Advanced Algorithms Course. Lecture Notes. Part 9

Algorithms for Problems on Special Instances

Dynamic Programming on Trees

Problems that are NP-complete in general graphs can become rather easy in special graph classes. Often it happens in practice that the input to a graph problem is a tree. (For example, many networks are hierarchically structured.) Most problems on trees can be solved by bottom-up dynamic programming. We illustrate the principle by the Weighted Vertex Cover problem which is also equivalent to the Weighted Independent Set problem.

In the given tree we distinguish an arbitrary node r as the root. All edges are oriented away from the root. This defines a directed tree T. For every node, let T_v denote the subtree with root v, consisting of v and all nodes reachable from v via directed edges. We denote the weight of a node v by $w(v)$. For every v we define $OPT(v,1)$ and $OPT(v,0)$ as the weight of a minimum vertex cover in T_v with v and without v, respectively. What we want is the minimum of $OPT(r,1)$ and $OPT(r,0)$.

These values are computed as follows. If v is a leaf, we immediately have $OPT(v,1) = w(v)$ and $OPT(v,0) = 0$. Now let v be an inner node, and $v_1, \ldots, v_d$ the children of v. If v is not in the vertex cover, we have to take all children, hence $OPT(v,0) = \sum_{i=1}^{d} OPT(v_i,1)$. If v is in the vertex cover, we can independently decide for any child to take it or not, and the minimum value is optimal. Hence we have $OPT(v,1) = w(v) + \sum_{i=1}^{d} \min(OPT(v_i,1), OPT(v_i,0))$.

That's all! The running time is $O(n)$, since every node is involved in only constantly many calculations for its parent node. It is recommended to reflect upon the question why our OPT function needed the second (Boolean) argument.

As a side remark, the unweighted Vertex Cover problem can even be solved by a greedy algorithm on trees.

Small Vertex Covers – XP and FPT

The Vertex Cover problem in graphs is NP-complete, but if the graph is already known (or expected) to have some vertex cover with a "small" number k of nodes, we can still solve it exactly and efficiently in practice.

Let n always denote the number of nodes in the given graph. A naive way to find a small vertex cover is to test all subsets of k nodes exhaustively. Elementary combinatorics tells us that this costs $O(kn^{k+1}/k!)$ time: Note that $O(kn)$ time is sufficient to test whether a given set of k nodes is a vertex cover, and the other factor comes from $\binom{n}{k}$. This time bound is feasible only for very small k. The bad thing is that k appears in the exponent of n. It would be much better to have a time bound of the form $O(b^k p(n))$, where b is a constant base, and p some fixed polynomial. (To get a feeling of the tremendous difference, try some concrete figures and compare the naive time bound for Vertex Cover with the bounds we will obtain below.)

A problem with input length n and another input parameter k is said to be in the **complexity class XP** if it can be solved in $O(n^{f(k)})$ time, where f is any computable function. A problem with input length n and another input parameter k is called **fixed-parameter tractable (FPT)** if it can be solved in $O(f(k) \cdot p(n))$ time, where f is any computable function (usually exponential) and p is some polynomial. We may write $O^*(f(k))$ instead of $O(f(k) \cdot p(n))$ if we want to suppress the polynomial factor and stress the more important parameterized part of the complexity.

In the following we show that Vertex Cover is not only an XP problem but an FPT problem. The basic algorithm is: Take an uncovered edge (i, j) and put node i or node j in the solution. Repeat this step recursively *in both branches*, until k nodes are chosen or all edges are covered.

Upon every decision (i or j) we create new branches, hence the whole process has the form of a recursion tree that we call a **bounded search tree**. Since at most k nodes of the graph are allowed in a solution, the tree has depth at most k, thus at most 2^k leaves and $O(2^k)$ nodes. If some leaf represents a vertex cover, we have found a solution, otherwise we know that there is no solution. To bound the time complexity, it remains to check how much time we need to process any node of the search tree: In a simple implementation we may copy the whole graph, delete in one copy all edges incident to i, and delete in one copy all edges incident to j (because these

edges are already covered). The main work is copying. Here we observe that the whole graph can have at most kn edges, otherwise no vertex cover of size k can exist. Hence copying costs $O(kn)$ time, and the overall time is $O(2^k kn) = O^*(2^k)$.

Although this is already much better than naive exhaustive search, further improvements would still be desirable. Here, the more important part is the exponential factor 2^k. Can we improve the base 2 and thus make the algorithm practical for somewhat larger k?

The weakness of the search tree algorithm above is that it considers single edges and selects only one vertex at a time. If we could select more vertices, we could generate our solutions faster. Now observe: For any node i, we have to take i *or all its neighbors*, in order to cover all edges incident to i. It might be good to apply this branching rule on nodes i of high degree. But what if the graph has no high-degree nodes?

If all degrees are at most 2, the graph consists of simple paths and cycles, and the problem is trivial. Thus we can assume (worst case!) that there is always a node of degree 3 or larger. In a branching step we take either 1 node or 3 nodes (or more). How large is our search tree?

This can be analyzed by recurrence equations, similar to the analysis of divide-and-conquer algorithms. Let $T(k)$ be the number of leaves of a search tree for vertex covers of size k. Due to our branching rule we have $T(k) = T(k-1)+T(k-3)$. To figure out what function T is, we assume that it has the form $T(k) = x^k$ with an unknown constant base x. Our recurrence becomes $x^k = x^{k-1} + x^{k-3}$, which simplifies to $x^3 = x^2 + 1$. This equation is called the **characteristic equation** of the recurrence. Numerical evaluation shows $x \approx 1.47$, which is much better than 2. Researchers have invented more tricky branching rules for Vertex Cover and further accelerated the branching process. Meanwhile the best known base is below 1.3.

Anyway, we have shown the time bound $O(1.47^k kn) = O^*(1.47^k)$.