Advanced Algorithms Course. Lecture Notes. Part 10

Kernelization

For the problem of finding a vertex cover of size at most k we have shown the time bound $O(1.47^k kn) = O^*(1.47^k)$. Can we also improve the polynomial factor?

Observe that any node i of degree larger than k is necessarily in the solution. (If we do not select i, we have to take all neighbors, but these are too many.) Thus we can put aside all nodes of degree larger than k. This can be done in $O(kn)$ time. There remains a graph where all nodes have degree at most k. Now, k vertices can cover at most k^2 of the remaining edges. Hence, if the remaining graph has more than k^2 edges, we know that there is no solution. This also means: In the positive case we have found a subgraph with at most k^2 edges, such that it remains to solve the hard Vertex Cover problem on this small graph only. The resulting time bound is $O(1.47^k k^2 + kn)$. Note that we got rid of the product of the exponential term and n.

The above process is called **kernelization**, and the remaining small graph is called a problem **kernel**. We skip the exact technical definition, however we observe that the size of the kernel depends only on the parameter k, but not on the original number n of nodes, and the kernelization needs only polynomial time.

To put these results in a much more general context: Kernelization is just a formal way of preprocessing an input, and it is widely used also outside FPT problems. The idea is to take away simple parts of an instance and give a miniaturized instance of the hard problem to the actual algorithm. Thus, the underlying algorithm has to deal only with the hard part of the instance, and if this is significantly smaller than the original instance, this saves much computation time.

Finding a Path by Color Coding

Color coding is a beautiful combination of ideas from both randomized and FPT algorithms. As an example we consider the following k-path problem. Given is a graph G and an integer k. We want to find a path of k nodes that does not cross itself, i.e., each of the k nodes shall be visited only once.

What is the motivation? Why should one be interested in finding such a k-path? Here is one real application from computational biology: Molecules like proteins, DNA, RNA are long sequences. Under some experimental conditions one cannot observe these sequences directly but obtain only information about pairs of short molecules that could possibly be neighbors in a sequence. The reconstruction of sequences from such local information leads to the problem of finding simple paths, of at least some prescribed length, in the graph of possible neighborhood relationships.

At first glance the k-path problem looks simple. We may start from some node and try all possible paths of length k. But if we do that naively by breadth-first-search, we need $O(n^k)$ time, showing that the problem is in XP. In fact, finding a k-path is easy if the graph has diameter at least k. Then it suffices to compute shortest paths between all pairs of nodes, which is possible in polynomial time. Since some of these paths has the desired length, and no nodes appear repeatedly on the path, we get a solution. But ironically, a smaller diameter makes the problem hard. Now we devise an algorithm that works also in this case.

The idea of color coding is to use k colors and assign one color to each node, randomly and independently. (Do not confuse it with the graph coloring problem where adjacent nodes must get different colors. This restriction is not applied here.) Then, we only search for a k-path where all k colors appear. The point is that this subproblem can be solved in $O(2^k n^2)$ time! Now we show how this is done, and how this result is used to solve the original problem.

Let $c(v)$ denote the color of node v. We do dynamic programming on all 2^k subsets C of the colors. For every set C and every node v we compute a value $p(C, v)$ which is 1 if there exists a path of $|C|$ nodes that ends in v and contains exactly the colors from C, otherwise we define $p(C, v) := 0$. For $|C| = 1$, we obviously have $p(C, v) = 1$ if and only if $C = \{c(v)\}$. Next suppose that we know all $p(C, v)$ with $|C| = i$. Then all $p(C', v)$ with $|C'| = i + 1$ are obtained as follows. We have $p(C', v) = 1$ if and only if $p(C' \setminus \{c(v)\}, u) = 1$ for some node u adjacent to v. The time bound is easy to see.

The algorithm succeeds if some path with k nodes (if it exists) actually carries all k colors. Let P be a fixed path of k nodes. It can be colored in k^k different ways, and $k!$ colorings are good. Hence the success probability in every attempt is $k!/k^k$, which is essentially $1/e^k$ due to Stirling's formula. It follows that we need $O(e^k)$ iterations to find a k-path with high probability, and if we do not detect some, we can report that no k-path exists, with an arbitrarily small error probability. The time is $O((2e)^k n^2) = O^*((2e)^k)$ in total.

Enumerating all Maximal Cliques

Despite NP-completeness of the Clique problem we can efficiently generate all maximal cliques (maximal with respect to set inclusion) in an undirected graph, say, with node set $\{v_1, \ldots, v_n\}$. The problem is of interest, e.g., in the analysis of networks. The time needed for enumeration is dominated by the number k of maximal cliques, which can be pretty small in some classes of graphs. The method is, essentially, dynamic programming on subsets again.

Let $N[v]$ denote the neighborhood of a node v, that is, v together with all its adjacent nodes. Suppose that we have already computed the family of all maximal cliques C in the subgraph induced by $\{v_1, \ldots, v_j\}$. (Induction base $j = 1$ is trivial.) We want to generate all maximal cliques C in the subgraph induced by $\{v_1, \ldots, v_{j+1}\}$.

Now observe: If $C \subset N[v_{j+1}]$, then $C \cup \{v_{j+1}\}$ is a maximal clique. Otherwise C surely remains a maximal clique, and $N[v_{j+1}] \cap C$ may also be a maximal clique, however its maximality must be checked. In either case, every C generates at least one maximal clique $\{v_1, \ldots, v_{j+1}\}$, and these maximal cliques are all distinct. It follows that the number of maximal cliques can never decrease when we go from j from $j+1$. Hence we never have to maintain more than k cliques. All auxiliary operations run in polynomial time. Thus, the problem is in FPT with parameter k.