A Proofs

A.1 Complexity Proof on Theorem 1

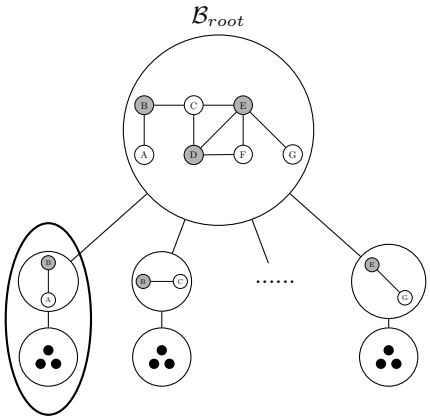


Fig. 6: ● stands for *auxiliary nodes*. The ellipse wrapping the two bags means that we are combining *auxiliary nodes* and the original leaf bag. ◍ stands for a vertex cover in the original graph. In this graph, $G = (V, E)$ is wrapped by $\mathcal{B}_{root}$, $V = \{A, B, C, D, E, F, G\}$

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* $(n - k - 1)$*?"*.

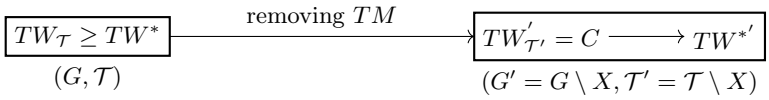


Fig. 7: The relationship between optimal and non-optimal tree-width before and after solving modulators on a given tree decomposition

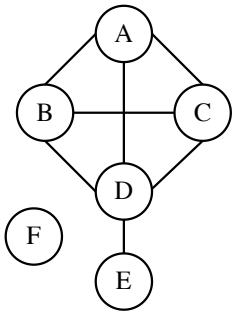
A.2 Proof of Lemma 1

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

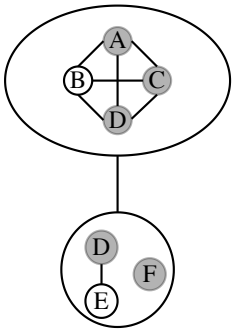
(2). We use $\mathcal{C}$ to represent the target width value η and use $\mathcal{T}$ to denote any obtained tree decomposition. No matter which tree decomposition we have, $\mathcal{C}$ is the target width value we need to reduce to. In Fig. 7, $\mathrm{TW}_{\mathcal{T}}$ represents the width of the 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

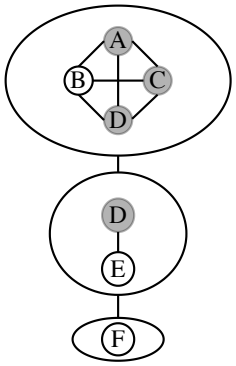
Optimally solving $TMGTD$ via optimal tree decompositions $\mathcal{T}^*$ is not a sufficient condition to access the optimal TM^*. On the other hand, optimally solving $TMGTD$ via non-optimal tree decompositions is possible to access the optimal TM^*.



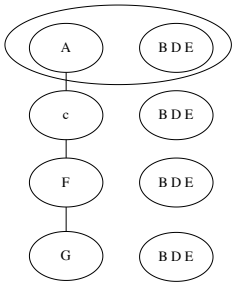
(a) $G = (V, E)$



(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***