23. Kernelization

Idea:
- Make the input smaller then the search will be quicker.
- Can be understood as a theory of preprocessing.

Observation:
- Many instances can be divided into
  - a part that is easy to solve and
  - a hard kernel of the instance.
- Use costly computation only on the hard kernel.
- Hope that kernel size is bounded by the parameter.

Technically:
- Reduce problem instance I to an equivalent instance I'.
- The size of I' is bounded by some $f(k)$.
- The instance I' is then analyzed, potentially using an exponential subroutine.
- If a solution exists for I', it can be lifted to a solution for I.

Hope: Often, kernelization will add an additive factor $f(k)$ rather than a multiplicative one.

Perspective:
- Iterative compression
- Interleaving of kernelization and bounded search trees.

23.1 Case Study

Goal: Apply kernelization to VERTEXCOVER.

Theorem (Buss & Goldsmith '93):
VERTEXCOVER is solvable in $O(sn + k^{4k})$.

Proof:

The proof is divided into a preprocessing and an analysis phase.

Preprocessing phase:

- For a graph $G=(V,E)$, any vertex of degree $> k$ must belong to every k-element vertex cover.
  Indeed, otherwise we have to pick all neighbors to cover the edges and this leads to a vertex cover $> k$.

- Moreover, if a vertex has degree 1 the vertex to which it is connected should be used without loss of generality.

The two observations lead to the following reduction rules, which we apply
↳ exhaustively
↳ in non-deterministic order
↳ but at most k-times.

Initially, $k' = k$

Reduction rule 1: If v has degree 1, delete v and its neighbor u.
Reduce the parameter to $k' := k' - 1$.

Reduction rule 2: Locate all vertices of degree $> k'$.
Let p be the number of such vertices.
If $p > k'$, there is no k-element vertex cover.
Otherwise, pick such a v, delete it,
and set $k' := k' - 1$.

Reduction rule 3: Remove isolated vertices.

Analysis phase:

- If the resulting graph G' has $> k'(k'+1)$ vertices, reject.

The bound is justified by the fact that
- a graph with k'-vertex cover and
- degrees bounded by k'

cannot have more than $k'(k'+1)$ vertices.

- Now search all size k' subsets of G' to check if one forms a vertex cover.
If yes, the detected vertices in the reduction phase and the found cover will constitute a vertex cover of G.
- To analyse the runtime, note that the worst case is that there are no reductions in the first phase.
Then G' has size at most $k(k+1)$.
The complete search would make $\binom{k\cdot(k+1)}{k}$ many steps, which is bounded by $O(k^{4k})$ □

Improvement: First kernelize, then apply the bounded search tree method.

Corollary:
VERTEXCOVER is solvable in time $O(\underbrace{n}_{\text{additive factor}} + \overbrace{f(k)\,k^2}^{\text{FPT}})$, where $f(k)$ is any constant bound by other methods, provided they work on a k^2-sized kernel (in linear time).

23.2 Kernelization as a Characterization of FPT

Goal: Show that kernelization is a universal method to construct FPT algorithms.

Needed: A formal definition of kernelization.

Definition (Kernelization):

Let $L \subseteq \Sigma^* \times \Sigma^*$ be a parameterized language.

A reduction to a problem kernel, also called kernelization, replaces an instance (I,k) by a reduced instance (I',k'), called the problem kernel, so that

(1) $k' \leq k$

(2) $|I'| \leq g(k)$ for some function g depending only on k

(3) $(I,k) \in L$ if and only if $(I',k') \in L$.

The reduction $(I,k) \mapsto (I',k')$ must be computable in time $\text{poly}(|I|+k)$.

Comment:
- To obtain FPT, Requirement (1) is not really necessary
- We could also reduce to a different parameterized problem L'. In this case, the method is called bi-kernelization.

Towards the main theorem:
- If a graph is big enough (and not a clique), it must have a size k independent set (set of vertices such that no two are adjacent).
- Hence, if the graph is large, answering the decision problem is easy.
- Otherwise, the small remaining instances can be solved by a table look-up.
- This reasoning applies to all FPT algorithms.

Theorem: A language L is FPT iff it is kernelizable.

Proof: "$\Leftarrow$" is easy.

"$\Rightarrow$": Let L be FPT in time $\mathcal{O}(f(k)\cdot |x|^c)$ on input (x,k).
Let the algorithm be $\mathbb{A}$.

- Observe that from some input length $|x|$ onwards,
  we have $f(k)\cdot |x|^c < |x|^{c+1}$.
- We simulate $\mathbb{A}$ for $|x|^{c+1}$ many steps.
  - ↳ If $\mathbb{A}$ terminates, we know the answer.
    Return a trivial yes or no instance of the problem.
  - ↳ If $\mathbb{A}$ does not terminate, we can conclude
    $$|x| < f(k).$$

  Choose: $k' = k$
  $g = f$

  The kernel is thus a look-up table
  for instances of size $\leq f(k)$. □

The look-up table formulation is a bit of an advertisement.
If one really wanted to construct the table,
one would have to put quite some computational effort into it.