such that $|\tau| \leq 2 \cdot |T_A|$ traveling T_A back and forth by Observation **O2**. However, as the optimal GIP solution τ^* is also a Group-ST solution by Observation **O1**, $|T^*| \leq |\tau^*|$, it follows that

$$|\tau| \leq 2 \cdot |T_A| \leq 2 \cdot O(\log n \cdot \log^2 k) \cdot |T^*| \leq O(\log n \cdot \log^2 k) \cdot |\tau^*|.$$

Thus, A_{gst} is an efficient $O(\log n \log^2 k)$-approximation algorithm for GIP. □

B Partial-coverage GIP

In this section we briefly describe how to extend the baseline MILP formulation for GIP (1a)–(1e) to support *partial* group coverage. This extension is orthogonal to the choice of subtour-elimination constraints, as it can be plugged into any of the discussed formulations. It is based on ideas proposed by Mizutani et al. [41], and we include it here for completeness.

Given a desired number of POIs to be covered, $q \leq k$, we introduce additional binary decision variables $z_i \in \{0, 1\}$ for all $i \in K$, indicating whether POI i is selected for inspection. The requirement to cover at least q POIs is enforced by the constraint

$$\sum_{i \in K} z_i \geq q. \tag{5}$$

The group-coverage constraint (1c) is then modified so that coverage is enforced only for selected POIs:

$$\sum_{v \in S_i} \sum_{u \in N^-(v)} x_{uv} \geq z_i, \quad \forall i \in K. \tag{6}$$

This GIP variant is particularly interesting in the aspect of model tightness tradeoffs. On one hand, partial coverage relaxes the full GIP problem constraints, easing the discovery of feasible solutions. On the other hand, the additional combinatorial explosion of having to chose q out of the k POIs significantly increases the solution space. When viewed through the lens of MILP, it looses model integrality gap, as fractional solutions may spread partial coverage across many groups via the z_i variables, weakening coverage constraints in the LP relaxation. Eventually, this leads to a problem where finding the *optimal* partial-coverage solution may be even more challenging than finding the optimal full-coverage solution. We leave empirical evaluation of this variant to future work.

C Additional Details on the Primal Heuristic for GIP

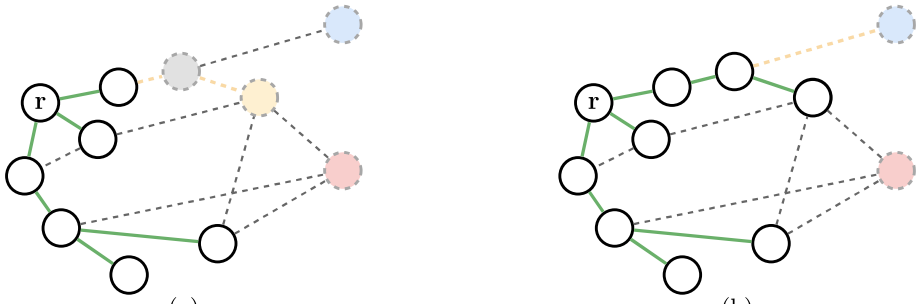
We provide additional details regarding our GIP-specific primal heuristic. The heuristic consists of the following three phases (visualized in Fig. 6 and 7 and detailed shortly). At a high level, we first modify edge costs based on the current LP relaxation, enabling the heuristic to leverage guidance from fractional solutions encountered during the search. Then, inspired by the Group-ST problem [33, 45], we greedily construct a tree that connects the root to at least one vertex from each POI group. Finally, we transform this group-covering tree into a feasible GIP tour by adding a small set of connecting edges.

We mention that a common approach to generate incumbent solutions in MILP problems is to round fractional solutions to enforce integrality. The LP-based cost modification strategy [33] generalizes this idea by using fractional values to guide the construction of integer solutions indirectly: instead of rounding variables explicitly, edge costs are discounted according to their fractional values in the current relaxation. This biases the heuristic toward structures already supported by the LP relaxation, improving solution quality and alignment with the solver's ongoing search. From this perspective, our heuristic can be viewed as a problem-aware LP-rounding step embedded within the BnC process.

We now provide a detailed description for each of the three phases.

Phase 1-LP-based cost-discounted graph. Let $\{x_e\}_{e\in E}$ denote a fractional solution obtained from an LP relaxation solved during the BnC process. We construct a new weighted graph $G_n = (V, E, c_n)$ over the same vertices and edges as the original graph $G = (V, E, c)$, where each edge $e \in E$ is assigned a modified cost $c_n(e) := c(e) \cdot (1 - x_e)$. This modification reflects the intuition that edges partially selected by the LP relaxation are more likely to belong to high-quality integer solutions. By lowering their cost, the heuristic is encouraged to follow the structure suggested by the relaxation, while still allowing alternative edges when necessary.

Phase 2-Group-covering tree. In the second phase, illustrated in Fig. 6, we construct a group-covering tree T rooted at r in the discounted graph G_n that covers all groups, i.e., $\forall S_i \in \mathcal{S},\ T \cap S_i \neq \emptyset$. To this end, we maintain a set $\mathcal{U}$ of uncovered groups, initialized as $\mathcal{U} := \mathcal{S}$, and incrementally expand the tree, which initially consists of the root vertex r alone.



(a) (b)

Fig. 6: Illustration of Phase 2 of the primal heuristic for GIP: A group covering tree (green) is built from the root r towards the closest vertex among all the vertices covering an uncovered group (colored vertices); (a) The yellow vertex is chosen as the closest vertex to the tree and a path from nearest tree vertex (orange edges) is found. (b) After the path was added, the tree continues to grow greedily until all groups are covered.

Before growing the tree, we compute all-pairs shortest paths in G_n, obtaining for every pair of vertices $u, v \in V$ a shortest (weighted) path $\pi^*(u, v)$ and its cost $d^*(u, v)$ with respect to the modified edge costs c_n. This forms the computational bottleneck of the heuristic, as it is recalculated whenever a new LP-relaxed solution is found and graph weights are updated. At each iteration, we define the set of candidate vertices

$$V_{\text{covered}} := \{v \in V \setminus T \mid \exists S_i \in \mathcal{U} \text{ such that } v \in S_i\},$$

consisting of vertices that belong to at least one uncovered group. Among these candidates, we greedily select the vertex $v' \in V_{\text{covered}}$ that is closest to the current tree, i.e.,

$$v' := \arg \min_{v \in V_{\text{covered}}} \min_{w \in T} d^*(w, v).$$

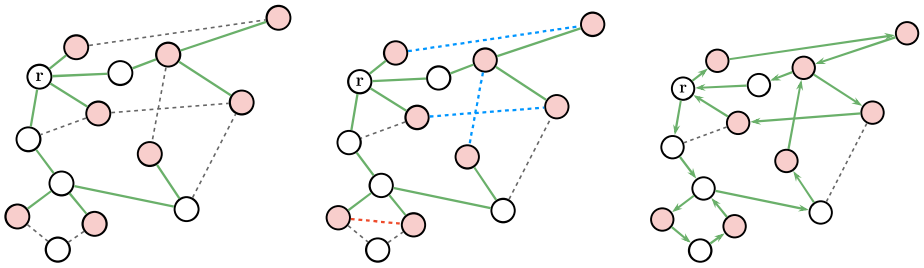
The selected vertex v' is added to the tree by inserting the shortest path $\pi^*(u, v')$, where

$$u := \arg \min_{w \in T} d^*(w, v').$$

This process is repeated until $\mathcal{U} = \emptyset$, yielding a tree that intersects all groups in $\mathcal{S}$.

Phase 3-Tree-to-tour conversion. In the final phase (illustrated in Fig. 7), we transform the group-covering tree T into a feasible GIP tour by adding a minimum-cost set of edges so as to make T an Eulerian subgraph.

Concretely, let O denote the set of vertices in T with odd degree. A standard approach to making the graph Eulerian is to compute a minimum-weight perfect matching M on the complete graph induced by O, where edge weights correspond to shortest-path distances in G_n, and to add the matched shortest paths to T. This procedure follows the matching phase of Christofides' algorithm [12] and can be implemented using Blossom-based algorithms for minimum-weight perfect matching [14, 18]. The matching adds exactly one incident edge to each odd-degree vertex, resulting in the augmented graph $T + M$ being Eulerian. In our



(a) (b) (c)

Fig. 7: Illustration of Phase 3 of the primal heuristic for GIP: (a) The phase starts with a tree (green) connecting the root to a covering set of vertices (red). (b) A matching is found between odd-degree tree vertices, resulting in an Eulerian graph. The matching may include actual graph edges (blue) or virtual edges representing shortest paths (red) (c) An Eulerian tour is imposed on the graph, orienting the selected edges to form a valid GIP tour.

context, using an optimal matching minimizes the additional cost required to make T traversable and typically yields higher-quality tours.

We refer to the heuristic described so far as *covering tree + minimum matching*. For scalability and speed, we consider a lightweight alternative based on *greedy* nearest-neighbor matching on O, in which the two closest unmatched vertices are repeatedly paired and removed [32, 46]. We refer to this heuristic as *covering tree + greedy matching*.

In practice, the greedy matching variant substantially reduces heuristic computation time and often produces solutions close to those obtained with optimal matching at a fraction of the computational cost. The trade-off between solution quality and runtime for these two variants is evaluated in Sec. D.3: while minimum-weight perfect matching gives slightly improved heuristics, the greedy matching approach enables better scaling to very large GIP instances. After augmenting T using the chosen matching strategy, we extract an Eulerian tour and shortcut repeated vertices to obtain the final GIP solution.

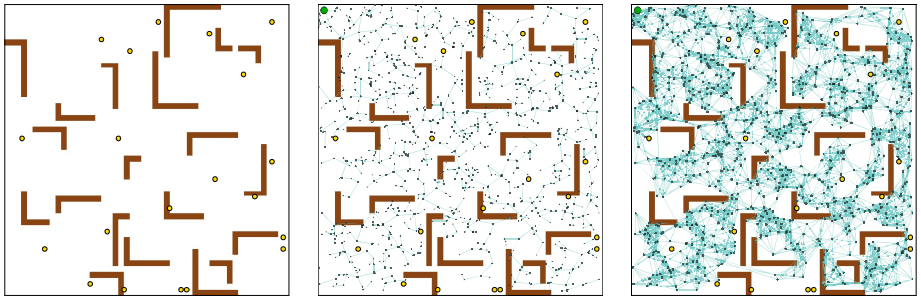
Throughout the experiments conducted in Sec. 6, the *covering tree + greedy matching* was the applied heuristic.

D Additional Experimental Results

In this section we complement the experimental discussion in Sec. 6 with the following: (i) We elaborate on the inspection-planning simulation, used in the `Controlled` instances. (ii) We complement the scale-range analysis with small scale experiments. (iii) We extend our ablation study regarding BnC algorithmic building blocks, with a primal-heuristic discussion.

D.1 Inspection-Planning Simulator

While real-world datasets capture practical inspection scenarios, simulation enables independent control over key problem parameters such as graph size and



(a) (b) (c)

Fig. 8: Inspection planning simulation. (a) The simulation environment is a planar maze of given size, with randomly allocated L-shaped obstacles (in brown). $k = 50$ POIs are scattered in the free space at random (in yellow). (b) RRT graph with $n = 1,000$ vertices built to span the robot's configuration space with the robot's starting configuration located in the top-left corner and marked in green. (c) Edges are locally added to create an RRG graph which is used as the inspection-planning graph.

number of POIs, allowing us to isolate and study the scaling behavior of different MILP formulations. To this end, we developed a configurable inspection-planning simulator that generates problem instances on a point robot model, preserving the structural characteristics of roadmap-based inspection graphs, while enabling precise control over instance scale and coverage complexity.

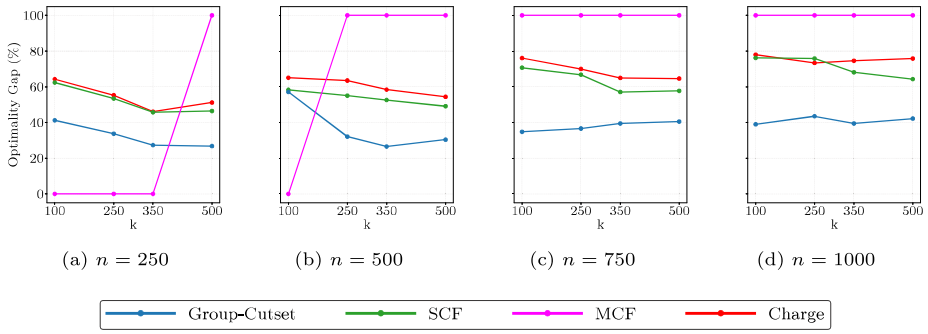
The point robot operates in a three-dimensional configuration space $\mathcal{C} = \mathbb{R}^2 \times \mathrm{SO}(2)$, representing planar position and orientation. It is equipped with a camera characterized by a fixed field-of-view (FOV) angle and a maximal inspection range. Each scenario is generated by randomly placing two-dimensional obstacles, a set of POIs $\mathcal{P}$, and a designated starting configuration r in a bounded planar workspace.

An RRG roadmap is then constructed in the robot's configuration space, respecting obstacle constraints using standard collision-checking procedures [30]. Vertices correspond to collision-free robot configurations, and edges represent dynamically feasible local motions with associated traversal costs. Based on the camera's FOV and inspection range, each vertex $v \in V$ is assigned a visibility set $\chi(v) \subseteq \mathcal{P}$, consisting of the POIs that can be inspected when the robot is positioned at v. This process yields a graph inspection planning instance as defined in Def. 1. In the experiments below, we systematically vary the roadmap size $n = |V|$ and the number of POIs $k = |\mathcal{P}|$ to analyze solver behavior across a wide range of problem scales.

The simulation environment is a planar maze populated with randomly placed L-shaped obstacles, inducing narrow passages and nontrivial connectivity patterns. POIs are uniformly distributed in the free space, and inspection plans must navigate the roadmap to ensure coverage while maintaining global connectivity. The process of generating a representative simulated instance is demonstrated in Fig. 8.

D.2 Evaluation of Small Instances

Recall that in Sec. 6.2 we studied the optimality gap on large-scale simulated instances (Fig. 4). We now examine the opposite end of the scale spectrum, where tighter formulations become feasible and exact convergence is sometimes attainable in the controlled scenarios. Fig. 9 reports the final optimality gap after a time limit of 500 s. Solver were applied in identical configuration to Sec. 6.



(a) $n = 250$ (b) $n = 500$ (c) $n = 750$ (d) $n = 1000$

Fig. 9: Optimality gap on small-scale simulated instances after 500 s for varying graph sizes (n) and number of POIs (k).

For instances in which the MCF solver fits within available memory, it consistently solves the problem to optimality. This behavior is expected, as the MCF solver provides a tight and complete encoding of connectivity and group-coverage constraints, making it the preferred choice whenever problem size permits.

In contrast, the more compact solvers (Charge and SCF) exhibit significantly larger optimality gaps on these instances. The Group-cutset formulation, however, maintains high performance even at small scales, indicating a degree of scale insensitivity.

One possible explanation for the weaker performance of the Charge and SCF formulations on these simulated instances, despite their competitive results on some larger real-world graphs (e.g., `Bridge-1000`), is the relatively small number of POIs. With fewer groups, coverage constraints become less restrictive, enlarging the feasible region of the LP relaxation and making tight lower bounds harder to obtain. In such settings, formulation strength, measured by lower-bound tightness, becomes the dominant factor, where Group-cutset excels. This interpretation is supported by the observed trend that the optimality gaps for the Charge and SCF solvers decrease as the number of POIs increases, whereas the gap slightly increases for group-cutset solvers.

D.3 Evaluation of Primal Heuristic

We evaluate the effectiveness of our problem-specific GIP heuristics discussed in App. C, and compare them against the heuristic proposed by Mizutani et al.

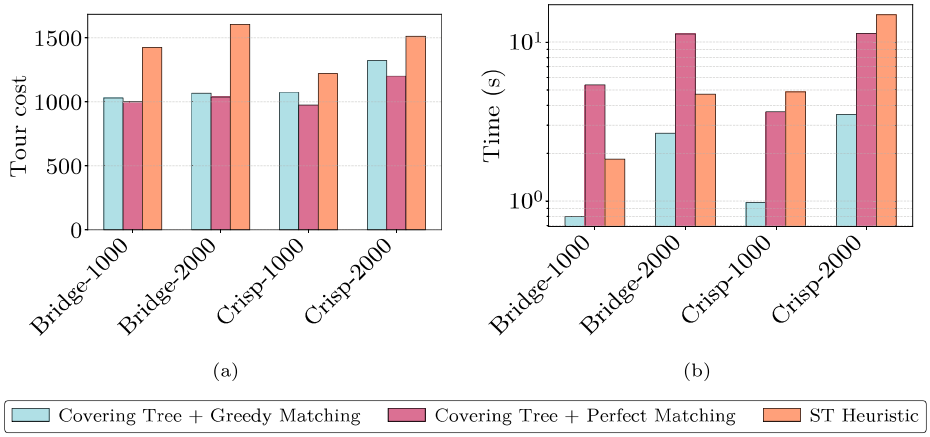


Fig. 10: Heuristic evaluation. **(a)** Tour cost comparison at the root of the BnC tree. Tour costs on `CRISP` instances are scaled by $1,000\times$. **(b)** Execution runtime in seconds.

[41], referred to as the *ST heuristic*. The ST heuristic first selects vertices greedily based on proximity to the root by finding the appropriate shortest-paths, until a group-covering set is obtained, then constructs a Steiner tree using a minimum spanning tree, which yields a 2-approximation of the optimal Steiner tree connecting those chosen vertices. Finally, their approach produces a tour by traversing the tree back and forth. We note that as presented, this heuristic is only applicable in an undirected graph regime.

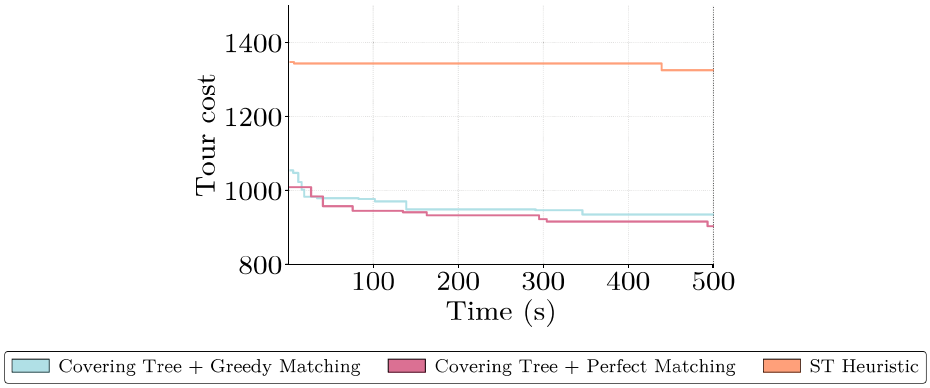


Fig. 11: Anytime performance on the `Bridge-1000` instance. The plot shows the tour cost improvement over time for the different heuristics.

In contrast, our GIP heuristic builds the group-covering tree iteratively, minimizing the incremental edge cost while covering new groups. This integrates vertex selection and tree construction into a single phase, which tends to yield more compact covering trees. Furthermore, the tree-to-tour conversion goes be-

yond simple back-and-forth traversal by augmenting the tree to an Eulerian graph and extracting an Eulerian tour.

We evaluate two variants of our "Covering-Tree" heuristic on real-world scenarios, which differ only in the final tour-construction phase (App. C): one using minimum-weight *perfect matching* and the other using *greedy matching*. Fig. 10 compares the resulting heuristic solutions[7] on the `CRISP` and `Bridge` instances. Specifically, Fig. 10(a) reports the cost of the heuristic-generated tour. In Fig. 10(b) we report the running time of a single heuristic execution on the scenario problem instance, i.e. independently from the BnC solver.

Finally, Fig. 11 illustrates a representative Group-Cutset solver run for the `Bridge-1000` instance, augmented with each heuristic variant, over a time horizon of $t = 500$ seconds.

Comparing the two Covering-Tree variants reveals a modest trade-off between solution quality and runtime (Fig. 10). Greedy matching substantially reduces heuristic computation time, at the cost of a mild degradation in solution quality of up to approximately 5%.

For solvers with sufficiently long time limits, this degradation has negligible impact on the final solution quality. Fig. 11 illustrates this behavior, showing that both variants, when used as primal heuristics within the BnC framework, consistently outperform the ST heuristic, with no clear dominance between them in this regime.

However, for the larger instances considered in Sec. 6.2, heuristic runtime becomes a dominant factor. In this regime, the faster greedy-matching variant provides a clear advantage, a trend we consistently observed during experimentation but do not report explicitly for brevity. Accordingly, we adopt the greedy-matching variant within the Group-Cutset BnC solver for all experimental evaluations (Sec. 6).

Comparing these results to those obtained using Gurobi's generic internal heuristics when applied to the Group-Cutset formulation highlights the importance of problem-specific primal heuristics. In additional experiments conducted during evaluation (not shown for brevity), our heuristic consistently produced substantially better incumbent solutions than Gurobi's default internal heuristics. On the `CRISP-1000` instance, where the difference is smallest, incumbent costs were approximately five times lower, while on more challenging instances the improvement reached up to two orders of magnitude. These results demonstrate the critical role of tailored primal heuristics for obtaining high-quality incumbents in lazy-constraint settings.

[7] To assess the intrinsic performance of the heuristics, both Covering-Tree variants are evaluated without LP-based cost discounting, i.e., using the original edge costs of the instance graph.