Advanced Algorithms Course. Lecture Notes. Part 9

Dynamic Programming on Subsets

In the k-path problem we are given a graph G and an integer k, and the problem is to find a path of exactly 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 of basic molecules. Under certain experimental conditions one cannot observe these sequences directly but obtain only information about pairs of basic 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 some prescribed length, in the graph of possible neighborhood relationships.

At first glance the k-path problem looks simple. We may start from each 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, merely showing that the problem is in XP.

Surprisingly, finding a k-path is easy if the graph has diameter at least k. (The diameter is the largest distance between any nodes.) Then it suffices to compute shortest paths between all pairs of nodes, which can be done in polynomial time. Since some of these paths has a length at least k, and no nodes appear repeatedly on that path, we get a solution. But ironically, a smaller diameter makes the problem hard.

Is there an FPT algorithm with parameter k? We will give a mainly positive answer, but first only for a simplified k-path problem. This also serves as an example of a general technique: a variant of dynmaic programming being well suited for FPT algorithms.

Here is our simplified k-path problem. In addition to the graph G and integer k, we are also given a function c that assigns a "color" $c(v)$ to every node v. (Do not confuse this with the usual notion of graph coloring where adjacent nodes must get different colors. This restriction does not apply here; the $c(v)$ can be arbitrary. Instead of "colors" we may also speak of "labels".) We have exactly k colors. The *colorful k-path problem* is: Find a path of exactly k nodes that uses every color exactly once.

Such additional constraints can make a problem harder or easier. In our case it will become much easier.

A rather simple FPT algorithm tries all $k!$ orderings (permutations) of the colors. For every fixed ordering of the colors, we keep only the edges that join nodes of neighbored colors. Then we can apply BFS to find a path where the k colors appear in the prescribed ordering. (We skip the details which should be obvious.) This algorithm takes $O(k!n^2)$ time.

The factor $k!$ depends only on the parameter, but it is a fast growing function. To greatly improve this factor we will do **dynamic programming on subsets**, specifically, on all 2^k subsets of the set of colors. We use C as a variable for subsets of colors.

For every set C and every node v we define $p(C, v) := 1$ if there exists a path of $|C|$ nodes that ends in v and contains exactly the colors from C, and $p(C, v) := 0$ otherwise. The trick is that the colors from C may appear in arbitrary ordering on the path, since this information is not needed anymore, for properly extending the path to a complete solution.

It remains to compute all $p(C, v)$. For $|C| = 1$, we obviously have $p(C, v) = 1$ if and only if $C = \{c(v)\}$. Next suppose that we have already computed 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 of this algorithm is only $O(2^k n^2)$; compare it to the previous result.

We stress that this method is not completely new. It is usual dynamic programming, with the only new twist that one argument of the function has a range of exponential size (in the parameter k) rather than polynomial size. Even the backtracinng procedure that reconstructs a solution from the $p(C, v)$ values works as usual, therefore we omit its details.

Finding a Path by Color Coding

So far we have only solved the colorful k-path problem in "FPT time", but not the original k-path problem without colors.

Here, the idea of **color coding** comes in: Take k colors and assign one color $c(v)$ to each node v, at random and independently. Then run the $O(2^k n^2)$-time algorithm for the colorful problem.

This randomized algorithm succeeds if some path with k nodes exists and actually carries all k colors in our random coloring. How likely is this?

Let P be some fixed path of k nodes. It can be colored in k^k different ways, and $k!$ colorings are good for us. 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 an existing 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.

This yields an $O((2e)^k n^2)$-time Monte Carlo algorithm. If P does not exist, all attempts will fail, but we cannot be sure whether there is really no solution or we only had bad luck so far.

In summary, we have not achieved an FPT algorithm but "randomized FPT", so to speak.