# Solving Maximum Weight Independent Set in Practice Using Tree Decompositions

BJARN DE JONG

We present a C++ implementation of an exact solver for the Maximum Weight Independent Set problem using dynamic programming over tree decompositions. Given a graph $G$ and a tree decomposition of width $k-1$, the algorithm runs in time $O(n \cdot k \cdot 2k \cdot 2^k)$, where $n$ is the number of vertices in the graph. The dominant cost arises from enumerating independent sets within each bag, which in the worst case can be as many as $2^k$ per bag. However, in practice, the number of independent sets encountered is typically much smaller. Our implementation reliably solves instances of graphs with tree decompositions for which the maximum bag size $k$ is less than 32, where memory usage remains practical. Although our encodings support tree decompositions with maximum bag size $k$ less than 128 as a data type constraint, memory demands generally limit effective computation to lower widths. Experimental evaluation confirms the solver's effectiveness on graphs with tree decompositions of low width.

## 1 Introduction

In this work, we give an approach for exactly solving the Maximum Weight Independent Set problem. Given an undirected graph $G = (V, E)$, where $V$ is its set of vertices and $E$ is its set of edges, a subset of vertices $S \subseteq V$ is an *independent set* if the elements of $S$ are pairwise nonadjacent in $G$. If $w(v)$ denotes the weight of vertex $v \in V$, the weight of an independent set $S$ is $w(S) = \sum_{v \in S} w(v)$. In the Maximum Weight Independent Set problem (MWIS) we are interested in finding an independent set $S$ such that $w(S) \geq w(S')$ for all independent sets $S'$. This problem is known to be NP-hard in general graphs.

Our approach is based on the concept of dynamic programming over tree decompositions. A tree decomposition is a cover of the vertex set satisfying certain structural properties, amongst which is the requirement that the elements of the cover form a tree. The tree structure facilitates a bottom-up computation where at any given point the problem is restricted to the induced subgraph of the current cover element. This leads to a parameterized algorithm, where the running time is exponential in the width of the decomposition but polynomial in the size of the input graph.

## 2 Preliminaries

The material in this section is adapted from Cygan et al. [4].

### 2.1 Separators

*Definition 2.1.* A *separation* of $G$ is a pair $(A, B)$, where $A, B \subseteq V(G)$, such that $A \cup B = V(G)$, and for which there are no edges from $A \setminus B$ to $B \setminus A$. The intersection $A \cap B$ is then called the *separator*.

For two independent sets $S$ of $G[A]$ and $S'$ of $G[B]$ we say that they *agree* on the separator if $S \cap (A \cap B) = S' \cap (A \cap B)$. When they agree, their union $S \cup S'$ is an independent set of $G$.

Figure 1 illustrates a separation of a graph, its separator $A \cap B$, and two independent sets that agree on the separator.

### 2.2 Tree Decompositions

*Definition 2.2.* A *tree decomposition* $TD$ of a graph $G$ is a tuple $(T, \{X_t\}_{t \in N(T)})$, where $T$ is a tree, the vertices of which are referred to as *nodes* and the set of which is denoted by $N(T)$, together with a family of *bags* $\{X_t\}_{t \in N(T)}$, such that for every node $t \in N(T)$, we have $X_t \subseteq V(G)$, satisfying the following requirements:

Author's address: Bjarn de Jong, bjarrn@gmail.com.

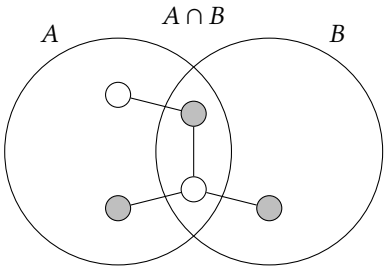



Fig. 1. Separation and separator

- *Vertex coverage*: For every $u \in V(G)$, there exists a node $t \in N(T)$ such that $u \in X_t$.
- *Edge coverage*: For every edge $\{u, v\} \in E(G)$, there exists a node $t \in N(T)$ such that $\{u, v\} \subseteq X_t$.
- *Running intersection property*: For every $u \in V(G)$, the induced subgraph $T[\{t \in N(T) \mid u \in X_t\}]$ is a subtree of $T$.

The *width* of a tree decomposition $TD$ is defined as $\mathrm{wd}(TD) = \max\{|X_t| \mid t \in N(TD)\} - 1$, and the *treewidth* of a graph $G$ is defined as $\mathrm{twd}(G) = \min\{\mathrm{wd}(TD) \mid TD \text{ is a tree decomposition of } G\}$.

Lemma 2.3. *For every pair of adjacent nodes $t, t' \in N(T)$, we have that $X_t \cap X_{t'}$ is a separator of the graph $G$.*

### 2.3 Rooted Tree Decompositions

We choose an arbitrary node as the *root* for the tree and will henceforth only work with *rooted tree decompositions*. The descendency relation induces a partial order on the nodes of the tree, where for $t, t' \in N(T)$ we write $t' \leq t$ if and only if $t'$ is a descendent of $t$. We will consider separations of the form

$$(A, B) = (G[\bigcup_{a \leq t'} X_a], G[\bigcup_{b \not\leq t'} X_b])$$

with separator $X_{t'} \cap X_t$, where $t$ is the parent of $t'$. For a node $t$, we write $G_t$ for the induced subgraph $G[\bigcup_{t' \leq t} X_{t'}]$ and $V_t$ for $V(G_t)$.

*Definition 2.4.* Given a rooted tree decomposition $TD$ of a graph $G$, we define for every node $t$ and every independent set $S \subseteq X_t$

$$c[t, S] = \max \left\{ w(\widehat{S}) \;\middle|\; \begin{array}{l} S \subseteq \widehat{S} \subseteq V_t, \\ \widehat{S} \text{ is an independent set,} \\ X_t \cap \widehat{S} = S \end{array} \right\}.$$

The aim will be to traverse up the tree and for each node $t \in N(T)$ and each independent set $S \subseteq X_t$ to compute $c[t, S]$. Finally, upon reaching the root, the weight of a MWIS will be

$$\max\{c[root, S] \mid S \subseteq X_{root}, S \text{ independent set}\}.$$

### 2.4 Nice Tree Decompositions

*Definition 2.5.* A rooted tree decomposition is called *nice* if it is a binary tree satisfying the following conditions:

- The *leaf nodes* and the *root* satisfy:
  - For every leaf node $l$, we have $X_l = \emptyset$.
  - For the root node $r$, we also have $X_r = \emptyset$.

- Every *non-leaf node* is one of the following three types:
  - *Introduce node*: a node $t$ with exactly one child $t'$, such that $X_t = X_{t'} \sqcup \{v\}$ for some vertex $v \in V$.
  - *Forget node*: a node $t$ with exactly one child $t'$, such that $X_t \sqcup \{v\} = X_{t'}$ for some vertex $v \in V$.
  - *Join node*: a node $t$ with exactly two children $t_1$ and $t_2$, such that $X_t = X_{t_1} = X_{t_2}$.

*Computation of $c[\cdot,\cdot]$ in a Nice Tree Decomposition* For a nice tree decomposition, the values $c[\cdot,\cdot]$ can be computed in a bottom-up manner as follows:

- For every leaf node $l$:
  - We have $X_l = \emptyset$, so:
$$c[l, \emptyset] = 0.$$
- For non-leaf nodes:
  - Introduce node: Let $t$ be an introduce node where vertex $v$ is introduced, and let $t'$ be its child. Then:
$$c[t,S] = \begin{cases} c[t',S], & \text{if } v \notin S, \\ c[t', S \setminus \{v\}] + w(v), & \text{if } v \in S. \end{cases}$$
  - Forget node: Let $t$ be a forget node where vertex $v$ is forgotten, and let $t'$ be its child. Then:
$$c[t,S] = \begin{cases} \max\{c[t',S],\ c[t', S \sqcup \{v\}]\}, & \text{if } S \sqcup \{v\} \text{ is independent,} \\ c[t',S], & \text{otherwise.} \end{cases}$$
  - Join node: Let $t$ be a join node with children $t_1$ and $t_2$. Then:
$$c[t,S] = c[t_1,S] + c[t_2,S] - w(S).$$

## 3 Algorithm

Instead of working with nice tree decompositions we will work with arbitrary rooted tree decompositions and borrow concepts from nice tree decompositions. The overall traversal of the tree decomposition is implemented using a depth-first traversal. During the traversal, we compute dynamic programming tables. Several vectors are used to represent the necessary information at each node. Most notably, we represent all independent sets local to a bag using natural numbers, where each number encodes a subset via its binary representation. These encoded sets are manipulated efficiently using bitwise operations and basic arithmetic.

### 3.1 Input

As input the program takes two arguments, a .td file representing the tree decomposition and a .graph file representing the vertex-weighted graph.

*.graph file* The .graph file consist of a first line `n m 10`, where $n$ is the number of vertices, $m$ is the number of edges and 10 indicates that vertices have weights. The vertices are 1-indexed and thus make up the range $[1, n]$. This is then followed by $n$ lines of the form `w(i) N(i)`, where `w(i)` is the weight of the vertex $i$ and where `N(i)` is the neighborhood of vertex $i$ listed in ascending order.

*.td file* The .td file format is as specified in the PACE 2017 Treewidth track [3]. Additionally, we require that the bags are sorted, that is, $X_i$ comes before $X_j$ if $i < j$ and the contents of the bags are sorted in ascending order. We also require the edge list to be sorted in ascending order where $(i < j) < (k < l)$ if $i < k$ or $i = k$ and $j < l$.

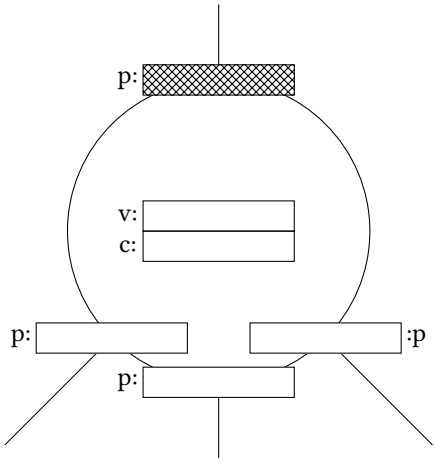



Fig. 2. Storage of the data structure locally at a node $t$.

### 3.2 Data Structure

*Encoding* Per node $t$, enumerate all independent sets local to the bag $X_t$ encoded as integers in the range $[0, 2^{|X_t|} - 1]$. The encoding is given such that the $i$-th bit in the binary representation of an independent set is set to 1 if and only if the $i$-th vertex of the bag is included in the independent set.

*Storage* Let $t$ be a node with bag $X_t$. Then the data structure used locally for $t$ consists of several vectors, all of which have the same length. The vector v contains all the independent sets $S \subseteq X_t$ encoded as integers in ascending order. The vector c is to contain the values $c[t, \cdot]$ from Definition 2.4. For each child $t'$ of $t$, we have a vector p to track how each independent set at $t$ relates to a corresponding independent set at $t'$. Figure 2 illustrates the local data structure at a node $t$.

### 3.3 Computation

As mentioned above, instead of working with a nice tree decomposition, we work with an arbitrary rooted tree decomposition. The depth-first traversal determines the order in which computations occur. Finishing a node involves three steps: initializing the node if it is a leaf, updating the current node's information to account for multiple children, and pushing the relevant information from the current node to its parent.

*Updating to the Parent Node* To update information from a current node $t'$ to its parent $t$, we traverse a *virtual path* consisting of forget nodes until we reach the intersection of the bags $X_t$ and $X_{t'}$, followed by a virtual path of introduce nodes until $X_t$ is reached. In particular, assuming $X_t \not\subseteq X_{t'}$ and $X_{t'} \not\subseteq X_t$, the virtual node corresponding to the intersection $X_t \cap X_{t'}$ acts as a forget node, while $t$ itself is considered an introduce node in this virtual path (see Figure 3).

Computing the vectors c upward follows the approach used in nice tree decompositions, except that $c[t, S]$ is initialized to zero, and the contributions from the virtual path at $t$ are added rather than assigned directly. Traversing this virtual path consists of *steps*, where a step either forgets a vertex at the $i$-th position from the bag or introduces a vertex at the $i$-th position to the bag. Figure 4 illustrates the behavior of these virtual steps for forgetting and introducing a vertex. For the purpose of computing our vectors v, c and p from the data structure, whilst forgetting the $i$-th vertex $v$ from the bag, every independent $S$ not containing $v$ is maintained and we only need to consider whether or not $S \sqcup \{v\}$ was also an independent set. In the case of introducing an $i$-th

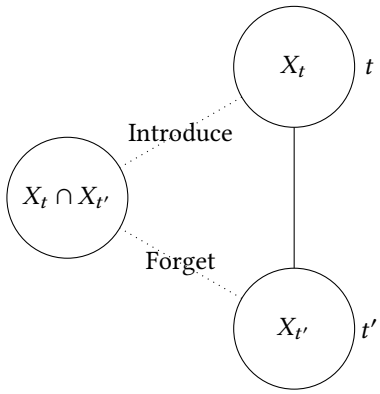
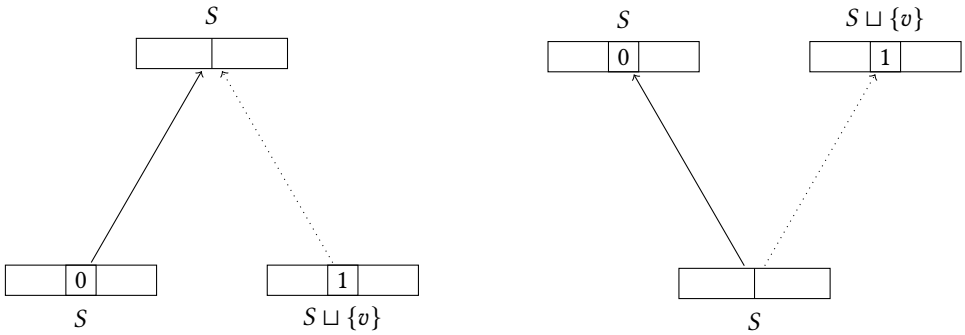



Fig. 3. Virtual path from node $t'$ to its parent $t$




(a) Virtual step forget (b) Virtual step introduce

Fig. 4. Treatment of independent sets during the traversal of the virtual path using the binary encodings.

vertex $v$ to the bag, every independent set $S$ is maintained, and we only need to consider whether or not such an independent set gives rise to an additional independent set $S \sqcup \{v\}$.

*Updating the Current Node* Since our tree decomposition nodes may have arbitrary degrees (no join nodes), we need to subtract the contribution of a set $S$ from $c[t, S]$ the appropriate number of times. Unless $t$ is the root (in which case the factor is 1 instead of 2), this update can be expressed as

$$c[t, S] = c[t, S] - (\deg(t) - 2) \cdot w(S),$$

where $\deg(t)$ denotes the degree of node $t$ and $w(S)$ the weight of set $S$.

*Initializing Leaf Nodes* Leaf initialization is a special case of updating to the parent, where we update from a virtual empty node to the leaf node.

### 3.4 Memory

For a node $t$ with bag $X_t$, the vectors used by the algorithm can be quite large, containing anywhere between 1 and $2^{|X_t|}$ entries. Since storing many such large vectors simultaneously can put significant pressure on memory resources, we consider their lifespans and how they are managed.

*The* p *vectors* The lifespan of the p vectors depends on whether the actual maximum weight independent set is computed, or if only its weight is required. The p vectors are optional and only computed alongside the c vectors if the maximum weight independent set itself is needed. Once computed, they must persist for the entire duration of the program, as they are used during a root-to-leaf traversal to reconstruct the solution. Because the p vectors are only needed when reading out the final solution, we manage them by writing to and reading from disk using the Zstandard compression library to reduce memory usage.

*The* v *vectors* The v vectors are only required for the current node during the depth-first traversal.

*The* c *vectors* The c vectors are required for the current node, but must also persist for all *active* nodes. A node $t$ is considered *active* if it has at least one finished child and at least one unfinished child. Because the number of active nodes can grow substantially for large tree decompositions, we provide an option to store these c vectors on disk as well, again using the Zstandard library for compression.

### 3.5 Output

The algorithm outputs the weight of a maximum weight independent set on the first line, followed by the time taken to compute this value in seconds on the second line. If the actual independent set is requested via an optional flag, an additional line is printed, listing the vertices included in the set in ascending order.

### 3.6 Running Time Analysis

Let $G$ be a graph and let $TD$ be a rooted tree decomposition of $G$ with $k = wd(TD) + 1$. Then the algorithm runs within $O(n \cdot k \cdot 2k \cdot 2^k)$, where $2^k$ is an upperbound on the size of a DP table, $2k$ is an upperbound on the length of of a virtual path, $k$ is an upperbound on the effort required to check if a set is independent and $n$ is an upperbound on the number of nodes in the tree decomposition.

## 4 Results

The primary bottleneck of our algorithm is the exponential number of independent sets in the bags of the tree decomposition. Although the theoretical upper bound for the number of independent sets encountered in a bag of a tree decomposition $TD$ with $k = \text{width}(TD) + 1$ is $2^k$, the actual number of sets encountered in practice is often much smaller. We can reliably solve the MWIS problem for all graphs with a tree decomposition of maximal bag size below 32, and provide support for decompositions with maximal bag size below 128, where the latter corresponds to the maximum size our encoding can handle.

### 4.1 Tree Decomposition Method

All tree decompositions used in this work were computed using the heuristic algorithm provided in the PACE 2017 Track A repository by Tamaki, Ohtsuka, Sato, and Makii [9], which incorporates the exact method described by Tamaki [8] as one of its components.

### 4.2 System Configuration for Performance Evaluation

All experiments and performance measurements were conducted on the following hardware and software environment:

- CPU: Intel Core i5-2400S @ 2.50 GHz, 4 cores (no hyper-threading), `x86_64` architecture
- RAM: 23 GiB
- Operating System: Ubuntu 24.04.2 LTS, Linux kernel 6.11.0-26-generic

- Compiler: g++ with flags:
  - `-O3`: Enables aggressive compiler optimizations
  - `-march=native`: Targets the host CPU with all available instruction sets.
  - `-std=c++20`: Uses the C++20 standard

### 4.3 Datasets

The benchmark set used in this work includes graphs drawn from standard datasets commonly used in MWIS research. These instances originate from publicly available sources, including OpenStreetMap (OSM), the Stanford Large Network Dataset Repository (SNAP), the SuiteSparse Matrix Collection (SSMC), and various 3D mesh and finite element datasets. We reused these datasets similar as to other works [2, 5, 7], all of which evaluate solvers for graph problems on subsets of the same graph families. While not distributed as a single unified benchmark suite, these graphs can be either reconstructed using the data sources and scripts referenced in the cited works or are shared amongst the community. To ensure consistency and comparability, we follow the instance naming conventions established in prior literature where applicable.

### 4.4 Results on the Dataset

Our dataset consists of 213 graphs grouped into five categories:

- (`mesh`) 14 graphs from well-known triangle meshes.
- (`fe`) 5 3D meshes derived from simulations using the finite element method.
- (`snap`) 40 graphs from the Stanford Large Network Dataset Repository.
- (`ssmc`) 6 graphs from the SuiteSparse Matrix Collection.
- (`osm`) 148 conflict graphs generated from OpenStreetMap data, following Barth et al. [1].

We excluded the `snap` graphs from the following analysis due to difficulties in computing suitable tree decompositions. For the remaining four groups, detailed tables presenting the tree decomposition widths and the corresponding results of our MWIS solver on the benchmark set are provided in the appendix, see Appendix A and Appendix B. Only the `osm` group contained graphs with widths within our guaranteed solvable range. In fact, 112 of the 148 `osm` graphs had widths supported by our datatype, and all of these were solvable by our algorithm.

Figure 5 plots for the `osm` graphs the maximum bag size occuring in the computed tree decompositions against the logarithm of the maximal number of independent sets in a bag. We use a grayscale color map to indicate the size of the corresponding bag, since the bag containing the maximal number of sets does not always correspond to the one with size closest to the decomposition's width.

The information in Appendix A and Appendix B also enables comparison against LearnAndReduce, a separate MWIS solver by Großmann et al [6]; see also the accompanying paper [5] for further details. While our solver could not solve any instances that LearnAndReduce could not fully reduce, we were able to successfully solve several reduced instances produced by LearnAndReduce by subsequently applying our algorithm. These additional instances and results are presented in Appendix C.

## 5 Future Work

Currently, the solver supports tree decompositions with maximum bag sizes less than 128. Since all `osm` graphs within this bound were solvable by the algorithm, extending support to maximum bag sizes below 255 might result in additional solved instances.

The tree decompositions used were computed using heuristics from [9]. These heuristics produce generally good decompositions, but were not designed with the specific needs of this MWIS solver in mind. Although our focus was on implementing the MWIS solver for given tree decompositions,

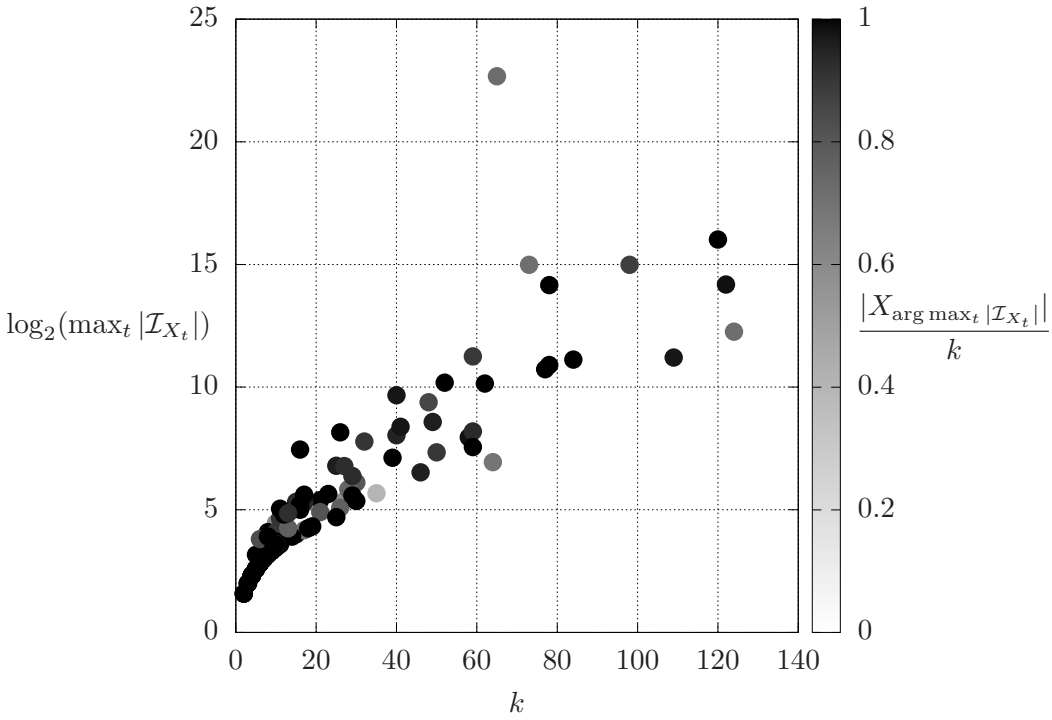



Fig. 5. All plotted points correspond to OSM instances for which we obtained a tree decomposition with maximal bag size $k$ less than 128, all of which are solvable by the algorithm described in this paper. The scatter plot shows the relationship between $k$ and the base-2 logarithm of the maximum value of $|\mathcal{I}_{X_t}|$ taken over all bags in the tree decomposition, where $\mathcal{I}_{X_t}$ denotes the set of independent sets of the induced subgraph of the bag $X_t$. The grayscale shading indicates the ratio between the size of the bag where this maximum occurs and $k$.

it would likely be beneficial to develop a tree decomposition algorithm that specifically minimizes the maximal number of independent sets occurring in a bag. If such an approach is feasible, it could significantly improve the efficiency and scalability of the solver.

In addition, extending the exact solver towards a heuristic approach could improve scalability.

## References


[1] Lukas Barth, Benjamin Niedermann, Martin Nöllenburg, and Darren Strash. Temporal map labeling: a new unified framework with experiments. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPACIAL '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345897. doi: 10.1145/2996913.2996957. URL https://doi.org/10.1145/2996913.2996957.

[2] Shaowei Cai, Wenying Hou, Jinkun Lin, and Yuanjie Li. Improving Local Search for Minimum Weight Vertex Cover by Dynamic Strategies. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1412–1418. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/196. URL https://doi.org/10.24963/ijcai.2018/196.

[3] PACE Challenge Committee. PACE Challenge 2017: Treewidth Track. https://pacechallenge.org/2017/treewidth/, 2017. Accessed: 2025-04-28.

[4] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, chapter 7, pages 157–166. Springer, Cham, May 2015. ISBN 978-3-319-21275-3. doi: 10.1007/978-3-319-21275-3. URL https://doi.org/10.1007/978-3-319-21275-3.

[5] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. Accelerating Reductions Using Graph Neural Networks and a New Concurrent Local Search for the Maximum Weight Independent Set Problem. Mar 2025. doi: 10.48550/arXiv.2412.14198. URL https://arxiv.org/abs/2412.14198v2.

[6] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. KarlsruheMIS/LearnAndReduce: v1.0 - ACDA2025, Apr 2025. URL https://doi.org/10.5281/zenodo.15229975.

[7] Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. Exactly Solving the Maximum Weight Independent Set Problem on Large Real-World Graphs. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 144–158. SIAM, 2019. doi: 10.1137/1.9781611975499.12. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611975499.12.

[8] Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *arXiv preprint arXiv:1704.05286*, 2017. doi: 10.48550/arXiv.1704.05286. URL https://arxiv.org/abs/1704.05286.

[9] Hisao Tamaki, Hiromu Ohtsuka, Takuto Sato, and Keitaro Makii. PACE2017-TrackA. https://github.com/TCS-Meiji/PACE2017-TrackA, 2017. Accessed: 2025-04-28.

## A Results on OSM

Table 1. Results on OSM instances (part 1 of 4). Each row corresponds to a graph instance derived from OpenStreetMap data. The columns report the number of vertices $n$, edges $m$, decomposition width plus one $k$ (i.e., the size of the largest bag), and number of bags $N$ in the computed tree decomposition. The columns $w$ and $t$ respectively show the weight of the computed maximum weight independent set and the runtime (in seconds) of our solver introduced in this work. A dash (–) indicates that no result could be obtained by our solver. Gray shading is used to indicate that an instance is known to be solvable by `LearnAndReduce`, independent of our solver.

| **OSM instance** | $n$ | $m$ | $k$ | $N$ | $w$ | $t$ |
|---|---|---|---|---|---|---|
| alabama-AM1 | 320 | 581 | 10 | 206 | 167 588 | 0.0006 |
| alabama-AM2 | 1 164 | 19 386 | 73 | 447 | 174 309 | 0.1071 |
| alabama-AM3 | 3 504 | 309 664 | 436 | 742 | - | - |
| alaska-AM1 | 31 | 31 | 4 | 13 | 20 266 | <0.0001 |
| alaska-AM2 | 54 | 156 | 11 | 20 | 20 900 | <0.0001 |
| alaska-AM3 | 86 | 475 | 21 | 36 | 22 325 | 0.0002 |
| arkansas-AM1 | 26 | 19 | 4 | 11 | 17 702 | <0.0001 |
| arkansas-AM2 | 55 | 233 | 15 | 20 | 20 771 | 0.0001 |
| arkansas-AM3 | 103 | 1 376 | 40 | 28 | 20 935 | 0.0007 |
| california-AM1 | 77 | 130 | 8 | 38 | 46 537 | <0.0001 |
| california-AM2 | 231 | 3 074 | 40 | 87 | 47 153 | 0.0029 |
| california-AM3 | 587 | 27 536 | 189 | 144 | - | - |
| canada-AM1 | 189 | 240 | 8 | 92 | 78 466 | 0.0002 |
| canada-AM2 | 449 | 2 947 | 30 | 153 | 81 799 | 0.0008 |
| canada-AM3 | 943 | 20 241 | 98 | 281 | 86 018 | 0.1397 |
| colorado-AM1 | 128 | 232 | 7 | 56 | 50 507 | 0.0001 |
| colorado-AM2 | 283 | 2 026 | 27 | 85 | 52 172 | 0.0004 |
| colorado-AM3 | 538 | 8 365 | 62 | 164 | 54 741 | 0.0040 |
| connecticut-AM1 | 87 | 96 | 5 | 44 | 55 131 | <0.0001 |
| connecticut-AM2 | 211 | 975 | 17 | 65 | 56 058 | 0.0002 |
| connecticut-AM3 | 367 | 3 769 | 46 | 118 | 57 650 | 0.0007 |
| delaware-AM1 | 2 | 1 | 2 | 1 | 1 060 | <0.0001 |
| delaware-AM2 | 3 | 3 | 3 | 1 | 1 060 | <0.0001 |
| delaware-AM3 | 5 | 9 | 4 | 2 | 1 060 | <0.0001 |
| district-of-columbia-AM1 | 2 500 | 24 651 | 65 | 1 378 | 196 475 | 9.8375 |
| district-of-columbia-AM2 | 13 597 | 1 609 795 | 857 | 2 644 | - | - |
| district-of-columbia-AM3 | 46 221 | 27 729 137 | 5 106 | 5 126 | - | - |
| florida-AM1 | 475 | 1 277 | 18 | 230 | 225 655 | 0.0005 |
| florida-AM2 | 1 254 | 16 936 | 58 | 387 | 230 595 | 0.0037 |
| florida-AM3 | 2 985 | 154 043 | 230 | 783 | - | - |
| georgia-AM1 | 294 | 434 | 8 | 144 | 205 068 | 0.0003 |
| georgia-AM2 | 746 | 7 753 | 64 | 256 | 216 346 | 0.0019 |
| georgia-AM3 | 1 680 | 74 126 | 267 | 480 | - | - |
| greenland-AM1 | 77 | 341 | 16 | 46 | 9 328 | 0.0002 |
| greenland-AM2 | 686 | 50 218 | 210 | 152 | - | - |
| greenland-AM3 | 4 986 | 3 652 361 | 2 830 | 357 | - | - |
| hawaii-AM1 | 411 | 1 423 | 15 | 227 | 113 792 | 0.0009 |

Table 2. Results on OSM instances (part 2 of 4). Continuation of Table 1.

| **OSM instance** | $n$ | $m$ | $k$ | $N$ | $w$ | $t$ |
|---|---|---|---|---|---|---|
| hawaii-AM2 | 2 875 | 265 158 | 320 | 554 | - | - |
| hawaii-AM3 | 28 006 | 49 444 921 | 9 208 | 1 426 | - | - |
| idaho-AM1 | 136 | 208 | 11 | 63 | 70 623 | 0.0002 |
| idaho-AM2 | 552 | 35 221 | 186 | 123 | - | - |
| idaho-AM3 | 4 064 | 3 924 080 | 2 957 | 272 | - | - |
| illinois-AM1 | 113 | 202 | 10 | 58 | 54 678 | 0.0002 |
| illinois-AM2 | 261 | 2 138 | 32 | 101 | 55 496 | 0.0011 |
| indiana-AM1 | 2 | 1 | 2 | 1 | 1 146 | <0.0001 |
| indiana-AM2 | 2 | 1 | 2 | 1 | 1 146 | <0.0001 |
| indiana-AM3 | 4 | 6 | 4 | 1 | 1 146 | <0.0001 |
| iowa-AM1 | 90 | 164 | 7 | 38 | 47 907 | <0.0001 |
| iowa-AM2 | 155 | 954 | 21 | 49 | 47 984 | 0.0003 |
| kansas-AM1 | 190 | 400 | 11 | 92 | 84 449 | 0.0002 |
| kansas-AM2 | 602 | 16 474 | 120 | 183 | 85 942 | 0.1932 |
| kansas-AM3 | 2 732 | 806 912 | 1 166 | 347 | - | - |
| kentucky-AM1 | 381 | 2 402 | 39 | 184 | 91 897 | 0.0010 |
| kentucky-AM2 | 2 453 | 643 428 | 914 | 398 | - | - |
| kentucky-AM3 | 19 095 | 59 533 630 | 12 625 | 966 | - | - |
| louisiana-AM1 | 157 | 181 | 6 | 93 | 51 446 | 0.0002 |
| louisiana-AM2 | 436 | 3 111 | 28 | 154 | 55 127 | 0.0008 |
| louisiana-AM3 | 1 162 | 37 077 | 137 | 299 | - | - |
| maine-AM1 | 38 | 29 | 4 | 19 | 24 921 | <0.0001 |
| maine-AM2 | 81 | 243 | 10 | 25 | 26 208 | <0.0001 |
| maine-AM3 | 143 | 850 | 17 | 49 | 26 734 | 0.0003 |
| maryland-AM1 | 104 | 216 | 10 | 47 | 43 930 | 0.0001 |
| maryland-AM2 | 316 | 4 715 | 50 | 99 | 45 300 | 0.0010 |
| maryland-AM3 | 1 018 | 95 415 | 281 | 168 | - | - |
| massachusetts-AM1 | 413 | 1 089 | 16 | 237 | 136 695 | 0.0009 |
| massachusetts-AM2 | 1 339 | 35 449 | 160 | 440 | - | - |
| massachusetts-AM3 | 3 703 | 551 491 | 1 014 | 709 | - | - |
| mexico-AM1 | 175 | 358 | 10 | 78 | 90 599 | 0.0002 |
| mexico-AM2 | 516 | 9 411 | 59 | 172 | 94 834 | 0.0025 |
| mexico-AM3 | 1 096 | 47 131 | 170 | 286 | - | - |
| michigan-AM1 | 133 | 112 | 4 | 69 | 51 076 | 0.0001 |
| michigan-AM2 | 241 | 750 | 14 | 87 | 51 928 | 0.0002 |
| michigan-AM3 | 376 | 2 459 | 29 | 138 | 52 674 | 0.0009 |
| minnesota-AM1 | 86 | 136 | 5 | 52 | 28 692 | 0.0002 |

Table 3. Results on OSM instances (part 3 of 4). Continuation of Table 1.

| **OSM instance** | $n$ | $m$ | $k$ | $N$ | $w$ | $t$ |
|---|---|---|---|---|---|---|
| minnesota-AM2 | 253 | 2 580 | 35 | 100 | 30 251 | 0.0007 |
| minnesota-AM3 | 683 | 34 188 | 182 | 171 | - | - |
| mississippi-AM1 | 74 | 60 | 5 | 39 | 32 273 | <0.0001 |
| mississippi-AM2 | 151 | 366 | 10 | 48 | 33 187 | 0.0001 |
| mississippi-AM3 | 242 | 1 116 | 16 | 90 | 33 318 | 0.0003 |
| missouri-AM1 | 10 | 6 | 2 | 6 | 7 928 | <0.0001 |
| missouri-AM2 | 13 | 12 | 3 | 6 | 7 928 | <0.0001 |
| missouri-AM3 | 17 | 24 | 4 | 8 | 7 928 | <0.0001 |
| montana-AM1 | 109 | 194 | 9 | 49 | 55 348 | 0.0001 |
| montana-AM2 | 307 | 5 154 | 59 | 100 | 56 068 | 0.0026 |
| montana-AM3 | 837 | 69 293 | 297 | 198 | - | - |
| nebraska-AM1 | 40 | 46 | 7 | 17 | 24 345 | <0.0001 |
| nebraska-AM2 | 93 | 734 | 26 | 31 | 26 680 | 0.0002 |
| nebraska-AM3 | 145 | 2 168 | 49 | 47 | 27 214 | 0.0014 |
| nevada-AM1 | 89 | 93 | 6 | 46 | 45 761 | 0.0001 |
| nevada-AM2 | 242 | 1 531 | 25 | 81 | 47 068 | 0.0004 |
| nevada-AM3 | 569 | 15 016 | 109 | 156 | 52 036 | 0.0156 |
| new-hampshire-AM1 | 195 | 302 | 7 | 98 | 108 186 | 0.0002 |
| new-hampshire-AM2 | 514 | 3 369 | 25 | 181 | 110 621 | 0.0009 |
| new-hampshire-AM3 | 1 107 | 18 021 | 59 | 367 | 116 060 | 0.0151 |
| new-jersey-AM1 | 4 | 6 | 4 | 1 | 256 | <0.0001 |
| new-jersey-AM2 | 4 | 6 | 4 | 1 | 256 | <0.0001 |
| new-jersey-AM3 | 4 | 6 | 4 | 1 | 256 | <0.0001 |
| new-mexico-AM1 | 3 | 3 | 3 | 1 | 182 | <0.0001 |
| new-mexico-AM2 | 3 | 3 | 3 | 1 | 182 | <0.0001 |
| new-mexico-AM3 | 3 | 3 | 3 | 1 | 182 | <0.0001 |
| new-york-AM1 | 42 | 118 | 11 | 20 | 13 187 | <0.0001 |
| new-york-AM2 | 224 | 6 399 | 78 | 72 | 14 330 | 0.0445 |
| new-york-AM3 | 837 | 88 728 | 329 | 137 | - | - |
| north-carolina-AM1 | 93 | 150 | 7 | 46 | 45 254 | 0.0001 |
| north-carolina-AM2 | 398 | 10 116 | 78 | 89 | 46 896 | 0.0103 |
| north-carolina-AM3 | 1 557 | 236 739 | 353 | 223 | - | - |
| ohio-AM1 | 78 | 96 | 6 | 42 | 50 964 | <0.0001 |
| ohio-AM2 | 211 | 1 815 | 30 | 67 | 51 289 | 0.0004 |
| ohio-AM3 | 482 | 11 376 | 84 | 151 | 52 634 | 0.0107 |
| oregon-AM1 | 381 | 996 | 17 | 203 | 161 298 | 0.0005 |
| oregon-AM2 | 1 325 | 57 517 | 202 | 361 | - | - |

Table 4. Results on OSM instances (part 4 of 4). Continuation of Table 1.

| **OSM instance** | $n$ | $m$ | $k$ | $N$ | $w$ | $t$ |
|---|---|---|---|---|---|---|
| oregon-AM3 | 5 588 | 2 912 701 | 2 332 | 705 | - | - |
| pennsylvania-AM1 | 193 | 276 | 8 | 99 | 133 914 | 0.0002 |
| pennsylvania-AM2 | 521 | 3 812 | 27 | 173 | 138 413 | 0.0010 |
| pennsylvania-AM3 | 1 148 | 26 464 | 124 | 359 | 143 870 | 0.0452 |
| puerto-rico-AM1 | 60 | 63 | 5 | 30 | 29 802 | <0.0001 |
| puerto-rico-AM2 | 165 | 1 285 | 29 | 40 | 32 921 | 0.0003 |
| puerto-rico-AM3 | 494 | 26 926 | 175 | 84 | - | - |
| rhode-island-AM1 | 455 | 1 973 | 26 | 269 | 171 224 | 0.0012 |
| rhode-island-AM2 | 2 866 | 295 488 | 402 | 692 | - | - |
| rhode-island-AM3 | 15 124 | 12 622 219 | 5 772 | 1 380 | - | - |
| south-carolina-AM1 | 75 | 69 | 5 | 42 | 50 033 | <0.0001 |
| south-carolina-AM2 | 165 | 713 | 16 | 64 | 51 446 | 0.0003 |
| south-carolina-AM3 | 317 | 4 508 | 48 | 103 | 52 087 | 0.0025 |
| tennessee-AM1 | 49 | 39 | 4 | 23 | 29 569 | <0.0001 |
| tennessee-AM2 | 100 | 418 | 19 | 30 | 31 567 | 0.0001 |
| tennessee-AM3 | 212 | 3 215 | 52 | 55 | 32 276 | 0.0062 |
| utah-AM1 | 230 | 309 | 6 | 125 | 87 856 | 0.0002 |
| utah-AM2 | 589 | 4 692 | 41 | 212 | 95 087 | 0.0018 |
| utah-AM3 | 1 339 | 42 872 | 181 | 386 | - | - |
| vermont-AM1 | 128 | 418 | 13 | 65 | 55 884 | 0.0002 |
| vermont-AM2 | 766 | 37 607 | 135 | 171 | - | - |
| vermont-AM3 | 3 436 | 1 136 164 | 1 012 | 319 | - | - |
| virginia-AM1 | 570 | 1 480 | 12 | 287 | 280 936 | 0.0007 |
| virginia-AM2 | 2 279 | 60 040 | 122 | 662 | 295 867 | 0.1249 |
| virginia-AM3 | 6 185 | 665 903 | 579 | 1 244 | - | - |
| washington-AM1 | 713 | 2 316 | 23 | 355 | 296 653 | 0.0019 |
| washington-AM2 | 3 025 | 152 449 | 255 | 759 | - | - |
| washington-AM3 | 10 022 | 2 346 213 | 1 470 | 1 606 | - | - |
| west-virginia-AM1 | 65 | 150 | 8 | 33 | 42 868 | <0.0001 |
| west-virginia-AM2 | 317 | 8 328 | 77 | 78 | 45 923 | 0.0053 |
| west-virginia-AM3 | 1 185 | 125 620 | 335 | 206 | - | - |
| wisconsin-AM1 | 54 | 51 | 4 | 24 | 44 608 | <0.0001 |
| wisconsin-AM2 | 89 | 219 | 10 | 27 | 44 651 | <0.0001 |
| wisconsin-AM3 | 136 | 588 | 13 | 56 | 47 904 | 0.0004 |
| wyoming-AM1 | 7 | 11 | 5 | 2 | 4 568 | <0.0001 |
| wyoming-AM2 | 8 | 16 | 6 | 2 | 4 568 | <0.0001 |
| wyoming-AM3 | 12 | 42 | 9 | 3 | 4 568 | <0.0001 |

## B Results on fe, mesh and ssmc

Table 5. Results on fe, mesh, and ssmc instance groups. Each row corresponds to a graph instance derived from the specified group. The columns report the number of vertices $n$, edges $m$, decomposition width plus one $k$ (i.e., the size of the largest bag), and number of bags $N$ in the computed tree decomposition. Rows shaded in gray indicate instances known to be solvable by `LearnAndReduce`. Since our solver did not produce solutions on any of these instances, the weight and runtime columns are omitted.

| | **Instance** | $n$ | $m$ | $k$ | $N$ |
|---|---|---|---|---|---|
| **fe** | body-uniform | 45 087 | 163 734 | 236 | 30 563 |
| | ocean-uniform | 143 437 | 409 593 | 143 398 | 40 |
| | pwt-uniform | 36 519 | 144 794 | 214 | 18 943 |
| | rotor-uniform | 99 617 | 662 431 | 99 617 | 1 |
| | sphere-uniform | 16 386 | 49 152 | 414 | 9 805 |
| **mesh** | blob-uniform | 16 068 | 24 102 | 166 | 13 262 |
| | buddha-uniform | 1 087 716 | 1 631 574 | 1 087 716 | 1 |
| | bunny-uniform | 68 790 | 103 017 | 349 | 56 333 |
| | cow-uniform | 5 036 | 7 366 | 43 | 4 301 |
| | dragonsub-uniform | 600 000 | 900 000 | 600 000 | 1 |
| | dragon-uniform | 150 000 | 225 000 | 232 | 125 637 |
| | ecat-uniform | 684 496 | 1 026 744 | 2 006 | 555 191 |
| | face-uniform | 22 871 | 34 054 | 129 | 19 126 |
| | fandisk-uniform | 8 634 | 12 818 | 97 | 7 159 |
| | feline-uniform | 41 262 | 61 893 | 141 | 34 000 |
| | gameguy-uniform | 42 623 | 63 850 | 241 | 35 759 |
| | gargoyle-uniform | 20 000 | 30 000 | 149 | 16 682 |
| | turtle-uniform | 267 534 | 401 178 | 1 278 | 220 527 |
| | venus-uniform | 5 672 | 8 508 | 75 | 4 750 |
| **ssmc** | ca2010 | 710 145 | 1 744 683 | 395 | 608 354 |
| | fl2010 | 484 481 | 1 173 147 | 297 | 422 722 |
| | ga2010 | 291 086 | 709 028 | 493 | 250 917 |
| | il2010 | 451 554 | 1 082 232 | 551 | 376 861 |
| | nh2010 | 48 837 | 117 275 | 142 | 42 850 |
| | ri2010 | 25 181 | 62 875 | 78 | 21 334 |

## C Results on Partially Reduced Instances

Table 6. Results on partially reduced instances provided by LearnAndReduce. Each row corresponds to a graph instance that was preprocessed by LearnAndReduce, after which we applied our solver to the reduced instance. The columns report the number of vertices $n$, edges $m$, decomposition width plus one $k$ (i.e., the size of the largest bag), and number of bags $N$ in the computed tree decomposition of the reduced instance. The columns $w$ and $t$ respectively show the weight of the computed maximum weight independent set and the runtime (in seconds) of our solver on these reduced intances. A dash (–) indicates that no result could be obtained.

| | **Reduced instance** | $n$ | $m$ | $k$ | $N$ | $w$ | $t$ |
|---|---|---|---|---|---|---|---|
| **fe** | body-uniform | 396 | 3 571 | 43 | 191 | 7 713 | 16.7516 |
| | pwt-uniform | 15 274 | 107 248 | 116 | 7 294 | - | - |
| | rotor-uniform | 89 294 | 690 242 | 2 128 | 40 946 | - | - |
| **osm** | alabama-AM3 | 815 | 61 637 | 239 | 157 | - | - |
| | california-AM3 | 359 | 19 705 | 159 | 65 | - | - |
| | district-of-columbia-AM2 | 2 361 | 274 528 | 584 | 465 | - | - |
| | district-of-columbia-AM3 | 26 728 | 13 406 329 | 3 481 | 3 010 | - | - |
| | florida-AM3 | 658 | 41 043 | 162 | 107 | - | - |
| | georgia-AM3 | 481 | 31 388 | 191 | 103 | - | - |
| | greenland-AM3 | 3 402 | 2 141 096 | 2 296 | 241 | - | - |
| | hawaii-AM3 | 22 849 | 38 081 016 | 8 031 | 911 | - | - |
| | idaho-AM3 | 2 920 | 2 413 139 | 2 386 | 130 | - | - |
| | kansas-AM3 | 1 209 | 292 997 | 753 | 117 | - | - |
| | kentucky-AM3 | 16 714 | 53 762 869 | 12 113 | 712 | - | - |
| | massachusetts-AM3 | 1 626 | 340 652 | 837 | 216 | - | - |
| | mexico-AM3 | 447 | 20 443 | 105 | 118 | 11 196 | 1.6672 |
| | montana-AM3 | 382 | 39 859 | 231 | 63 | - | - |
| | new-york-AM3 | 526 | 50 572 | 266 | 98 | - | - |
| | north-carolina-AM3 | 997 | 133 106 | 293 | 135 | - | - |
| | oregon-AM3 | 3 159 | 1 817 045 | 1 950 | 214 | - | - |
| | rhode-island-AM2 | 498 | 46 183 | 187 | 147 | - | - |
| | rhode-island-AM3 | 12 189 | 11 225 470 | 5 349 | 866 | - | - |
| | vermont-AM3 | 1 946 | 526 463 | 790 | 167 | - | - |
| | virginia-AM3 | 2 385 | 293 068 | 408 | 359 | - | - |
| | washington-AM3 | 6 671 | 1 955 568 | 1 388 | 687 | - | - |
| | west-virginia-AM3 | 864 | 98 399 | 325 | 141 | - | - |
| **snap** | as-skitter-uniform | 1 996 | 27 527 | 222 | 993 | - | - |
| | loc-gowalla_edges | 304 | 2 938 | 104 | 184 | - | - |
| | soc-LiveJournal1-uniform | 1 360 | 21 602 | 137 | 780 | - | - |
| | soc-p.-rel.-uniform | 782 844 | 14 236 948 | 782 844 | 1 | - | - |
| | web-NotreDame | 75 | 1 178 | 37 | 24 | 2 728 | 0.0337 |
| | web-NotreDame-uniform | 97 | 1 357 | 38 | 43 | 3 323 | 0.9553 |