Karlsruher Institut für Technologie

Solution to Exercise Sheet 6

Submission due by July 16, 2026

Problem 1: Dominating Set on Special Graphs — 10 points

Consider the following parameterized variant of Dominating Set. The graph G is given together with an ordering of the vertices $V = \{v_1, \dots, v_n\}$. The *length* of an edge $\{v_i, v_j\}$ is $|i-j|$. The parameter k is the maximum edge length in G. Provide an FPT algorithm that solves this parameterization of Dominating Set.

Solution 1:

We are given a graph $G = (V,E)$ with a vertex ordering $V = \{v_1, \dots, v_n\}$ such that for every edge $\{v_i,v_j\} \in E$ we have $|i - j| \leq k$. We observe that we obtain a path decomposition of width $2k$ by placing every consecutive block of $2k$ vertices into a single bag.

Using this path decomposition, we can solve Dominating Set (DS) via dynamic programming. For each bag X we store partial solutions that, for every vertex $x \in X$, indicate whether it is in the DS, is dominated by another vertex, or is neither in the DS nor dominated. Additionally, for each partial solution, we store the current size of the constructed DS. For the first bag of the path decomposition, we can enumerate all 2^{2k} possibilities of placing vertices into the DS.

We must now think about how to update partial solutions when vertices are introduced or forgotten. When a vertex is introduced, it may either be added to the DS or not, giving us for each old partial solution two new ones; we update the size of the DS and the domination status of each vertex accordingly. When a vertex v is forgotten, any partial solution in which v is neither in the DS nor dominated is discarded as invalid.

Because of the construction of the path decomposition, we never have to consider bags with more than $2k$ vertices. Since each vertex can be in three possible states, each bag yields at most 3^{2k} partial solutions. Hence, the overall running time is $6^k \cdot n^{O(1)}$.

Problem 2: Miscellaneous facts about treewidth — 6 + 6 points

Part (a) Let X_i, X_k, and X_j be bags of a tree decomposition of a graph G such that X_k lies on the path between X_i and X_j. Show that $X_i \setminus X_k$ is separated from $X_j \setminus X_k$ by X_k.

Solution 2a:

Let X_i, X_k, and X_j be bags of a tree decomposition of a graph G such that X_k lies on the path

between X_i and X_j. Let $a \in X_i \setminus X_k$ and $b \in X_j \setminus X_k$. Assume that X_k does not separate a and b. Then there exists a path $P = s_1, \dots, s_\ell$ with $s_1 = a$ and $s_\ell = b$ that uses no vertex from X_k.

Denote by $X(v)$ the set of bags that contain a vertex v. By induction one can show that the bags $\bigcup_{i=1}^{\ell} X(s_i)$ form a connected subtree. For s_1 this follows directly from the definition of a tree decomposition. If it holds for the vertices $s_1, \dots, s_i$, then it also holds for $s_1, \dots, s_{i+1}$, because the edge $\{s_i, s_{i+1}\}$ guarantees a bag containing both s_i and s_{i+1}, so the two connected subtrees intersect.

Every connected subtree that contains both X_i and X_j must also contain X_k. This contradicts the fact that P contains no vertex from X_k, i.e. $X_k \notin \bigcup_{i=1}^{\ell} X(s_i)$.

Part (b) Let G be a graph of treewidth at most k with n vertices and m edges. Show that $m \leq n \cdot k$.

Solution 2b:

Let G be a graph of treewidth at most k. We claim that G contains a vertex v of degree at most k. Such a vertex is found by starting at a leaf of a nice tree decomposition of width k and moving upward until the first forget-node is encountered. The vertex forgotten in this bag can be adjacent to at most the k other vertices of the bag. Removing v gives us a graph whose treewidth remains at most k, so we can iteratively delete vertices of degree $\leq k$. Consequently $m \leq n \cdot k$.

Problem 3: Connected Dominating Sets **18 points**

A vertex set $D \subseteq V$ in a graph $G = (V,E)$ is a *dominating set* if every vertex $v \in V$ is either contained in D or has a neighbor in D. The Connected Dominating Set problem asks for a smallest possible dominating set D such that the subgraph induced by D is connected.

Given a graph G, a parameter k, and a tree decomposition of width k, provide an FPT algorithm for this parameterization of Connected Dominating Set.

Hint: Use dynamic programming on the tree decomposition. Using Courcelle's Theorem (from lecture 10) is not allowed.

Solution 3:

We present a DP algorithm on a nice tree decomposition that computes the size of a minimum connected dominating set (CDS).

For each bag, we store partial solutions that partition the vertices of the bag into three sets. A partial solution for a bag B has a partition $X_0 \dot\cup X_1 \dot\cup X_2 = B$ with the following meaning:

- X_0: Vertices that belong to the CDS. This set is further partitioned into sets that indicate which vertices are already linked by the current bag or previously processed bags.
- X_1: Vertices that are not in the CDS but are already dominated by it.

X_2: Vertices that are neither in the CDS nor dominated yet.

For each partition, we also store the current size s of the CDS up to this bag.

Each partial solution of a bag is based on the partial solutions of the child nodes. For the partial solutions of the current bag, we distinguish according to the bag type:

- Forget:
  - *Node was in X_0:* If the node was in a partition together with other nodes in X_0, then it can be deleted. Otherwise the partial solution is invalid because the CDS is no longer connected and can never become connected again (the bags of each node form a subtree).
  - *Node was in X_1:* The node can be deleted and the current partial solution remains valid.
  - *Node was in X_2:* The partial solution can be deleted because the node can no longer be added to the dominating set nor be covered by it.
- Introduce: Here, from each partial solution, two new partial solutions arise, representing the different ways of adding to X_0.
  - *Do not add to X_0:* The assignment to X_1 or X_2 follows implicitly.
  - *Add to X_0:* s is increased by 1. The node sets X_1 and X_2 change because nodes that were previously in X_2 are now in X_1 due to their connection to the new node. A node cannot move from X_1 to X_2. The partitioning within X_0 changes as the new node is assigned to the partition with which it has a connection. If necessary, several partitions merge into one.
- Join: Let (X_0', X_1', X_2', s') and $(X_0'', X_1'', X_2'', s'')$ be the partial solutions of the two child nodes of the join bag. Partial solutions where X_0' and X_0'' are equal are combined pairwise. The new CDS size s is computed as $s = s' + s'' - |X_0|$. The resulting set X_1 is the union of X_1' and X_1''. The set X_2 is derived from X_0 and X_1. It may happen that the join operation creates partial solutions that do not differ in their assignment to the sets X_0, X_1, and X_2 and that also partition the nodes in X_0 identically. In such cases, the solution with the smallest s should be chosen.

For the root node of the tree, it is important that only partial solutions are accepted for which $|X_2| = 0$ and, additionally, the partitioning of X_0 consists of a single partition.

Analysis: The number of partial solutions per bag can be estimated by the partitioning into three sets, giving 3^{k+1}. Before the partial solutions are restricted, the same partitioning may appear multiple times in a bag. The worst case occurs when many unions take place in a join bag. The

number of unions does not exceed the size of the power set. Thus, the number of partial solutions in the worst case is at most $2^{(3^{k+1})}$.

The size of the tree decomposition is bounded from above by $\mathcal{O}(n)$.

Since handling the partial solutions has a time complexity that depends only on the number of partial solutions and their size, the processing of each bag has a time complexity of $\mathcal{O}(f(k))$, where f is a computable function.

Consequently, the overall runtime is $\mathcal{O}(f(k) \cdot n)$.

Problem 4: Treewidth DP

10 bonus points

In this exercise, you have to implement a dynamic programming algorithm on a tree decomposition to solve the Independent Set problem. On the course website, you will find five instances, each consisting of a graph and a corresponding tree decomposition.

State the size of a maximum independent set for each instance and briefly explain how you realized the lecture's approach and any optimizations you applied.

Input format of a graph (`.gr`): The first line `p tw n m` contains the letter `p`, the word `tw`, the number of vertices `n`, and the number of edges `m`. The following `m` lines each describe an edge; each line contains two numbers `a b` with $1 \leq$ a,b $\leq$ n, denoting an edge between vertex `a` and vertex `b`. Since the graphs are undirected, each edge is defined only once.

Example:

```
p tw 5 6
1 2
1 3
2 3
2 4
3 4
2 5
```

Input format of a tree decomposition (`.td`): The first line starts with `s td N W n`, where `N` is the number of bags, `W` the maximum bag size, and `n` the original graph's vertex count. The following `N` lines have the form `b i v1 v2 ...`, where `i` is the bag number ($1 \leq$ `i` $\leq$ `N`) and `v1`, … are the vertices contained in that bag. All remaining lines define the edges of the tree decomposition; each line contains two integers `i j` ($1 \leq$ `i` $<$ `j` $\leq$ `N`), denoting an edge between bag `i` and bag `j`.

Example:

```
s td 3 3 5
b 1 1 2 3
```

b 2 2 3 4
b 3 2 5
1 2
2 3

Solution 4:

Graph	Solution
DBLP-v1	12
Chebyshev2	684
memplus	7686
scc_infect-dublin	36
bio-grid-mouse	595