RWTH Aachen **WS 2011/12**
Lehrgebiet Theoretische Informatik Problem Set 10
Langer—Reidl—Rossmanith—Sikdar 20.01.2012

Parameterized Algorithms Tutorial

Tutorial Exercise T26

Let $G = (L \uplus R, E)$ be a bipartite graph. Suppose that $L_1 \uplus L_2 = L$ and $R_1 \uplus R_2 = R$ 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].

Proposed 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,$_R$1 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.

Tutorial Exercise T27

Use the insights you gained from T26 to design a $O(3^k n^{O(1)})$-algorithm for Odd Cycle Transversal using iterative compression.

Proposed 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 $Y \uplus L \uplus R = S$ 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 $L_1 = L, L_2 = N(R) \setminus L, R_1 = R, R_2 = N(L) \setminus R$. 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.

Homework H22

Given a graph $G = (V, E)$, a *perfect code* for G is a vertex set $S \subseteq V(G)$ such that for all $v \in V(G)$ there is exactly one vertex in $N[v] \cap S$. The Perfect Code problem is defined as follows: given a graph $G = (V, E)$ and an integer parameter k, decide whether G has a perfect code with k vertices. This problem is W[1]-complete on general graphs. Show that this problem is fixed-parameter tractable if we assume that the input graph is planar. Use the fact that every planar graph has a vertex of degree at most five.

Proposed Solution

We want to employ a simple branching algorithm, however, we somehow need to preserve the information that a) a vertex has a neighbour part of the *perfect code* and b) that a vertex cannot be picked as this would cover some neighbour multiple times.

To this end, we mark vertices as *covered* if they have a neighbour that is part of the perfect code and *forbidden* if it is not covered itself but has a covered neighbour.

Now, our recursive algorithm proceeds as follows: pick a vertex v of degree at most five from the graph and branch on the at most six possibilities of including one vertex of $N[v]$ in the solution—of course avoiding vertices marked as *covered* or *forbidden*. In each branch, if vertex $w \in N[v]$ is picked, we remove w from the graph and mark all vertices in $N(w)$ as *covered* and all vertices in $N^2(w)$ (the set of all vertices with distance 2 to w) as *forbidden*.

The correctness of the above algorithm can be verified easily.

Homework H23

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 $O((r+2)^k \cdot \text{poly}(n))$.

Proposed 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.