Karlsruher Institut für Technologie

Solution to Exercise Sheet 2

Submission due by May 14th, 2026

Problem 1: Weighted Vertex Cover

8 + 4 points

In the previous version, the desired running time was $O(1.342^k \cdot n)$, where n is supposed to be the size of the input. However, in the context of graphs this is not clear.

Given is an undirected graph G with weight function $w : V \to \mathbb{N}$ for the vertices. The goal of the Weighted Vertex Cover problem is to find a vertex cover of G with minimum total weight. In the parameterized problem, we ask if there is a weighted vertex cover with weight at most k. We show that there is an FPT algorithm with the same running time as the algorithm for Vertex Cover (parameterized by the same parameter k) from the lecture, namely $O(1.342^k \cdot (n + m))$.

Part (a) Assume for now that all weights are bounded by some constant $c \in \mathbb{N}$. Describe an FPT algorithm that solves parameterized Weighted Vertex Cover on graphs with constant weights in $O(1.342^k \cdot (n + m))$. Briefly argue its correctness.

Hint: Use an FPT algorithm for Vertex Cover as a black box.

Solution 1a:

We reduce Weighted Vertex Cover with constant weights to Vertex Cover. Given a graph G for Weighted Vertex Cover, we construct a graph G' for Vertex Cover as follows: For each vertex u in G, there is an independent set V_u of size $w(u)$ in G'. If two vertices u and v are connected in G, then we add all possible edges between V_u and V_v in G'. We show that there is a vertex cover of weight at most k in G if and only if there is a vertex cover of size at most k in G'.

Let C be a vertex cover of weight at most k in G. Then, choosing all vertices in the corresponding independent sets yields a vertex cover of size at most k in G'. On the other hand, let C' be a vertex cover of size at most k in G'. We choose all vertices u in G, where V_u is *fully* included in C', and show that this is indeed a vertex cover of G: If an edge uv is not covered in G, then both V_u and V_v are not fully included in C', i.e., there are vertices $x \in V_u$ and $y \in V_v$ that are not in C'. Since u and v are connected in G, xy is an edge in G' that is not covered by C'. Thus, the chosen vertices are a vertex cover in G and have weight at most k.

Since all weights are constant, the size of G' is linear in the size of G, we can solve Weighted Vertex Cover in the same running time as Vertex Cover.

Part (b) Describe an FPT algorithm that solves parameterized Weighted Vertex Cover on general graphs in $O(1.342^k \cdot (n + m))$.

Hint: How do we obtain instances with constant weights? How should we choose c, the upper bound for the weights?

Solution 1b:

We first describe the algorithm and choose c suitably later. To obtain instances with constant weights, we branch on vertices with weight at least c. We can either choose the vertex or all vertices in its neighborhood. This gives us a branching vector of $(c, 1)$ in the worst case. By applying the branching rule exhaustively, we obtain instances with constant weights, which we can solve as described in the previous part.

By choosing $c = 6$, we get the basis 1.2852 for the first part. Since this is dominated by the second part, we obtain a running time of $O(1.342^k \cdot (n + m))$ in total.

Problem 2: 3-Hitting Set **10 points**

An instance of 3-Hitting Set consists of a universe U and a family of sets $\{S_1, \ldots, S_n\}$ with $|S_i| \leq 3$ and $S_i \subseteq U$. The goal is to decide if there is a set $H \subseteq U$ with $|H| \leq k$ such that $H \cap S_i \neq \emptyset$ for all S_i.

Give an algorithm that uses a bounded search tree with running time $2.562^k \cdot n^{O(1)}$.

Solution 2:

We solve 3-Hitting Set using a bounded search tree with the dominating branching vector $(1, 2, 2, 2, 2)$. We define the following reduction rules and branching rules.

1. Reduction rule: if there is a set S consisting of a single element x, then we include x in the solution and remove all sets that contain x.
2. Branching rule: if there is a set S with two elements, then choose one of the two elements. This yields a branching vector of $(1, 1)$.

After applying these rules exhaustively, all remaining sets have size exactly 3. We can now apply the remaining branching rules:

3. Reduction rule: if there is a set S that has no common element with any other set, then include an arbitrary element of S in the solution and remove S.
4. Branching rule: If there are two sets S and T with $|S \cap T| = 1$, then either choose the element in the intersection or one element from each set that is not in the intersection. For example, if $S = \{a, b, c\}$ and $T = \{a, d, e\}$, we can either choose $\{a\}$, $\{b, d\}$, $\{b, e\}$, $\{c, d\}$, or $\{c, e\}$. This yields a branching vector of $(1, 2, 2, 2, 2)$.
5. Branching rule: If there are two sets S and T with $|S \cap T| = 2$, then either choose one element from the intersection or both elements that are not in the intersection. This yields a branching vector of $(1, 1, 2)$.

The rules above cover all possible cases. From the branching vector $(1, 2, 2, 2, 2)$, we get a tree size of 2.562^k. Selecting and applying a rule can be done in polynomial time. Thus, we can decide 3-Hitting Set in $2.562^k \cdot n^{O(1)}$.

Problem 3: d-Hitting Set

12 points

An instance of d-Hitting Set consists of a universe U and a family of sets $\{S_1, \ldots, S_n\}$ with $|S_i| \leq d$ and $S_i \subseteq U$. The goal is to decide if there is a set $H \subseteq U$ with $|H| \leq k$ such that $H \cap S_i \neq \emptyset$ for all S_i. Give an algorithm with $(d - 0.658)^k \cdot n^{O(1)}$ running time.

Hint: Use iterative compression to obtain an algorithm with $2.342^k \cdot n^{O(1)}$ running time for 3-Hitting Set.

Solution 3:

We define d-Hitting Set Compression and Disjoint d-Hitting Set as shown in the lecture. In the Disjoint d-Hitting Set problem, a d-Hitting Set instance is given, along with a hitting set H of size $k + 1$. We ask if there is a hitting set X of size k such that $X \cap H = \emptyset$. Similarly, in the d-Hitting Set Compression problem, a d-Hitting Set instance is given, along with a hitting set H of size $k + 1$. We ask if there is a hitting set X of size k.

We first note that assuming we can solve Disjoint d-Hitting Set, we can solve the compression problem as follows. Partition the given hitting set H into all possible subsets X, Y. Let S' be the family of sets that are not hit by X. Then, Y is a hitting set on S'. Together with X, a hitting set on S' with size $|Y| - 1$ that is disjoint from hitting set Y yields a hitting set on the compression instance of size $|H| - 1$.

Furthermore, assuming we can solve the compression problem, we can solve d-Hitting Set. Begin with a universe of size k, and restrict all sets to the initial universe. We maintain a hitting set of size k. In each iteration, we extend the instance by one element, which we also add to the current hitting set. By solving the compression problem, we obtain a hitting set of size k.

We show that Disjoint $d + 1$-Hitting Set is equivalent to d-Hitting Set. Let H be a given hitting set of size $k + 1$ for Disjoint $d + 1$-Hitting Set. Then, H intersects each set in at least one element, which we cannot choose for the disjoint hitting set X. In each set, there are at most d elements that can be chosen for X, and X must contain one element of each set. This is equivalent to solving d-Hitting Set.

We prove by induction that d-Hitting Set can be solved in $O((d-0.658)^k \cdot n^{c+d})$ for some constant c. For $d = 2$, d-Hitting Set is equivalent to Vertex Cover, which can be solved in $O(1.342^k \cdot n^c)$ with the algorithm in the lecture. We have already shown that Disjoint $(d + 1)$-Hitting Set can be solved in the same running time as d-Hitting Set, namely $O((d-0.658)^k \cdot n^{c+d})$. By the lecture, we obtain a running time of $O((k + 1)(d + 1 - 0.658)^k \cdot n^{c+d})$ for $(d + 1)$-Hitting Set. Using $k \leq n$ yields $O((d + 1 - 0.658)^k \cdot n^{c+d+1})$. Since c and d are constants, this is $O((d + 1 - 0.658)^k \cdot n^{O(1)})$.

Problem 4: Vertex Cover implementation — 6 points + 8 bonus points

In this task, you should implement a program (using any programming language you prefer) that computes a minimum vertex cover for a given graph. You should use the rules given in lecture 3, but you can also implement additional rules and optimizations.

Describe in the PDF submission which rules you implemented and what further optimizations you made. Also, for each graph, specify in the PDF submission the size of the vertex cover you found. Additionally, submit the source code as well as your computed solutions (in the format described below) as a ZIP file.

You get two points for every minimum vertex cover.

Input format: The first line consists of n and m, the number of vertices and edges, respectively. The following m lines each consist of two vertices, which are connected by an (undirected) edge. The vertices are numbered from 1 to n.

Output format: The first line should contain the solution size k. The following k lines should contain the IDs of k vertices that form a minimum vertex cover.

Note: You can validate your vertex cover using the file `validator.py`. Note that minimality is not checked. Please check if your vertex covers are valid before submitting.

Solution 4:

Graph	Minimum Vertex Cover
Graph 0	35
Graph 1	119
Graph 2	102
Graph 3	81
Graph 4	72
Graph 5	80
Graph 6	112