Karlsruher Institut für Technologie

Solution to Exercise Sheet 4

Submission due by June 18th 2026

Problem 1: Application of Lenstra

4 + 8 points

Formulate an ILP to show that the following problems are in FPT:

Part (a) Variety Subset Sum: Given a multiset A of integers in $\mathbb{N}$ (unlike in a normal set, elements in a multiset can occur multiple times) and a value $b \in \mathbb{N}$. Is there a sub-multiset $X \subseteq A$ with $\sum_{x \in X} x = b$? Consider the number of distinct numbers in A as the parameter.

Solution 1a:

Let $z_1, \ldots, z_k$ be the distinct numbers in A and let m_i be the number of occurrences of z_i. We introduce integer variables $x_1, \ldots, x_k$, where x_i represents how many times z_i is chosen in the solution.

- Each number can only be chosen as often as it exists in A:

$$0 \leq x_i \leq m_i \quad \text{for } 1 \leq i \leq k$$

- The chosen elements must sum to exactly b:

$$\sum_{i=1}^{k} x_i \cdot z_i = b$$

This formulation requires no maximization function, just feasibility bounds. Because there are only k variables, Lenstra's Theorem guarantees that this problem is in FPT.

Part (b) Makespan Scheduling: Given m machines, n jobs with processing times $p_1, \ldots, p_n \in \mathbb{N}$, and an upper bound for the maximum processing duration $k \in \mathbb{N}$. Is there an assignment of the jobs to the machines such that no machine requires more than k time to process all its jobs? Consider the bound k on the processing duration as the parameter.

Hint: How can the possible job assignments to a machine look like?

Solution 1b:

Instead of tracking which machine executes which job individually, we keep track of *machine configurations*. A configuration $c = (c_1, \ldots, c_k)$ describes exactly how many jobs of each length a

single machine processes. In a solvable instance all processing times will be smaller than k and therefore a configuration can be described by at most k integers. In addition, a machine can only process jobs of a specific length at most k times and thus $c_i \leq k$ for all i. Therefore, the set of all valid configurations C is strictly bounded by a function of k, $|\mathcal{C}| \leq (k+1)^k$.

For each valid configuration $c \in \mathcal{C}$, we introduce an integer variable $x_c \geq 0$ representing the number of machines utilizing that specific configuration. The constraints are:

- We can use at most m machines in total:
$$\sum_{c \in \mathcal{C}} x_c \leq m$$
- Let n_t be the total number of jobs of length t. Exactly n_t jobs of that length must be processed across all machines:
$$\sum_{c \in \mathcal{C}} c_t \cdot x_c = n_t \quad \text{for } 1 \leq t \leq k$$

This formulation requires no maximization function. The number of variables is exactly $|\mathcal{C}|$, which only depends on the parameter k, Lenstra's Theorem states this problem is in FPT.

Problem 2: Max Sat

8 + 8 = 16 points

Given a boolean formula (in CNF) with n variables and m clauses. Given a parameter k, the parameterized Max Sat problem asks for a variable assignment that satisfies at least k clauses.

Part (a) Provide safe reduction rules that yield a kernel with at most $2k$ clauses and k variables.

Hint: Use Hall's Theorem to reduce the number of variables.

Satz (Hall's Theorem)**.** *Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. There exists a matching in G that covers all vertices of V_1 if and only if $|X| \leq |N(X)|$ for every subset $X \subseteq V_1$. Otherwise, an inclusion-minimal set $X \subseteq V_1$ with $|X| > |N(X)|$ can be efficiently found.*

Solution 2a:

Reduction Rule 1 ($m > 2k$): For any variable assignment A and a given clause, A or its complement $\overline{A}$ will satisfy that clause. Thus, at least one of these two assignments must satisfy $\geq \frac{m}{2} > k$ clauses and the instance is solvable.

Reduction Rule 2 ($n > k$): We construct a bipartite incidence graph $G = (V \cup C, E)$ where an edge exists if a variable appears in a clause (regardless of whether it is positive or negative). If a matching covers V, we can satisfy $\geq n$ clauses by setting each variable to satisfy its matched clause. As $n > k$, the instance is solvable. Otherwise, by Hall's Theorem, we can efficiently find an

inclusion-minimal $X \subseteq V$ with $|X| > |N(X)|$. The reduction rule removes X and $N(X)$, reducing the new parameter to $k' = k - |N(X)|$.

If the original instance is solvable with a solution of size k, then the new instance is solvable with a solution of size k', as the variables in X can only satisfy at most $|N(X)|$ clauses. On the other hand, if the new instance is solvable with a solution of size k', then the original instance is solvable with a solution of size k. To see this, note that X is inclusion-minimal (for every proper subset $Y \subset X : |Y| \leq |N(Y)|$), so we can apply Hall's theorem to $X' = X \setminus \{x\}$ for any $x \in X$ to find a matching that covers X'. It is easy to see that $|X'| = |N(X')| = |N(X)|$, and we obtain a perfect matching between X' and $N(X)$. Therefore, we can safely assign the variables in X' to satisfy all clauses in $N(X)$, extending a solution of size k' to a solution of size k.

Applying both rules exhaustively yields a kernel with a maximum of $m \leq 2k$ clauses and $n \leq k$ variables.

Part (b) Provide an FPT algorithm for the following parameterization "above $\frac{m}{2}$" with parameter k: Is there a variable assignment that satisfies at least $\frac{m}{2} + k$ clauses?

Hint: Consider clauses with exactly one variable separately from larger clauses and show first that many larger clauses lead to a large solution.

Solution 2b:

For any conflicting unary clauses (x) and $(\neg x)$, we remove both. This reduces the number of clauses by 2 and the number of solved clauses by 1, so the parameter remains unchanged. We apply this exhaustively so each variable appears at most once in unary clauses. Let u be the number of unary clauses and g the number of larger clauses, such that $m = u + g$. Because there are now no conflicting unary clauses, we can satisfy all u clauses independently. If $u \geq \frac{m}{2} + k$, the instance is immediately solvable.

If $u < \frac{m}{2} + k$, we consider the expected number of satisfied clauses for a uniformly random assignment. Such an assignment satisfies each unary clause with probability $\frac{1}{2}$ and larger clauses with probability $\geq \frac{3}{4}$. Let T_i be the indicator variable for whether clause i is satisfied. By linearity of expectation, the expected number of satisfied clauses is

$$E\left[\sum_{i=1}^{m} T_i\right] = \sum_{i=1}^{m} E[T_i] \geq \frac{u}{2} + \frac{3g}{4} = \frac{m}{2} + \frac{g}{4}.$$

If $g \geq 4k$, the expectation is $\geq \frac{m}{2} + k$, and the probabilistic method guarantees a solution.

If neither case applies, we know that $u < \frac{m}{2} + k$ and $g < 4k$. Summing these bounds gives:

$$m = u + g < \frac{m}{2} + 5k \implies m < 10k.$$

Since $m < 10k$, the total number of clauses required to solve the instance is bounded purely by a

function of k. Therefore, we can apply the reduction rules from part (a) to obtain a small kernel. This kernel can then be solved via brute force.

Problem 3: Weighted Vertex Cover on Trees **12 points**

Given an undirected graph G with weight function $w : V \to \mathbb{N}$ for the vertices. The goal of Weighted Vertex Cover is to find a vertex cover of G with minimum total weight. In this task we only consider trees. Formulate a dynamic program that determines the minimum weight of a vertex cover for a given tree. What is the running time of your DP?

Solution 3:

We first root the tree T at some vertex r and denote the subtree rooted at v by T_v. For each vertex v, we want to determine the minimum weight of a vertex cover of T_v. Our DP distinguishes between the cases where v is included in the vertex cover ($\text{vc}[v,1]$) and where it is not ($\text{vc}[v,0]$). The minimum of $\text{vc}[r,0]$ and $\text{vc}[r,1]$ then contains the minimum weight of a vertex cover of T.

We compute these values bottom-up, starting from the leaves. For a leaf v, we set $\text{vc}[v,0] = 0$ and $\text{vc}[v,1] = w(v)$. For an internal vertex v, the following holds: If we do not include v in a vertex cover of v, then the edges between v and its children must be covered by v's children. That is, a vertex cover without v contains all of v's children. If, on the other hand, we include v in a vertex cover of T_v, then the edges between v and its children are already covered by v. Thus, for each child u of v, we can choose a minimum-weight vertex cover of T_u (with u or without). We obtain:

$$\text{vc}[v,0] = \sum_{u \in \text{child}(v)} \text{vc}[u,1] \quad \text{and} \quad \text{vc}[v,1] = w(v) + \sum_{u \in \text{child}(v)} \min_{x \in \{0,1\}} \text{vc}[u,x].$$

The dynamic programming can be computed in linear time since each edge is considered only a constant number of times.