1 Iterative compression: disjoint variant in FPT time

As mentioned last week, Feedback Vertex Set in undirected graphs in an example of a problem that admits an FPT algorithm via the iterative compression, but the disjoint variant of the problem is (likely) not solvable in polynomial time.

In the Disjoint Feedback Vertex Set, we are given a graph $G = (V, E)$, a subset of forbiden vertices $W \subset V$, which is an FVS (i.e., $G[V \setminus W]$ is a forest), and an integer k. The goal is to find a FVS in G of size at most k disjoint from W. Note that in the definition we do not assume that the size of W is bounded by k or a function of k. This more general problem definition will be convenient to formulate a recursive algorithm.

Please read Chapter 4.3.1 in the FPT book (pages 87–88) for an $\mathcal{O}^*(4^k)$ time algorithm for Disjoint FVS. Note that the algorithms is an interesting example of the bounded search tree technique where the progress measure is different than the prarameter.

2 Algorithms for computing treewidth

The exact treewidth of a graph can be found in $k^{\mathcal{O}(k)} \cdot n$ time, with an algorithm due to Bodlaender. Alternatively, there is a 4-approximate algorithm – i.e., it outputs a tree decomposition of width $4w + 4$ or concludes that the treewidth is greater than than w – running in time $\mathcal{O}(8^k k^2 \cdot n^2)$, and a 5-approximate algorithm running in time $2^{\mathcal{O}(k)} \cdot n$.

Usually, when we talk about running times of FPT algorithms for graph problems parameterized by treewidth, we assume that a tree decomposition is given on input.

3 Nice decomposition

Constructing dynamic programming algorithms working with tree decomposition is often much easier if we assume the decomposition is a *nice*. A nice tree decomposition is a rooted tree that satisfies the basic requirements of a tree decomposition and two additional ones. First, the root and all the leaves have empty bags. Second, each node is either an *introduce node*, a *forget node*, or a *join node*. We say that a node t is:

- an introduce node if it has exactly one child s, and $B_t = B_s \cup \{v\}$, for a vertex $v \in V$;
- a forget node if it has exactly one child s, and $B_t = B_s \setminus \{v\}$, for a vertex $v \in V$;
- a join node if it has exactly two children s_1 and s_2, and $B_t = B_{s_1} = B_{s_2}$.

Given a width w tree decomposition T of graph $G = (V, E)$, one can compute, in time $\mathcal{O}(w^2 \cdot \max(|V|, |T|))$, a nice tree decomposition of the same width that has $\mathcal{O}(w \cdot |V|)$ nodes.

4 Win-win

We often use an algorithm for small treewidth graphs as a subroutine in an algorithm for general graphs. A common method to do that is to notice that the problem has a trivial solution if the treewidth is too large.

For example, consider the Vertex Cover problem paramterized by solution size. If a graph admits a vertex cover of size at most k, then it has treewidth at most k. Indeed, consider a tree decomposition that is a path of length $n-k$ and each of its nodes contains the same k vertices from a vertex cover and one other vertex. To exploit this idea algorithmically, we can first compute (an approximation of) the treewidth, and if it is too large, we immediately answer NO, otherwise we run an FPT algorithm parameterized by treewidth.

More interesting results of this kind are based on grid theorems. For an integer t, t-grid is the graph $G=(V,E)$ with $V=[t]\times[t]$ and

$$E=\big\{\big((u,v),(u+1,v)\big)\mid(u,v)\in[t-1]\times[t]\big\}\cup\big\{\big((u,v),(u,v+1)\big)\mid(u,v)\in[t]\times[t-1]\big\}.$$

Theorem 1 (Chekuri-Chuzhoy, STOC'14)**.** *If a graph does not contain t-grid as a minor, then its treewidth is bounded by $t^{98+o(1)}$.*

Theorem 2 (Robertson-Seymour-Thomas and Gu-Tamaki)**.** *If a planar graph does not contain t-grid as a minor, then its treewidth is bounded by $\frac{9t}{2}$, and its $(\frac{9t}{2}+\varepsilon)$-width tree decomposition can be found in $\mathcal{O}(n^2)$ time.*

Note that t-grid contains a matching of size $t^2/2$, hence its vertex cover size is at least $t^2/2$, and taking a minor can only decrease the vertex cover size. Therefore, if a planar graph has treewidth larger than $\sqrt{k}\cdot const$, then it cannot have a vertex cover smaller than k. This easily leads to an algorithm for Vertex Cover in planar graphs (parameterized by solution size) running in time $2^{\mathcal{O}(\sqrt{k})}n$.

Similar arguments lead to $2^{\mathcal{O}(\sqrt{k})}$ or $k^{\mathcal{O}(\sqrt{k})}$ time algorithms for, e.g., Independent Set, Dominating Set, Feedback Vertex Set, or Longest Path, all in planar graphs.