ZOOM-IN on Modulator: An Evolutionary Algorithm Enhanced Tree-width Dynamic Programming*

Jialiang Li Aneta Neumann Frank Neumann
Hung Nguyen Mingyu Guo

April 18, 2024

Abstract. *Tree-width*(TW), obtained via *tree decomposition*, measures the similarity between a given graph and an exact tree. It is extensively utilized alongside with dynamic programming to design *fixed-parameter tractable* algorithms with complexity $\mathcal{O}(f(TW)\cdot n^{O(1)})$ when solving NP-HARD graph combinatorial optimization problems. With the increase on TW, an exponential growth on f is inevitable, which directly restricts scalability only on graphs with bounded tree-widths. However, in practice, it is unrealistic to expect tree-width remains within a desired threshold.

In this work, we propose a general framework to improve the standard tree decomposition based dynamic programming by harnessing the power of elementary evolutionary algorithms such as (1+1)EA. To accelerate the efficiency, we put efforts on reducing the tree-width and reusing the constructed tree decomposition. On reducing tree-width, we adopt *tree-width modulator* with necessary adjustments, propose a more practical version of modulator. On reusing tree decomposition, we first fixate a *partial solution* on the vertices of modulator through a ZOOM-IN EA. With the known partial solution, the dynamic programming can proceed and scale to much larger TWs.

To demonstrate the applicability of the model, we select three graph combinatorial problems to execute three sets of experiments analyzing the performance of the model in term of *solution quality*, *the number of fitness evaluations*, *wall-clock time* and *the first hitting time* of the obtained best solutions. As the *first study* to combine structural parameters and evolutionary computations, we focus on demonstrating the extent to which basic (1+1)EA and tree-width based dynamic programming can mutually benefit. Our experimental results indicate that the

*Some technical details and important results are included in the appendix due to space limitations. The appendix and code are available at as supplementary materials.

proposed framework significantly improves the scalability of the dynamic programming with non-optimal tree decompositions. It also substantially reduces the number of fitness evaluations, time cost and quickly yields a near-optimal solution. The improvements will be stronger by tuning the target tree-width.

Keywords: (1+1) EA · fixed parameter tractable · tree decomposition · dynamic programming · graph combinatorial optimization

1 Introduction

Evolutionary computation community has studied NP-HARD combinatorial optimization problems through the lens of parameterized complexity. Theoretical results on selected parameters make substantial progress for a wide spectrum of problems including vertex cover [15], maximum leaf spanning trees [14], Euclidean TSP [20], makespan scheduling [23], closest string [22], and so on [13]. Commonly studied parameters include the solution size [15] and hand-crafted problem-specific graph structures [14, 20]. Other recent developments include jump-and-repair operators [8] and multivariate analysis [3].

This paper focuses on a specific fixed parameter algorithm called *tree decomposition based dynamic programming* (TD-DP) [5], which is one of the most well-known fixed parameter algorithms and has been widely applied on graph combinatorial optimization. We start with a high-level description and defer the more technical descriptions to Section 3. As mentioned earlier, fixed parameter analysis studies how to identify and exactly solves easy instances of NP-hard problems. Many NP-hard combinatorial optimization problems that are hard on general graphs become tractable when the actual graph instances are *similar* to trees. Focusing on graph structural invariants, *tree-width* (TW) measures the similarity between a given graph and an exact tree. We present its formal definition in Definition 1. TW is calculated via a complicated *tree decomposition* process [10], which maps any given general graph to a tree structure, where each tree node, often called a *bag*, corresponds to a subset of vertices from the original graph. Graphs closer to trees tend to lead to smaller bags. The tree-width is defined as the maximum bag size minus one. A significant result on tree-width is *Courcelle's theorem* [9], which states that **every** graph property expressible by *Monadic Second Order Logic* (MSO_2) is decidable in linear time on graphs with bounded tree-width — there is often a *straightforward* way to construct TD-DP with time complexity $O(f(TW) \cdot n^{O(1)})$. For NP-hard problems, f is exponential, therefore, TD-DP only scales on graphs with bounded tree-widths.

Despite the general applicability of TD-DP, in practice, direct usage of TD-DP seems out of reach because the graph instances do not necessarily have small enough tree-widths. In this paper, we apply evolutionary computations, specifically (1+1)EA, to scale up TD-DP. We focus on vertex selection problems, where the solution is a subset of vertices, for example, *independent set*, *vertex cover* and *max cut*. For presentation purposes, for the rest of this paragraph, we utilize *maximum independent set* (MIS) as our context. For MIS, there exists a TD-DP

with complexity $\mathcal{O}^*(2^{TW})$[1]. When the tree-width is below a threshold (i.e., 10), TD-DP is certainly scalable. For graphs with larger tree-widths, we resort to a concept called *Tree-width Modulator* (TM), which concerns how to optimally delete a small set of vertices in order to lower the tree-width of the remaining graph below a threshold. A naive way to utilize TM is to simply ignore all vertices in TM (i.e., the final MIS solution does not contain any vertex from TM), which forces TD-DP to be able to scale on the remaining graph. However, naively ignoring all vertices in TM generally alters the solution space. For example, the optimal MIS solution may indeed need to include some nodes from TM. Additionally, directly ignoring TM may also break the feasibility. For example, for *minimum vertex cover* (MVC), simple TM deletion may lead to uncovered edges. Therefore, instead of naively ignoring all vertices in TM, we use evolutionary algorithm(EA) to generate partial solutions involving vertices only in TM, named ZOOM-IN EA. For MIS, the partial solution specifies which vertices inside TM are selected (and discarded) as a part of the final MIS solution. With the partial solution fixated, we run TD-DP to complete the full solution on the remaining graph, which is scalable as the remaining graph's tree-width is below our desired threshold. The fitness of ZOOM-IN EA is evaluated based on the completed solution after running TD-DP. When mutating a partial solution within TM, we need to respect *local constraints*. Specifically for MIS, when selecting vertices from TM to be included in the final MIS solution, we cannot select connected vertices. Otherwise, such local conflicts cause infeasibility. Assume EA is built by bit strings, after fixating a partial solution with local constraints satisfied, for each bag, there are three vertex states, including *selected*(1), *discarded*(0) and *uncertain*(?). We then apply an additional *marking mechanism*, uploading the partial solution to the DP tabulation used by TD-DP. Specifically, for each bag, the searching space is reduced by revealing some of the ? bits to 0 or 1, which are filled by the partial solution from EA. Overall, our approach is problem-agnostic, although we do need problem-specific local constraints, which are rather trivial to apply.

A key challenge we have yet mentioned is the computation of the tree-width modulator. Unfortunately, the computation of tree-width modulator is known NP-HARD, and trivial but efficient heuristics seem beyond reach [11, 18].

On the challenge of application aspect, vertex deletion would trigger the recalculation of tree decomposition for the remaining graph[2], which is made even worse by the fact that optimal tree decomposition is also NP-HARD and commonly used tree decomposition heuristics are expensive for large graphs. This motivates us to propose a more scalable version of tree width modulator, named as *Tree-width Modulator on a Given Tree Decomposition (TMGTD)*. The idea is to run an expensive tree decomposition heuristic *only once*. We then focus

[1] For simplicity, we use $\mathcal{O}^*(\cdot)$ to highlight the dominating terms and omit polynomial terms. The notation is widely used.

[2] The scope of our discussions focus on calculating tree-widths by constructing tree decompositions. Other exact or approximation methods directly calculating tree-widths are out of scope.

on optimally deleting vertices from the existing tree decomposition without ever regenerating the tree decomposition (i.e., never recreate the bags nor modify the tree structure). We show that our more scalable version is still NP-hard. However, in practice, it can often be solved efficiently via mixed integer programming or via cheap heuristics.

In summary, we apply evolutionary algorithm, specifically (1+1)EA, to scale up TD-DP, leading to a practically scalable and more importantly a *problem-agnostic general* approach that can be applied to a variety of graph combinatorial optimization problems. In our experiments, we demonstrate the effectiveness of our approach on *maximum independent set*, *minimum vertex cover* and *max cut*. We run exhaustive experiments on a variety of graphs and provide comprehensive experimental results in terms of *solution qualities*, *the number of fitness evaluations*, *wall-clock time* and the *first hitting time* of the best solutions obtained. By integrating (1+1)EA, TD-DP can scale to much larger tree-widths, and the number of evaluations of EA is significantly shortened compared to (1+1)EA alone. Meanwhile, near-optimal results will appear at much earlier stages. Notably, previous works on combining parameterized complexity and EA focused on specific optimization problems or problem-specific parameters, while our work is the *first study* that combines problem-agnostic graph structural parameters and EA.

2 Preliminaries

Given a graph $G = (V, E)$, we note $|V| = n, |E| = m$. Given $V' \subseteq V$, we define the induced subgraphs as $G(V')$, and we use $N[\cdot]$ and $N(\cdot)$ to represent the inclusion-wise and exclusion-wise neighborhood, respectively. We define *boundary* of V' as $\partial_G(V') = \{v \in V' : \exists u \in G \setminus V', (u, v) \in E\}$.

The formal definitions of tree decomposition and tree-width are given as:

Definition 1 (Tree Decomposition & Tree-width [10]). *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 a tree decomposition to be valid, we must have*

- *The union of all bags contain 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 **tree-width** *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 tree-width as $\mathcal{T}^*$.*

Using tree decomposition, we can construct *fixed-parameter tractable* dynamic programming for many graph combinatorial optimization problems [5].

The time complexity of these NP-hard problems is typically $O(f(TW) \cdot n^{O(1)})$, where f is inevitably exponential in TW. For practical scalability, we need small tree-widths. That is, it is desirable to work with the optimal tree-width TW^* as it is the smallest possible. However, the decision version of optimal tree-width is NP-COMPLETE [2], and it remains NP-COMPLETE for many special graphs [6]. Furthermore, for many graphs, even the optimal tree-width can still be too large. To handle graphs with large tree-widths, we resort to the following concept called *tree-width modulator*, which concerns how to optimally delete a small set of vertices in order to reduce the remaining graph's tree-width to below a threshold.

Definition 2 (Tree-width Modulator [11](TM)). *Let $\eta \geq 0$ be an integer and G be a graph. A set $X \subset V(G)$ is called an η-tree-width modulator in G if $TW^*(G \setminus X) \leq \eta$. We use $TM^*_\eta(G)$ to denote the optimal tree-width modulator with the minimum size (i.e., when $|X|$ is minimized).*

As a special case of *vertex deletion problem* with hereditary property [11,18], the decision version of tree-width modulator is NP-complete. Deriving the optimal tree-width modulator $\mathrm{TM}^*_\eta(G)$ involves two optimization subtasks. We need to optimally decide X (which vertices to delete) and also need to calculate the optimal tree-width of the remaining graph $\mathrm{TW}^*(G \setminus X)$. As mentioned above, the decision version of optimal tree-width is NP-complete. Furthermore, even if we assume that we have access to an oracle that returns the optimal tree-width, the optimal vertex deletion part is still NP-complete — We use $MVC(G)$ and $MFVS(G)$ to denote the *minimum vertex cover* (MVC) and the *minimum feedback vertex set* (MFVS), respectively. It is easy to see that $\mathrm{TM}^*_0(G)$ is MVC and $\mathrm{TM}^*_1(G)$ is MFVS, and these two problems only involve trivial tree decomposition, so these problems are hard even with an optimal tree-width oracle.

Considering the above hardness results, a natural idea is to settle for heuristics. That is, we could heuristically select the vertices for deletion and also heuristically calculate the tree-width of the remaining graph. For example, the vertex deletion subtask can perhaps be carried out via evolutionary computation or reinforcement learning, and the tree decomposition subtask can perhaps be carried out using standard heuristics such as *min degree* or *min fill-in* [7]. Unfortunately, in practice, tree decomposition is expensive on large graphs even if we settle for heuristics. If we use evolutionary computation to heuristically select the vertices for deletion, then we cannot afford too many fitness evaluations. Similarly, reinforcement learning for vertex deletion is also not scalable as it also requires frequent regeneration of tree decompositions.

Therefore, for practical scalability, we propose a more scalable version of tree-width modulator, which we call *Tree-width Modulator on a Given Tree Decomposition (TMGTD)*.[3] Our more scalable version only generates the tree decomposition once, which is affordable even with an expensive heuristic. We only focus on deleting vertices from the bags of this constructed tree decomposition. Thus, the bags are neither regenerated nor the tree structure is modified. We provide the formal definition of TMGTD, together with an example in Fig. 1.

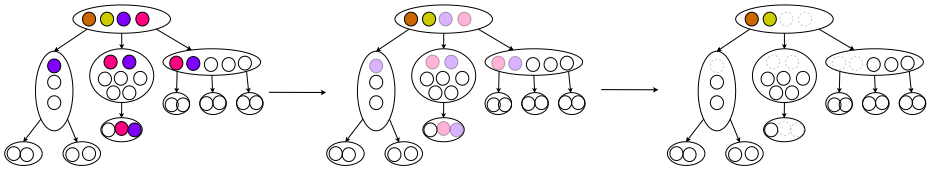


Fig. 1: Recall that the tree-width is the largest bag size minus one. By removing the purple and the red vertex from the existing bags without regenerating the tree structure, we reduced the tree-width from 6 to 4.

Definition 3 (Tree-width Modulator on a Given Tree Decomposition (TMGTD)**).** *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 tree-width 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.

First of all, we show that deleting vertices from the bags without regenerating the tree structure guarantees the validity of the resulting 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.

Proof. Assume the set of deleting vertices is Y, and we execute the deletion on a valid tree decomposition. By removing Y from each X_t, we eliminate the existence of all vertices in Y from T, so as all related incident edges. Clearly, this operation satisfies the first two properties in Def. 1. Additionally, the third property in Def. 1 will never be violated since we remove Y completely from every bag.

[3] The edge-centric version of this problem has previously been studied in the field of bioinformatics [19], named as **Tree-diet**, and a more general form of the problem is recognized by the term $\mathcal{F}$-M-Deletion **[4]**.

Next, we claim that even though TMGTD avoid regenerating tree decomposition, it is still NP-COMPLETE. For space limit, the proof is deferred to **Appendix A.1**.

Theorem 1 (Hardness of Def. 3). *Tree-width modulator on a given tree decomposition is* NP-COMPLETE.

To wrap up the complexity discussion, we provide two trivial bounds to roughly illustrate the size of TMGTD. Proof is in **Appendix A.2**.

Lemma 1 (bounds).

1. *Given an optimal tree-width* TW^*, $|TM^*_\eta| \geq |TM^*_{\eta'}|$ *when* $\eta \leq \eta'$
2. $|TMGTD^*(TW_\mathcal{T}, \mathcal{C})| \geq |TM^*(TW^*, \mathcal{C})|$

In practice, despite the NP-completeness, optimally solving TMGTD on practical graphs is a scalable task due to the limited number of large bags. The goal is to reduce every bag's size to at most $\eta+1$. Initially, bags of size at most $\eta+1$ are discarded. For a bag with size x, at least $x-\eta-1$ vertices must be removed so the resulting TW is desired. The task of minimizing the number of vertices removed can be easily formulated as a mixed integer program. With mixed integer programming solvers such as Gurobi, deriving the optimal TMGTD is scalable. For particularly difficult or large instances, incumbent results can serve as heuristic solutions.

Lastly, given that TMGTD only generates a tree decomposition once, intuitively speaking, the quality of the initial tree decomposition could significantly influence the quality of the modulator. It raises the following question: whether better initial tree decompositions (i.e., start with smaller tree-widths) always lead to better modulators (i.e., end up having to delete fewer vertices)? The answer is no. We provide the following observation, and defer the examples and detailed discussions to **Appendix. A.3** due to the space limit.

Observation 1 *Optimally solving* $TMGTD$ *via optimal tree decompositions* $\mathcal{T}^*$ *is not a sufficient condition to access the optimal* TM^*. *On the other hand, optimally solving TMGTD via non-optimal tree decompositions is possible to access the optimal* TM^*.

3 Framework

Our proposed approach utilizes evolutionary computation to scale up TD-DP. To simplify the discussion, assume a tree modulator TM for a given tree decomposition TD is known, which is obtained via solving the scalable TMGTD problem. Based on the definition of TM, the removal of all vertices in TM from TD reduce the width of TD below a threshold, thereby allowing TD-DP to proceed. The high-level idea of our approach is to apply evolutionary computation to mutate the partial solution, focusing exclusively on the vertices within the TM. In this paper, we use the elementary (1+1)EA. Due to its simplicity, we

defer the pseudo-code of (1+1)EA to Alg. 5 in **Appendix C**. With the partial solution fixated, the solution for the remaining graph can then be completed via TD-DP. The fitness evaluation of the partial solution provided by (1+1)EA focuses on the quality of the completed solution. We name our approach ZOOM-IN EA-enhanced dynamic programming, reflecting the fact that the ZOOM-IN (1+1)EA only cares about the tree modulator vertices.

When evolving the partial solution, adhering problem-specific *local constraints* is essential. Specifically, a partial solution is feasible if and only if it can be completed into a feasible full solution. For example, for MIS, the partial solution cannot contain two connected vertices. We also need a *marking mechanism*, in the sense that the decisions within the tree modulator may affect the decisions on the rest of the graph. For example, for MIS, for selected vertices in TM, their neighbors outside of TM must not be selected. Integrating the marking mechanism into TD-DP is trivial, because TD-DP is exhaustive in nature and the marking mechanism merely filters out some of the irrelevant states based on the partial solution given by EA.

3.1 Technical Details of Our Approach

We first outline the fundamental principles of *tree decomposition based dynamic programming* (TD-DP) and the details of *local constraints* and *marking mechanism* used in our ZOOM-IN *EA-enhanced dynamic programming*.

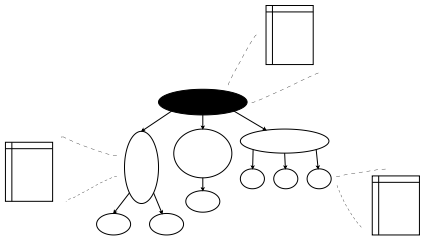
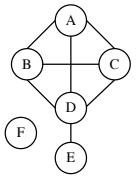
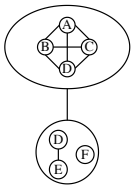


Fig. 2: Tree decomposition based dynamic programming, where every bag is associated with a DP table and the final result is assembled at the root bag.



(a) $G = (V, E)$



(b) An example of optimal $\mathcal{T}_1^*$ of G.

Fig. 3: Simple example of optimal tree decomposition.

Tree Decomposition based Dynamic Programming (TD-DP)
We defer the pseudo-code of TD-DP to **Appendix C** due to space constraint. Tree decomposition allows the execution of dynamic programming in a bottom-up fashion. The primary difference to conventional tabular DPs has been highlighted in Fig. 2, TD-DP disassembles the global table into segments, with each *bag* possessing its own local table. After exactly filling each local table, global solutions are assembled at the root *bag*. The general TD-DP procedures are shown in Alg. 6. In the pseudo-code, both X and Y represent the dynamic programming tabulation. We traverse the tree decomposition via breath-first search from leaf bags to the root bag. For each bag $\mathcal{B}_t$, there is at most one parent

bag and multiple child bags. The valid state collection $\mathcal{S}_t$ of each bag $\mathcal{B}_t$ is calculated by problem-specific constraints. To elaborate the process, we use MIS and Fig. 3 as a concrete example. For the leaf bag $\{D, E, F\}$, the valid states $\mathcal{S} = \{\{\}, \{D\}, \{E\}, \{F\}, \{D, F\}, \{E, F\}\}$. To calculate the valid state collection, we apply MIS's *no-edge* constraint to the *power set* of $\{D, E, F\}$ to filter out all invalid states. Then, we use table Y to decide which state is able to pass to the parent bag $\{A, B, C, D\}$. Every *locally valid* state needs to be passed via the *intersection* between child bag and parent bag. Therefore, in our example, all valid states $s \in \mathcal{S}$, able to pass from child bag to the parent bag must satisfy: $s \subseteq \mathcal{B}_{parent} \bigcap \mathcal{B}_{child} = \{D\}$. As outlined by the seventh line in Alg. 6, we use table Y to record the **best** state from the subtree rooted at $\mathcal{B}_t$. After that, we utilize table X to assemble the partial solutions from the children bags. In our example, we use the mentioned method to enumerate all valid states of $\{A, B, C, D\}$. Next, in the sixth line, we use the *intersection* $(s \bigcap \mathcal{B}_{child})$ as the **key** to extract the corresponding entry from the Y table filled by the child bag. By using the two-stage strategy, we finish computing one bag.

The pseudo-code assumes the objective is maximization. If minimization is the goal, the 'max' function in the code can simply be replaced with 'min'.

Local Constraint Property and Marking Mechanism

Given a tree decomposition $\mathcal{T}$ and a modulator of it $\mathrm{TM}_{\mathcal{T}}$, derived from the problem TMGTD, we apply (1+1)EA on the vertices in $\mathrm{TM}_{\mathcal{T}}$. (1+1)EA will occasionally make mistakes due to its inherent randomness. If the EA decisions on vertices in $\mathrm{TM}_{\mathcal{T}}$ already lead to conflicts, then we consider this partial solution infeasible. Fig. 4 shows an example for MIS. In this figure, the circle represents the area of $\mathrm{TM}_{\mathcal{T}}$. There are two nodes in the circle which are connected. An example local constraint violation is when EA selects both vertices in the circle. When this happens, this partial solution can never be completed into a full solution that is feasible. In terms of marking mechanism, in Fig 4, the black vertex represents a vertex selected by (1+1)EA. For MIS, if a vertex is included in the solution set, all its neighbours must be excluded. That is, the three vertices to the left of the circle must be excluded. We can summarise similar local constraints and marking mechanism for other combinatorial optimization models.

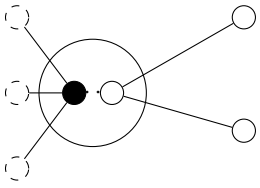


Fig. 4: If the black vertex is picked by (1+1)EA, dashed vertices are ensured out of the current MIS solution. And we can safely remove them. If the white vertex (inside the circle that represents the tree modulator) is not picked by (1+1)EA, then we can not make decisions for the two vertices to the right of the circle. We keep them.

It is obvious that if (1+1)EA makes the correct decisions on the tree modulator vertices, then the completed solution after TD-DP is globally optimal.

For MIS, MVC and MC, we list the problem-specific **local constraints** and **marking mechanism**. We assume the local decisions produced by (1+1)EA is $\mathcal{S} = \mathcal{S}_1 \bigcup \mathcal{S}_0$, where $\mathcal{S}_1$ represent vertices selected as a part of the final solution, and $\mathcal{S}_0$ refers to vertices out of the final solution.

- **(LC-MIS)** $\mathcal{S}$ can be *safely* removed from $\mathcal{T}$ by leaving marks on $N[\mathcal{S}_1] \setminus \mathcal{S}$
- **(LC-MVC)** $\mathcal{S}$ can be *safely* removed from $\mathcal{T}$ by leaving marks on $N[\mathcal{S}_0] \setminus \mathcal{S}$
- **(LC-MC)** $\mathcal{S} \setminus \partial_G(\mathcal{S})$ can be safely removed from $\mathcal{T}$ by leaving marks on $\partial_G(\mathcal{S})$

Algorithm 1: ZOOM-IN Mutation

Input : A random bit string $x : \{0,1\}^n$, n=|V| Modulator of a given tree decomposition $\mathrm{TM}_{\mathcal{T}}$

$z = (z_1, z_2, ...z_n)$

1 $z_i = \begin{cases} 1 & i \in \mathrm{TM}_{\mathcal{T}}, \\ 0 & i \in V \setminus \mathrm{TM}_{\mathcal{T}} \end{cases}$

2 $y \leftarrow \mathrm{BitwiseFlipping}(x, \frac{1}{|TM_{\mathcal{T}}|})$;

3 $y \leftarrow y \wedge z$;

4 **return** y;

After removing the *safe* vertices, we need to inform the dynamic programming what (1+1)EA has done. As mentioned earlier, $\mathcal{S}$ can be safely removed from G since the vertices inside $\mathcal{S}$ has no connection to the rest of the graph. However, for MIS and MVC, decisions on $N[\mathcal{S}_1] \setminus S$ and $N[\mathcal{S}_0] \setminus S$ will be influenced by decisions made by (1+1)EA on $\mathcal{S}$. Same principle applies to the $\partial_G(S)$ in the case of MC. We use Fig. 3 and MIS as an example. Suppose (1+1)EA decides to select the vertex **A** to the solution set $\mathcal{S}$, we know that **B, C, D** must not be in the solution set. In the context of (1+1)EA, the **bit** at the position of **A** is assigned with **1**, and the **bits** at the positions of **B, C, D** must be **0**. We then take the tree decomposition in Fig.3b to continue this example. For the root bag, we do not need to compute valid states anymore, because the valid state has been confirmed. We have $< 1, 0, 0, 0 >$ lexicographically corresponds to $\{A, B, C, D\}$ based on (1+1)EA's decision. Comparing with plain TD-DP, when we firstly encounter the bag $\{A, B, C, D\}$, the bit string status is $<?^4, ?, ?, ? >$. Thus, we need to enumerate all $\mathbf{2^4}$ to construct the valid state collection. Reducing from 16 state validity checks to none, it is a big improvement. On the child bag $\{D, E, F\}$, as we already know D is assigned by **0**, we have $< 0, ?, ? >$ corresponds to $\{D, E, F\}$. Therefore, the number of states for the child bag $\{D, E, F\}$ is already halved from 2^3 to 2^2 than using TD-DP only. Considering we use TMGTD, we successfully reduce the TW to the target TW′. Thus, in the worst case, we need to compute $O(2^{TW'+1})$ options for each bag for MIS. If the target TW′ is small enough, the TD-DP has been reduced to the time $O(2^{TW'} \cdot poly(TW') \cdot n)$, where $poly(TW')$ is the time spent on state validity checking. In summary, local constraints and marking mechanism allows us to proceed TD-DP on the remaining graph in a scalable fashion, without modifying the solution space. There is the additional computational cost for validity checking, but that is only polynomial in the tree-width.

[4] We use '?' to represent uncertain bits.

Having established all preconditions, we now present the complete zoom-in EA enhanced dynamic programming algorithm.

ZOOM-IN EA Enhanced Dynamic Programming

We use Alg. 1 to explain the ZOOM-IN EA. We represent the graph with a bit string $x : \{0,1\}^n$, where each bit corresponds to a $v \in V$. Firstly, we build a mutation mask z based on the calculated $\mathrm{TM}_{\mathcal{T}}$ via the indicator function specified in the first line. Next, we apply the standard bit flipping mutation to x with $\frac{1}{|\mathrm{TM}_{\mathcal{T}}|}$ as the mutation ratio. Finally, we apply the mutation mask z to eliminate the mutations on all irrelevant bits. This approach confines the mutations within the $\mathrm{TM}_{\mathcal{T}}$ area. Following this, we apply the local constraint property to $\mathcal{T}$ and at the same time, use the marking mechanism to inform the TD-DP which bits are confirmed at (1+1)EA stage. For the fitness function, we define it by **summing up** the fitness value from (1+1)EA and the exact solution from TD-DP. TD-DP will **never produce conflicts** due to the correctness of the framework. That is to say, the penalties will always come from (1+1)EA by violating the local constraint properties within the area of $\mathrm{TM}_{\mathcal{T}}$. We complete the model description by showing Alg. 2.

Algorithm 2: ZOOM-IN EA enhanced TD-DP

Input : Graph $G = (V, E)$, a tree decomposition $\mathcal{T}$,
Modulator of the given tree decomposition $\mathrm{TM}_{\mathcal{T}}$,
Fitness function $f : \{0,1\}^n \to \mathbb{R}$,
A bit string: $x : \{0,1\}^n$,
LC: problem-specific local constraint property,
DP tabulations X and Y

```
while not terminate criteria do
    y ← zoom-in mutation(x, TM_T);
    r^IN ← f(y');
    Calculate S by applying LC(G, y);
    for X_t ∈ T do
        X_t ← X_t \ S
    Mark the table X and Y;
    r^OUT ← TD-DP(G, T);
    if r^IN + r^OUT ≥ r^BEST then
        x ← y;
        r^BEST ← r^IN + r^OUT
    for X_t ∈ T do
        X_t ← X_t ⋃ S
```

4 Experiments

In our experiments, we aim to demonstrate the following:

- (1+1)EA significantly improves the scalability of TD-DP to handle larger tree-width.
- TD-DP efficiently reduces the number of fitness evaluations typically required by (1+1)EA. Good solutions will appear at an early stage.
- Allocated the same time budget, our model shows a distinct advantage over (1+1)EA with various initialization strategies.
- Our framework can adapt to various graph combinatorial problems.

It is important to note that our objective is to utilize (1+1) EA to strengthen the scalability of TD-DP. Thus, our experiments focus on graphs whose tree-width is moderately larger than the upper limit manageable by standard TD-DP. We note that the magnitude of tree-width in our experiments is sufficient for many practical graphs, such as road networks, bio-information graphs, and etc. In general, these graphs do not possess extremely large cliques.

For the sake of reproducibility, we use **Appendix.B.1** and **Appendix.B.2** to explain **experiment configurations** and **EA implementation details**, respectively.

4.1 Standalone TD-DP in Practice

In our initial set of experiments, we demonstrate the practical efficiency of plain TD-DP. We utilize graphs generated by random regular model and *Erdős–Rényi* model. We manually adjust the parameters of the models to ensure the width of the tree decompositions falling within a range that allows us to observe the appearance of the exponential runtime increase. For each instance, TD-DP is given with only **1 shot** and timeout after **30 minutes**. As indicated in Table. 4 in **Appendix. D**, with the calculated tree-width increases, the runtime escalates exponentially. The results align with the theoretical time complexity of TD-DP. Therefore, it will be prohibitive to proceed the TD-DP beyond a certain threshold of tree-width, since the single exponential term always leads to impractically long runtime.

4.2 Target TW

In this set of experiment, we use small-size instances to assess the impact of the various size of TMs on our model. For each instance, we set three different target **TW=5,7,9**. For both our model and (1+1)EA, we gave **3,000 and 1,000,000** steps of fitness evaluations, respectively. We let our model and (1+1)EA start with the exact same bit string, which is initialized by basic feasible solutions. Specifically, for MVC, we use $\mathbf{1}^n$; for MIS, we use $\mathbf{0}^n$ and for MC, $\mathbf{0}^n$ and $\mathbf{1}^n$ will have the same impact, we choose to use $\mathbf{0}^n$ in our experiments. In the visualization in Fig. 5, we follow the rules: *1) Once the curves converge to the same value, we choose not to draw the rest of the steps because they are not relevant to our emphasis, and no benefits helping readers understand.*, *2) We drop the results at the* **very first step***, because the starting point are the same for all curves based on our initialization.*

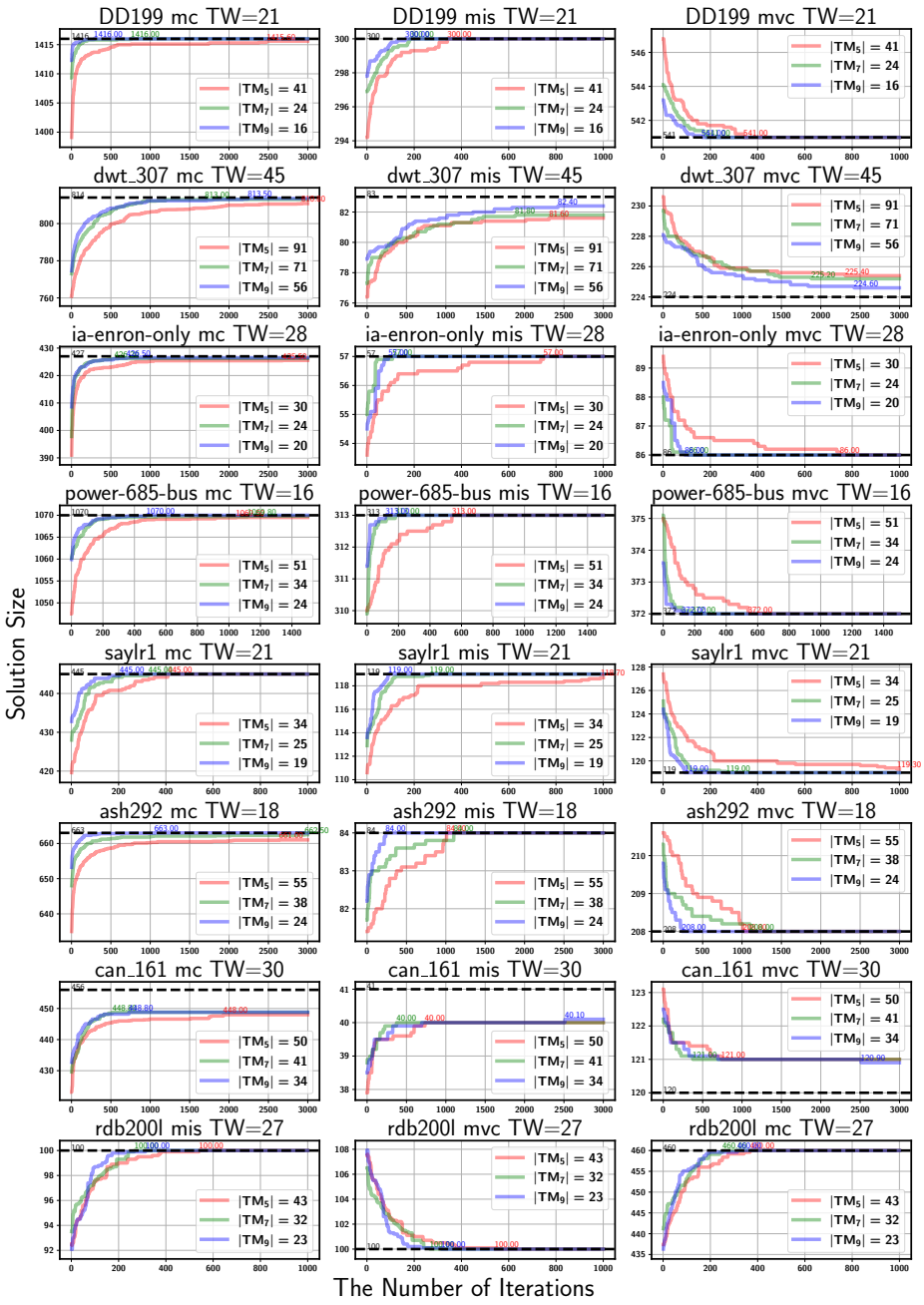


Fig. 5: The colorful numbers represents *the average first hitting time of best solutions for each line*. More details refer to Table. 1. The - - - - - indicates the Gurobi solution.

Table 1: For each data pair, the FIRST VALUE is *the average best solution size*, the SECOND VALUE (in parentheses) is *the average first hitting time of the best solution*. (5), (7) and (9) are different *target tree-width* values.

Graph	Problem	(1+1)EA	ZOOM-IN(5)	ZOOM-IN(7)	ZOOM-IN(9)	GUROBI
DD199	MC	1393.6 (553.52)	1415.6 (230.77)	1416.0 (80.89)	**1416.0 (68.32)**	**1416**
	MIS	273.0 (604.30)	300.0 (23.66)	**300.0 (16.55)**	300.0 (16.70)	**300**
	MVC	568.0 (622.84)	541.0 (23.36)	**541.0 (16.03)**	541.0 (16.05)	**541**
dwt_307	MC	802.3 (82.94)	810.8 (152.70)	813.0 (291.82)	**813.5** (763.10)	**814**
	MIS	77.4 (255.50)	81.6 (24.59)	81.8 (52.01)	**82.4** (100.02)	**83**
	MVC	229.6 (258.66)	225.4 (24.69)	225.2 (43.19)	**224.6** (71.79)	**224**
ia-enron-only	MC	422.9 (23.03)	425.5 (36.28)	426.2 (31.29)	**426.5** (170.50)	**427**
	MIS	56.9 (65.66)	57.0 (2.00)	**57.0 (0.39)**	57.0 (0.88)	**57**
	MVC	86.1 (66.11)	86.0 (2.11)	**86.0 (0.34)**	86.0 (0.66)	**86**
power-685-bus	MC	1039.9 (280.53)	1069.6 (98.78)	1069.8 (96.26)	**1070.0** (173.46)	**1070**
	MIS	297.1 (436.09)	313.0 (25.24)	313.0 (7.51)	**313.0 (4.38)**	**313**
	MVC	387.9 (444.00)	372.0 (25.86)	372.0 (7.31)	**372.0 (4.09)**	**372**
saylr1	MC	440.5 (5.91)	**445.0 (13.16)**	445.0 (16.60)	445.0 (33.21)	**445**
	MIS	110.9 (139.41)	119.0 (11.19)	**119.0 (8.21)**	119.0 (11.23)	**119**
	MVC	127.1 (142.74)	119.0 (11.30)	**119.0 (7.81)**	119.0 (10.45)	**119**
ash292	MC	655.6 (162.55)	661.0 (104.27)	662.5 (304.59)	**663.0** (245.65)	**663**
	MIS	81.2 (230.45)	84.0 (16.69)	84.0 (20.92)	**84.0 (11.34)**	**84**
	MVC	210.8 (234.57)	208.0 (16.33)	208.0 (17.59)	**208.0 (8.22)**	**208**
can_161	MC	**450.4 (2.33)**	448.0 (29.48)	448.8 (39.56)	448.8 (109.05)	**456**
	MIS	**40.2** (60.69)	40.0 (2.68)	40.0 **(1.88)**	40.1 (7.12)	**41**
	MVC	**120.8** (62.19)	121.0 (2.88)	121.0 **(1.79)**	120.9 (6.01)	**120**
rdb200l	MC	441.4 (109.44)	**460.0 (11.66)**	460.0 (22.35)	460.0 (77.16)	**460**
	MIS	94.2 (108.77)	**100.0 (7.65)**	100.0 (12.01)	100.0 (28.57)	**100**
	MVC	105.8 (108.43)	**100.0 (7.73)**	100.0 (11.49)	100.0 (26.34)	**100**

The specific initial tree-width, target tree-width and the corresponding modulator sizes are given in Fig. 5. The results indicates that, mostly, smaller TMs will bring good solutions at an earlier evolution stage in terms of the number of fitness evaluations. However, smaller TMs will sometimes undermine the runtime for TD-DP due to the fact that a larger target TW will make TD-DP harder to proceed (imagine for each bag, 2^5 and 2^9 will pose a clear difference). Additionally, our model also shows a strong performance in terms of the first hitting time of the best solutions. As shown by Table. 1, our framework will quickly yield the good results in terms of wall-clock timing. And the first hitting time of the obtained best solution is superior, especially when we set the target TW to 7 and 9.

4.3 Large Instances

In this set of experiments, we extensively test the scalability of our model on medium and large instances. We give (1+1)EA and our framework **1 hour** limit. We give Gurobi **3-hour** limit, and use the best solution of the 10 trials as our truth. We also use 4 different initialization strategies, including basic feasible initialization, random feasible initialization, 30% random greedy initialization and 60% random greedy initialization, to generate different initial bit strings. We fix the **target tree-width** in our model to **5**. Our model shows a stable performance on instances with tree-width around one hundred. The solutions are mostly near-optimal. Due to the space limit, we **direct the readers to the complete Table. 5 and Table. 6 in Appendix D.3**.

References

1. Pace 2019 (track vertex cover exact), https://pacechallenge.org/2019/vc/vc_exact/, accessed: 2024-01-28
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k -tree. Siam Journal on Algebraic and Discrete Methods **8**, 277–284 (1987), https://api.semanticscholar.org/CorpusID:123254044
3. Baguley, S., Friedrich, T., Neumann, A., Neumann, F., Pappik, M., Zeif, Z.: Fixed parameter multi-objective evolutionary algorithms for the w-separator problem. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1537–1545. GECCO '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3583131.3590501, https://doi.org/10.1145/3583131.3590501
4. Baste, J., Sau, I., Thilikos, D.M.: Hitting minors on bounded treewidth graphs. i. general upper bounds. SIAM J. Discret. Math. **34**, 1623–1648 (2020), https://api.semanticscholar.org/CorpusID:225410953
5. Bodlaender, H.L.: Dynamic programming on graphs with bounded treewidth. In: International Colloquium on Automata, Languages and Programming (1988), https://api.semanticscholar.org/CorpusID:37438883
6. Bodlaender, H.L., Bonnet, É., Jaffke, L., Knop, D., Lima, P.T., Milanic, M., Ordyniak, S., Pandey, S., Suchý, O.: Treewidth is np-complete on cubic graphs. In: International Symposium on Parameterized and Exact Computation (2023), https://api.semanticscholar.org/CorpusID:266192964
7. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations i. upper bounds. Inf. Comput. **208**, 259–275 (2010), https://api.semanticscholar.org/CorpusID:722714
8. Branson, L., Sutton, A.M.: Focused jump-and-repair constraint handling for fixed-parameter tractable graph problems closed under induced subgraphs. Theor. Comput. Sci. **951**, 113719 (2023)
9. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Inf. Comput. **85**, 12–75 (1990), https://api.semanticscholar.org/CorpusID:14435655
10. Cygan, M., Fomin, F., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized algorithms. In: Cambridge International Law Journal (2015), https://api.semanticscholar.org/CorpusID:19436693
11. Cygan, M., Lokshtanov, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: On the hardness of losing width. Theory of Computing Systems **54**, 73–82 (2011), https://api.semanticscholar.org/CorpusID:15548026
12. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using networkx. In: Varoquaux, G., Vaught, T., Millman, J. (eds.) Proceedings of the 7th Python in Science Conference. pp. 11 – 15. Pasadena, CA USA (2008)
13. Kearney, J., Neumann, F., Sutton, A.M.: Fixed-parameter tractability of the (1 + 1) evolutionary algorithm on random planted vertex covers. In: FOGA. pp. 96–104. ACM (2023)
14. Kratsch, S., Lehre, P., Neumann, F., Oliveto, P.S.: Fixed parameter evolutionary algorithms and maximum leaf spanning trees: A matter of mutation. In: Parallel Problem Solving from Nature (2010), https://api.semanticscholar.org/CorpusID:12975856
15. Kratsch, S., Neumann, F.: Fixed-parameter evolutionary algorithms and the vertex cover problem. In: Proceedings of the 11th Annual Conference on Genetic and

Evolutionary Computation. pp. 293–300. GECCO '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1569901.1569943, https://doi.org/10.1145/1569901.1569943
16. Kunegis, J.: KONECT – The Koblenz Network Collection. In: Proc. Int. Conf. on World Wide Web Companion. pp. 1343–1350 (2013), http://dl.acm.org/citation.cfm?id=2488173
17. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
18. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is np-complete. J. Comput. Syst. Sci. **20**, 219–230 (1980), https://api.semanticscholar.org/CorpusID:10658756
19. Marchand, B., Ponty, Y., Bulteau, L.: Tree diet: reducing the treewidth to unlock fpt algorithms in rna bioinformatics. Algorithms for Molecular Biology : AMB **17** (2021), https://api.semanticscholar.org/CorpusID:234349139
20. Nallaperuma, S., Sutton, A.M., Neumann, F.: Fixed-parameter evolutionary algorithms for the euclidean traveling salesperson problem. In: 2013 IEEE Congress on Evolutionary Computation. pp. 2037–2044 (2013). https://doi.org/10.1109/CEC.2013.6557809
21. Rossi, R.A., Ahmed, N.K.: An interactive data repository with visual analytics. SIGKDD Explor. **17**(2), 37–41 (2016), http://networkrepository.com
22. Sutton, A.M.: Fixed-parameter tractability of crossover: Steady-state gas on the closest string problem. Algorithmica **83**, 1138–1163 (2021)
23. Sutton, A.M., Neumann, F.: A parameterized runtime analysis of simple evolutionary algorithms for makespan scheduling. In: International Conference on Parallel Problem Solving from Nature. pp. 52–61. Springer (2012)

A Proofs

A.1 Complexity Proof on Theorem 1

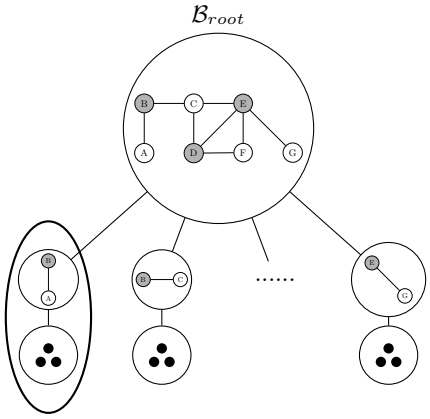


Fig. 6: ● stands for *auxiliary nodes*. The ellipse wrapping the two bags means that we are combining *auxiliary nodes* and the original leaf bag. ◍ stands for a vertex cover in the original graph. In this graph, $G = (V, E)$ is wrapped by $\mathcal{B}_{root}$, $V = \{A, B, C, D, E, F, G\}$

Proof. We firstly claim that given a certified YES/NO-instance of the problem, it is obvious that we can verify it in polynomial time. Therefore, the problem is NP. Next, we prove the completeness of the problem by the reduction from the well-known VERTEX COVER problem. For an arbitrary graph $G = (V, E)$, given a vertex cover instance (G, k), we construct a trivial bag, denoted by $\mathcal{B}_{root}$, by wrapping the entire graph into it. We then build a set of bags $B_{leaf} = \{B_1, B_2, ..., B_m\}$, where $m = |E|$. We assign only one edge e to each bag B_i. We then build a set of edges $E_{\mathcal{T}} = \{(\mathcal{B}_{root}, B_1), (\mathcal{B}_{root}, B_2), ..., (\mathcal{B}_{root}, B_m)\}$. This edge-linking process results in a trivial tree decomposition $\mathcal{T} = ((B_{root}|B_{leaf}), E_{\mathcal{T}})$. And it is clear that the trivial tree decomposition is valid, whose tree-width is $TW_{\mathcal{T}} = |\mathcal{B}_{root}| - 1$. We then fill each leaf bag B_i with $(n - k - 1)$ auxiliary nodes $\tilde{v} \notin V$. This operation will neither break the integrity of the reduction nor the validity of the trivial tree decomposition. After this operation, each leaf bag has the cardinality of $(n - k + 1)$. We now utilize the VERTEX COVER instance (G, k) as a TM of our trivial tree decomposition. For $\mathcal{B}_{root}$, we have $n - k$ nodes left. For B_{leaf}, we have either $(n - k)$ or $(n - k - 1)$ left. Therefore, the modified tree decomposition will have $TW'_{\mathcal{T}} \leq n - k - 1$. So it is obvious that every single instance (G, k) for asking *"Is there a size-k* VERTEX COVER *in G?"* always corresponds to the instance $(TW_{\mathcal{T}}, n - k - 1)$, which is asking *"Is there a* MODULATOR OF THE GIVEN TREE DECOMPOSITION*, whose size is at most k, is able to reduce the tree-width from* $(n - 1)$ *to* $(n - k - 1)$*?"*.

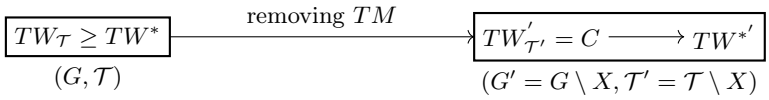


Fig. 7: The relationship between optimal and non-optimal tree-width before and after solving modulators on a given tree decomposition

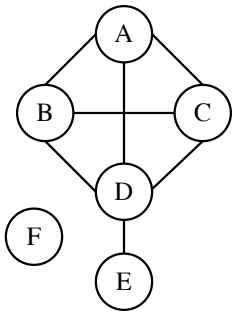
A.2 Proof of Lemma 1

Proof. (1). Based on the definition of tree-width modulator, the contradiction is easy to obtain.

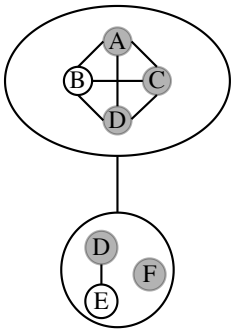
(2). We use $\mathcal{C}$ to represent the target width value η and use $\mathcal{T}$ to denote any obtained tree decomposition. No matter which tree decomposition we have, $\mathcal{C}$ is the target width value we need to reduce to. In Fig. 7, $\mathrm{TW}_{\mathcal{T}}$ represents the width of the current $\mathcal{T}$, TW^* indicates the optimal tree-width of the current graph G. By reducing $TW_{\mathcal{T}}$ to C, we remove $X \subseteq V$ from G, and at the same time, we remove the vertices from the bags of $\mathcal{T}$. We denote the tree decomposition after the deletion by $\mathcal{T}'$. Clearly, there is no guarantee on the optimality of $\mathcal{T}'$. The optimal tree-width $TW^{*\prime}$ of the current graph $G' = G \setminus X$ is clearly bounded by $\mathcal{C}$ from above. Therefore, we have $|TMGTD^*(TW_{\mathcal{T}}, C)| \geq |TM^*(TW^*, TW^{*\prime})|$. From Lem. 1, we have $|TM^*(TW^*, TW^{*\prime})| \geq |TM^*(TW^*, C)|$. Conclusion proved.

A.3 Observation 1

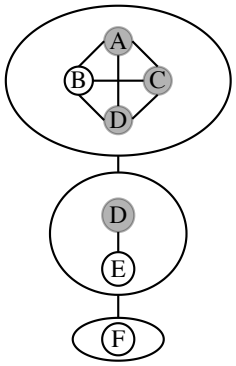
Optimally solving $TMGTD$ via optimal tree decompositions $\mathcal{T}^*$ is not a sufficient condition to access the optimal TM^*. On the other hand, optimally solving $TMGTD$ via non-optimal tree decompositions is possible to access the optimal TM^*.



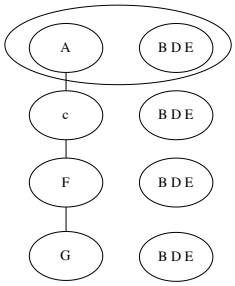
(a) $G = (V, E)$



(b) An example of optimal $\mathcal{T}_1^*$ of G. If the target $\eta = 0$, the optimal $\text{TM}^*_{\mathcal{T}_1^*} = \{A, C, D, F\}$



(c) $\mathcal{T}_2^*$ of G. If the target $\eta = 0$, the optimal $\text{TM}^*_{\mathcal{T}_2^*} = \{A, C, D\}$



(d) Left half: An optimal tree decomposition of G from the $\mathcal{B}_{root}$ in Fig. 6 (after deleting $\{B, D, E\}$). Right half: Adding $\{B, D, E\}$ back to the optimal tree decomposition.

Fig. 8: Examples to explain Obs. 1

Proof. We prove the observation by construction. Given the original graph G in Fig. 8a, we can have two different optimal tree decompositions $\mathcal{T}_1^*$ as shown in Fig. 8b and $\mathcal{T}_2^*$ as shown in Fig. 8c. Assume the target width value $\eta = 0$, for $\mathcal{T}_1^*$, the optimal solution to TMGTD is 4 vertices. However, for $\mathcal{T}_2^*$, the optimal solution is 3 vertices. For the other half of the observation, we use the original graph G in $\mathcal{B}_{root}$ from Fig. 6 to prove. Assume the target tree-width value $\eta = 0$, $\{B, D, E\}$ must be one of the optimal tree-width modulators. Therefore, there must be an optimal tree decomposition like the left side of Fig. 8d, after removing the optimal tree-width modulator $\{B, D, E\}$ from the graph G. We trivially add $\{B, D, E\}$ to each bag, we have a non-optimal tree decomposition of G.

B Experiment Notes

B.1 Data and Configurations

We implement the experiments using Python 3.10.12 on a platform equipped with Intel(R) Xeon(R) Platinum 8360Y CPUs @ 2.40GHz, hosting Red Hat Enterprise Linux release 8.4 (Ootpa). On the library front, we use NetworkX 3.2.1 and Numpy 1.26.2(C++ backend) for our codebase construction. (1+1)EA highly depends on Numpy and our model relies on NetworkX's existing MIN-DEGREE heuristic to build tree decompositions. For each run, we use one core, one thread and use 4GB as the hard limit on memory.

We use Gurobi 9.5 to compute the optimal solutions. For small instances, we allocate **10-minute** to Gurobi. For large graphs, we give Gurobi **3-hour**. Detailed Gurobi configurations are provided in **Appendix. B.5**. We also use Gurobi to solve TMGTD as a mixed integer programming, since our priority is not to research how to efficiently solve TMGTD, but rather on its theoretical implications to inform our algorithm design. For all our instances, the construction of tree decompositions, via NetworkX's MIN-DEGREE heuristic, will take **less than 0.5** seconds in average, with the longest instance vc_exact-133 last for **3.2 seconds** in average. All instances finish TMGTD stage **strictly less than 1 second**, with the longest one cost **less than 0.7 seconds**. For all our experiments and instances, we repeat for **10 trials** with different random seeds.

Our dataset includes generated graphs, real-world graphs and competition graphs from PACE[1]. The common ground of the data is that they have *not-too-large* non-optimal tree-width. The non-optimal tree-width value ranges from tens to hundreds. Details are available at Fig. 5 in Section. 4.2, and Table. 4, and Table. 2 in Appendix. D. Our generated graphs use NetworkX's built-in functions[12]. For random regular graphs, we set $seed = 100$. For *Erdős–Rényi* graphs, we use $\frac{\text{degree}}{\#\text{vertices}}$ as *the edge probability* to create graphs with low tree-width. Our large instances are accessible via [21],[17] and[16]. We preprocess graph instances to ensure graphs are undirected, and free from multi-edges and self-loops.

B.2 Problem Settings and Implementation

We use three combinatorial optimization problems to execute our experiments. They are *maximum independent set*, *minimum vertex cover* and *max cut*. Essentially, the three problems can be summarized by *how to split the vertex set into two so that the objective can be maximized or minimized*. Assume the solution set is S, the constraint for *independent set* is that for any pair of $u, v \in S$, $(u, v) \notin E$. Similarly, the constraint for *vertex cover* can be summarized by for any edge $e = (u, v) \in E$, $\{u, v\} \bigcap S \neq \emptyset$. For the *cut* problem, the requirement is to split the vertex set into two subsets S and $V \setminus S$ and count the crossing edges. Therefore, we can summarize the optimization version of the three problems by the following objectives:

- MIS: $\arg\max_{S \subseteq V} f(S) \coloneqq |\{v \in S : N_G(v) \bigcap S = \emptyset\}|$

- MVC: $\arg\min_{S\subseteq V} f(S) \coloneqq |\{e \in E : e \bigcap S \neq \emptyset\}|$
- MC: $\arg\max_{S\subseteq V} f(S) \coloneqq |\{e \in E : |e \bigcap S| = 1 \wedge |e \bigcap (V \setminus S)| = 1\}|$

We brief our implementation of (1+1)EA. For each instance, we use a bit string $\mathbf{x} = \{x_0, x_2, ..., x_{|V|-1}\}$ to represent the solution, $\{x_i \in \{0,1\} : i \in [0, ..., |V|-1] \bigcap \mathbb{N}\}$. We use S to represent the solution vertex set. If $i \in S$, $x_i = 1$, and if $i \notin S$, $x_i = 0$. We use (1+1)EA with standard bit flipping mutation using $\frac{1}{n}$ as the mutation ratio, where $n = |\mathbf{x}|$. For the three problems, we define the fitness functions as the following list. For MIS and MVC, we use set e and e_u to represent: edges in the solution set, and uncovered edges in the solution set, respectively. For MC, we define $\overline{S} = V \setminus S$. Our model and (1+1)EA share the same fitness functions in all experiments.

- $f(x)_{MIS} = -|e|$ if $|e| > 0$ else $\sum_{i\in[0,|V|-1]\bigcap\mathbb{N}} x_i$
- $f(x)_{MVC} = -|e_u| - |V|$ if $|e_u| > 0$ else $-\sum_{i\in[0,|V|-1]\bigcap\mathbb{N}} x_i$
- $f(x)_{MC} = \sum_{u\in S, v\in\overline{S}} x_u(1 - x_v)$

The initialization strategies for the bit strings used in the experiments are explained in Appendix. B.3.

B.3 Initialization Notes

In the main content of the paper, we utilize different initialization strategies in our final experiment. The *basic feasible initialization* is explained in the Section. 4.2. Here, we only explain the rest two methods. For the *random feasible initialization*, we start with bit strings of $\mathbf{0}^n$ and $\mathbf{1}^n$ for MIS and MVC respectively. For MIS, we randomly shuffle the sequence of vertices. We then decide which vertex is in the solution set and which vertex is out of the solution set. If we pick a vertex i in the solution set, we accordingly set $N(i)$ to zero in the initial string. For MVC, we randomly shuffle the sequence of edges before we iterate over the sequence. For each edge (u, v), we randomly decide to pick u or to pick v with 50% probability each. For MC, we randomly sample bit strings from $\{0,1\}^n$ as its initial string.

Regarding greedy initialization, we use two settings: randomly sampling 30% and 60% vertices from the original graphs by extracting the induced subgraphs. Next, we construct the initial solution in a greedy manner. For MIS, we use the MIN-DEGREE heuristic by always picking the vertex with the minimum degree. Tie breaks by the lexicographical order. Conversely, for MVC, we use the MAX-DEGREE heuristic by always picking the vertex with the maximum degree. For MC, we apply the *greedy vertex switching* method. We don't allow an exhaustive running of greedy vertex switching, as our research focus is not to directly compete with the limits of greedy methods. We start with a random partition A and $G \setminus A$, and give the partition one round greedy vertex switching by iterating the entire graph. We observe that given different starting bit strings, our model shows stable performance. That indicates our algorithm does not highly rely on the quality of initial solutions, it can always approach to near-optimal solutions given different types initial strings.

B.4 Tree Decomposition Operations

For reproducibility, we introduce two techniques we used in this paper. The purpose of the techniques is to simplify the tree decompositions, it does not have apparent contributions to the TD-DP performance.

The first one is outlined in Alg. 3, the idea is to eliminate subset relationship in the tree decomposition. We only present the pseudo-code in this paper, as the tree operations involved are commonly understood.

Algorithm 3: Subset Bag Elimination

```
   Input  : A tree decomposition 𝒯
 1 Q ← Queue(ℬ_root)
 2 while Q is not empty do
 3     // parent ⊂ child
 4     currentNode ← Q.top();
 5     nodeSuccessors ← getSuccessors(currentNode);
 6     newRoot ← max(nodeSuccessors, currentNode);
 7     relink(newRoot, nodeSuccessors);
 8     remove currentNode from 𝒯;
 9     Q.put(newRoot);
10     // child ⊂ parent
11     for s in getSuccessors(newRoot) do
12         if s subset newRoot then
13             relink(newRoot, getSuccessors(s));
14             delete s;
15         else
16             Q.put(s);
17 return 𝒯;
```

The second Alg. 4 is used to merge large bags in tree decomposition.

Algorithm 4: Merge Large Bags

```
   Input  : A tree decomposition 𝒯,
            k_1: absorber TW,
            k_2: absorbee TW,
 1 for bag in 𝒯 from ℬ_leaf to ℬ_root do
 2     if len(bag) > k_1 then
 3         successors ← getSuccessors(bag);
 4         bag ← ⋃ all successors whose length ≥ k_2;
 5         Remove all absorbed successors;
 6 return 𝒯;
```

The above two operations will not break the validity of tree decompositions. The purpose of these operations is to reduce the number of bags of the tree decompositions, and avoid *unnecessary* replicated computings.

B.5 Gurobi Configuration

```
import gurobipy as gp
gp.setParam(GRB.Param.Threads, 1)
gp.setParam("LogToConsole", 0)
```

C Algorithms From Main Paper

C.1 (1+1)EA

Algorithm 5: (1+1) EA

Input : A fitness function $f : \{0,1\}^n \to \mathbb{R}$

Choose x uniformly at random from $\{0,1\}^n$;
while not *terminate criteria* **do**
    $y = \text{BitwiseFlipping}(x, \frac{1}{n})$;
    if $f(y) \geq f(x)$ **then**
        $x \leftarrow y$;
return x;

C.2 Tree Decomposition based Dynamic Programming

Algorithm 6: Tree decomposition dynamic programming

Input : Graph $G = (V, E)$, a tree decomposition $\mathcal{T}$

for $\mathcal{B}_t$ *in BFSTraversal*$(\mathcal{B}_{leaf} : \mathcal{B}_{root})$ **do**
    $\mathcal{B}_p \leftarrow \text{predecessor}(\mathcal{B}_t)$;
    $\mathcal{B}_c \leftarrow \{\mathcal{B}_{c_1}, \mathcal{B}_{c_2}..\mathcal{B}_{c_k} \mid |\text{successors}(\mathcal{B}_t)| = k\}$;
    $\mathcal{S}_t \leftarrow$ all **valid** local states of $\mathcal{B}_t$;
    for s *in* $\mathcal{S}_t$ **do**
        $X(t, s \bigcap \mathcal{B}_{c_j}) = |s| + \sum_j (Y(c_j, t, s \bigcap \mathcal{B}_{c_j}) - |s \bigcap \mathcal{B}_{c_j}|)$;
        $Y(t, p, s \bigcap \mathcal{B}_p) = \max X(t, s)$
return $\max_{s \in S_{root}} X(root, s)$;

D Full Tables References

D.1 Graph Information

Instance	(|V|, |E|)	TW (MIN-DEGREE)	$\|TM_5\|$
bio-CE-HT	(2617, 2985)	101	123
bio-yeast-protein-inter	(1870, 2203)	51	53
ca-Erdos992	(5094, 7515)	152	215
er-1000-3-100	(957, 1476)	127	159
hep-th	(7610, 15751)	367	606
inf-power	(4941, 6594)	25	77
power-bcspwr09	(1723, 2394)	15	34
rajat06	(10922, 18061)	181	1674
roadminnesota	(2642, 3303)	31	87
rt-http	(8917, 10314)	62	94
shar_te2-b1	(17160, 34314)	218	219
tech-routers-rf	(2113, 6632)	178	298
vc-exact_032	(1490, 2680)	107	141
vc-exact_131	(2980, 5360)	213	297
vc-exact_133	(15783, 24663)	178	248
vc-exact_157	(2982, 5361)	218	303
vc-exact_175	(3523, 6446)	232	311
web-edu	(3031, 6474)	29	87

Table 2: This table demonstrates the information of all large instances in this paper. The first column displays (number of nodes, number of edges). The second column shows tree-widths calculated by MIN-DEGREE from NetworkX. The third column is for the cardinality of TMGTD calculated by integer programming. We fix the target width $\eta = 5$.

Instance	(|V|, |E|)
DD199	(841, 1902)
dwt_307	(307, 1108)
ia-enron-only	(143, 623)
power-685-bus	(685, 1282)
saylr1	(238, 445)
ash292	(292, 958)
can_161	(161, 608)
rdb200l	(200, 246)

Table 3: Small instances information. The tree-width values and modulator values have been provided in Fig. 5

D.2 Experiments for Standalone TD-DP

Table 4: Performance report of *standalone* TD-DP.

(Type, |V|, |E|, min-degree TW)	Problem	Runtime
(3^5-reg, 50, 75, **9**)	mis	0.0268797238667806
	mvc	0.02666271527608236
	mc	0.02666271527608236
(3-reg, 60, 90, **12**)	mis	0.21498955090840657
	mvc	0.25149513880411783
	mc	0.5755066553751628
(5-reg, 70, 175, **24**)	mis	OOT[6]
	mvc	OOT
	mc	OOT
($\frac{5}{60}$-ER, 60, 146, **18**)	mis	3.9180885791778564
	mvc	2.1841339270273843
	mc	56.816769981384276
($\frac{6}{80}$-ER, 80, 237, **29**)	mis	OOT
	mvc	OOT
	mc	OOT

D.3 Experiments on Large Instances

Due to resource constraints, we use the statistics produced by (1+1)EA every **100 steps**. It is the reason that *the average number of fitness evaluations* for (1+1)EA in the following tables are *always zeros* after the decimal points. On the other hand, we use the statistics of our algorithm **step by step**. We emphasize one simple fact that the downsampling applied to (1+1)EA will not give our algorithm any comparative advantage. **Conversely, it emphasizes the growth rate of (1+1)EA.** Nonetheless, the difference on presenting data will not affect the results, as we use the same timeout period as the cut-off for both methods.

[5] Number prefix is the degree number.
[6] The abbreviation of Out-Of-Time

Table 5: **1/2** Table for medium to large instances. In this set of experiments, basic feasible initialization and random feasible initialization strategies are used for the starting bit strings. For each problem, each instance is run by 10 trials. The hard cut-off time is 1 hour for both methods. We are showing *the average solution size* and *the average number of fitness evaluations* in parentheses. In this set of experiments, we fix the target tree-width to 5.

Instance	Problem	Basic Feasible Start		Random Feasible Start		Gurobi (3 hours)
		(1+1)EA	Zoom-in	(1+1)EA	Zoom-in	
bio-CE-HT	MC	2734.2(2450339.0)	**2850.2(23082.5)**	2740.3(2422809.0)	**2854.0(20233.3)**	**2859***
	MIS	1689.9(2592659.0)	***1801.0(19977.2)**	1679.8(2646069.0)	***1801.0(19785.0)**	**1801***
	MVC	926.0(2628249.0)	***816.0(20069.8)**	866.1(2695079.0)	***816.0(19521.2)**	**816***
bio-yeast-protein-inter	MC	1953.7(3513519.0)	**2028.2(37635.7)**	1944.7(3401269.0)	**2024.5(33753.5)**	**2029***
	MIS	1208.4(3744519.0)	***1244.0(35583.7)**	1201.5(3587389.0)	***1244.0(35263.6)**	**1244***
	MVC	660.8(3706189.0)	***626.0(32553.4)**	650.4(3671899.0)	***626.0(32800.4)**	**626***
ca-Erdos992	MC	6361.4(1201749.0)	**6762.7(7673.0)**	6357.7(1189699.0)	**6708.3(5545.6)**	**6765***
	MIS	4499.9(1294479.0)	***4633.0(8077.0)**	4411.3(1274659.0)	***4633.0(8420.3)**	**4633***
	MVC	593.4(1306339.0)	***461.0(10905.9)**	472.9(1351799.0)	***461.0(10685.0)**	**461***
er-1000-3-100	MC	1292.9(6056739.0)	**1325.9(43051.2)**	1290.8(5703969.0)	**1323.4(41463.9)**	**1333***
	MIS	468.7(5890769.0)	***483.0(72296.4)**	467.5(5948239.0)	***483.0(68097.4)**	**483***
	MVC	488.5(5892769.0)	***474.0(65785.5)**	485.7(6133199.0)	***474.0(65931.9)**	**474***
hep-th	MC	11249.7(627039.0)	**11428.8(4744.9)**	11246.3(624679.0)	**11432.1(4626.1)**	**11488***
	MIS	3263.5(654969.0)	**3682.1(7382.7)**	3222.8(659929.0)	**3681.3(7231.5)**	**3684***
	MVC	4346.5(651769.0)	**3928.3(6984.0)**	4091.6(672589.0)	**3927.5(7003.2)**	**3926***
inf-power	MC	5743.1(1229469.0)	**5956.2(5368.5)**	5752.4(1231579.0)	**5956.0(5467.5)**	**5958***
	MIS	2432.5(1309689.0)	***2738.0(6001.5)**	2403.2(1328039.0)	***2738.0(5987.6)**	**2738***
	MVC	2508.6(1287509.0)	***2203.0(6012.5)**	2401.3(1356519.0)	***2203.0(5825.7)**	**2203***
power-bcspwr09	MC	2092.7(3445499.0)	***2153.0(15243.3)**	2086.0(3346419.0)	***2153.0(14687.4)**	**2153***
	MIS	894.5(3471819.0)	***948.0(17107.2)**	891.0(3408729.0)	***948.0(16908.3)**	**948***
	MVC	827.8(3502789.0)	***775.0(16735.1)**	812.5(3416689.0)	***775.0(16534.9)**	**775***
rajat06	MC	14035.8(503339.0)	**14422.6(3533.8)**	14039.6(509869.0)	**14076.6(2826.6)**	**14461***
	MIS	*3661.0(539549.0)	***3661.0(3694.3)**	*3661.0(538209.0)	***3661.0(3880.8)**	**3661***
	MVC	*7261.0(550569.0)	***7261.0(3476.9)**	*7261.0(559849.0)	***7261.0(3488.4)**	**7261***
roadminnesota	MC	3029.0(2348479.0)	**3101.2(9432.0)**	3029.7(2409839.0)	**3101.2(9130.3)**	**3103***
	MIS	1165.0(2457899.0)	***1323.0(10012.2)**	1160.5(2500929.0)	***1323.0(9516.5)**	**1323***
	MVC	1477.1(2416329.0)	***1319.0(9342.3)**	1449.7(2505329.0)	***1319.0(9069.3)**	**1319***
rt-http	MC	9657.3(805939.0)	**10185.7(669.6)**	9661.5(820619.0)	**10157.3(676.8)**	**10192***
	MIS	8145.0(845879.0)	**8293.7(1177.8)**	8031.5(886459.0)	**8293.2(1137.6)**	**8294***
	MVC	772.0(906889.0)	***623.0(2993.2)**	672.8(910459.0)	***623.0(3130.2)**	**623***
tech-routers-rf	MC	4986.8(1761499.0)	**5093.5(11314.8)**	4979.1(1753399.0)	**5069.1(9267.1)**	**5096***
	MIS	1239.5(1742659.0)	***1318.0(27040.5)**	1232.0(1762949.0)	***1318.0(26043.7)**	**1318***
	MVC	872.6(1779029.0)	***795.0(26219.1)**	834.4(1750319.0)	***795.0(25314.6)**	**795***
shar_te2-b1	MC	26628.4(312609.0)	**33707.8(467.2)**	25625.1(286859.0)	**29982.0(399.8)**	**33748***
	MIS	16739.7(286789.0)	**16829.9(412.4)**	16628.0(284149.0)	**16767.1(411.7)**	**16874***
	MVC	414.6(299829.0)	**309.8(2043.5)**	287.1(285919.0)	***286.0(2136.2)**	**286***
vc-exact_032	MC	2140.3(3574129.0)	**2223.4(23916.2)**	2136.6(3433199.0)	**2225.2(23959.4)**	**2230***
	MIS	489.1(3505459.0)	**527.8(43553.1)**	490.8(3541269.0)	**527.7(41447.8)**	**530***
	MVC	1000.1(3566909.0)	**962.4(37957.1)**	1005.7(3390599.0)	**962.8(39956.0)**	**960***
vc-exact_131	MC	4259.8(1795859.0)	**4439.9(10444.6)**	4259.4(1780009.0)	**4442.3(10657.3)**	**4451***
	MIS	952.1(1814189.0)	**1051.0(20982.5)**	947.2(1802379.0)	**1051.2(21445.0)**	**1060***
	MVC	2027.9(1791349.0)	**1929.3(18305.1)**	2040.7(1836899.0)	**1927.9(18971.2)**	**1920***
vc-exact_133	MC	20002.3(362629.0)	**20503.5(2130.9)**	20021.5(349759.0)	**20496.3(2088.7)**	-
	MIS	5753.9(374409.0)	**6014.1(2134.7)**	5741.4(359179.0)	**6012.9(2115.1)**	-
	MVC	10029.1(371989.0)	**9770.0(1852.2)**	10147.2(360439.0)	**9767.3(1867.9)**	**9755***
vc-exact_157	MC	4262.5(1729449.0)	**4440.9(10794.6)**	4261.2(1776499.0)	**4442.7(10231.1)**	**4453***
	MIS	950.1(1760999.0)	**1052.0(21138.9)**	948.9(1775909.0)	**1051.5(20218.4)**	**1061***
	MVC	2031.7(1751769.0)	**1930.7(18378.7)**	2048.3(1780309.0)	**1930.6(18870.4)**	**1921***
vc-exact_175	MC	5109.3(1507689.0)	**5331.0(9024.1)**	5105.2(1467169.0)	**5326.8(8697.3)**	**5374***
	MIS	1109.1(1463509.0)	**1228.3(17308.7)**	1111.4(1563999.0)	**1227.0(16560.9)**	**1241***
	MVC	2413.9(1506459.0)	**2294.8(15748.0)**	2433.4(1560139.0)	**2296.1(14564.2)**	**2282***
web-edu	MC	4569.4(1641159.0)	***4743.0(11349.4)**	4564.8(1635199.0)	***4743.0(11377.2)**	**4743***
	MIS	1438.0(1714379.0)	***1580.0(12805.8)**	1431.6(1645609.0)	***1580.0(13037.9)**	**1580***
	MVC	1592.9(1727129.0)	***1451.0(11186.7)**	1577.2(1668579.0)	***1451.0(11192.7)**	**1451***

Table 6: **2/2** Table for medium to large instances. In this set of experiments, random greedy heuristics are used on 30% and 60% of the instances to construct the starting bit strings. For each problem, each instance is run by 10 trials. The hard cut-off time is 1 hour for both methods. We are showing *the average solution size* and *the average number of fitness evaluations* in parentheses. In this set of experiments, we fix the target tree-width to 5.

Instance	Problem	Random Greedy Start (0.3)		Random Greedy Start (0.6)		Gurobi (3 hours)
		(1+1)EA	Zoom-in	(1+1)EA	Zoom-in	
bio-CE-HT	MC	2738.4(2524949.0)	**2852.8(22879.3)**	2738.8(2530429.0)	**2854.6(22291.9)**	**2859***
	MIS	1714.1(2749189.0)	***1801.0(21363.5)**	1738.4(2722639.0)	***1801.0(20721.5)**	**1801***
	MVC	892.3(2777489.0)	***816.0(21051.2)**	864.1(2804149.0)	***816.0(20698.3)**	**816***
bio-yeast-protein-inter	MC	1949.5(3605039.0)	**2025.8(34370.9)**	1954.8(3554839.0)	**2024.9(36070.5)**	**2029***
	MIS	1208.4(3811129.0)	***1244.0(38129.5)**	1217.5(3823429.0)	***1244.0(38080.1)**	**1244***
	MVC	653.4(3867399.0)	***626.0(34521.7)**	643.4(3876769.0)	***626.0(33902.8)**	**626***
ca-Erdos992	MC	6375.0(1279819.0)	**6734.6(6328.9)**	6352.5(1282259.0)	**6688.8(6212.7)**	**6765***
	MIS	4573.7(1364599.0)	***4633.0(8564.9)**	4614.3(1353269.0)	***4633.0(8686.9)**	**4633***
	MVC	496.9(1395949.0)	***461.0(11534.4)**	470.0(1395219.0)	***461.0(11369.9)**	**461***
er-1000-3-100	MC	1296.7(6173589.0)	**1326.9(43215.1)**	1297.9(6212589.0)	**1322.3(43204.2)**	**1333***
	MIS	469.4(6093319.0)	***483.0(76423.2)**	470.3(6209959.0)	***483.0(76691.1)**	**483***
	MVC	489.4(6134579.0)	***474.0(70014.7)**	487.5(6176189.0)	***474.0(68757.7)**	**474***
hep-th	MC	11248.5(658159.0)	**11430.2(4939.4)**	11250.4(650589.0)	**11437.2(4940.3)**	**11488***
	MIS	3306.0(681699.0)	**3681.8(7830.2)**	3436.7(678079.0)	**3682.5(8033.4)**	**3684***
	MVC	4201.9(683729.0)	**3928.4(7540.7)**	4063.6(689279.0)	**3927.3(7667.3)**	**3926***
inf-power	MC	5745.7(1278539.0)	**5955.8(5748.2)**	5755.0(1277559.0)	**5956.4(5931.6)**	**5958***
	MIS	2433.8(1332859.0)	***2738.0(6444.0)**	2511.9(1341869.0)	***2738.0(6669.4)**	**2738***
	MVC	2460.3(1338149.0)	***2203.0(6294.0)**	2373.3(1352269.0)	***2203.0(6461.4)**	**2203***
power-bcspwr09	MC	2091.2(3555789.0)	***2153.0(15514.9)**	2092.9(3574179.0)	***2153.0(15453.6)**	**2153***
	MIS	894.9(3627459.0)	***948.0(18471.7)**	903.1(3634959.0)	***948.0(18025.0)**	**948***
	MVC	823.6(3602839.0)	***775.0(17435.0)**	809.9(3678909.0)	***775.0(17335.2)**	**775***
rajat06	MC	14053.6(526579.0)	**14089.5(3014.1)**	14050.0(528409.0)	**14100.7(3096.7)**	**14461***
	MIS	*3661.0(559679.0)	***3661.0(4136.0)**	*3661.0(552269.0)	***3661.0(4157.4)**	**3661***
	MVC	*7261.0(557509.0)	***7261.0(3646.2)**	*7261.0(567359.0)	***7261.0(3685.5)**	**7261***
roadminnesota	MC	3030.7(2477129.0)	**3100.9(9819.2)**	3028.6(2469239.0)	**3100.6(9938.4)**	**3103***
	MIS	1169.1(2581079.0)	***1323.0(10123.8)**	1200.8(2571099.0)	***1323.0(10731.0)**	**1323***
	MVC	1465.6(2547909.0)	***1319.0(9913.3)**	1438.4(2572289.0)	***1319.0(10251.4)**	**1319***
rt-http	MC	9688.7(858149.0)	**10159.4(691.8)**	9700.7(845069.0)	**10160.4(686.7)**	**10192***
	MIS	8205.3(909599.0)	**8293.9(1223.7)**	8244.7(912289.0)	***8294.0(1290.2)**	**8294***
	MVC	693.1(940749.0)	***623.0(3285.0)**	653.4(935569.0)	***623.0(3339.9)**	**623***
tech-routers-rf	MC	4982.2(1822439.0)	**5069.9(9623.0)**	4976.9(1815419.0)	**5070.8(9745.7)**	**5096***
	MIS	1250.5(1808739.0)	***1318.0(28436.7)**	1270.6(1802519.0)	***1318.0(28931.6)**	**1318***
	MVC	851.6(1814739.0)	***795.0(27275.6)**	828.4(1825809.0)	***795.0(27270.5)**	**795***
shar_te2-b1	MC	26140.2(312439.0)	**30030.5(404.7)**	26044.1(312809.0)	**29854.1(409.0)**	**33748***
	MIS	**16863.9(300049.0)**	16850.2(424.9)	**16864.5(294699.0)**	16850.2(422.4)	**16874***
	MVC	295.2(303699.0)	***286.0(2224.0)**	293.9(300729.0)	***286.0(2248.0)**	**286***
vc-exact_032	MC	2139.5(3754099.0)	**2225.5(24736.9)**	2135.7(3758039.0)	**2225.5(24935.3)**	**2230***
	MIS	491.7(3728229.0)	**527.1(46916.7)**	493.7(3653409.0)	**527.4(46615.9)**	**530***
	MVC	1000.9(3718809.0)	**962.8(41528.7)**	1003.1(3761549.0)	**962.9(41361.8)**	**960***
vc-exact_131	MC	4258.3(1842219.0)	**4439.6(11010.4)**	4260.7(1847549.0)	**4442.0(10991.7)**	**4451***
	MIS	955.1(1885119.0)	**1052.0(22390.2)**	970.2(1838739.0)	**1052.0(22526.3)**	**1060***
	MVC	2038.5(1878189.0)	**1928.2(20088.8)**	2033.5(1888289.0)	**1928.3(19092.5)**	**1920***
vc-exact_133	MC	20026.6(383119.0)	**20499.6(2258.4)**	20008.8(382189.0)	**20500.3(2189.4)**	-
	MIS	5785.4(386449.0)	**6014.6(2237.8)**	5862.3(384649.0)	**6014.0(2237.4)**	-
	MVC	10038.0(390979.0)	**9768.8(1984.0)**	10098.3(396119.0)	**9769.4(1962.1)**	**9755***
vc-exact_157	MC	4266.1(1850839.0)	**4442.8(11079.0)**	4259.7(1817359.0)	**4442.0(11145.2)**	**4453***
	MIS	953.1(1875979.0)	**1051.8(22545.7)**	973.3(1823299.0)	**1052.4(22793.0)**	**1061***
	MVC	2039.9(1870479.0)	**1931.0(20235.1)**	2041.1(1891029.0)	**1929.8(20527.0)**	**1921***
vc-exact_175	MC	5107.9(1569649.0)	**5338.2(9336.3)**	5109.9(1572659.0)	**5329.7(9285.2)**	**5374***
	MIS	1115.9(1589069.0)	**1228.3(19190.1)**	1134.3(1567839.0)	**1229.4(18301.4)**	**1241***
	MVC	2424.9(1589839.0)	**2295.2(16552.5)**	2419.5(1626079.0)	**2293.9(16318.8)**	**2282***
web-edu	MC	4572.1(1735859.0)	***4743.0(12069.0)**	4568.4(1726439.0)	***4743.0(12386.3)**	**4743***
	MIS	1445.9(1742339.0)	***1580.0(13364.8)**	1457.8(1747829.0)	***1580.0(13973.9)**	**1580***
	MVC	1586.6(1728789.0)	***1451.0(11916.9)**	1581.4(1736289.0)	***1451.0(12262.2)**	**1451***