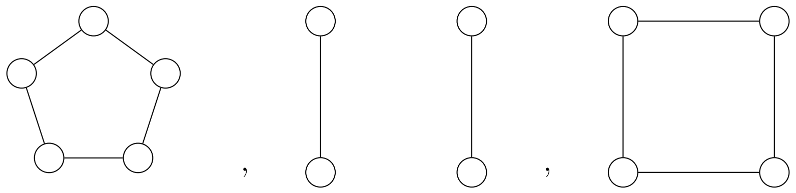
Parameterized Algorithms WS 2021
Prof. Dr. P. Rossmanith
Dr. E. Burjons, M. Gehnen, H. Lotze, D. Mock

Date: November 22th, 2021

Exercise Sheet with solutions 05

Task T16

A graph $G = (V, E)$ is a *split graph* if its vertex set can be partitioned into sets C and I such that C is a clique and I is an independent set. Show that a graph is a split graph if and only if it does not contain the following three graphs as induced subgraphs:



In the *Split Vertex Deletion* problem, given a graph G and an integer k, the task is to check if one can delete at most k vertices from G to obtain a split graph. Can you find an algorithm which solves this problem in $5^k n^{\mathcal{O}(1)}$? Can you find a different algorithm solving this problem in $2^k n^{\mathcal{O}(1)}$?

Solution

We can prove this property by a case distinction.
For the 5^k algorithm, we first search for a forbidden subgraph in polynomial time and branch on the removal of 5 (or 4) vertices. For the 2^k algorithm, we use iterative compression. The task reduces to showing that the disjoint version of the problem is polynomial-time solvable. The main observation is that a split graph on n vertices has at most n^2 partitions of the vertex set into the clique and independent set part: for a fixed one partition, at most one vertex can be removed from the clique side to the independent one, and at most one vertex can be moved in the opposite direction. (In fact one can show that it has at most $O(n)$ split partitions.) Hence, you can afford guessing the “correct” partition of both the undeletable and the deletable part of the instance at hand.

Task T17

The r-Regular Vertex Deletion problem is defined as follows: given a graph G and an integer k, decide whether there is a set $S \subseteq V(G)$ of size at most k whose deletion results in an r-regular graph. A graph is r-regular if every vertex has degree exactly r. Show that this problem admits an algorithm with running time $O((r + 2)^k \cdot \text{poly}(n))$.

Solution

First observe that any vertex of degree $< r$ must necessarily be taken into the solution as we cannot increase the degree of any vertex by removing other vertices. This reduction will be applied after each branching step.
The branching itself proceeds as follows: pick any vertex v of degree larger than r. If no such vertex exists we are done; as the above reduction rule already took care of all vertices of degree less than r. Otherwise, pick any set $A \subseteq N(v)$ of $r + 1$ neighbours of v. Note that from the set $v \cup A$ we *must* delete at least one vertex to obtain an r-regular graph. Therefore, we can branch on $r + 2$ different cases of choosing a vertex from $v \cup A$ to be part of the solution.
This yields the desired $O((r + 2)^k\, poly(n))$-algorithm.

Task H11 (5 credits)
Let $G = (L \dot\cup R, E)$ be a bipartite graph. Suppose that $L_1 \dot\cup L_2 = L$ and $R_1 \dot\cup R_2 = R$ are partitions of the vertex sets L and R. Prove the following:

1. $(L_1 \cup R_1, L_2 \cup R_2, E)$ is a bipartite graph iff there are no paths for the following pairs of vertex sets: L_1 and L_2; L_2 and R_2; R_2 and R_1; R_1 and L_1.
2. One can find a minimum set X such that $G - X$ does not contain any of the above paths in polynomial time [Hint: use a flow algorithm].

Solution

1. First note that a path from L_1 to L_2 or from R_1 to R_2 would have even length as L, R is a biparition of the graph. As L_1 and L_2 / R_1 and R_2 should occur on opposite sides in the new bipartition, such paths must not exist. For the paths between L_2,R_2 and L_1,$_R1$ the same argument holds, but in the other direction: as $L_1 \cup R_1, L_2 \cup R_2$ should be a valid bipartition, these paths would have even length, but that would contradict L, R being a valid bipartition.
2. The idea is to create a flow network $G_{s,t}$ as follows: add source s and sink t to the graph (now interpreted as a network) and connect s to all vertices in L_1 and R_2 and, similarly, connect t to all vertices in L_2 and R_1. All edges are assigned a capactiy of one.

   We now want to calculate a maximal flow from s to t and use the max-flow/min-cut theorem to obtain X—however, the direct application does not work as this would give us a minimal *edge* cut. By a simple construction, however, we can transform our network $G_{s,t}$ to another network $G'_{s,t}$ such that the minimum edge-cut of $G'_{s,t}$ corresponds to a minimum vertex-cut in $G_{s,t}$. This is possible by transforming each vertex $v \in G_{s,t}, v \notin \{s, t\}$ into two vertices v_{in}, v_{out} connected by and arc (v_{in}, v_{out}) in such a way that all incoming arcs of v are now going into v_{in} and all outgoing arcs of v now are originating from v_{out}. We will skip the proof of this construction as it can be found in various textbooks on efficient algorithms.

Task H12 (5 credits)
Use the insights you gained from H11 to design a $O(3^k n^{O(1)})$-algorithm for Odd Cycle Transversal using iterative compression.

Solution
Consider an instance (G, S, k) of the compression routine for Odd Cycle Transversal, i.e. S is a vertex set of G of size $k + 1$ such that $G - S$ is bipartite. We iterate through all three-partitions $Y \dot\cup L \dot\cup R = S$ of the previous solution S, where Y denotes the set of vertices which we want to keep for the new solution and L and R denote the vertices that will *not* be part of the new solution. This further assigns a side (left or right) for each such vertex.
If L, R is not a bipartition we can immediately continue with the next three-partition as this cannot be possibly extended to a valid solution.
Now, given L and R, our task is to find a set $X \in G - S$ such that $G - (X \cup Y)$ is a bipartite graph. We further restrict our search to sets X which admit a bipartition of $G - (X \cup Y)$ such that the sets L and R occur on opposite sides. Assume in the following that Y has been removed from G. Observe that $N(L) \setminus R$, the neighbours of the set L in $G - S$, must be put on the right side of the final bipartition, and similarly the vertices of $N(R) \setminus L$ must be put on the left side. Vertices in $N(L) \cap N(R)$ *cannot* occur in a bipartite graph, so we have to take

them into X immediately (again assume for simplicity that we remove these vertices from the graph).

Now we have essentially the situation described in exercise T26: let $L_1 = L, L_2 = N(R) \backslash L, R_1 = R, R_2 = N(L) \setminus R$. The only difference now is there also exists a set of not-assigned vertices (namely the vertices in $G - S$ not connected to L or R), however, the above proofs work exactly the same with this small addition. It follows that we can now use the above outlined flow algorithm to find X in $G - S$ in polynomial time, yielding a $O(3^k\, poly(n))$ algorithm for Odd Cycle Transversal.