Computational Complexity — tutorial 12

Fine-grained complexity 2, FPT 1

From **Strong Exponential Time Hypothesis (SETH)** it follows that CNF-SAT cannot be solved in $(2-\varepsilon)^n$ time for any $\varepsilon > 0$.

Orthogonal Vectors Conjecture (OVC). The following decision problem cannot be solved in $O(n^{2-\varepsilon} \cdot \text{poly}(d))$ time for any $\varepsilon > 0$:

> Orthogonal Vectors
> Input: two sets $A = \{v_1, v_2, \ldots, v_n\}$, $B = \{w_1, w_2, \ldots, w_n\}$ of bit vectors, each of length d
> Output: are there two vectors $v_i \in A$, $w_j \in B$ such that $\langle v_i, w_j \rangle = 0$? *That is, on each position, at least one of these two vectors should have a* 0 *bit.*

1. Prove that the following statements are equivalent. Note that (a) is equivalent to the negation of OVC.

(a) For some $\varepsilon > 0$, there exists an $O(n^{2-\varepsilon} \cdot \text{poly}(d))$ algorithm solving Orthogonal Vectors.

(b) For some $\varepsilon > 0$, there exists an $O(n^{2-\varepsilon} \cdot \text{poly}(d))$ algorithm for Orthogonal Vectors which additionally returns a pair of orthogonal vectors if it exists.

(c) For some $\varepsilon > 0$, there exists an $O(n^{2-\varepsilon} \cdot \text{poly}(d))$ algorithm for Orthogonal Vectors constrained to $A = B$.

(d) For some $\varepsilon > 0$, there exists an $O(n^{1.5-\varepsilon} \cdot \text{poly}(d))$ algorithm solving the following problem:

> Square Root Orthogonal Vectors
> Input: two sets $A = \{v_1, \ldots, v_n\}$, $B = \{w_1, \ldots, w_{\sqrt{n}}\}$ of bit vectors, each of length d
> Output: are there two vectors $v_i \in A$, $w_j \in B$ such that $\langle v_i, w_j \rangle = 0$?

(e) For some $\varepsilon > 0$, there exists an $O(n^{2-\varepsilon} \cdot \text{poly}(d))$ algorithm for the following problem:

> Subset Vectors
> Input: two sets $A = \{v_1, v_2, \ldots, v_n\}$, $B = \{w_1, w_2, \ldots, w_n\}$ of bit vectors, each of length d
> Output: are there two vectors $v_i \in A$, $w_j \in B$ such that for each bit set in v_i, the corresponding bit in w_j is also set?

2. Assuming SETH, prove that the following problem cannot be solved in $O(n^{k-\varepsilon} \cdot \text{poly}(d))$ time for any $\varepsilon > 0$:

> k-Orthogonal Vectors
> Input: k sets $A_1, A_2, \ldots, A_k$ of n bit vectors, each of length d
> Output: are there k vectors $w_1 \in A_1$, $w_2 \in A_2$, …, $w_k \in A_k$ such that, on each of d positions, at least one of the chosen vectors has a 0 bit?

Note that 2-Orthogonal Vectors is exactly Orthogonal Vectors.

A problem $L \subseteq \Sigma^* \times \mathbb{N}$ is **parameterized** if the inputs (w, k) to this problem consist of the word $w \in \Sigma^*$ and a parameter $k \in \mathbb{N}$. E.g.

- Given a graph G and a parameter k, does G have a vertex cover of size k?
- Given a graph G and a parameter k, does G have a clique of size k?
- Given a graph G and a parameter k such that G has treewidth $\leqslant k$, does G have a Hamiltonian cycle?

A problem $L \subseteq \Sigma^* \times \mathbb{N}$ is **fixed-parameter tractable** (FPT) if it works in $f(k) \cdot \text{poly}(n)$ time, where f is a computable function and $n = |w| + k$ is the size of the input (input encodes both w and k).

3. Recall from the lecture the $2^k \cdot (n + m)$ time algorithm solving Vertex Cover.

An **FPT reduction** from problem L to problem M is an algorithm mapping the inputs (w, k) of the problem L to the inputs (w', k') of the problem M such that:

- $(w, k) \in L \Leftrightarrow (w', k') \in M$,
- the reduction works in $f(k) \cdot \text{poly}(n)$ time for some computable function f,
- $k' \leqslant g(k)$ for some computable function g.

There are some problems predicted to **not** be in FPT, e.g. Clique, Dominating Set.

4. Take any two problems L, M such that there is an FPT reduction from L to M. Prove that if $M \in \mathsf{FPT}$, then also $L \in \mathsf{FPT}$.
(Conversely, if $L \notin \mathsf{FPT}$, then $M \notin \mathsf{FPT}$.)

5. Find an FPT reduction from Clique (parameterized by the size k of the clique) to Independent Set (parameterized by the size k of the independent set). Note that it follows that we don't expect Independent Set to be FPT.

6. Propose a reduction from Independent Set to Vertex Cover. Why doesn't it work as an FPT reduction?

7. Find an FPT reduction from from Clique to Clique on Regular Graphs (given a graph G such that each vertex has the same degree, and an integer k, decide if G has any k-clique).

8. Find an FPT reduction from Clique on Regular Graphs to Independent Set on Regular Graphs.

9. Find an FPT reduction from Independent Set on Regular Graphs to Partial Vertex Cover defined below:

> Partial Vertex Cover
> Input: graph G, two integers k, s
> Parameter: k (the size of the solution)
> Output: is there a subset of k vertices which is incident to at least s edges?

Note that it follows that we don't expect Partial Vertex Cover to have an FPT algorithm (even though Vertex Cover has many such algorithms!)

10. Find an FPT reduction from Dominating Set to Bipartite Dominating Set defined below:

> Bipartite Dominating Set
> Input: a bipartite graph G ($V(G) = A \cup B$ and there are only edges between A and B), integer k
> Parameter: k (the size of the solution)
> Output: is there a subset of k vertices of A such that every vertex of B is dominated by some selected vertex of A?

11. Consider the following recursive algorithm solving the parameterized variant of Vertex Cover:

```
# Returns True if [graph] has a vertex cover of size at most [k].
def HasVertexCover(graph, k):
    if k < 0: return False
    if graph.empty(): return True
    if graph has no vertices of degree more than 2:
        # Each component of [graph] is either a path or a cycle.
        # The minimum vertex cover in each such component can be computed using
        #   simple formulas.
        min_vertex_cover = ComputeVertexCoverForPathsAndCycles(graph)
        return (k >= min_vertex_cover)
    else:
        v = vertex of maximum degree in [graph]
        A = the set of neighbors of [v]
        # Two cases:
        #  a) we take [v] to the vertex cover; or
        #  b) we don't take [v] to the vertex cover; then, we must take
        #    all vertices in [A] to the cover.
        return (
            HasVertexCover(graph.remove(v), k - 1) or
            HasVertexCover(graph.remove(v).remove(A), k - len(A)))
```

This is the algorithm from the lecture, with an additional case where the graph has only vertices of small degree.

Prove that this algorithm solves Vertex Cover in time $1.466^k \cdot (n + m)$.

Hint: let $f(k)$ be maximum number of recursive calls to `HasVertexCover` *if the initial parameter is equal to k. Prove that $f(k) \leqslant f(k-1) + f(k-3)$ for $k \geqslant 3$.*