A Learning Framework for Twin-Width and Related Problems*

Ryan O'Connor
University College Dublin
Dublin, Ireland

Johannes Meintrup
University of Applied Sciences Mittelhessen
Giessen, Germany

Maximilian Huber
University of Applied Sciences Mittelhessen
Giessen, Germany

Alexander Leonhardt
Goethe University
Frankfurt am Main, Germany

Manuel Penschuck
University of Southern Denmark
Odense, Denmark

Yosuke Mizutani
University of Utah
Salt Lake City, U.S.A.

Oscar Yeoh
University College Dublin
Dublin, Ireland

Deepak Ajwani
University College Dublin
Dublin, Ireland 0000-0001-7269-4150

Abstract

Efficiently solving NP-hard network optimization problems often relies on exploiting the underlying topological structure of the graph. Structural parameters such as treewidth and the recently introduced twin-width offer a pathway to tractable solutions via parameterized algorithms. However, computing these parameters and their associated decompositions is itself a computational bottleneck. This paper explores whether machine learning (ML) can augment the development of effective computational routines for novel graph parameters across diverse graph classes. Focusing first on twin-width, we show that classification models built from simple graph features accurately predict twin-width, and that ML can produce contraction sequences with widths comparable to leading solvers. We then apply the same generic ML framework to treewidth, proving its efficacy in producing optimal elimination orderings. These results highlight the potential of ML to accelerate research in parameterized algorithms by reducing the reliance on extensive manual engineering, thereby unlocking the potential of structural parameters for practical, efficient network optimization.

Keywords

Parameterized Algorithms, Twin-width, Treewidth, Algorithm Engineering, Machine Learning

1 Introduction

A pervasive challenge in network optimization is the combinatorial explosion inherent to NP-hard problems, which renders exact solvers intractable for large-scale instances. To circumvent this, Fixed-Parameter Tractable (FPT) algorithms exploit structural graph measures, such as the classical treewidth and the recently introduced twin-width, to confine combinatorial explosions to a small parameter rather than the input size. This approach theoretically renders NP-hard problems, including Maximum Independent Set and Minimum Dominating Set, tractable on structured networks. For these theoretical advantages to materialize in practice, however, two prerequisites are essential: efficient computation of the parameters and a systematic characterization of their behavior across diverse network topologies.

Traditionally, establishing these properties for new graph parameters has required years of intensive manual effort and algorithmic engineering. A compelling recent example is twin-width: developing effective solvers to compute this parameter, a prerequisite for running twin-width-based optimization routines, demanded long-term, hands-on work, ultimately leading to major progress in the PACE 2023 challenge (e.g., [5, 11, 14, 33, 34]).

Given the high cost of manual engineering, data-driven automation offers an attractive alternative. However, most of the existing work on using machine learning to engineer algorithms for NP-hard combinatorial optimization problems has focused on subset problems, where the optimal solution is defined as a subset of nodes or edges that satisfy certain constraints (e.g., Vertex Cover [31], Facility Location [41], Vehicle Routing [26, 30] and Maximum Clique [32]). These learning approaches focus on pruning the search space by leveraging features derived from algorithmic insights to identify elements that are either unlikely or likely to be in the optimal solution. We note that these techniques are not directly applicable to learning contraction sequences for twin-width computation, as this problem cannot be described as a subset problem, but is instead an *order-based optimization problem*. Thus, there is a need for a novel machine-learning approach to discover heuristics for and glean insight on order-based optimization problems, which is exactly what we contribute in this paper.

In this paper, we explore a fundamental question: Can machine learning (ML) techniques automate the development of computational heuristics to estimate novel parameters across diverse problem instances? We answer this affirmatively. We demonstrate that both twin-width and treewidth can be effectively estimated using interpretable classification models based on simple topological features. This suggests that ML can provide a rapid, automated pathway to gauge the tractability of networks without solely relying on complex, hand-tuned heuristics.

Furthermore, solving a network optimization problem via FPT algorithms requires more than just a parameter estimate; it requires a structural certificate – a contraction sequence for twin-width or an elimination ordering for treewidth – to guide the Dynamic Programming process. This leads to our next critical question: Can we learn to efficiently generate these decomposition certificates while still producing estimates comparable to those from carefully engineered, state-of-the-art solvers? We propose a generic learning framework that generates certificates for both twin-width and treewidth. Our approach outperforms a standard greedy baseline and yields results only slightly inferior to GUTHM [33], the highly specialized PACE 2023 winning

*A poster based on this work, with limited details, will be presented at SofSem 2026

INOC '26, Liège (Belgium)

10.48786/inoc.2026.23

technique, while utilizing minimal domain knowledge of the two parameters.

To construct complete contraction sequences, we iteratively expand partial sequences managed in a priority queue, with priorities guided by learned twin-width estimates. Efficiently navigating the vast search space necessitates further algorithm engineering: we limit the number of contractions at each step and bias the queue toward near-complete sequences. Both techniques focus the solver on more promising sequences and complement accurate predictions.

A persistent barrier to employing parameters like twin-width in practical optimization is the opacity of their structural underpinnings. For many graph classes, we lack closed-form expressions or general reduction rules, making it difficult to predict algorithm performance. This raises our third critical question: Can ML provide interpretability regarding how these parameters are influenced by specific network characteristics? We demonstrate that it can. By analyzing feature importance within our classification models, we uncover key topological predictors across different graph classes. For instance, in Erdős–Rényi random graphs $G(n, p)$, we identify edge count and density as the primary predictors of twin-width, a finding that empirically aligns with known closed-form expectations [1, 2]. Similar insights emerge from feature selection on other graph classes, establishing that ML can illuminate the structural properties of emerging parameters even when theoretical domain knowledge is limited.

The success of this data-driven framework suggests a paradigm shift in the development of parameterized algorithms. Rather than dedicating years to the manual engineering of computational routines for new graph parameters, researchers can leverage learning techniques to rapidly prototype robust solvers. This approach effectively augments the algorithm design pipeline, significantly reducing the cost and effort required to understand how novel parameters behave across diverse problem instances. By providing a method to generate high-quality decomposition certificates comparable to state-of-the-art solvers, our framework facilitates the immediate practical application of theoretical parameters in network optimization tasks.

We generate a substantial dataset [35] comprising 115,307 graphs from various distributions along with 295,238 contraction sequences of minimum twin-width, requiring over 15k core hours using state-of-the-art solvers. We make this dataset available for its independent utility to the broader research community working on graph parameters, algorithm engineering and network optimization. [35]

Outline. Section 2 reviews preliminaries on twin-width and treewidth, and Section 3 surveys related work on these parameters and ML for combinatorial optimization problems on graphs. Section 4 details the generation of labeled datasets for training our models. Sections 5 and 6 present our twin-width solver methodology and experimental results, while Section 7 briefly covers the ML-guided treewidth solver. Section 8 concludes with a discussion of the framework's strengths and limitations.

2 Preliminaries

Twin-width and treewidth are fundamental graph parameters that quantify the structural complexity of graphs.

Twin-width measures how "close" a graph is to a cograph, a graph that can be reduced to a single vertex by iteratively merging node pairs with identical closed neighborhoods (twins).

A *contraction sequence* of a simple graph is a sequence of simple graphs obtained by repeatedly merging two nodes into a single node. Initially, all edges are considered *black*. During each contraction of a pair of nodes u and v, edges are colored red if they do not belong to the intersection of their neighborhoods $N(u) \cap N(v)$. Note that edges colored red at some point remain red. The contraction process is applied iteratively until a single node remains. The number of incident red (black) edges of a node are called its *red degree* (*black degree*). The *width* of a contraction sequence is the maximum red degree any node has in any of the intermediate graphs. The *twin-width* of a graph G is the minimum k such that G admits a contraction sequence of width k.

Treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, B) where $T = (W, F)$ is a tree and B is a mapping $W \to \{V' \mid V' \subseteq V\}$ with the following properties: (i) $\bigcup_{w \in W} G[B(w)] = G$, and (ii) for each node $v \in V$, the nodes w with $v \in B(w)$ induce a subtree in T. For each node $w \in W$, $B(w)$ is called the *bag* of w. The *width* of a tree decomposition is the size of the largest bag plus one. The *treewidth* of a graph G is the smallest k such that a tree decomposition of G with width k exists.

An alternative certificate for the treewidth is a so-called elimination order, which is a sequence of the nodes of a graph. To *eliminate* a node u, the neighborhood $N(u)$ is made into a clique and u is removed from the graph. This process is repeated until no node remains. The *width* of an elimination order is defined as the size of the largest clique plus one encountered in any intermediate graph during elimination. These two definitions are equivalent, and a tree decomposition of the same width can be constructed in polynomial time from an elimination order, and vice-versa. Thus, treewidth can be considered an order-based optimization problem.

3 Related Work

Twin-width. The parameter twin-width is of interest as all problems that can be formulated in First Order (FO) logic can be solved in FPT time parameterized by the twin-width [21]. Distinguishing between graphs of twin-width 4 and 5 is NP-complete [9]. For further theoretical results, see [13, 15–21]. Graphs of twin-width 0 and 1 are recognizable in linear time [3]. It remains open whether graphs of twin-width 2 or 3 can be detected in polynomial time. Notably, sparse graphs of twin-width 2 possess bounded treewidth [10], underscoring the connections between twin-width and treewidth.

We are aware of only a single practical solver for twin-width outside of PACE 2023 [7], a SAT-based formulation [39] limited to very small graphs. In contrast to many NP-complete problems, no practical reduction rules are known, except for trivial merging of twins and fringe cases that effectively exploit small separators [34]. Recent exact solvers use branch-and-bound [33, 34], and prior solvers used SAT formulations [39]. Heuristically, greedily merging two nodes resulting in the minimum red degree is successful, as witnessed in the PACE 2023 winner [33].

Treewidth is motivated by the fact that Monadic Second Order (MSO) problems can be solved in FPT time when parameterized by the treewidth [25]. Deciding whether a graph has treewidth k is FPT [12]. State-of-the-art solvers combine dynamic programming (DP) ideas [4] with potential-maximal-clique techniques [22], and utilize sophisticated preprocessing, including reduction rules [42] and special node separators [40]. Greedy algorithms iteratively building elimination orders are fast and commonly used for initial

upper bounds or quick filtering; common choices are *min-degree* and *minimum fill-in* (least number of edges).

ML for Combinatorial Optimization on Graphs. ML for hard graph problems is well studied (e.g., surveys [8, 28, 38, 43]). As noted earlier, most of these are subset problems. For ordering problems, some work exists using mixed-integer linear programming branch-and-bound solvers (e.g., [6, 44]), but these methods are not directly applicable to learning contraction sequences or elimination orderings. Techniques like imitation learning assume the existence of strong greedy rules to imitate, which is unlikely for novel parameters and measures.

4 Labeled Data Generation

Training supervised models requires large, diverse datasets, but obtaining labels via exact solvers is costly. We computed optimal twin-widths for all graphs using the top two exact solvers [33, 34] from PACE 2023 [7]. A key strength of our ML approach is that it requires minimal domain knowledge: the exact solvers only provide the labels and no additional insight. SAT solvers could generate the same labels, but at far greater computational cost. We generated and labeled a set of 3911 random hyperbolic graphs (also denoted *RHG*), with $2.5 \leq \gamma \leq 3$ and $35 \leq n \leq 45$, resulting in twin-widths of at most 4. This class is of limited utility as instances with twin-width of at most 1 can be recognized in linear time [3]. Preliminary experiments were also run on Erdős-Rényi graphs with $30 \leq n \leq 50$, and $p \in \{0.05, 0.1, \ldots, 0.45\}$ yielding twin-widths ranging from 3 to 11. This graph class has a closed-form [1] bound: With high probability and for every $\varepsilon > 0$ and $1/n \leq p := p(n) \leq 1/2$

$$\text{twin-width}(G(n,p)) > 2p(1-p)n - (2\sqrt{2}+\varepsilon)\sqrt{p(1-p)n\ln n}.$$

We sampled 1699 of the resultant graphs to ensure diversity of twin-widths over the number of edges. This approach aims to discourage models from relying purely on edge counts to classify graphs. We also generated a dataset of Erdős-Rényi graphs with $8 \leq n \leq 48$ and $p = 1/2$. This value of p yields the highest expected twin width value [1], making it of particular interest in our solver evaluation: With high probability, the twin-width of $G(n, 1/2)$ is

$$\text{twin-width}(G(n,1/2)) = \frac{n}{2} - \frac{\sqrt{3n\ln n}}{2} + o(\sqrt{n\ln n})$$

We also considered another synthetic graph class, referred to as "perturbed rook graphs". These graphs are $n \times m$ grids where each node connects to all others in its row and column. To introduce variety, we randomly add edges with probability a between nodes not in the same row or column, and delete edges with probability b between nodes in the same row or column. This yields high twin-width graphs, which are otherwise difficult to generate. In fact, we are not aware of any other techniques to consistently generate graphs of high twin-width (outside of the Erdős-Rényi approach above, with $p = 0.5$). We used all $n, m \in [5, 9]$ and $a, b \in 0.1, 0.2$, producing twin-widths from 6 to 17. We also include 8000 real-world planar graphs with $n < 100$, whose twin-widths lie in the range $0 - 5$. Such data is useful, so as to keep in scope possibilities for real-world applications of twin-width. We provide the dataset [35] and our code implementations[1]. Since the data is computationally expensive to produce and we are unaware of any established source of solved high twin-width graphs, we believe it is useful for practitioners.

[1] https://github.com/ryan-o-c/Learning_Framework_Twin_Width

5 Learning Twin-Width

Here we outline our two-pronged methodology which utilizes machine learning for twin-width computation. Each approach addresses a distinct challenge: We first train supervised classification models using simple graph-theoretic features to predict twin-width across graph classes. Performant classifiers provide insight into how different features relate to twin-width. Feature selection identifies the most discriminative features, which is crucial for successful ML applications. Next, we leverage the trained twin-width classification models to generate a contraction sequence. Navigating the vast solution space is challenging, but our ML classifiers facilitate efficient search, producing sequences comparable in width to state-of-the-art solvers.
Before describing the above methodologies, we first discuss the common thread between these approaches: feature generation.

5.1 Feature Generation for Twin Width

Supervised models are underpinned by features which provide useful information and context to the model. For new parameters, we often lack knowledge of which features correlate with the parameter. We therefore use standard graph-theoretic features, along with some features that simply follow from the definition of the parameter. The full set of features considered for twin-width computation include (i) the total number of nodes and edges, (ii) graph red and black edge density, (iii) graph diameter, (iv) average clustering coefficient, (v) node connectivity, (vi) average shortest path length, (vii) average co-occurrence, (viii) average Jaccard coefficient, (ix) average degree centrality, (x) `GL2Vec` graph embeddings, (xi) maximum red degree, (xii) the total number of red edges, (xiii) average red degree (of nodes with nonzero red degree), (xiv) proportion of nodes with a nonzero red degree, (xv) a binary feature to indicate if the graph is a tree, and (xvi) a binary feature to indicate if the graph is bipartite.

For the classifiers in Section 5.2, features involving red edges are excluded since initial models are trained on input graphs without red edges. Red-edge features are included later in the twin-width solver (Section 5.3). We use standard graph metrics and include a feature set from the GL2Vec [24] graph neural network, which aggregates graphlet substructures into a fixed-length embedding. We use a 32-dimensional vector, as longer embeddings yield diminishing returns according to feature selection results.

5.2 Predicting Twin-Width

We frame these experiments as a supervised learning task, training separate models for each graph class to examine how feature correlations with twin-width vary across classes. For each graph class, graph-level features are extracted from labeled datasets, and results are reported on hold-out test sets. Features serve as input to Random Forests [23] and feedforward neural networks (later denoted MLP) [27], chosen for their availability and strong performance. Hyperparameters are tuned via grid search, with feature selection applied per model using Mutual Information [37] and scikit-learn's SequentialFeatureSelector [36]. Additionally, we tested a popular end-to-end GNN classifier based on graph convolutional networks [29] (GCN), but it compared poorly to models using hand-crafted features. Common GNN embeddings were still somewhat useful as an additive feature.

We also explored modeling this task as a regression problem. While regression minimized the "mean-squared error" loss function, it performed poorly in terms of accuracy due to rounding

errors, normalization issues, and slower training, so we focus on classification models instead.

5.3 Learning Approach for Generating Contraction Sequences

We generate contraction sequences using our twin-width predictor models by exploring a search tree of partial sequences, tracking the maximum red degree throughout each sequence. We start with sequences of length 0, corresponding to the original graph of size n; a complete solution is a sequence of length $n-1$, corresponding to a single-node graph. Partial sequences are stored in a priority queue, each linked to its partially contracted graph and historical maximum red degree. Of note: a single partially contracted graph can be linked to multiple partial contraction sequences, each potentially having a different historical maximum red degree.

When a partial contraction sequence is popped from the queue, the solver considers several candidate node pairs for contraction. Each candidate contraction produces a new partial sequence—its "child"—one pair longer, which has a corresponding partially contracted graph one node smaller. The ML model evaluates this graph's feature vector to assign a priority, which is then used to insert the child sequence back into the queue.

Priorities are defined as $\theta_{\text{prior}} = \hat{\theta} - \epsilon\,(\text{depth})^c$, where $\hat{\theta}$ is the predicted twin-width, depth the sequence length, and ϵ, c tuneable to favor solution quality or speed/early bounding.

This model uses all features, including red-edge ones, making it suitable for partially contracted graphs, unlike the prior model trained only on original graphs. It also expands the training data: each labeled graph yields n points corresponding to all partially contracted graphs along its optimal sequence. This generic order-based approach could be applied to other graph measures.

Optimizations. The following optimizations reduce running time by over 95% in many cases, allowing us to process instances up to four times larger.

- Branching-factor reduction via Jaccard filtering: Pre-select a subset of node pairs based on Jaccard coefficient.
- Solver restarts with pruning: After finding a sequence of width k, restart and prune any states with current red degree $\geq k$ to shrink the search space.
- Feature sampling: Approximate expensive features (e.g., diameter, clustering, densities) via random node samples or incremental computation from the prior partial contraction step.
- Exclude embedding: We drop GL2Vec features due to high cost; red-degree features help offset the loss in inference.

6 Experimental Results

Our evaluation consists of two parts. In Section 6.1 we assess direct prediction of twin-width without certificates. This serves as a stepping stone to the fully-fledged heuristic solver in Section 6.3, which relies on accurate classifiers to efficiently navigate the search space.

6.1 Evaluating Twin-Width Classifiers

We present our evaluation of classification models here. Regression experiments yielded similarly positive results, but since twin-widths are integers, we view the evaluation metrics for classification to be more relevant/interesting. In this direct prediction setting, extensive feature calculation enables the training of relatively complex models. Table 1 summarizes the strong performance of the trained classifiers across all datasets from Section 4, though the end-to-end GNN approach performs noticeably worse. Such models are popular in the literature for graph-based applications, but their inference is not easily explained. This also limits their utility in our setting, where we are interested in how features relate to the parameter of interest, in pursuit of a deeper understanding of the problem.

Table 1: Overall performance for twin-width predictions.

	RF		MLP		GNN	
Dataset / Graph Class	Acc.	F1	Acc.	F1	Acc.	F1
Perturbed Rook	0.869	0.851	0.871	0.856	0.514	0.442
Random Hyperbolic	0.761	0.736	0.727	0.704	0.724	0.516
Planar	0.907	0.777	0.887	0.678	0.822	0.180
$G(n,p)$	0.858	0.747	0.813	0.775	0.745	0.552
$G(n,p=0.5)$	0.821	0.715	0.825	0.717	0.259	0.166

Table 2: Confusion Matrix for $G(n,p)$ and Random Forests

True TW	Predicted TW								
	3	4	5	6	7	8	9	10	11
3	16	5	0	0	0	0	0	0	0
4	1	84	1	0	0	0	0	0	0
5	0	1	39	2	0	0	0	0	0
6	0	0	1	11	1	0	0	0	0
7	0	0	0	2	21	4	0	0	0
8	0	0	0	0	2	48	5	0	0
9	0	0	0	0	0	13	39	2	0
10	0	0	0	0	0	0	2	14	0
11	0	0	0	0	0	0	0	3	0

To gain a more fine-grained understanding of the classifier's performance on the most difficult-to-classify datasets, we provide a confusion matrix for Erdős–Rényi random graphs in Table 2. For all graph classes, there is a strong concentration along the diagonal. **In fact, across all datasets, the classifier accurately predicts the twin-width, or deviates by at most one.**

There is substantial overlap between features selected for the initial twin-width classifiers and their sampled counterparts used during solving. While evaluation metrics decline somewhat in the sampled case, these classifiers preserve solver performance and, due to large runtime gains, improve overall efficiency.

6.2 Feature Importance for Trained Classifiers

Studying feature importance is valuable both for solver efficiency and problem insight. Feature selection allows us to eliminate low-importance features and their runtime cost, reducing overhead during solving. Conversely, high-importance features potentially reveal connections between well-understood graph features and twin-width, which may lead to structural insights and further exploitation by hand-crafted algorithms. Different graph classes have different associated closed-form expected values or bounds for twin-width due to their underlying characteristics. As such, we train separate classifiers for each graph class to investigate these properties. Resultant feature importance scores vary across datasets, though graph size (nodes or edges) appears consistently. We outline the most pertinent results below.

We observe that for Erdős–Rényi random graphs, the number of edges is the most important feature, and together with the

Table 3: Comparison on G(n,p), Planar and RHG Graphs

Solver	$G(n,p)$		Planar		*RHG*	
	Opt	Time [s]	Opt	Time [s]	Opt	Time [s]
Greedy	1.65	300	1.72	300	1.78	300
ML	1.20	13.72	1.33	75.65	1.095	43.41
GUTHM	1.04	10	1.04	10	1.004	10

Table 4: Perturbed Rook Graphs and G(n,p=0.5)

Solver	Perturbed Rook Graphs		G(n,p=0.5)	
	Opt	Time [s]	Opt	Time [s]
Greedy	1.27	300	1.24	300
ML	1.13	65.81	1.058	108.1
GUTHM	1.05	10.0	1.06	10

correlated features of graph density and average degree centrality, it dominates the mutual information feature importance for this graph class. This is in line with the tight bounds obtained on twin-width for random graphs [1, 2]. The fact that our classifiers recover these known relationships provides strong evidence of their ability to identify structural correlates of twin-width. Results on perturbed rook graphs provide further evidence of this, where number of nodes, density, and average degree centrality correspond directly to the parameters defining the graphs and their resulting twin-widths.

6.3 Evaluating the Contraction Sequences

We now integrate our previous experimental insights into a learning-based heuristic that computes contraction sequences of small twin-width. Evaluation is conducted across all datasets from Section 4, comparing against a simple greedy solver and GUTHM, the winning heuristic from the PACE Challenge [33]. The greedy solver, implemented in Rust, merges the node pair with minimum red degree at each step. A randomized node order is generated for each greedy solver call to break ties, and this process is repeated until timeout. All solvers are given a 5-minute time limit, reporting their best solution within this window. This is the same as the evaluation framework for the PACE challenge heuristic track. GUTHM is designed to run until timeout, so its average solve time always matches the provided limit. While it often converges well before the full 300 seconds, it does not expose intermediate solution times. We experimented with shorter timeouts and could run solves for as short as 10 seconds without a significant drop in optimality ratio. This illustrates how quickly it converges to best-in-class solution quality. Tables 3 and 4 summarizes our findings.

The relative solver performance is consistent across datasets. While the PACE heuristic generally achieves the best optimality ratios, the performance gap between it and the ML solver narrows on graph classes with higher twin-widths. Notably, the ML solver yields slightly better optimality ratios on $G(n, p = 0.5)$ graphs. Conversely, despite the computational efficiency associated with its Rust implementation, the greedy solver never surpasses the ML solver in solution quality. However, regarding runtime, the ML solver is less competitive; the greedy approach can identify solutions comparable to its 300s benchmark in under one second. These results are to be expected, given that GUTHM was extensively optimized for both speed and quality in a competitive setting: it selects specialized solvers based on input characteristics, uses extensive look-ahead routines to score contractions, and employs advanced data structures and sampling techniques in the low-level language Zig. In contrast, our framework is generic, relying only on standard ML techniques and minimal domain knowledge. While slower, this design offers the flexibility to target various graph parameters beyond just twin-width.

Table 5: Treewidth Solver Results

Dataset	Full		Sampled	
	Opt Ratio	Time [s]	Opt Ratio	Time [s]
MinDegree	1.031	0.0002	1.031	0.0002
MinDegree-ML	1.031	30.6	1.033	7.04
MinFill	1.038	0.0004	1.038	0.0004
MinFill-ML	1.017	30.1	1.017	6.9
ML-select	1.354	22.2	1.315	5.2

7 Learning-Augmented Treewidth

To exhibit this flexibility, we extend our framework to treewidth, another ordering-based optimization problem. Using the same feature sets as for twin-width (excluding red-edge features), classifiers trained on perturbed k-tree data achieve strong performance, paralleling our twin-width results.

Beyond prediction, we also evaluate treewidth solvers guided by these classifiers. A pure ML method, *ML Select*, chooses contractions solely from classifier output, while two ML-augmented heuristics query the *ML Select* classifier to break ties in the *MinDegree* and *MinFill* heuristics. We compare these to the standard heuristics. Results for the treewidth solvers are displayed in Table 5, for solvers which use the sampled and full feature calculation (as in the twin-width setting) models respectively. *ML-select* is slow and produces relatively poor solutions. The ML-extension for *MinDegree* does not consistently improve optimality, but applying it to *MinFill* yields marked improvement, at the cost of increased runtime.

8 Discussion

We present a generic, learning-augmented framework for estimating the structural parameters that govern the complexity of network optimization problems. The first component of our approach —rapid parameter estimation— serves as a crucial filter for algorithm selection, allowing solvers to assess the tractability of specific network instances before committing computational resources. The second component —leveraging these estimates to generate certificates— bridges the gap between theoretical parameterized complexity and practical solver implementation.

This generation of certificates is particularly impactful for order-based parameters such as twin-width, treewidth, clique-width, and boxicity, where the certificate (e.g., a contraction sequence) effectively acts as the "blueprint" for dynamic programming. For parameters defined by subset-based certificates (e.g., vertex cover), techniques such as learning-to-prune [41], are highly effective.

To the best of our knowledge, this work represents the first successful application of machine learning to estimate fundamental, computationally challenging graph parameters like twin-width and treewidth. By automating the discovery of these structural decompositions, we remove a primary bottleneck in deploying structure-aware algorithms for large-scale networks. We foresee significant potential in integrating advanced ML paradigms, such as tree-search-based Reinforcement Learning, to further refine the quality of these decompositions. Such methods, combined

with learned evaluation functions, could produce certificates that minimize the downstream runtime of optimization solvers.

Finally, we provide our labeled datasets, representing a substantial CPU investment, as a foundational resource for the community. These benchmarks are essential for training the next generation of data-driven heuristics. Notably, the inclusion of the perturbed rook-graph family introduces a novel, high twin-width benchmark.

Acknowledgments. The research of the first and last author is supported in part by a grant from Science Foundation Ireland (Grant number 18/CRT/6183). For the purpose of Open Access, the authors have applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission. This work was partially supported by the Deutsche Forschungsgemeinschaft (DFG) – ME 2088/5-2 (FOR 2975), and the Novo Nordisk Foundation grants NNF21OC0066551.

References

[1] Jungho Ahn, Debsoumya Chakraborti, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. 2022. Twin-width of random graphs. *CoRR* (2022).
[2] Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. 2022. Bounds for the Twin-Width of Graphs. *SIAM J. Discret. Math.* (2022).
[3] Jungho Ahn, Hugo Jacob, Noleen Köhler, Christophe Paul, Amadeus Reinald, and Sebastian Wiederrecht. 2025. Twin-width one. arXiv:2501.00991
[4] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. 1987. Complexity of Finding Embeddings in a k-Tree. *SIAM J. on Algebraic Discrete Methods* (1987).
[5] Emmanuel Arrighi, Pål Grønås Drange, Kenneth Langedal, Farhad Vadiee, Martin Vatshelle, and Petra Wolf. 2023. PACE Solver: Zygosity. In *IPEC*, Vol. 285. 39:1–39:3.
[6] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. 2018. Learning to Branch. In *ICML*, Vol. 80. 344–353.
[7] Max Bannach and Sebastian Berndt. 2023. The PACE 2023 Challenge: Twin-width. In *IPEC*.
[8] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. *EJOR* (2021).
[9] Pierre Bergé, Édouard Bonnet, and Hugues Déprés. 2022. Deciding Twin-Width at Most 4 Is NP-Complete. In *ICALP*.
[10] Benjamin Bergougnoux, Jakub Gajarskỳ, Grzegorz Guśpiel, Petr Hliněnỳ, Filip Pokrỳvka, and Marek Sokołowski. 2023. Sparse graphs of twin-width 2 have bounded tree-width. *arXiv preprint arXiv:2307.01732* (2023).
[11] Gaétan Berthe, Yoann Coudert-Osmont, Alexander Dobler, Laure Morelle, Amadeus Reinald, and Mathis Rocton. 2023. PACE Solver: Touiouidth. In *IPEC*, Vol. 285. 38:1–38:4.
[12] Hans L. Bodlaender. 1996. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.* (1996).
[13] Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. 2022. Twin-Width VIII: Delineation and Win-Wins. In *IPEC*.
[14] Édouard Bonnet and Julien Duron. 2023. PACE Solver: RedAlert - Heuristic Track. In *IPEC*.
[15] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. 2022. Twin-width II: small classes. *Comb. Theory* (2022).
[16] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. 2024. Twin-Width III: Max Independent Set, Min Dominating Set, and Coloring. *SIAM J. Comput.* (2024).
[17] Édouard Bonnet, Colin Geniet, Romain Tessera, and Stéphan Thomassé. 2022. Twin-width VII: groups. *CoRR* (2022). arXiv:2204.12330
[18] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. 2024. Twin-Width IV: Ordered Graphs and Matrices. *J. ACM* (2024).
[19] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. 2023. Twin-Width V: Linear Minors, Modular Counting, and Matrix Multiplication. In *40th Int. Symp. on Theoretical Aspects of Computer Science, STACS*.
[20] Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. 2022. Twin-width VI: the lens of contraction sequences. In *SODA*.
[21] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. 2022. Twin-width I: Tractable FO Model Checking. *J. ACM* (2022).
[22] Vincent Bouchitté and Ioan Todinca. 2002. Listing all potential maximal cliques of a graph. *Theoretical Computer Science* (2002).
[23] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.
[24] Hong Chen and Hisashi Koga. 2019. Gl2vec: Graph embedding enriched by line graphs with edge features. In *ICONIP*.
[25] Bruno Courcelle. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation* (1990).
[26] James Fitzpatrick, Deepak Ajwani, and Paula Carroll. 2024. A scalable learning approach for the capacitated vehicle routing problem. *Computers & Operations Research* (2024).
[27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.
[28] Tingfei Huang, Yang Ma, Yuzhen Zhou, Honglan Huang, Dongmei Chen, Zidan Gong, and Yao Liu. 2019. A Review of combinatorial optimization with graph neural networks. In *BigDIA*.
[29] TN Kipf. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[30] Wouter Kool, Herke Van Hoof, and Max Welling. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475* (2018).
[31] Kenneth Langedal, Johannes Langguth, Fredrik Manne, and Daniel Thilo Schroeder. 2022. Efficient Minimum Weight Vertex Cover Heuristics Using Graph Neural Networks. In *20th Int. Symp. on Experimental Algorithms, SEA*.
[32] Juho Lauri, Sourav Dutta, Marco Grassia, and Deepak Ajwani. 2023. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *J. of Heuristics* (2023).
[33] Alexander Leonhardt, Holger Dell, Anselm Haak, Frank Kammer, Johannes Meintrup, Ulrich Meyer, and Manuel Penschuck. 2023. PACE Solver: Exact (GUTHMI) and Heuristic (GUTHM). In *IPEC*.
[34] Yosuke Mizutani, David Dursteler, and Blair D. Sullivan. 2023. PACE Solver: Hydra Prime. In *IPEC*.
[35] Ryan O'Connor, Johannes Meintrup, Maximilian Huber, Alexander Leonhardt, Manuel Penschuck, Yosuke Mizutani, Oscar Yeoh, and Deepak. 2026. Collection of graphs including optimal twin-width contraction sequences. https://doi.org/10.5281/zenodo.18886833
[36] Fabian Pedregosa et al. 2011. Scikit-learn: Machine learning in Python. *JMLR* (2011).
[37] Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence* 27, 8 (2005), 1226–1238.
[38] Yun Peng, Byron Choi, and Jianliang Xu. 2021. Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art. *Data Science and Engineering* (2021).
[39] André Schidler and Stefan Szeider. 2022. A SAT approach to twin-width. In *(ALENEX)*.
[40] Hisao Tamaki. 2019. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.* (2019).
[41] Dena Tayebi, Saurabh Ray, and Deepak Ajwani. 2022. Learning to Prune Instances of k-median and Related Problems. In *ALENEX*.
[42] Frank van den Eijkhof, Hans L. Bodlaender, and Arie M. C. A. Koster. 2007. Safe Reduction Rules for Weighted Treewidth. *Algorithmica* (2007).
[43] Junchi Yan, Shuang Yang, and Edwin Hancock. 2020. Learning for Graph Matching and Related Combinatorial Optimization Problems. In *IJCAI*.
[44] Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. 2021. Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies. In *AAAI*.