Taming Treewidth DP with Modulators: A General Booster for Graph Heuristics

Jialiang Li[1] , **Aneta Neumann**[1] , **Frank Neumann**[1] , **Hung Nguyen**[1] and **Mingyu Guo**[1]
[1]Adelaide University
{j.li, aneta.neumann, frank.neumann, hung.nguyen, mingyu.guo}@adelaide.edu.au

Abstract

Treewidth is a fundamental graph invariant that quantifies how tree-like a given graph is. It is extensively used with dynamic programming to design *fixed-parameter tractable* algorithms for many NP-hard graph combinatorial optimization problems. However, despite broad theoretical applicability, treewidth dynamic programming (TDP) does not scale in practice beyond graphs with very small treewidth. Rather than applying TDP as a standalone technique, in this paper, we demonstrate that TDP can serve as a broadly applicable *enhancer* for a wide range of graph combinatorial optimization algorithms. Our framework leverages the concept of *treewidth modulators*, which refer to vertex sets whose removal significantly reduces the treewidth. We further propose an empirically efficient procedure for generating such treewidth modulators. To enhance an algorithm $\mathcal{A}$, we use $\mathcal{A}$ to heuristically make decisions on the modulators vertices, after which the remaining decisions outside the treewidth modulators become scalable for TDP.

To demonstrate the general applicability of our proposed framework. We experimented with three classic graph combinatorial optimization models: *Maximum Independent Set*, *Minimum Vertex Cover*, and *Max Cut*. We apply TDP to enhance algorithms across diverse paradigms, including *evolutionary search*, *greedy heuristics*, and *graph-neural-network-based heuristics*. For all combinations of optimization models and base algorithms, TDP significantly improves performance over the original methods. In many settings, TDP-enhanced greedy heuristics are *competitive with*, and *sometimes clearly outperform*, state-of-the-art commercial solvers.

1 Introduction

Treewidth (TW) is a prominent graph invariant that measures the similarity between a given graph and an exact tree, whose value can be calculated via a *tree decomposition* process. Many NP-hard graph combinatorial optimization problems become easy for special graph instances with small treewidths. According to the well-known *Courcelle's Theorem* [Courcelle, 1990], there is a general way to construct *treewidth dynamic programs (TDP)* [Downey *et al.*, 2013] that can optimally solve a wide spectrum of hard graph problems in *linear time* for graphs with bounded treewidths. These problems include but are not limited to *independent set*, *vertex cover*, *3-coloring*, *TSP* and etc [Bodlaender, 1988]. These problems are referred to as *fixed-parameter tractable* parameterized by treewidth. Their time complexities are of the form $O(f(TW) \cdot n)$, where $f(TW)$ is a constant factor generally exponential in TW when it comes to NP-hard problems.[1] That is, despite its general applicability and the impressive "linear" time complexity, TDP is practically **not** scalable, unless we restrict to special graph instances with tiny treewidths.

We take the *Maximum Independent Set (MIS)* as an example. There exists a treewidth dynamic program with complexity $O(2^{TW}n)$. Going by real-world graphs [Maniu *et al.*, 2019], for a power network with treewidth 10, the above TDP is certainly scalable. For a road network with treewidth 50, the above TDP becomes not scalable due to the large constant factor 2^{50}. Again, it is *expected* that TDP is not scalable, as MIS is, after all, an NP-hard problem.

In summary, the advantage of TDP is its broad applicability and its disadvantage is its limited practically scalability. Instead of applying TDP as a standalone technique, we propose to apply TDP to enhance existing graph optimization algorithms. We employ the concept of *treewidth modulator*. It is a vertex deletion problem [Lewis and Yannakakis, 1980], with the goal of deleting as few vertices as possible to bring down the treewidth to below a desired threshold (i.e., to a threshold that makes TDP scalable). Our usage of treewidth modulator certainly does not involve actually deleting any vertices, as otherwise that would be modifying the original optimization model and/or the solution space. We use the treewidth modulator as a *backdoor* [Williams *et al.*, 2003], which is a concept borrowed from SAT (Boolean satisfiability). For SAT, it is known that *hard instances can become much easier by fixating assignments to a small subset of variables*. The assignments for reducing computational difficulty are called the backdoor. We apply TDP to enhance an existing graph optimization algorithm $\mathcal{A}$ as follows. (Our approach is broadly applicable

[1]For $O(f(TW) \cdot n)$, if f is polynomial in TW, then the whole expression becomes polynomial in n as $TW < n$.

to many optimization models, which is why our presentation intentionally avoids referencing a specific model. The readers are welcome to adopt *Maximum Independent Set* as an example, if it aids in understanding.) We use $\mathcal{A}$ to make decisions within the treewidth modulator. We refer to these decisions as the *advice string*, which serves to significantly reduce the TDP search space (i.e., it is the "backdoor" to TDP). We propose a modified version of TDP that takes as input an advice string, and focuses on generating the decisions outside of the treewidth modulator, which is scalable (as it is effectively equivalent to working on the remaining graph after vertex deletion). Essentially, our modified version of TDP produces the *optimal solution* **conditional on** *the advice string*. The above interpretation also shows that for any existing algorithm $\mathcal{A}$, the TDP enhanced version of $\mathcal{A}$ never performs worse than $\mathcal{A}$ itself in terms of solution quality.

1.1 Summary of contributions

- We propose a general framework on using TDP to enhance existing graph combinatorial optimization algorithms. We experimented on three different graph optimization models, including *Maximum Independent Set*, *Minimum Vertex Cover*, and *Max Cut*. We apply TDP to enhance algorithms following a variety of paradigms, including *(1+1)EA evolutionary algorithm*, *greedy*, and *graph neural networks based heuristics*. For all combinations of optimization models and existing algorithms, TDP significantly enhances the existing algorithms.
- Our framework builds on the efficient generation of treewidth modulators. Optimal treewidth modulator generation is known to be notoriously difficult. We propose a scalable variant called *Optimal Treewidth Modulator on a Given Tree Decomposition*. We prove that our variant is still NP-hard, but it is empirically cheap, costing only 1 second on average in our experiments.
- For easy graph datasets, TDP enhanced (1+1)EA almost always reaches the optimal solution, and it is significantly superior to standalone (1+1)EA in terms of wall-clock time and number of iterations before hitting optimality.
- In many scenarios with hard graph datasets, TDP enhanced greedy heuristics are *competitive with*, and *sometimes clearly outperform*, the solutions by GUROBI. Our *pure Python* implementation manages to reach *better solutions* while using *less time* than GUROBI.
- TDP enhanced graph neural networks based heuristics outperform standalone graph neural networks based heuristics. Furthermore, we observe the following common phenomenon in training. The network quickly stagnates on decisions within the treewidth modulator. Effectively, we have already entered into a local optimum. Afterwards, there is a prolonged period of minor improvements on decisions outside the treewidth modulator. That is, judging by the cost function, it is decreasing and therefore *appears to be learning*. However, the decisions outside the treewidth modulator are not worth the training efforts as they can be optimally and efficiently derived via TDP. In other words, our framework can function as an early detection test to prevent such wasteful training.

Related Work. Treewidth modulator is related to the *forbidden minor* problem, which has been extensively studied in the parameterized complexity community [Cygan *et al.*, 2011; Baste *et al.*, 2020; Jansen *et al.*, 2021]. To the best of our knowledge, efficient algorithms for acquiring optimal treewidth modulators appear highly improbable by many conjectures [Baste *et al.*, 2020]. Even the fixed-parameter tractable results are based on the precondition of *finite graph collections*, but yet, exhibit a *superexponential* growth rate. The progress on treewidth modulators serves more for theoretical interests. In this paper, we do not pursue the exact calculation of treewidth modulators. Instead, we adopt a more practical approach by proposing an empirically scalable heuristic. Despite the resemblance between treewidth modulator and *strong backdoor*, the difference has already been clarified in [Fomin *et al.*, 2015]. In terms of algorithm design, other research focused on parameterized approximation scheme [Lampis, 2013], which shares the same interest on accelerating treewidth dynamic programming, but via algebraic operations and non-trivial hand-crafted rules.

2 Preliminary

In this paper, we follow the standard notations and terminologies in graph theory and parameterized complexity. Contents marked by ($\bigstar$) are deferred to the appendix.

Notations. Given a graph $G = (V, E)$, we note $|V| = n$, $|E| = m$. We define the induced subgraph by a subset $S \subseteq V$ as $G[S]$. The neighbors of a vertex or a set of vertices are denoted by $N[\cdot]$ (inclusion-wise) and $N(\cdot)$ (exclusion-wise). We define the *boundary* of S as $\partial_G(S) = \{v \in S : \exists u \in V \setminus S, (u, v) \in E\}$. We focus on vertex selection problems, which involve searching for a solution within the binary combinatorial space 2^n. We represent an arbitrary *assignment* within the space as $\boldsymbol{s} \in 2^n$. Unless otherwise specified, we assume *maximization* objective, such as maximizing the number of vertices selected subject to feasibility.

We start by introducing a few relevant concepts from parameterized complexity:

Definition 1 (Fixed-parameter Tractable). *For a problem $L \subseteq \Sigma^* \times \mathbb{N}$, if there exists an algorithm $\mathcal{A}$ that can determine whether an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ in time $f(k) \cdot |x|^{O(1)}$, where f is a computable function, then the problem is fixed-parameter tractable (FPT) parameterized by k.*

In our context, we focusing on using the treewidth (TW) as the parameter, which is formally defined below. We say a graph combinatorial optimization problem is FPT parameterized by TW if we have an algorithm with complexity $O(f(TW) \cdot n^{O(1)})$. That is, if the graph instances have bounded treewidths, then the algorithm is polynomial-time.

Definition 2 (Tree Decomposition [Cygan *et al.*, 2020]). *A tree decomposition is a mapping from G to a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree and every tree node $t \in V(T)$ corresponds to a subset of vertices from the original graph, denoted as $X_t \subseteq V(G)$. X_t is often referred to as a* **bag**. *For valid tree decomposition, we must have:*

- *The union of all bags contains all the vertices from the original graph: $\bigcup_{t \in V(T)} X_t = V(G)$;*

- *For any edge (u, v) in the original graph, there must exist at least one bag that contains both u and v: $\forall(u, v) \in E(G)$, $\exists t \in T$ with $\{u, v\} \subseteq X_t$;*
- *For any vertex u from the original graph, all bags containing u must form a subtree of T: $\forall u \in V(G), T_u = \{t \in V(T) | u \in X_t\}$ is a connected component of T.*

The **width** *of a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ is defined by*

$$\max_{t \in V(T)} |X_t| - 1$$

The optimal **treewidth** *of a graph G is denoted by TW*, which is the minimum width of all valid tree decompositions of G. We denote the tree decomposition producing the minimum treewidth as $\mathcal{T}^*$.*

It should be noted that the computation of the minimum treewidth is proven NP-hard [Robertson and Seymour, 1984], hard to approximate [Wu *et al.*, 2014], and hard even for special graph classes [Bodlaender *et al.*, 2023]. For practical purposes, treewidth dynamic programming does not require the minimum treewidth. It can be based on sub-optimal treewidth values obtained via polynomial-time tree decomposition heuristics. In our presentation, treewidth refers to a heuristically generated value.

The following theorem proves that bounded treewidth implies linear-time algorithms for many problems.

Theorem 1 (Courcelle's Theorem [Courcelle, 1990]). *Given a monadic second order logic (MSO) formula φ, and a graph instance G. Then, there exists a linear-time algorithm (in time $f(|\varphi|, k)$) to decide if G satisfies φ if TW(G) is bounded by k.*

Problems such as *3-coloring*, *Hamiltonian cycle*, *independent set*, and *vertex cover* are all expressible by MSO. According to the above theorem, they all are linear-time solvable for graph instances with bounded treewidths. There is also a systematic way to construct treewidth dynamic programs for these problems. Theoretically, this sounds nice, but practically, direct application of TDP is often not scalable due to the inevitable exponential constant factor in the treewidth. We apply TDP as a broadly applicable tool to enhance existing graph optimization algorithms. Our framework employs a similar concept to treewidth modulator.

Definition 3 (Treewidth Modulator [Cygan *et al.*, 2011]). *Let $\eta \geq 0$ be an integer and G be a graph. A set $\mathcal{M}_\eta \subset V(G)$ is called an η-treewidth modulator in G if $TW^*(G \setminus \mathcal{M}_\eta) \leq \eta$. We use $\mathcal{M}^*_\eta(G)$ to denote the optimal treewidth modulator to a target width value η with the minimum size (i.e., we want to minimize the number of vertices deleted $|\mathcal{M}_\eta|$).*

We use $\mathcal{M}^*$ to denote the task of generating the optimal treewidth modulator $\mathcal{M}^*_\eta(G)$, which is NP-hard due to its hereditary property [Lewis and Yannakakis, 1980]. Its hardness can also be easily illustrated via the following example.

Example 1. *When $\eta = 1$, $\mathcal{M}^*$ is equivalent to feedback vertex set. When $\eta = 0$, $\mathcal{M}^*$ is equivalent to vertex cover.*

Despite a long series of existing theoretical works on $\mathcal{M}^*$, such as [Baste *et al.*, 2020], there do not exist scalable exact algorithms, approximation algorithms, or even practically useful bounds. In this paper, we propose a variant of $\mathcal{M}^*$ called *Treewidth Modulator on a Given Tree Decomposition*.[2] Our variant was designed for *practical scalability* – it is still NP-hard, but its exact solution takes only 1 second on average in our experiments.

Definition 4. ***(Treewidth Modulator on a Given Tree Decomposition)*** *Given a graph G and a tree decomposition of G denoted as $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ and a parameter k. We simply delete vertices from the existing bags (the X_t) and do not regenerate the tree structure T. That is, after deleting $Y \subseteq V(G)$, the resulting tree decomposition is $\mathcal{T}' = (T, \{X_t \setminus Y\}_{t \in V(T)})$ and the resulting treewidth is $\max_{t \in V(T)} |X_t \setminus Y| - 1$. We want to decide whether there exists a vertex set $Y \subseteq V(G), |Y| \leq k$, whose deletion will reduce the width of $\mathcal{T}'$ to at most η.*

We use $\mathcal{M}$ to denote the above variant.[3] The idea behind it is simple. Instead of regenerating tree decomposition after vertex deletion, we reuse the same tree decomposition by simply removing vertices from its existing bags. We first show that the deleting vertices from a tree decomposition's existing bags indeed results in a valid tree decomposition.

Proposition 1 (★). *Given a graph G and a tree decomposition of G denoted as $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree and every tree node $t \in V(T)$ corresponds to a subset of vertices from the original graph, denoted as $X_t \subseteq V(G)$.*

For any set of vertices $Y \subseteq V(G)$, after deleting Y from $\mathcal{T}$ without regenerating the tree decomposition, the result $\mathcal{T}' = (T, \{X_t \setminus Y\}_{t \in V(T)})$ is still a valid tree decomposition.

Next, we claim that even though $\mathcal{M}$ avoids regenerating tree decompositions, it remains NP-complete.

Theorem 2 (★). *Treewidth modulator on a given tree decomposition is* NP-COMPLETE.

In our experiments, we use the polynomial-time MIN-DEGREE heuristic to generate the initial tree decomposition and apply mixed integer programming to remove the minimum set of vertices from the initial tree decomposition's existing bags (★). The above is a fast heuristic for $\mathcal{M}$.

3 Technical Description of Framework

We start by describing our *core routine*, which applies TDP to enhance an existing algorithm specified as a subroutine. We describe the *(1+1)EA evolutionary algorithm* subroutine and the *ϵ-greedy* subroutine in this section. We defer the *graph neural networks (GNN) based heuristics* to the experiments. In our presentation, we use P to denote the optimization problem with solution space 2^n. Without loss of generality, we assume that P has a *maximization* objective.

Core Routine. There are no known efficient algorithms for generating the optimal treewidth modulator. Our more scalable variant of treewidth modulator on a given tree decomposition ($\mathcal{M}$) offers an empirically scalable way to generate

[2] Our proxy problem is on deleting vertices. A variant on deleting edges from a given tree decomposition has been previously studied in the field of bioinformatics [Marchand *et al.*, 2021].

[3] We abuse $\mathcal{M}$ for both treewidth modulator and the problem of computing treewidth modulator.

a treewidth modulator that suffices for our TDP application. While treewidth modulators are defined as vertex sets whose deletion would reduce the treewidth of the remaining graph, when we apply a treewidth modulator, we do not *actually delete* any vertices, but we rely on the existing algorithm $\mathcal{A}$ that we aim to enhance (i.e., the subroutine we plug in) to assign values to $v \in \mathcal{M}$ from a search space of $2^{|\mathcal{M}|}$. The assignments are referred to as $\mathcal{A}$'s *advice*, denoted by s. Via TDP, we can efficiently derive the optimal overall solution *conditional* on advice s. To illustrate this idea, imagine that we are processing a specific bag $\mathcal{B}$ from the tree decomposition. Typically, TDP involves enumerating all feasible states/assignments of $\mathcal{B}$. For example, suppose $|\mathcal{B}| = 4$. Our initial understanding of the optimal assignments can be described as the string `<?,?,?,?>`, representing that the optimal assignments are completely unknown. TDP needs to enumerate 2^4 states. Suppose an *advice* is provided, we can settle some of the ?s. Assume $\mathcal{A}$ advised that the first coordinate must be 1 and the third coordinate must be 0, i.e., our understanding of the optimal assignments becomes `<1,?,0,?>`. Consequently, we only have 2^2 states to enumerate. Thus, given an advice string describing the decisions within the treewidth modulator, TDP is effectively focusing on the decisions outside the treewidth modulator, as if the treewidth modulator vertices have been deleted. Our core routine's pseudo code is provided as follows. We follow the standard description of treewidth dynamic program,[4] only slightly modified to accommodate advice.

Algorithm 1 Treewidth Dynamic Programming with *advice*

Input: $G = (V, E)$, $\mathcal{T}_w = (T, \{X_t\}_{t \in V(T)})$: an arbitrary tree decomposition with width w, $\mathcal{M}_\eta$: a modulator to $\mathcal{T}_w$ with target width η, $\boldsymbol{s} \subseteq \mathcal{M}$: an *advice string* (subset) given by external subroutine $\mathcal{A}$, P: a maximization problem
Output: OPT to P

Let C, D be two tables.
for $t \in V(\mathcal{T})$ in a *bottom-up* manner **do**
  $\mathcal{X}_p \leftarrow \{k | k \in V(T)$ and $k \in$ Parent$(t)\}$
  $\mathcal{X}_c \leftarrow \{j | j \in V(T)$ and $j \in$ Children$(t)\}$
  Apply advice here
  $\mathcal{F}_t \leftarrow \{x|\ x \subseteq X_t \setminus \boldsymbol{s}$ and P(x) is satisfied$\}$
  for $x \in \mathcal{F}_t$ **do**
    Complete state with advice
    $x \leftarrow x \cup \boldsymbol{s}$
    Merge states from children
    $C(t, x \cap X_j) = |x| + \sum_{j \in \mathcal{X}_c} (D(j, t, x \cap X_j) - |x \cap X_j|)$
    Upload states to parent
    $D(t, k, x \cap X_k) = \max_{k \in \mathcal{X}_p} C(t, x \cap X_k)$
$OPT \leftarrow \max_{x \in \mathcal{F}_{root}} C(root, x)$
return OPT

(1+1)EA Evolutionary Algorithm Subroutine. We use (1+1)EA to search over advice assignments on the modulator $\mathcal{M}$. The algorithm maintain an incumbent $x \in \{0,1\}^{|\mathcal{M}|}$. Each iteration generates an offspring y by mutating x: for every coordinate i, the bit x_i is flipped independently with probability $1/|\mathcal{M}|$. The selection rule is elitist: y replaces x if it achieves at least the same fitness value.

The fitness function f is defined through conditional optimization. For an advice vector x, we fix the decisions of vertices in $\mathcal{M}$ according to x, and compute an optimal completion over $V \setminus \mathcal{M}$ that inherits these fixed decisions. The resulting objective value is $f(x)$. This evaluation is performed by the treewidth DP with advice (Alg. 1), which makes fitness computation scalable after modulator restriction. The complete (1+1)EA procedure is given in Alg. 2.

Algorithm 2 (1+1)EA Subroutine

Input: modulator $\mathcal{M} \subseteq V$ with $|\mathcal{M}| = m$; fitness function $f : \{0,1\}^m \to \mathbb{R}$; iteration (or time) budget B
Output: the best advice $x \in \{0,1\}^m$ found

Sample $x \sim \mathrm{Unif}(\{0,1\}^m)$; $f_x \leftarrow f(x)$
for $\ell = 1$ **to** B **do**
  $y \leftarrow x$
  for $i = 1$ **to** m **do**
    if $\mathrm{Bernoulli}(1/m) = 1$ **then**
      $y_i \leftarrow 1 - y_i$
  $f_y \leftarrow f(y)$
  if $f_y \geq f_x$ **then**
    $x \leftarrow y$; $f_x \leftarrow f_y$
return x

ϵ-greedy with Voting Subroutine. The second class of algorithms we investigate is *greedy*. For each problem P, we instantiate a problem-specific greedy heuristic $\mathcal{H}_p$: *max-degree* for MVC, *vertex-flipping* for MC, and *min-degree* as an effective heuristic for MIS.

We augment $\mathcal{H}_p$ with an *exploration* rate ϵ, interpreted as the probability of taking a non-greedy step rather than the greedy one. Algorithm 3 runs $\mathcal{H}_p(G, \epsilon)$ for R trails to generate solutions $x^{(r)}$, then converts each trail into an advice bitvector $a^{(r)} \in \{0,1\}^m$ by restricting to the modulator , i.e., $a^{(r)} \leftarrow x^{(r)}\ |_{\mathcal{M}}$; each trail is scores by $\alpha^{(r)} \leftarrow obj_P(x^{(r)})$. The final advice is selected by either: 1. WTA: choose $r^* \in \arg\max_{r \in [R]} \alpha^{(r)}$ and output $a^{(r)}$. 2. Maj: for each $i \in [m]$, set $a_i = 1 \iff \left|\left\{r \in [R] \mid a_i^{(r)} = 1\right\}\right| > R/2$. In this way, WTA trusts the single best run under obj_P, while Maj stabilizes the modulator decisions through aggregation over repeated ϵ-greedy trails.

Correctness. TDPA conditions the dynamic program on an advice assignment over the treewidth modulator $\mathcal{M}$. The key correctness requirement is that this conditioning must neither exclude feasible completions that remain consistent with the advice nor admit infeasible solutions. The required consistency mechanism is problem dependent, and we use MC versus MIS/MVC to illustrate the difference.

For MC, the advice only fixes decisions internal to $\mathcal{M}$ and does not impose constraints on choices outside $\mathcal{M}$. Consequently, committing to any assignment on $\mathcal{M}$ cannot introduce

[4] Please refer to Chapter 7.1-7.3 of [Cygan *et al.*, 2020] for a detailed description of treewidth dynamic programming.

Algorithm 3 ϵ-Greedy with Voting Subroutine

Input: modulator $\mathcal{M}$ with $|\mathcal{M}| = m$; heuristic $\mathcal{H}_P$; exploration rate ϵ; trials R; rule $\mathsf{Vote} \in \{\mathsf{WTA}, \mathsf{Maj}\}$
Output: advice vector $a \in \{0,1\}^m$

for $r = 1$ **to** R **do**
  $x^{(r)} \leftarrow \mathcal{H}_P(G, \epsilon)$
  $a^{(r)} \leftarrow x^{(r)}|_{\mathcal{M}} \in \{0,1\}^m$
  $\alpha^{(r)} \leftarrow \mathrm{obj}_P(x^{(r)})$
if $\mathsf{Vote} = \mathsf{WTA}$ **then**
  $r^\star \leftarrow \arg\max_{r \in [R]} \alpha^{(r)}$
  return $a^{(r^\star)}$
else
  Initialize $a \leftarrow \mathbf{0} \in \{0,1\}^m$
  for $i = 1$ **to** m **do**
    if $|\{r \in [R] \mid a_i^{(r)} = 1\}| > R/2$ **then**
      $a_i \leftarrow 1$
  return a

cross-boundary conflicts, and the DP over the remainder of the graph remains an exact optimization over all feasible completions.

Fro MIS and MVC, in contrast, decisions on $\mathcal{M}$ necessarily constrain vertices outside $\mathcal{M}$ through adjacency. We therefore enforce boundary consistency via $\partial_{\mathcal{M}}$, the set of vertices in $\mathcal{M}$ that have neighbors outside $\mathcal{M}$. For MIS, selecting $v \in \partial_{\mathcal{M}}$ forces all neighbors in $N(v)$ to be non-selectable. For MVC, excluding $v \in \partial_{\mathcal{M}}$ forces all neighbors $v \in N(v)$ to be included. These implications ensure that the DP enumerates exactly the feasible completions consistent with the advice in the next stage.

Finally, for randomized algorithms like (1+1)EA, the advice itself maybe locally infeasible within $\mathcal{M}$ (e.g., selecting adjacent vertices for MIS). Such conflicts are resolved by a simple randomized repair step that restores feasibility on $\mathcal{M}$ before invoking TDPA.

4 Experiments

We evaluate on *Maximum Independent Set* (MIS), *Minimum Vertex Cover* (MVC), and *Max Cut* (MC). We apply TDP to enhance *(1+1)EA*, *greedy*, and *graph neural networks*. For every combination of optimization models and existing algorithms, our TDP enhanced versions significantly improve over the original algorithms. In many scenarios, TDP-enhanced greedy achieve solutions that are comparable to, and sometimes clearly outperform, those obtained by GUROBI, a state-of-the-art commercial solver. It is worth emphasizing that while occasionally surpassing GUROBI is an *impressive achievement, especially considering that our implementation is pure Python*, our primary experimental goal is to demonstrate that TDP can be used as a broadly applicable enhancement tool.

Dataset. We construct two benchmark sets, EASY and HARD, based on treewidth. EASY consists of real-world graphs, e.g., from *fluid dynamics*, *road networking* and others that typically have relatively low treewidth. HARD contains synthetic instances generated from the frequently cited models [Ahn *et al.*, 2020; Sun and Yang, 2023] *Erdős–R'enyi* (ER) and *Barab'asi–Albert* (BA) models, configured to yield moderately large treewidth so that commercial solvers such as GUROBI struggle to obtain high-quality solutions within a limited time budget; we generate *100* instances per model. We exclude near-clique graphs (clique number close to $|V|$), since they usually have treewidth linear to $|V|$ and are thus uninformative for treewidth dynamic programming. Full instance statistics and sources are deferred (★) to the appendix.

Baselines and Configurations. We evaluate baseline algorithms with the TDPA-enhanced counterparts under our framework. As a reference point, we include GUROBI, which typically yields strong feasible solutions for combinatorial optimization even when it does not complete the optimality proof. To ensure a fair comparison, we restrict GUROBI to a single thread, consistent with all other methods in this paper. We report results at checkpoints of $\{30, 60, 300, 600\}$ seconds under two settings: the default configuration and `MIPFocus=1`, which prioritizes finding high-quality feasible solutions.

For heuristic baselines, we integrate evolutionary algorithm, greedy heuristics and GNN models into our framework, demonstrating broad composability across various paradigms.

Greedy heuristics. We tune ϵ in the range of $\{0, 0.01, 0.05\}$ as exploration ratio on both MIS and MVC. For MC, we use vertex-flipping, here ϵ controls the refinement budget (number of iterations), and we set it in $\{0, 1000, 5000\}$. Details (★) are deferred to appendix.

GNN baselines. We train a simple GNN (★) by unsupervised minimization of the QUBO objective (★). Given GNN advice, TDPA computes the optimal completion consistent with that advice.

All methods have 10 runs with 10 different seeds per instance; additional details can be found in appendix.

Decompositions and Modulators. Prior results [Bannach *et al.*, 2017; Maniu *et al.*, 2019] indicate that *min-degree* (★) heuristic often produces *near-optimal* treewidth in practice and remains competitive with more sophisticated treewidth algorithms. We therefore apply this heuristic to construct initial tree decompositions. And we defer the details of this heuristic to appendix. For the modulator $\mathcal{M}$, we formulate the computation as a mixed-integer programming (★), since in practice the MIP takes about *one second on average* to obtain $\mathcal{M}$. Although the optimality gap between $\mathcal{M}$ and $\mathcal{M}^*$ can in principle be large, tightening $|\mathcal{M}|$ is not our focus; any reduction in $|\mathcal{M}|$ *will only* strengthen our results.

4.1 Results & Analysis

On Randomized Algorithm. Across all instances and objectives (MC/MIS maximization and MVC minimization), Figure 1 highlights two consistent effects: incorporating $\mathcal{M}$ reduces stagnation, and smaller modulators typically yield stronger gains. We denote TM_τ as the modulator computed by $\mathcal{M}$ at target treewidth τ (larger $\tau \Rightarrow$ smaller $|TM_\tau|$). We vary the target treewidth $\tau \in \{5, 7, 9\}$; as the legends indicate, a larger τ produces a smaller modulator, e.g., $|TM_5| > |TM_7| > |TM_9|$. Under a much smaller budget (3,000 vs. 1 million iterations), the modulator-guided runs reach high-quality solutions much earlier: the annotated first-hitting times occur sooner and the trajectories improve more

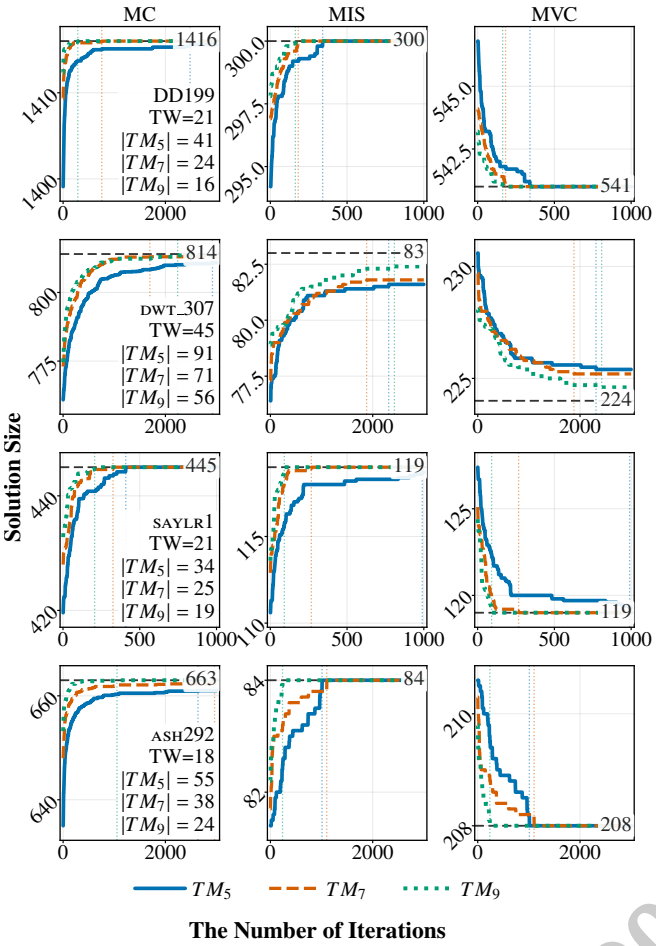

Figure 1: Colored vertical lines mark the *average first hitting time — the first moment the best solution appears for each curve*. The - - -indicates the GUROBI solutions. For presentation, once the curves converge to a single value, we omit the rest steps because they are not informative. The instance name and TW are shown above TM in each legend group. More results (★) are deferred to the appendix.

steadily, with TM_9 most often leading throughout the run. Additionally, the smaller-$|\mathcal{M}|$ curves more frequently approach, and in several cases match the GUROBI reference line, while larger-$|\mathcal{M}|$ runs exhibit longer plateaus, whereas leveraging $|\mathcal{M}|$ substantially increases the chance of attaining the global optimum. The table echoes this effect quantitatively: for many instances (e.g., DD199 and power-685-bus across MC/MIS/MVC), EA+TDPA attains the GUROBI-optimal objective values whereas $(1+1)$EA remains far below/above them, consistent with the baseline being trapped on suboptimal plateaus. While the smaller modulator is preferable in best objective values, the first-hitting times in seconds can exhibit a trade-off (e.g., dwt_307), and there are instances where the baseline is competitive under our settings (e.g., can_161), suggesting room for adaptive choices of τ.

On Greedy Heuristics. Table 2 summarizes the MIS/MVC results (Results on MC(★) has been deferred to the appendix), and reveals two patterns. First, ϵ-greedy baselines are insensitive and problem-specific: on both ER and BA, increasing ϵ from 0 to 0.05 consistently worsens the runtime but does not reliably improve quality, and can even worsen it and yields larger gaps on both ER and BA. Second, TDPA provides a consistent improvement on solution quality and largely removes the tuning brittleness. For MIS, TDPA yields near-solver quality: on ER, (0.01)-TDPA attains a gap of about -0.20% relative to the 600s GUROBI baseline, and on BA it reaches 0.51% – both dramatically better than greedy's 2-4% gaps. For MVC, TDPA reduces the gap roughly by half compared to greedy, e.g., BA:3.78%→ 1.84%. Compared with MIS, MVC seems more sensitive to the remaining hard structure outside the reduced-treewidth region. In terms of time, TDPA typically runs in ∼25-60s, which place it near the 30-60s solver budgets; in this regime it can match or exceed short-budget solver performance in several settings while remaining purely heuristic. These trends suggest that TDPA primarily improves the search landscape and robustness (replace fragile ϵ choices with stable performance). Additionally, restricting TDP to reduced-treewidth components can systematically elevate simple heuristics toward near-optimal solutions while preserving the scalability advantages that make greedy methods attractive on hard instances.

INSTANCES	(1+1)EA	EA+TDPA(5)	EA+TDPA(7)	EA+TDPA(9)	GUROBI
DD199	1393.6 (553.52)	1415.6 (230.77)	1416.0 (80.89)	**1416.0 (68.32)**	**1416**
	273.0 (604.30)	300.0 (23.66)	**300.0 (16.55)**	300.0 (16.70)	**300**
	568.0 (622.84)	541.0 (23.36)	**541.0 (16.03)**	541.0 (16.05)	**541**
DWT_307	802.3 (82.94)	810.8 (152.70)	813.0 (291.82)	**813.5** (763.10)	**814**
	77.4 (255.50)	81.6 (24.59)	81.8 (52.01)	**82.4** (100.02)	**83**
	229.6 (258.66)	225.4 (24.69)	225.2 (43.19)	**224.6** (71.79)	**224**
IA-ENRON-ONLY	422.9 (23.03)	425.5 (36.28)	426.2 (31.29)	**426.5** (170.50)	**427**
	56.9 (65.66)	57.0 (2.00)	**57.0 (0.39)**	57.0 (0.88)	**57**
	86.1 (66.11)	86.0 (2.11)	**86.0 (0.34)**	86.0 (0.66)	**86**
POWER-685-BUS	1039.9 (280.53)	1069.6 (98.78)	1069.8 (96.26)	**1070.0** (173.46)	**1070**
	297.1 (436.09)	313.0 (25.24)	313.0 (7.51)	**313.0 (4.38)**	**313**
	387.9 (444.00)	372.0 (25.86)	372.0 (7.31)	**372.0 (4.09)**	**372**
SAYLR1	440.5 (5.91)	**445.0 (13.16)**	445.0 (16.60)	445.0 (33.21)	**445**
	110.9 (139.41)	119.0 (11.19)	**119.0 (8.21)**	119.0 (11.23)	**119**
	127.1 (142.74)	119.0 (11.30)	**119.0 (7.81)**	119.0 (10.45)	**119**
ASH292	655.6 (162.55)	661.0 (104.27)	662.5 (304.59)	**663.0** (245.65)	**663**
	81.2 (230.45)	84.0 (16.69)	84.0 (20.92)	**84.0 (11.34)**	**84**
	210.8 (234.57)	208.0 (16.33)	208.0 (17.59)	**208.0 (8.22)**	**208**
CAN_161	**450.4 (2.33)**	448.0 (29.48)	448.8 (39.56)	448.8 (109.05)	**456**
	40.2 (60.69)	40.0 (2.68)	40.0 (**1.88**)	40.1 (7.12)	**41**
	120.8 (62.19)	121.0 (2.88)	121.0 (**1.79**)	120.9 (6.01)	**120**
RDB200L	441.4 (109.44)	**460.0 (11.66)**	460.0 (22.35)	460.0 (77.16)	**460**
	94.2 (108.77)	**100.0 (7.65)**	100.0 (12.01)	100.0 (28.57)	**100**
	105.8 (108.43)	**100.0 (7.73)**	100.0 (11.49)	100.0 (26.34)	**100**

Table 1: For each $\cdot(\cdot)$ pair, the first value is *the average best solution size*, the value in $(\cdot)$ is *the average first hitting time of the best solution* in seconds. (5), (7) and (9) are different *target treewidth* values. For the three rows in each cell, from top to the bottom, are the size of MC, MIS, MVC, respectively.

On Graph Neural Network. Neural solvers for combinatorial optimization are commonly built upon GNNs with reinforcement learning [Ahn *et al.*, 2020] or diffusion-style sampling [Sun and Yang, 2023]. In practice, they often suffer from *(a) large training budgets*, *(b) sensitivity to the training distribution*, and *(c) limited competitiveness against strong heuristics*. We study an unsupervised alternative [Angelini and

METHOD	ER			BA		
	SIZE ↑ / ↓	GAP% ↓	TIME(S) ↓	SIZE ↑ / ↓	GAP% ↓	TIME(S) ↓
GUROBI(30S)	715.94/1518.04	7.85/3.93	30	879.23/1383.30	3.13/2.03	30
GUROBI(60S)	729.36/1505.15	6.13/3.11	60	885.83/1376.41	2.41/1.54	60
GUROBI(300S)	760.17/1474.66	2.16/1.10	300	904.01/1358.79	0.40/0.26	300
*GUROBI(600S)	776.95/1458.39	0.00/0.00	600	907.67/1355.20	0.00/0.00	600
↑GUROBI(30S)	718.58/1515.68	7.51/3.78	30	887.17/1374.36	2.26/1.39	30
↑GUROBI(60S)	734.35/1500.19	5.48/2.79	60	893.09/1369.25	1.61/1.03	60
↑GUROBI(300S)	768.29/1467.18	1.11/0.60	300	905.50/1357.18	0.24/0.15	300
↑GUROBI(600S)	781.58/1453.98	-0.60/-0.30	600	909.19/1353.66	-0.17/-0.11	600
(0.0)-GREEDY	760.50/1515.44	2.12/3.76	0.37/0.79	885.90/1402.61	2.40/3.38	0.39/0.71
(0.01)-GREEDY	759.12/1520.55	2.29/4.09	7.30/15.99	883.08/1408.42	2.71/3.78	7.73/14.31
(0.05)-GREEDY	750.46/1541.06	3.40/5.36	7.11/16.09	870.04/1431.79	4.15/5.35	7.52/14.47
(0.0)-TDPA	767.63/1496.46	1.20/2.54	24.68/25.60	895.80/1380.93	1.31/1.86	45.71/57.38
(0.01)-TDPA	778.41/1494.38	-0.20/2.40	32.18/36.34	903.08/1380.63	0.51/1.84	52.43/57.64
(0.05)-TDPA	775.68/1496.65	0.16/2.50	42.90/47.43	899.98/1381.49	0.85/1.90	61.88/58.89

Table 2: ·/· denotes the results for MIS/MVC. Largest improvement has been highlighted by the gray block. * indicates the baseline used to calculate optimal gap. ↑ indicates the tuned GUROBI. For greedy and TDPA, the values of ϵ are in (·).

Ricci-Tersenghi, 2022] that frames vertex problems as node classifications and minimize a QUBO-form objective [Glover *et al.*, 2019]. Empirically, as shown in Fig. 2, we find that optimization progress is not reflected by the loss: the loss decreases monotonically, yet solution quality (Performance) plateaus early, consistent with convergence to a poor local optimum; continuing training then yields diminishing returns rather than meaningful improvements.

TDPA provides a principled way to diagnose and exploit this plateau. We monitor the labels on the treewidth modulator $\mathcal{M}$ via *TM-stability*, defined at checkpoint t as $\frac{|L_p \cap L_c|}{|\mathcal{M}|}$, where L_t is the predicted label set on $\mathcal{M}$. In Fig. 2 (10,000 gradient-descent steps, evaluated every 50 steps), TM-stability rises to near 1.0 within the earliest checkpoints, substantially earlier than the loss and performance curves stabilize. This indicates that the model's decisions on $\mathcal{M}$ "freeze" quickly. This decoupling explains the stagnation: *subsequent updates mainly reshuffle predictions outside $\mathcal{M}$ and rarely change the globally relevant commitments captured by $\mathcal{M}$*. From this early stabilization, TDPA uses GNN's current labels on $\mathcal{M}$ as advice to produce a high-quality feasible solution early; the orange curves track this prediction and consistently dominate the raw GNN solutions during the long plateau for MC and MIS, and yield markedly better solutions for MVC as well (where smaller the better). Consequently, TM-stability serves as an actionable *early-stopping signal*, and TDPA turns partial training progress into near-best solutions without paying the full training cost. Results on BA reflect the similar improvements, further confirming the effectiveness while coupling with GNNs.

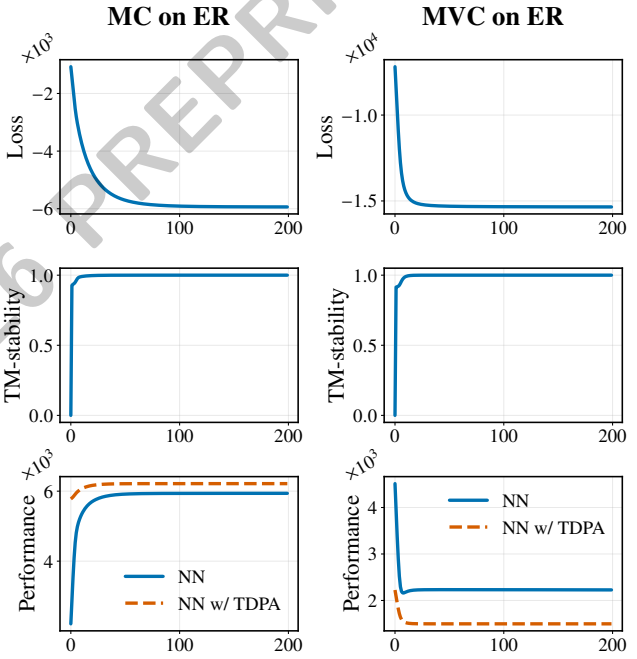

Figure 2: GNN training for ER. Each unit on *x-axis* is 50 gradient descend steps. Results for MIS and BA (★) are similar.

5 Conclusion

We introduced treewidth dynamic programming with advice, a general framework for enhancing graph combinatorial optimization algorithms. TDPA integrates TDP with heterogeneous advice sources, including $(1+1)$EA, greedy heuristics, and GNN-based models. To enable this integration, we show that treewidth modulator on given tree decomposition is hard, and we propose a MIP formula that enables fast computation in practice. We then prove that the resulting advice-integrated DP remains correct. Empirically, TDPA consistently improves solution quality over the underlying methods and stays competitive with GUROBI in first-hitting time. We further showed that TDPA provides a useful early-stage prediction, enabling earlier feedback during training and sharper improvement of downstream performance. These findings suggest that treewidth DP can be practically effective beyond small-treewidth instances when coupled with targeted advice. Future work will focus on practical methods for constructing smaller treewidth modulators to further expand the range of scalable instances.

Acknowledgements

We would like to particularly thank **Professor Michael Fellows** and **Professor Frances Rosamond** for their valuable comments and support to this work. I am deeply grateful to **Dr. Weitong Chen** for his tremendous guidance and unwavering support. We are also profoundly thankful to **Dr. Fabien Voisin**, who consistently provided timely support with computing resources. This work was supported with resources provided by the **Phoenix HPC** service at Adelaide University. Without the support of any of these parties, this work would not have reached its current stage.

References

[Ahn *et al.*, 2020] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International conference on machine learning*, pages 134–144. PMLR, 2020.

[Angelini and Ricci-Tersenghi, 2022] Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5:29–31, 2022.

[Bannach *et al.*, 2017] Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A modular library for computing tree decompositions. In *The Sea*, 2017.

[Baste *et al.*, 2020] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. i. general upper bounds. *SIAM J. Discret. Math.*, 34:1623–1648, 2020.

[Bodlaender *et al.*, 2023] Hans L. Bodlaender, Édouard Bonnet, Lars Jaffke, Dušan Knop, Paloma T. Lima, Martin Milanič, Sebastian Ordyniak, Sukanya Pandey, and Ondřej Suchý. Treewidth is np-complete on cubic graphs (and related results), 2023.

[Bodlaender, 1988] Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *International Colloquium on Automata, Languages and Programming*, 1988.

[Boettcher, 2022] Stefan Boettcher. Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems. *Nature Machine Intelligence*, 5:24–25, 2022.

[Courcelle, 1990] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85:12–75, 1990.

[Cygan *et al.*, 2011] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory of Computing Systems*, 54:73–82, 2011.

[Cygan *et al.*, 2020] Marek Cygan, F. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Parameterized algorithms. In *Beyond the Worst-Case Analysis of Algorithms*, 2020.

[Downey *et al.*, 2013] Rodney G Downey, Michael R Fellows, et al. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

[Erdos and Rényi, 1984] Paul L. Erdos and Alfréd Rényi. On the evolution of random graphs. *Transactions of the American Mathematical Society*, 286:257–257, 1984.

[Fomin *et al.*, 2015] F. Fomin, Daniel Lokshtanov, Neeldhara Misra, M. S. Ramanujan, and Saket Saurabh. Solving d-sat via backdoors to small treewidth. In *ACM-SIAM Symposium on Discrete Algorithms*, 2015.

[Glover *et al.*, 2019] Fred W. Glover, Gary A. Kochenberger, and Yu Du. Quantum bridge analytics i: a tutorial on formulating and using qubo models. *4OR*, 17:335 – 371, 2019.

[Gurobi Optimization, LLC, 2024] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[Jablonský, 2015] Josef Jablonský. Benchmarks for current linear and mixed integer optimization solvers. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 63:1923–1928, 2015.

[Jansen *et al.*, 2021] Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[Karalias and Loukas, 2020] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *ArXiv*, abs/2006.10643, 2020.

[Lampis, 2013] Michael Lampis. Parameterized approximation schemes using graph widths. In *International Colloquium on Automata, Languages and Programming*, 2013.

[Lewis and Yannakakis, 1980] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20:219–230, 1980.

[Maniu *et al.*, 2019] Silviu Maniu, Pierre Senellart, and Suraj Jog. An experimental study of the treewidth of real-world graph data (extended version). In *International Conference on Database Theory*, 2019.

[Marchand *et al.*, 2021] Bertrand Marchand, Yannick Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock fpt algorithms in rna bioinformatics. *Algorithms for Molecular Biology : AMB*, 17, 2021.

[Mittelmann, 2002] Hans D Mittelmann. Benchmark for optimization software. *http://plato. asu. edu/bench. html*, 2002.

[Mittelmann, 2010] Hans Mittelmann. Mixed integer linear programming benchmark (serial codes), 2010.

[Mittelmann, 2017] Hans D Mittelmann. Latest benchmarks of optimization software. In *INFORMS Annual Meeting. Houston, TX*, 2017.

[Robertson and Seymour, 1984] Neil Robertson and Paul D. Seymour. Graph minors. iii. planar tree-width. *J. Comb. Theory B*, 36:49–64, 1984.

[Schuetz *et al.*, 2021] Martin J. A. Schuetz, John Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4:367 – 377, 2021.

[Sun and Yang, 2023] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *ArXiv*, abs/2302.08224, 2023.

[Sun *et al.*, 2022] Haoran Sun, Etash K. Guha, and Hanjun Dai. Annealed training for combinatorial optimization on graphs, 2022.

[Williams *et al.*, 2003] Ryan Williams, Carla Pedro Gomes, and Bart Selman. Backdoors to typical case complexity. In *International Joint Conference on Artificial Intelligence*, 2003.

[Wu *et al.*, 2014] Ledell Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *J. Artif. Intell. Res.*, 49:569–600, 2014.