Decompositions for Semidefinite Programming with Reinforcement Learning

Asbjørn Ebenezer Magnussen[1], Bissan Ghaddar[2,3][0000−0003−4695−200X], and Stefan Ropke[1][0000−0002−6799−9934]

[1] Technical University of Denmark, amagnussen98@gmail.com, ropke@dtu.dk
[2] Ivey Business School, Western University, bghaddar@ivey.ca
[3] IE University, bghaddar@faculty.ie.edu

Abstract. Large-scale semidefinite relaxations arising in network structured optimization problems rely critically on chordal decompositions for computational tractability. While heuristic merging strategies are widely used, their performance can vary substantially across problem instances and decomposition choices. This paper proposes a learning-based framework for improving decomposition quality in semidefinite relaxations. We introduce a Random Forest surrogate model that ranks decompositions by expected solver runtime. Building on this surrogate, we design a reinforcement learning agent that sequentially constructs clique merge decisions, combining heuristic guidance with learned feedback. Empirical results on large optimal power flow instances show that the learned policy generalizes to unseen instances and in most cases outperforms standard heuristic merging strategies, achieving meaningful reductions in total solve time.

Keywords: semidefinite programming · chordal decomposition · reinforcement learning · optimal power flow

1 Introduction

Quadratic optimization is a cornerstone of mathematical optimization, with widespread applications across many fields. While convex quadratic problems can be solved efficiently, many real-world applications lead to inherently non-convex formulations that are significantly more challenging, especially at large scales. This non-convexity arises from underlying problem structure in energy systems [5,11], water networks [6], gas networks [12], and engineering design [15]. For large instances, such problems are often tackled using semidefinite programming (SDP) relaxations, which provide high-quality bounds on the optimal solution [20]. However, enforcing the positive semidefinite (PSD) constraint becomes computationally prohibitive as problem size grows. As real-world problems demand larger scales and near real-time solutions, this bottleneck motivates the need for strong yet scalable relaxations that explicitly exploit problem structure.

In many applications, including power grids, water distribution networks, and gas networks, the underlying structure of the network and the resulting optimization problem are sparse: components interact only with small subsets of the system. Thus, we focus on scalable semidefinite relaxations that exploit

correlative sparsity, where constraints and objectives depend on limited variable subsets. Instead of enforcing the PSD constraint on a single large matrix, it is imposed on multiple smaller matrices, with additional linking constraints to maintain consistency [10]. This approach scales SDP relaxations to much larger instances, but the performance depends on the chosen decomposition. Since many decompositions are possible, selecting a near-optimal one remains difficult, and existing heuristic strategies often yield suboptimal performance [2]. The difficulty of selecting high-quality decompositions suggests that purely heuristic rules may be insufficient for capturing the complex interactions that determine solver performance. In this context, machine learning methods offer a natural complement, as machine learning has shown promise in guiding algorithmic decisions that are hard to formalize analytically [4,9].

To the best of our knowledge, there has been little work focused on integrating machine learning into the design of SDP decomposition strategies. We briefly review these efforts to position this work. The study in [13] uses reinforcement learning to imitate the minimum-degree heuristic for chordal extension. While it reproduces the expert policy, it cannot generate novel or improved decompositions. In contrast, [2] learns to select the best decomposition from a candidate set produced by multiple heuristics and their random permutations, outperforming state-of-the-art strategies in most cases. This result motivates our work by showing that learning can surpass classical heuristics. A limitation of the paper is that it ranks decompositions, but does not generate new decompositions, restricting exploration to the small subset produced by heuristic candidates and their random permutations. A common assumption is that minimizing fill-in during chordal extension yields better decompositions. However, [20] shows that adding fewer edges does not necessarily lead to better performance, underscoring the complex link between structure and solver efficiency and the limits of minimal fill as a proxy for quality. Other work bypasses decomposition by predicting solutions directly. For example, [18] maps optimal power flow instances to solutions with a graph neural network, but generalizes only within a fixed network and is demonstrated only on small systems (30 and 118 nodes). Related methods [19] predict subsets of variables and recover the rest by post-processing. Although fast at inference, these approaches depend on ground-truth solutions and do not scale to large problems, where decomposition is still needed to generate training data. Moreover, they are specialized to optimal power flow, whereas learning-based SDP decomposition could apply more broadly.

This paper proposes a learning-based framework for generating chordal decompositions of large-scale semidefinite relaxations, with optimal power flow (OPF) as a motivating use case. We develop a data generation pipeline and a Random Forest surrogate to assess decomposition quality, and introduce a reinforcement learning policy that sequentially constructs merge decisions. The proposed approach outperforms established heuristics on unseen instances. The results demonstrate that learning-guided decomposition can achieve meaningful solver runtime reductions beyond fixed or greedy merging strategies.

2 Solution Methodology

SDP lifting converts nonconvex quadratic constraints into linear constraints over a lifted positive semidefinite matrix, but the resulting PSD constraint quickly becomes a computational bottleneck at scale. An SDP problem can be stated as

$$\begin{aligned} \min & \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}^0 x_{ij} \\ \text{s.t.} & \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}^k x_{ij} = b_k, \quad k = 1, \dots, m \\ & X \in \mathbb{S}^n, X \succeq 0 \end{aligned}$$

where $\mathbb{S}^n$ is the set of symmetric $n \times n$ matrices, $X \succeq 0$ constrains X to be a PSD matrix and x_{ij} are the elements of the X matrix. $Q^0 = \left(q_{ij}^0\right)$ and $Q^k = \left(q_{ij}^k\right)$ are symmetric $n \times n$ matrices (parameters of the problem). If Q^0 and Q^k are sparse we may be able to speed up computational efforts. We construct an undirected *sparsity graph* $G = (V, E)$ with nodes $V = \{1, \dots, n\}$ and E containing edges (i, j) if the corresponding element is non-zero in at least one of the Q^0 or Q^k, matrices. By constructing a chordal extension of this sparsity graph, the original large PSD constraint can be equivalently replaced by a collection of smaller PSD constraints defined on the maximal cliques of the chordal graph, coupled through linear consistency constraints on their overlaps [10]. This transformation relies on the fact that, for chordal graphs, global positive semidefiniteness is fully characterized by positive semidefiniteness on the clique submatrices. In practice, however, different chordal completions and clique merging strategies lead to substantially different computational performance, as they determine both the number and size of the resulting PSD blocks. Thus, as a first step, we investigate a regression task using a Random Forest (RF) model to determine reliable predictions of the computational performance of different decompositions. A regression model enables the selection of the most promising decomposition from a pool of candidates, thereby improving efficiency in solving the original problem. The predictive approach however relies on having already generated a diverse set of decompositions, and its effectiveness is inherently limited by the quality of the best decomposition in that set. This approach does not itself generate decompositions, it merely selects among them.

To overcome this limitation, a second component of our methodology explores a reinforcement learning (RL) approach. Here, a learned regression model acts as a surrogate evaluator, providing a reward signal to guide an agent in learning how to construct efficient decompositions autonomously. With this setup, the agent is able to explore a broader space of possible decompositions and to identify structures that may outperform those in the dataset. Additionally, at inference time, a successful agent would directly form high-quality decompositions, bypassing the candidate-generation and prediction stage entirely. For the reinforcement learning setup, we design an environment tailored to the use case,

specifying the state and action spaces, the reward structure, and the parameterization of both a policy and a value function.

2.1 Surrogate Model

We design a surrogate model to predict the computational performance of candidate SDP decompositions, with the goal of enabling data-driven decomposition selection without solving the underlying optimization problem. Among standard regression models, we adopt a Random Forest regressor as the surrogate. This choice is motivated by three considerations. First, RFs are well suited for heterogeneous, nonlinearly interacting features, which naturally arise from graph-level summary statistics. Second, RFs are robust to feature scaling and outliers, both of which are present due to large variation in problem size and decomposition structure. Third, RFs provide strong ranking performance in practice, which is critical in our setting where the relative ordering of decompositions matters.

Representing decompositions. Three closely related graph structures are used throughout this work: the original graph, its chordal extension graph, and the clique graph. The original graph corresponds to the sparsity pattern of the SDP relaxation, where nodes, $v \in V$, represent variables and edges encode pairwise interactions appearing in constraints or the objective. Since this graph is generally non-chordal, enforcing a sparse semidefinite relaxation directly is not possible. A chordal extension is therefore constructed by adding fill edges such that all cycles of length greater than three are triangulated, enabling a decomposition of the PSD constraint into smaller PSD constraints associated with maximal cliques. These maximal cliques form the nodes of the clique graph, where edges represent shared variables between cliques. The clique graph captures the interaction structure of the decomposed SDP and determines both the size of the local semidefinite blocks and the number of consistency constraints to ensure that shared variables among the cliques agree. As a result, properties of all three graphs jointly influence solver performance and are explicitly reflected in the feature representation. We use a tabular representation computed from summary statistics, aligned with prior work [2,13]. For each decomposition we compute 36 features: nine graph statistics computed on each of the original graph, its chordal extension, and the clique graph and nine clique-specific descriptors capturing the distribution of clique sizes and the complexity of separator constraints, including quartiles of clique sizes, the sum of cubed clique sizes (reflecting cubic SDP scaling), and the separator load induced by consistency constraints. Together, these features provide a compact yet expressive representation of the structural factors known to influence SDP solver performance (see Table 1).

Evaluation. To evaluate generalization in the regression task, we need to distinguish between two settings: (1) predicting solve times for unseen decompositions of the same problem instance, and (2) predicting for decompositions of entirely unseen problem instances. Due to the comparatively smaller intra-instance variation in graph structure and solve times, generalization to unseen instances is considered more challenging and practically important. Thus, our evaluation

Table 1: Summary of features computed for each decomposition.

Category	**Feature Name**	**Formula / Description**
Graph-specific	Number of nodes	$\lvert V\rvert$
	Number of edges	$\lvert E\rvert$
	Maximum degree	$\max_{v\in V} \deg(v)$
	Minimum degree	$\min_{v\in V} \deg(v)$
	Mean degree	$\frac{1}{\lvert V\rvert}\sum_{v\in V} \deg(v)$
	Median degree	$\text{median}\left(\{\deg(v) \mid v \in V\}\right)$
	Degree variance	$\text{Var}_{v\in V}(\deg(v))$
	Global clustering coefficient	$\frac{3\times\#\text{ triangles}}{\#\text{ connected triplets}}$
	Density	$\frac{2\lvert E\rvert}{\lvert V\rvert(\lvert V\rvert-1)}$
Clique-specific	Number of cliques	Number of maximal cliques (n_c)
	Mean clique size	$\frac{1}{n_c}\sum_{i=1}^{n_c} \lvert C_i\rvert$
	Minimum clique size	$\min_i \lvert C_i\rvert$
	Q1 clique size	First quartile of $\{\lvert C_i\rvert\}$
	Median clique size	Median of $\{\lvert C_i\rvert\}$
	Q3 clique size	Third quartile of $\{\lvert C_i\rvert\}$
	Maximum clique size	$\max_i \lvert C_i\rvert$
	Sum of cubed clique sizes	$\sum_{i=1}^{n_c} \lvert C_i\rvert^3$
	Separator load	Number of equalities needed

focuses on the model's ability to generalize across different entirely new problem instances taken from the optimal power flow usecase. This is in line with the approach from [2]. To assess this, we adopt an instance-based leave-one-instance-out cross-validation scheme. In each fold, all decompositions associated with one instance are held out for testing, while the surrogate is trained on decompositions from the remaining instances. This setup provides an unbiased estimate of generalization performance to previously unseen networks. Model performance is evaluated using both regression and ranking metrics. Standard regression metrics (MAE, MSE, RMSE) quantify absolute prediction accuracy. However, since the primary downstream objective is to select an efficient decomposition, ranking quality is equally important. We therefore also report Pearson and Spearman correlation coefficients, which measure linear and monotonic agreement between predicted and true solve times, respectively. Beyond standard regression metrics, we evaluate the practical usefulness of the surrogate model for guiding decomposition selection using top-k performance measures. In many realistic settings, a model is not required to identify the globally optimal decomposition with perfect accuracy; rather, it must reliably rank high-quality candidates near the top of its predictions. To capture this behavior, we report `top-1`, `top-5`, and `top-10` metrics, defined as the average of the true solver runtime of the best-ranked decomposition among the k lowest predicted runtimes.

While chordal decomposition enables scalable SDP formulations, the initial set of maximal cliques obtained from a chordal extension is not unique and often overly fine-grained. Minimum Degree (MD) and Approximate Minimum Degree (AMD) are graph-based heuristics used to compute elimination orderings for

chordal graph construction. Given an undirected graph, the MD heuristic iteratively eliminates the vertex with the smallest degree, introducing fill edges between its neighbors to preserve sparsity structure. This process yields a chordal extension of the original graph and determines the resulting set of maximal cliques. While MD aims to minimize fill-in exactly, maintaining and updating exact degrees is computationally expensive for large graphs. AMD is a scalable approximation that estimates vertex degrees using local structural information and aggressive update strategies, significantly reducing runtime while producing orderings of comparable quality in practice. Due to its favorable trade-off between quality and computational cost, AMD is widely used in large-scale sparse SDP applications. To reduce the number of semidefinite blocks and the associated consistency constraints, neighboring cliques can be merged, trading off fewer but larger PSD constraints against a reduced number of linking equalities. This merging process has a direct and often nontrivial impact on solver performance: excessive merging increases the cost of interior-point iterations due to larger matrix blocks, whereas insufficient merging leads to a proliferation of separator constraints that can degrade numerical stability. Consequently, selecting an effective merging strategy is critical, and small differences in merge decisions can lead to substantial differences in runtime. This sensitivity motivates treating clique merging as a decision problem rather than a fixed heuristic choice, and forms the basis for both the surrogate-based selection and the reinforcement learning approaches developed in this work.

2.2 Reinforcement Learning - Generation Through Policy Learning

Even with a successful and reliable surrogate model in place, we can only evaluate decompositions; how do we generate new ones? To address this, we introduce a reinforcement learning-based approach that utilizes the surrogate model to guide the search and efficiently identify high-quality decompositions.

Environment Design. In designing a reinforcement learning setup, careful consideration must be given to the environment structure. Several variants were initially explored. Sequential construction strategies such as adding fill edges until chordality or selecting elimination nodes step by step were considered. However, intermediate states do not correspond to valid decompositions, making the surrogate evaluation unreliable and forcing learning to rely on sparse terminal rewards. Another option explored was to generate an initial elimination ordering (e.g., via AMD or MD) and modify it through node swaps. While this preserves chordality throughout the trajectory, empirical tests showed that local swap actions had limited impact on decomposition quality.

To address these issues, the environment is defined directly over the clique graph merging process and in particular the action space is defined over the edges of the clique graph. Each episode begins with a chordal extension generated using AMD, from which the corresponding clique graph is constructed. Let $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ denote the clique graph at timestep t, where each node $i \in \mathcal{V}_t$ corresponds to a clique with a variable set C_i. For an edge $a = (i, j) \in \mathcal{E}_t$ connecting two cliques, we assign a weight $w(a) = |C_i|^3 + |C_j|^3 - |C_i \cup C_j|^3$, inspired by [1]. The intuition behind this measure stems from the cubic complexity of each

iteration of the interior point method with respect to the size of the positive semi-definite matrices. At each timestep t, the admissible action set is restricted to $\mathcal{A}_t = \{\, a \in \mathcal{E}_t \mid w(a) > 0 \,\}$, which significantly reduces the action space and naturally prunes poor merge decisions. A positive weight therefore indicates that merging the two cliques is expected to reduce the overall computational cost. As merging progresses, the size of $\mathcal{A}_t$ decreases, yielding a natural termination condition: an episode ends when $\mathcal{A}_t = \emptyset$. The state-transition dynamics are deterministic. Given an action $a_t = (i, j) \in \mathcal{A}_t$, the two cliques C_i and C_j are removed and replaced by a new clique $C_k = C_i \cup C_j$ and all nodes and edges features of the clique graph are updated accordingly, resulting in the next state s_{t+1}.

With the state representation, action space, and transition dynamics defined, the final component is the reward signal. Initially, only the surrogate prediction was used, but early findings revealed that relying solely on it led to unstable convergence. To support learning, a heuristic reward based on the edge weight is added to guide the agent. Let s_t and s_{t+1} denote the clique graph states before and after executing action a_t. The total reward is defined as $R_t = r_{\text{heur}}(a_t) + r_{\text{sur}}(s_t, s_{t+1})$, where r_{heur} encourages merges with high structural benefit and r_{sur} provides feedback based on the surrogate-predicted change in solve time. Let $w(a_t)$ denote the weight of the selected action, and define

$$w_{\max} = \max_{a \in \mathcal{A}_t} w(a), \qquad \text{rank}(a_t) = \frac{1}{|\mathcal{A}_t|} \left|\{\, a \in \mathcal{A}_t \mid w(a) > w(a_t) \,\}\right|,$$

that is, the fraction of admissible actions with strictly larger weight than a_t. From this, we compute

$$\text{value\_score}(a_t) = \frac{w(a_t)}{w_{\max}} \qquad \text{and} \qquad \text{rank\_score}(a_t) = 1 - \text{rank}(a_t).$$

To prevent inflated rewards near the end of an episode when few actions remain, a decay factor based on episode progress is applied: $\mathrm{d}_t = 1 - \frac{t}{|\mathcal{A}_0|}$, where $|\mathcal{A}_0|$ denotes the number of admissible actions at the start of the episode. The heuristic reward is then defined as $r_{\text{heur}}(a_t) = \mathrm{d}_t(\beta_v \,\text{value\_score}(a_t) + \beta_r \,\text{rank\_score}(a_t))$, where β_v and β_r are scaling factors selected empirically.

For the surrogate reward, let $f_{\text{surr}}(\cdot)$ denote the surrogate model predicting the solve time of a decomposition. Due to high variance in absolute predictions, the raw values are not used directly. Instead, we define $\Delta f_t = f_{\text{surr}}(s_t) - f_{\text{surr}}(s_{t+1})$, and employ a thresholded reward

$$r_{\text{sur}}(s_t, s_{t+1}) = \begin{cases} 1, & \text{if } \Delta f_t > \delta, \\ -1, & \text{if } \Delta f_t < -\delta, \\ 0, & \text{otherwise.} \end{cases} \qquad \text{with } \delta = 0.01.$$

This reward design encourages the policy $\pi_\theta(a_t \mid s_t)$ to balance structural heuristics with data-driven feedback. Early in an episode, the heuristic reward dominates and guides coarse decisions, while later stages rely increasingly on the

surrogate reward to refine merge choices. Although the agent observes only the clique graph, the environment maintains the corresponding decomposition graph in parallel. Each merge operation is reflected by adding the appropriate fill edges.

Policy and Value Parametrization. Both the policy π_θ and the value function V_ψ are parameterized using a Graph Neural Network where the nodes are represented using two usecase-specific features (detailed in Section 3.2) along with the clique size and edge features are assigned using $w(a)$. We adopt a shared feature-extraction backbone followed by separate policy and value heads, a standard design in policy-gradient methods [21], which balances representational efficiency and parameter economy. The feature extractor consists of two Graph Attention Network (GATv2) layers with a hidden dimension of 16 [7]. For each node (clique) $i \in \mathcal{V}_t$, the final-layer embedding is denoted by $\mathbf{h}_i^{(L)} \in \mathbb{R}^{16}$. This shared representation is used by both heads. At timestep t, each admissible action corresponds to selecting an edge $a_t = (i, j) \in \mathcal{A}_t$. For a candidate edge (i, j), we construct an edge-level representation by concatenating the embeddings of its incident nodes with the scalar edge attribute $w(i, j)$:

$$\mathbf{z}_{ij} = \left[\mathbf{h}_i^{(L)} \,\|\, \mathbf{h}_j^{(L)} \,\|\, w(i,j)\right] \in \mathbb{R}^{33}.$$

This representation is passed through a single-hidden-layer multilayer perceptron (MLP) with ReLU activation to produce a scalar logit

$$\ell_{ij} = \mathbf{w}_2^\top \sigma(\mathbf{W}_1 \mathbf{z}_{ij} + \mathbf{b}_1) + b_2,$$

where $\sigma(\cdot) = \mathrm{ReLU}(\cdot)$, $\mathbf{W}_1$ and $\mathbf{w}_2$ are learnable weight matrices, and $\mathbf{b}_1$ and b_2 are bias terms. To enforce feasibility, edges with non-positive heuristic weight are masked by setting $\ell_{ij} = -\infty$. The resulting logits define a categorical distribution $\pi_\theta(\cdot \mid s_t)$ over valid merge actions, from which the action a_t is sampled. The value head estimates the expected return from state s_t. A graph-level embedding is obtained by mean pooling over node embeddings,

$$\mathbf{h}_{\mathcal{G}_t} = \frac{1}{|\mathcal{V}_t|} \sum_{i \in \mathcal{V}_t} \mathbf{h}_i^{(L)}.$$

This pooled representation is passed through a separate MLP with one hidden layer of size 16 to produce the scalar value estimate $V_\psi(s_t)$, which serves as a baseline for variance reduction.

Training. Training is performed using the REINFORCE algorithm with a learned baseline [21]. This choice is motivated by the large and dynamically changing action space induced by the clique-merging process. The baseline plays a critical role in our setting, as rewards are predominantly positive; subtracting the value estimate $V_\psi(s_t)$ ensures that only above-average merge decisions are reinforced, thereby reducing gradient variance and stabilizing learning. Although the initial goal was to adopt a leave-one-group-out training strategy analogous to the surrogate evaluation, this approach resulted in unstable learning dynamics and inconsistent performance. As a more stable alternative, the agent was trained using a single fixed optimal power flow instance as the sole environment.

To maintain consistency under this single-instance training regime and avoid data leakage, the surrogate model used within the environment is trained exclusively on the decompositions generated from the same instance, ensuring that the agent did not gain indirect exposure to evaluation environments through the reward signal. An entropy regularization term is added to the policy objective to encourage exploration and prevent premature convergence to deterministic policies. This technique is widely used in policy-gradient methods [16] to maintain sufficient stochasticity and avoid poor local optima. Let $\hat{A}_t = G_t - V_\psi(s_t)$ denote the advantage estimate at timestep t, where G_t is the empirical return. The resulting policy loss is given by

$$\mathcal{L}_\pi(\theta) = -\mathbb{E}\Big[\log \pi_\theta(a_t \mid s_t)\, \hat{A}_t\Big] - \alpha\, \mathbb{E}\big[\mathcal{H}\big(\pi_\theta(\cdot \mid s_t)\big)\big]\,,$$

where $\mathcal{H}\big(\pi_\theta(\cdot \mid s_t)\big) = -\sum_{a \in \mathcal{A}_t} \pi_\theta(a \mid s_t) \log \pi_\theta(a \mid s_t)$ denotes the policy entropy, and $\alpha = 10^{-3}$ is the entropy coefficient. This value is chosen to balance exploration without encouraging excessively random behavior. To further improve training stability, gradient clipping with a maximum norm of 1 is applied during backpropagation. In addition, although the learned baseline already reduces variance by centering returns, advantage estimates are normalized to zero mean and unit variance prior to policy updates. This normalization prevents excessively large or small advantage values from dominating gradient steps, resulting in more stable and faster convergence. Notice that the surrogate model is only used during training. Inputs to the RL agent after training, are only the node and edge features mentioned earlier.

Evaluation. In reinforcement learning, the primary performance metric is the expected return, as this is the objective the agent is explicitly optimized to maximize. During training, we log both the episode return and the average policy entropy. The episode return serves as a direct measure of the agent's progress in optimizing its objective, while policy entropy provides insight into the agent's exploration behavior. Monitoring these two metrics allows us to diagnose convergence and assess whether the agent is effectively learning in the training environment. However, in our setting, the return signal is not the ultimate metric of interest. Our true objective is to determine whether the learned policy produces decompositions that lead to faster solve times. To test whether the agent genuinely learns beyond the heuristic reward, we introduce an additional baseline that always selects the edge with the highest heuristic weight at each step. This strategy is analogous to the clique graph merging heuristic described in [1] and we refer to this approach as CGM.

3 Use Case and Computational Results

This section presents the use case and the data generation scheme, along with computational results showcasing the performance of the surrogate modeling approach, followed by an evaluation of the reinforcement learning method.

3.1 Data Generation

The optimal power flow (OPF) problem provides a natural benchmark for learning-based decomposition strategies. OPF is a large-scale network optimization prob-

lem whose semidefinite relaxation becomes computationally challenging as grid size grows, making chordal and clique-based decompositions essential for scalability. This also makes OPF well suited for decomposition learning: modest changes in the induced chordal structure and clique merging decisions can lead to noticeable differences in solver runtime and numerical behavior. The formulation of the OPF is taken from [8].

OPF instances. All instances are taken from Power Grid Lib–Optimal Power Flow (PGLib-OPF) v23.07 [3]. PGLib-OPF provides 65 test cases with multiple operating-condition variants. We use only the base-case versions to reflect realistic operating conditions and to avoid artificially stressed scenarios that could bias decomposition performance. To balance scale and tractability, we restrict to networks in the range 1,000-6,000 buses, yielding 22 instances. Table 2 summarizes the selected cases along with the size of the resulting conic formulation under the baseline decomposition (AMD chordal extension, no merging).

Generating a library of decompositions. For each OPF instance, we generate a diverse set of chordal decompositions and record the corresponding solver runtimes. The pipeline is inspired by [2] and introduces variability at three stages: graph perturbation, chordal extension, and clique merging. We first create structural perturbations of the original network to alter the induced chordal structure. Specifically, we consider (i) no perturbation, (ii) weak perturbation by adding 20 edges, and (iii) strong perturbation by adding 50 edges. Added edges connect pairs of nodes at graph distance two, which primarily creates triangles and changes the elimination dynamics. Each perturbed setting is repeated three times, giving 7 graph variants per instance. Each variant is chordalized using two elimination heuristics: minimum degree and approximate minimum degree. Finally, we apply the clique merging heuristic of [8] using a single threshold tol. Let C_k denote a clique in the clique tree with parent $C_{\mathrm{pa}(k)}$, and separator $\eta_k = C_k \cap C_{\mathrm{pa}(k)}$. A merge is performed if

$$(|C_{\mathrm{pa}(k)}|-|\eta_k|)(|C_k|-|\eta_k|) \le tol \quad \text{or} \quad \max\Big(|C_k|-|\eta_k|,\ |C_{\mathrm{pa}(k)}|-|\eta_{\mathrm{pa}(k)}|\Big) \le tol.$$

The first condition controls fill-in created by merging, while the second prevents merges that create large "exclusive" clique parts. Varying tol yields a continuum of trade-offs between the number of cliques and clique sizes, which is useful for learning. Overall, the pipeline generates 434 decompositions per instance, for a total of $22 \times 434 = 9{,}542$ decompositions. Each decomposition is solved using the implementation in [8], and the solve time is stored as the supervision signal.

To assess whether the features presented in Table 1 capture meaningful variation, we apply PCA separately within each OPF instance and examine the relationship between the first principal component (PC1) and normalized solve time. In many cases, larger PC1 values correspond to lower runtimes (Fig. 1), suggesting that PC1 captures a performance-relevant structural direction, though exceptions exist (e.g., case 2312). Across instances, PC1 loadings are qualitatively consistent: features related to clique-graph connectivity (e.g., clique-graph edge count and degree statistics), number of cliques, and separator load tend to contribute positively, indicating that decomposition efficiency is driven by system-

atic trade-offs between separator complexity, clique sizes, and induced sparsity patterns.

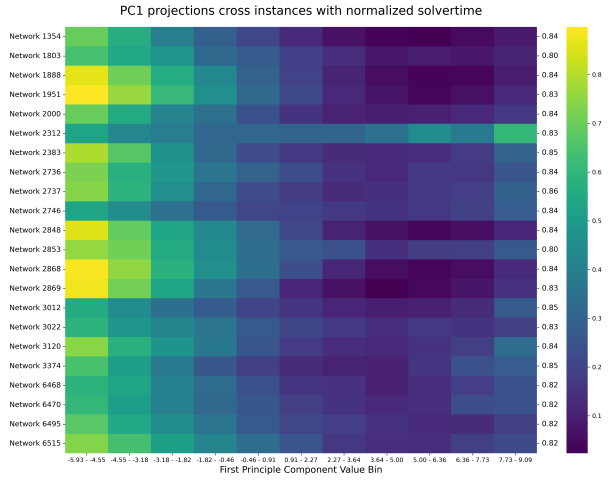

Fig. 1: Projections of decomposition features onto bins of the PC1.

3.2 Surrogate and RL Results

The Random Forest regressor is implemented using the scikit-learn library[4], with all hyperparameters left at their default values. In particular, bootstrap samples are used to build each tree, all features are considered at each node split within the trees, and the squared error criterion is used for measuring the splits. The resulting random forest surrogate shows a clear overall correlation between predicted and observed solve times across hold-out OPF instances (see Fig. 2), indicating its ability to capture meaningful performance trends across candidate decompositions.

Regarding RL training, training is conducted on a single mid-sized instance (case 3120). The surrogate model f_{surr} used in the environment reward is trained exclusively on decompositions from the same instance. Among the evaluated surrogates, a random forest regressor with 100 trees and no depth limit achieved the best predictive performance and is therefore selected. The mean Spearman correlation on the hold-out test sets is 0.875, and the mean Pearson correlation on the hold-out test sets is 0.861. The standard error of these measures is small (0.033), indicating low variability across instances. For comparison, the method in [2] reported a Pearson correlation of 0.57 on their test set, highlighting the stronger predictive performance achieved here. As mentioned in Section 2.2, the RL agent uses two problem-specific node features corresponding to the average active and reactive power of the buses within the clique. The agent is trained for 500 episodes using the Adam optimizer. The policy learning rate is set to 2.2×10^{-2}, while the value-function learning rate is set to 1×10^{-3}, following

[4] https://scikit-learn.org

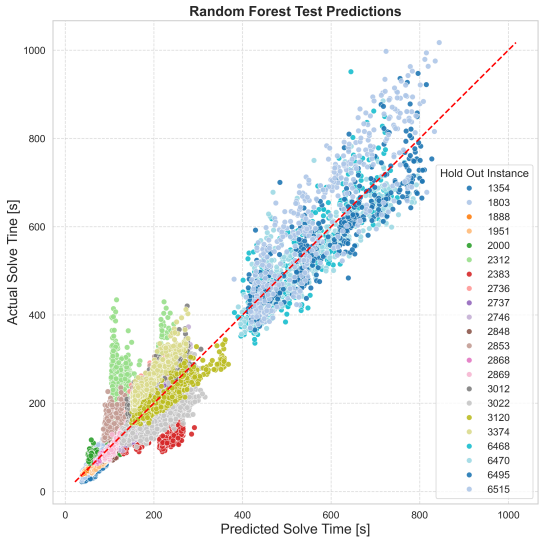

Fig. 2: Surrogate test predictions vs actual solve times for the combined test set, aggregated across all hold-out instances. Color indicates each instance.

standard practice for actorcritic-style training. All experiments are implemented in PyTorch and executed on the DTU HPC cluster using Tesla A100 GPUs. A fixed random seed is used to ensure reproducibility. The implementation is publicly available [14]. After training the RL agent on instance 3120, we applied the agent to all instances in our data set and reported the MOSEK [17] solve time for the resulting decomposition. The results are shown in Table 2 (column `RL`). The table compares the solve time (in seconds) to that obtained by other decomposition methods. Bold numbers indicate the best performance for a particular instance (disregarding results in the Best column). The other methods included are:

- **`CGM`**: Heuristic defined in Section 2.
- **`top-k`**: the average solve time over the predicted k best decompositions (using the RF model) out of 434 decompositions generated for each instance.
- **`No Merge`**: Solve time using the chordal decomposition from AMD ordering with no clique merging.
- **`Fixed Merge`**: Solve time from AMD ordering followed by merging using a fixed threshold of 16, as recommended in [8].
- **`Best Merge`**: The best solve time achieved using AMD ordering and merging, with the merge threshold, tol, selected from the candidate range (0 to 30).
- **`Best`**: The fastest solve time observed among all 434 decompositions for that instance, regardless of edge-perturbation, ordering or merging strategy.

Of the methods listed, only `RL`, `CGM`, `No Merge` and `Fixed Merge` can be applied to a new instance without excessive computation. `Best merge` requires solving 31 decompositions, and `Best` requires generating and solving all 434

decompositions. The `top-1` results could also be seen as practically feasible; this requires generating 434 decompositions and evaluating each one using the RF model and selecting the best. The `top-5` and `top-10` results require solving 5 or 10 decompositions, and are therefore computationally impractical.

Table 2 shows that `RL` provides decompositions with the shortest solve time overall (see "total" row), and it is also the method with the most "wins" (better than the other methods in 12 out of 22 cases). Of the other practical methods, `CGM` and `top-1` do well, while the performance of `Fixed Merge` is disappointing when compared to the alternatives. It is interesting to note that `RL` in 9 out of 22 cases finds a better decomposition than the best decomposition in the training set, indicating that the method can generate decompositions that do not have an equivalent in the training set. A post hoc bootstrap comparing `RL` to `CGM` yields an estimated mean advantage of 9.5s with a 95% CI of [0.35, 21.08] and a one-sided bootstrap p-value of 0.0308, supporting that `RL` is faster on average.

Table 2 only shows MOSEK runtime on the generated decomposition and not how long it takes to generate the decomposition. It turns out that the running time of the RL agent is low. It takes between 0.6 and 1.8 seconds to execute the RL agent (longer time for larger instances). For comparison, it takes between 0.2 and 0.7 seconds to execute the CGM heuristic.

Table 2: OPF instance characteristics and solve times (in seconds) for different decomposition strategies. Green row indicate training environment for RL.

Case	Problem Size				RL / Heuristic		RF Selection			Baselines			Best
	Scalar Const.	Cone Const.	Scalar Vars.	Matrix Vars.	`RL`	`CGM`	`top-1`	`top-5`	`top-10`	`No Merge`	`Fixed Merge`	`Best Merge`	
case1354	27,520	3576	18,458	1287	**20**	**20**	21	23	23	27	40	24	21
case1803	35,040	5590	26,874	1685	30	**23**	34	35	35	35	59	34	26
case1888	35,776	4820	24,440	1816	**37**	39	38	39	40	46	67	42	38
case1951	36,786	4942	25,364	1879	**40**	43	42	41	42	50	74	44	40
case2000	58,587	6676	32,354	1854	83	**69**	82	75	73	98	100	73	63
case2312	55,815	6068	29,800	2217	**128**	155	267	259	256	342	265	206	165
case2383	52,970	5792	28,980	2312	**82**	90	91	94	94	124	122	105	85
case2736	60,312	6538	32,232	2652	130	**107**	112	125	120	140	171	127	108
case2737	60,010	6538	32,068	2653	113	**106**	128	121	118	145	149	116	97
case2746	61,714	6614	32,768	2653	107	163	**105**	109	110	186	144	126	92
case2848	54,701	7124	36,536	2739	69	78	69	**68**	70	81	124	75	65
case2853	63,618	7842	40,350	2724	**109**	137	114	118	122	143	164	167	114
case2868	55,214	7178	37,002	2763	69	72	**65**	68	69	83	121	72	63
case2869	66,922	8368	42,046	2700	75	76	**74**	75	75	86	124	**74**	71
case3012	67,035	7144	35,886	2916	**134**	168	149	170	169	227	235	150	139
case3022	68,988	8380	40,872	2880	123	**107**	109	116	122	138	140	120	101
case3120	69,770	7386	36,700	3029	**148**	155	165	172	172	226	201	169	153
case3375	76,572	8300	41,626	3248	**185**	188	195	204	204	258	246	202	184
case6468	136,836	16,576	82,148	6153	**381**	405	405	384	388	506	582	422	336
case6470	137,980	16,856	84,088	6149	**318**	406	406	394	391	408	477	408	356
case6495	137,671	16,888	84,234	6171	**354**	358	461	433	428	445	550	427	366
case6515	137,991	16,926	84,438	6193	454	443	481	427	440	553	498	**410**	370

4 Conclusion

The random forest surrogate demonstrated useful rank-based predictive capability across OPF instances, despite exhibiting high absolute prediction errors on some cases. This behavior aligns with its intended role: guiding decomposition

selection rather than predicting exact solve times. RF-based selection consistently outperformed the no-merge and fixed-threshold baselines, confirming that decomposition features capture informative signals. However, the RF did not reliably match the strongest heuristic baseline, CGM, nor consistently identify the best decomposition within the candidate library. In practice, this suggests that RF-based top-k selection is more robust than single-candidate selection, at the cost of evaluating multiple decompositions. A key limitation of this approach is its dependence on a pre-generated candidate pool, which constrains exploration to the support of the dataset.

The reinforcement learning approach overcomes this limitation by learning a sequential merging policy rather than selecting from a fixed library. Despite relying on an imperfect surrogate signal, the learned policy generalized well to unseen OPF instances and achieved lower total solve times than heuristic baselines, even after accounting for action-selection overhead. Importantly, the policy did not simply imitate CGM decisions, indicating that it exploited additional structural information beyond greedy edge selection.

Stable training was achieved when using a single mid-sized instance as the training environment. While strong generalization suggests that the learned behavior is not instance-specific, extending training to multiple instances will likely require further stabilization and tuning. A second limitation arises from reward misalignment: the surrogate was trained on final decompositions from a threshold-based heuristic, whereas the RL agent evaluates intermediate merge actions drawn from a different distribution. This mismatch necessitated heuristic reward shaping and restricts exploration. Promising directions include training surrogates on intermediate states, incorporating cumulative rewards relative to the initial decomposition, and integrating active learning to selectively query exact solve times for uncertain states. Extending the framework to other sparse SDP or conic relaxations would further clarify its generality beyond OPF.

Acknowledgments. The authors gratefully acknowledge the financial support of the Danish Council for Independent Research (Grant ID: DFF - 5244-00162B), the Natural Sciences and Engineering Research Council of Canada Discovery RGPIN-2025-04585, and the John Thompson Chair Fellowship.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. A clique graph based merging strategy for decomposable SDPs. IFAC-PapersOnLine **53**(2), 7355–7361 (2020), 21st IFAC World Congress
2. Alizadeh, C., Alizadeh, P., Anjos, M.F., Létocart, L., Traversi, E.: How to learn the optimal clique decompositions in solving semidefinite relaxations for OPF. In: 2022 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2022)
3. Babaeinejadsarookolaee, S., Birchfield, A., Christie, R.D., Coffrin, C., DeMarco, C., Diao, R., Ferris, M., Fliscounakis, S., Greene, S., Huang, R., Josz, C., Korab, R., Lesieutre, B., Maeght, J., Mak, T.W.K., Molzahn, D.K., Overbye, T.J., Panciatici, P., Park, B., Snodgrass, J., Tbaileh, A., Hentenryck, P.V., Zimmerman, R.: The

power grid library for benchmarking AC optimal power flow algorithms (2021), https://arxiv.org/abs/1908.02788
4. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour dhorizon. European Journal of Operational Research **290**(2), 405–421 (2021)
5. Bingane, C., Anjos, M.F., Le Digabel, S.: Tight-and-cheap conic relaxation for the optimal reactive power dispatch problem. IEEE transactions on power systems **34**(6), 4684–4693 (2019)
6. Bragalli, C., DAmbrosio, C., Lee, J., Lodi, A., Toth, P.: On the optimal design of water distribution networks: a practical minlp approach. Optimization and Engineering **13**, 219–246 (2012)
7. Brody, S., Alon, U., Yahav, E.: How attentive are graph attention networks? (2022), https://arxiv.org/abs/2105.14491
8. Eltved, A., Dahl, J., Andersen, M.S.: On the robustness and scalability of semidefinite relaxation for optimal power flow problems. Optimization and Engineering **21**(2), 375–392 (2019)
9. Fan, Z., Ghaddar, B., Wang, X., Xing, L., Zhang, Y., Zhou, Z.: Artificial intelligence for optimization: Unleashing the potential of parameter generation, model formulation, and solution methods. European Journal of Operational Research (2025)
10. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting sparsity in semidefinite programming via matrix completion i: General framework. SIAM Journal on Optimization **11**(3), 647–674 (2000)
11. Kocuk, B., Dey, S.S., Sun, X.A.: New formulation and strong misocp relaxations for ac optimal transmission switching problem. IEEE Transactions on Power Systems **32**(6), 4161–4170 (2017)
12. Li, Y., Dey, S.S., Sahinidis, N.V.: A reformulation-enumeration minlp algorithm for gas network design. Journal of Global Optimization **90**(4), 931–963 (2024)
13. Liu, D., Lodi, A., Tanneau, M.: Learning chordal extensions. Journal of Global Optimization **81**(1), 3–22 (2021)
14. Magnusson, A.: Master's thesis code repository. https://github.com/amagnus98/MastersThesis (2025), accessed: 2025-06-18.
15. Misener, R., Floudas, C.A.: Advances for the pooling problem: Modeling, global optimization, and computational studies. Applied and Computational Mathematics **8**(1), 3–22 (2009)
16. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. International Conference on Machine Learning (ICML) pp. 1928–1937 (2016)
17. MOSEK ApS: Mosek optimization software. https://www.mosek.com (2024)
18. Owerko, D., Gama, F., Ribeiro, A.: Optimal power flow using graph neural networks. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5930–5934 (2020)
19. Pan, X., Chen, M., Zhao, T., Low, S.H.: DeepOPF: a feasibility-optimized deep neural network approach for AC optimal power flow problems. IEEE Systems Journal **17**(1), 673–683 (2023)
20. Sliwak, J., Anjos, M., Létocart, L., Maeght, J., Traversi, E.: Improving clique decompositions of semidefinite relaxations for optimal power flow problems (2019), https://arxiv.org/abs/1912.09232
21. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (2018)