Exact Algorithms
Lecture 9: Structural Parameters, DP, Treewidth

February 19, 2016 Lecturer: Michael Lampis

1 FPT algorithms with structural parameters

Reminder:

Definition 1. *A parameterized problem is a function from* $(\chi, k) \in \{0,1\}^* \times \mathbb{N}$ *to* $\{0,1\}$. *In other words, a parameterized problem is a decision problem, where we are supplied together with each instance a positive integer. This integer is called the parameter.*

Definition 2. *An algorithm solves a parameterized problem in FPT (fixed-parameter tractable) time if it always runs in time at most* $f(k)n^c$, *where* $n = |\chi|$, f *is any function and* c *is a constant that does not depend on* n, k.

So far we have seen two types of algorithms: algorithms that run in c^n time (exponential-time algorithms) and parameterized algorithms where k is the size of the solution. Observe, however, that the definition of FPT does not specify that k has to be the size of the solution: it can be any integer associated with the input instance.

Motivated by this observation, much of parameterized complexity theory is built on parameterized problems where k measures the "distance from triviality" of the input instance. More specifically, a parameterized problem typically consists of:

- An NP-hard problem
- A polynomially solvable class of instances of that problem (the "trivial case").
- A measure of distance k (the parameter) which, for any input χ can tell us how close χ is to the solvable class of instances. Ideally, when $k = 0$ the instance will belong in the class.

In the above setting the goal is to design an algorithm whose complexity deteriorates as smoothly as possible as the distance k increases. Note that the parameterizations we have already seen (k-Vertex Cover, k-Clique) do indeed fall into the above scheme: if k is the target value, all of these problems become trivial for very small values of k, and gradually become harder as k increases. The point of today's lecture is to see that there are much more sophisticated ways to define and measure the complexity of an instance.

2 Base Cases and Structural Parameters

To consider a concrete example, think of a class of graphs where most NP-hard problems are in P: trees.

Theorem 1. *The following problems can be decided in polynomial time on trees: Max Independents Set, Min Vertex Cover, Max Clique, Min Dominating Set, Min Coloring.*

Proof: Easy exercise! □

Recall that a feedback vertex set of a graph $G(V, E)$ is a set $S \subseteq V$ such that $G[V \setminus S]$ contains no cycles. We denote by $\text{fvs}(G)$ the size of the minimum feedback vertex set of G.

Consider the following problem: We are given a graph $G(V, E)$ and a feedback vertex set S of G. We are asked to solve Max Independent Set/Min Dominating Set/Min Coloring on G. The parameter now is $k = |S|$. What is the most efficient algorithm we can come up with?

Theorem 2. *There is a $2^k n^{O(1)}$ algorithm which, given a graph $G(V, E)$ on n vertices and a feedback vertex set S of G with size k computes the minimum vertex cover of G.*

Proof:

Equivalently, we compute the maximum independent set of G. Our first step is to "guess" a subset $S' \subseteq S$ that will be included in the independent set. (By "guess" we mean that the following steps will be repeated for each $S' \subseteq S$ and the best solution among them will be selected). If S' is not an independent set we can reject it, so in the remainder we assume S' is independent.

We delete from the graph $S \setminus S'$, since we have decided these vertices will not be in the maximum independent set. We also delete S' and all vertices connected to S'. The remaining graph has no cycles, since we removed all of S. We can therefore compute in polynomial time its maximum independent set. Therefore, the whole process takes time $2^k n^{O(1)}$.

□

We now have a general setting: let $\mathcal{C}$ be a class of graphs where a problem can be solved efficiently. We define the class $\mathcal{C} + kv$ as the class of graphs which belong to $\mathcal{C}$ *if we delete at most k vertices*.

Example: if $\mathcal{T}$ is the class of all acyclic graphs, then $\mathcal{T} + kv$ is the class of graphs G with $\text{fvs}(G) \leq k$.

Example: if $\mathcal{N}$ is the class of empty (that is, edgeless) graphs, then $\mathcal{N} + kv$ is the class of graphs that have vertex cover at most k.

Informally, the idea we are trying to capture here is the following: we know how to solve a problem well if an input has some properties (e.g. it is a tree/bipartite graph/planar graph/etc.). What if we are given an input that *almost* has this property? We say here that a graph *almost* has this property if we can make the property true by deleting a few (k) vertices. Can we then obtain an efficient algorithm?

As the example above on graphs with $\text{fvs} \leq k$ shows, this can indeed happen sometimes: we know how to solve Max Independent Set on trees, and we can extend this quite well to "almost-trees". Observe that we cannot hope to get a $2^{o(\text{fvs}(G))}$ algorithm, since any graph has $\text{fvs}(G) \leq n$, therefore such an algorithm would violate the ETH.

Because the example above is quite natural, you may be tempted to think that this is what usually happens: if I have a graph that is "almost" an easy graph, I should be able to get an algorithm that is "almost" as good as the one I have for the easy case. This is in fact untrue:

Theorem 3. *Let $\mathcal{B}$ be the class of bipartite graphs. Then 3-Coloring is NP-complete on the class $\mathcal{B} + 3v$.*

We will not look at the proof of the above theorem (it is a reduction from 3-SAT). The important observation here is that 3-Coloring is a natural problem which is *trivial* on bipartite graphs. But, adding a constant number of vertices makes it NP-hard. Not only is it impossible to get a 2^k algorithm in the class $\mathcal{B} + kv$, it is also impossible to get a n^k algorithm (since this would run in polynomial time for $k = 3$).

In the remainder of this lecture we look at parameterized problems with "structural parameters", that is, k will be a value that measures how far the input graph is from being easy. Even though in all cases we will have that the problem in question will be in P for $k = 0$, as we saw this does not automatically imply much for the parameterized problem: its complexity can go from FPT (2^k), to n^k, to 2^n. One of the type of parameters we will consider is the number of vertices that need to be deleted to reach a certain graph class (that is, we will consider several classes of the form $\mathcal{C} + kv$). But, as we will see there are other, more clever ways to measure a graph's structure.

3 Trees, Separators and Dynamic Programming

Consider the Weighted Maximum Independent Set problem: we are given a graph $G(V, E)$ and a weight function $w : V \to \mathbb{N}$. The problem is to find an independnt set whose vertices have the maximum total weight. This problem obviously generalizes Maximum Independent Set, and is therefore NP-hard. However, the obvious greedy approach that we can use to solve it on trees does not work: for example in $K_{1,2}$, normally the best independent set is the set of two leaves, however, it is easy to give appropriate weights so that the max independent set becomes the middle vertex.

Nevertheless, the problem is still in P for trees.

Theorem 4. *There is a polynomial time algorithm which calculates the Maximum Weighted Independent graph of G, if G is a tree.*

Proof:

Let $G(V, E)$ be a tree on n vertices. Pick a vertex r arbitrarily and call it the root. For any other vertices u, v such that $(u, v) \in E$ we will say that u is the parent of v is u is closer to r than v.

For every vertex u we denote by V_u the set of vertices v for which the path that connects v to r goes through u. In other words, the set V_u contains u, its children, its childrens' children etc.

Our algorithm will calculate two values for each vertex: $M_1(u)$ and $M_0(u)$. The value $M_1(u)$ will be the weight of the maximum weight independent set of $G[V_u]$. The value of $M_0(u)$ will be the value of the maximum weight independent set of $G[V_u \setminus \{u\}]$. Observe that if we can calculate all these values in polynomial time, we are done, because then $M_1(r)$ is the value of the optimal solution.

First, observe that it is trivial to compute these values for all leaves: if u is a leaf then $M_1(u) = w(u)$ and $M_0(u) = 0$.

We now show how these values can be computed in polynomial time for a vertex u, if they have been computed for all its children. Let $C(u)$ be the set of children of u. We have $M_0(u) = \sum_{v \in C(u)} M_1(v)$. Furthermore, we have $M_1(u) = \max\{M_0(u), w(u) + \sum_{v \in C(u)} M_0\}$. The former relation is true because the graph $G[V_u \setminus u]$ is a collection of disjoint trees, one for each child of u, so its maximum independent set is just their sum. The second relation is true because the max independent set of $G[V_u]$ either does not include u (so it is equal to $M_0(u)$), or it does, but then it cannot include any child of u.

□

Why does the algorithm of the previous theorem work? The key idea is that in a tree, every vertex u is a **separator**. In order to calculate the best solution in the whole tree, we essentially pick a separator and solve the problem on both parts of the remaining graph. We then combine the solutions into a solution for the whole graph, making sure that the solutions agree on their decision for u.

It is easy now to observe the following:

Theorem 5. *There exists an algorithm which, given a graph G and a feedback vertex set S of G with size $|S| \leq k$, computes the maximum weight independent set of G in time $2^k n^{O(1)}$.*

4 Balanced Separators

As we saw in the previous section, separators are very important algorithmically because they allow us to break down a graph into smaller parts and then solve the problem using Dynamic Programming. Trees are a special case of this, because all vertices are separators. However, trees actually have a much stronger property: they always contain a **balanced separator**.

Definition 3. *Let $G(V, E)$ be a graph. We say that a set $S \subseteq V$ is a balanced separator of size k if $|S| = k$ and in $G[V \setminus S]$ all connected components have size at most $\frac{2}{3}|V|$.*

Theorem 6. *Let $G(V, E)$ be a tree on n vertices. Then, there exists $u \in V$ such that all components of $G[V \setminus \{u\}]$ have at most $n/2$ vertices.*

Proof:

Take an edge $(u, v) \in E$. We will say that this edge "points" to v if the components that contains v in the graph $G(V, E \setminus \{(u, v)\})$ has at least as many vertices as the one containing u. If an edge (u, v) is pointing to v, then the component that contains u if we delete this edge has at most $n/2$ vertices.

If there is an edge (u, v) that points to both u and v then both u and v are balanced separators. Furthermore, each edge must be pointing to one of its endpoints. So, assume each edge is pointing to exactly one of its endpoints, and therefore consider G as a directed graph without cycles.

We now claim two facts: first, there exists a vertex u such that all its incident edges point to u. Second, therefore u is a balanced separator. For the first, observe that if this were not true, the

graph would contain a directed cycle: start at any vertex and start moving by following edges that point away from the current vertex. This process must eventually return to a visited vertex (since the graph is finite), detecting a cycle. But G contains no cycles, contradiction! For the second, this follows from the fact that if we delete u we also delete all its incident edges.

□

Balanced separators are even more useful than separators, because they allow us to *quickly* break down a graph into smaller pieces. The added value is somewhat hard to see on trees, because most problems are already easy on these graphs. A good example of the difference that balanced separators make is given by planar graphs.

Theorem 7. *Every planar graph on n vertices has a balanced separator S of size at most $4\sqrt{n}$. Furthermore, such a separator can be found in polynomial time.*

Proof:

The proof is based on two claims:

1. If a planar graph has a spanning tree with diameter r, then it has a balanced separator of size at most $r + 1$.

2. If a BFS tree of a planar graph has diameter more than $\sqrt{n}$ then we can transform G into a graph with a spanning tree of diameter at most $\sqrt{n}$ by removing at most $\sqrt{n}$ vertices.

Claim 1: The hard part is the first claim. Take a plane drawing of G, and suppose that we have a spanning tree T of G such that the spanning tree has diameter r. We first begin by noting that we may assume that G is triangulated: if some face is not a triangle we add an edge to it while preserving planarity. This does not change T, and it can only make finding a separator harder.

Consider now an edge $(x, y) \in E \setminus T$. Adding this edge to T creates a cycle. Let C_{xy} be the set of edges of this cycle. The crucial observation is that in a planar graph any cycle is a separator. Furthermore, the number of vertices of C_{xy} is at most $r+1$, since the path connecting x, y in T has length at most r. So, what remains is to select the edge (x, y) in such a way that C_{xy} is a balanced separator.

For each non-tree edge $(x, y) \in E \setminus T$ we do the following: we check if the cycle C_{xy} obtained by adding it to T is a balanced separator (this can be done in polynomial time). If such an edge is found we are done. If not, we will make some analysis that leads to a contradiction, proving that such an edge must exist, and thus it will be found.

So, suppose that no edge $(x, y) \in E \setminus T$ produces a balanced separator C_{xy}. Among all the possible "bad" separators C_{xy} we select the one that is most balanced, that is, the one that has fewer vertices in the side with larger size. If several separators are equally good in this respect, we select the one that has the fewest faces in the graph induced by C_{xy} and the larger side. We will denote the larger connected component of $G \setminus C_{xy}$ by A.

The reason C_{xy} is a separator is that a cycle is a closed curve on the plane, that separates it into two regions ("inside" and "outside"). Let A be the "inside" vertices, without loss of generality.

In the plane drawing of G, (x, y) is part of some face with some vertices of A ("inside"). Since all faces are triangles, there exists $z \in A$ such that $(x, z), (y, z) \in E$. We will now examine some cases about the edges $(x, z), (y, z)$.

1. If $(x,z),(y,z) \in C_{xy}$ we already have a contradiction, because this means that the cycle only has length three and contains no vertices!

2. If $(x,z) \in C_{xy}$ and $(y,z) \notin C_{xy}$ then we have $(y,z) \notin T$ (otherwise we would have the previous case). So, C_{yz} is another separator. This separator contains the same number of vertices inside, but has fewer faces, which is a contradiction to the choice of (x,y). The case $(x,z) \notin C_{xy}$ and $(y,z) \in C_{xy}$ is symmetric.

We can therefore assumet that $(x,z),(y,z) \notin C_{xy}$. Let us consider some further cases:

1. If $(x,z),(y,z) \in T$ then T contains a cycle! A contradiction.

2. If $(x,z) \in T$ and $(y,z) \notin T$, we consider again C_{yz}. This is the same cycle as C_{xy}, except it also contains the edge (x,z) and it has fewer edges.

So, we conclude that $(x,z),(y,z) \notin T$. Let us now consider the two separators C_{xz}, C_{yz}. Observe that these partition A into two parts, say A_1, A_2. Assume, without loss of generality that $|A_1| \geq |A_2|$. Therefore, $|A_1| \geq n/3$. If $|A_1| \leq 2n/3$ then C_{xz} is a balanced separator, contradiction! But if $|A_1| > 2n/3$ then it is a bad separator, that is more balanced than C_{xy}, again contradiction!

Claim 2: Given now an arbitrary planar graph G, take a BFS tree T of G. If the tree has diameter at most $2\sqrt{n}$ we are done. Suppose then that it has larger diameter. Let r be an arbitrary root of T, and let L_i be the set of vertices at distance exactly i from r in T (and also in G). We observe that, because of the structure of the BFS tree, each L_i is a separator.

We will say that L_i is fat if $|L_i| > \sqrt{n}$, otherwise we will say it's thin. Let L_m be the median layer of the tree T, that is, m is the minimum number such that $\sum_{i=0}^{m} |L_i| \geq n/2$. Clearly, if L_m is thin, it is a balanced separator and we are done. So, suppose that it is fat.

Let i_1 be the maximum index such that $i_1 < m$ and L_{i_1} is thin. If such an i_1 exists, we select L_{i_1} in our separator. Similarly, let i_2 be the smallest index such that $i_2 > m$ and L_{i_2} is thin. If i_2 exists, we add L_{i_2} to our separator.

We have now separated the graph into three parts: the vertices "before" i_1, the vertices "after" i_2, and the vertices in between. The vertices before and after are each $< n/2$. So if the vertices in between are $< 2n/3$ we are done.

Suppose that there are $> 2n/3$ vertices between layers i_1 and i_2. We observe that $i_2 - i_1 < \sqrt{n}$, because all layers between these two are fat. So, the graph induced by these middle layers has at most $\sqrt{n}$ levels. Now, by using Claim 1 we can separate this graph into balanced components by deleting at most $2\sqrt{n}$ vertices.

□

Theorem 7 has some important and direct algorithmic implications.

Theorem 8. *There is a $2^{O(\sqrt{n})}$ algorithm which computes the maximum independent set of planar graphs on n vertices.*

Proof:

We are given a graph $G(V, E)$. The algorithm is the following: find a balanced separator S of the graph, by Theorem 7. S partitions the graph into $V = S \cup A \cup B$ where A, B are disjoint set of vertices, there are no edges from A to B and $|A|, |B| \leq 2n/3$. We also have $|S| \leq c_1\sqrt{n}$ for some constant c_1. We now "guess" the set of vertices of S that will be selected in the maximum independent set. (In other words, we will repeat the following $2^{|S|}$ times, and select the best outcome). We delete from the graph the vertices of S not selected, and we also delete the selected vertices, along with their neighbors. The remaining graph consists of two sets of vertices $A' \subseteq A$ and $B' \subseteq B$. We then calculate the maximum independent set on $G[A']$, $G[B']$ with the same algorithm. Our solution is the union of these two sets, together with our guess for a subset of S.

The running time of the algorithm on a graph on n vertices is at most $T(n) \leq 2^{|S|} \cdot 2 \cdot T(2n/3) \leq 2^{c_1\sqrt{n}} T(2n/3)$. Consider the infinite sum $\sum_{i=0}^{\infty} (\sqrt{\frac{2}{3}})^i$. This sum converges, and it is upper-bounded by some constant c_2. We now have,

$$\begin{aligned} T(n) &\leq 2^{c_1\sqrt{n}} T(2n/3) \\ &\leq 2^{c_1\sqrt{n}} 2^{c_1\sqrt{2n/3}} T(4n/9) \\ &\leq 2^{c_1\sqrt{n}(1+\sqrt{2/3}+\sqrt{4/9}+\ldots)} \\ &\leq 2^{c_1 c_2 \sqrt{n}} \end{aligned}$$

□

The above exemplifies a typical situation that holds for many graph problems: when a problem need time c^n to be solved exactly in general graphs, it can be solved in time $c^{\sqrt{n}}$ in planar graphs, and this is "optimal" under the ETH. The reason is almost always the existence of balanced separators on planar graphs.

If $\mathcal{P}$ is the class of planar graphs, we immediately get the following:

Theorem 9. *There is an algorithm solving maximum weighted independent set in time $2^k 2^{O(\sqrt{n})}$ on the class $\mathcal{P} + kv$.*

5 Almost-trees and Separators

Let us now consider another definition of graphs which are almost trees, which is more general than feedback vertex set.

Definition 4. *We say that a graph $G(V, E)$ has treewidth k if and only if there exists a super-graph $G'(V, E')$ with $E' \supseteq E$ such that G' is chordal and $\omega(G') \leq k + 1$.*

The intuition behind this definition is that (as we have seen in a previous course) chordal graphs have a **tree structure**. In particular, a chordal graph with small clique size, looks like a tree, because in a chordal graph all minimal separators are cliques (and therefore, if all cliques are small, all minimal separators are small).

Another equivalent definition of treewidth can be given by the cops-and-robber game we have also seen in a previous course. Recall that, in that game, there are k cops who are trying to catch a robber hiding on a vertex of a graph. The cops have helicopters and can fly from one vertex to the other, but at any point the robber can see them and can run through any path of the graph that does not go through a stationary cop. The question is how many cops we need to catch the robber.

Theorem 10. *A graph has treewidth k if and only if $k+1$ cops can catch the robber on the graph.*

One direction is easy: if G has treewidth k, then it has a chordal super-graph with clique-size $k+1$. As we have seen, on chordal graphs the number of cops needed to win is exactly $\omega(G)$. The other direction is not as obvious, and we skip the proof here.

Let us think a little bit whether the definition of treewidth we have given agrees with the notion of "tree-like":

Example: Trees have treewidth 1. Indeed, trees are chordal, with clique size 2.

Example: Cliques have treewidth $n-1$. Indeed, cliques are chordal, with clique size n.

Example: $K_{n,m}$ has treewidth at most $\min(n,m)$. Observe that $\min(n,m)+1$ cops can catch the robber by stationing all but one cops on one side and using the remaining cop to clear the other.

Example: $n \times m$ grids have treewidth at most $\min(n,m)$. Suppose $n \leq m$ is the number of rows. We sketch a strategy for $n+1$ cops. First, place n cops on the first column. Then place the extra cop on the top vertex of the second column. This frees the cop on the top vertex of the first column. Place him on the second vertex of the second column. Etc.

Let us now see an equivalent definition of treewidth which is usually used to design algorithms on graphs of treewidth k.

Definition 5. *For a graph $G(V,E)$ a tree decomposition is a tree $T(X,I)$ such that $X \subseteq 2^V$ and we have the following:*

1. *For all $v \in V$ there exists $x \in X$ such that $v \in X$.*
2. *For all $(u,v) \in E$, there exists $x \in X$ such that $u,v \in x$.*
3. *For all $v \in V$ the set $\{x \in X \mid v \in x\}$ induces a connected sub-tree of T.*

We say that the width of a tree decomposition is $\max_{x \in X}(|x|-1)$. We say that the treewidth of a graph is the minimum width of any tree decomposition of the graph.

The above definition may seem a little strange at first. However, it follows exactly the tree structure of chordal graphs. Indeed, suppose that we have a chordal graph G. We can make a tree decomposition of G as follows:

Theorem 11. *Let $G(V,E)$ be a chordal graph. Then there exists a tree decomposition of G with width $\omega(G)-1$.*

Proof:

We proceed by induction proving a slightly stronger statement: if G is chordal there exists a tree decomposition such that every maximal clique is contained in some $x \in X$ and $\max_{x \in X}|x| = \omega(G)+1$.

For a single vertex graph it is trivial. Suppose that we have proved the statement for all chordal graphs with at most n vertices. Take a graph G with $n+1$ vertices and find a simplicial vertex u. Consider the graph $G-u$. By the inductive hypothesis, we have a tree decomposition $T(X,I)$ of this graph with the width $\omega(G-u)$. Furthermore, there is an $x \in X$ such that $N(u) \subseteq x$, because $N(u)$ is a clique, and all maximal cliques of $G-u$ are contained in some x. We now make a tree decomposition of G by adding to T a new x' that is connected to x and such that $x' = N(u) \cup \{u\}$. Observe that all maximal cliques of G are contained in some node of the new tree. Also, the width of the decomposition either stayed the same, or increased in the case where $N[u]$ is the maximum clique of G.

□

Thus, if a graph has treewidth k, it also has a tree decomposition of width k. What about the other direction?

Theorem 12. *If a graph has a tree decomposition of width k then it has treewidth k.*

Proof:

Take the vertices of the graph G and for any u, v such that there exists an $x \in X$ in the decomposition with $u, v \in x$ add the edge (u, v). We thus have a supergraph of G. It is left as an exercise to show that the supergraph is chordal and that it has the appropriate clique size.

□

Given a graph G, how do we know if it has small treewidth? In general, we don't! Just like fvs, computing the smallest k such that G has a decomposition of width k is NP-hard. However, as we will see, if we manage to obtain such a decomposition then it can significantly help us solve some NP-hard problems.

Theorem 13. *There exists an algrorithm which, given a weighted graph $G(V,E)$ and a tree decomposition $T(X,I)$ of G with width k, computes the maximum weighted independent set of G in time $O(2^k n^{O(1)}$.*

Proof:

The idea of the algorithm is a generalization of the dynamic programming algorithm that we used to solve this problem in polynomial time on trees. The crucial observation is that, because of the connectivity property of the decomposition, every $x \in X$ that is not a leaf is a **separator**, just like every internal vertex of a tree is a separator. This means that if we know the correct decision for each of the vertices of x, we can decompose the problem into independent sub-problems.

More precisely, let r be a root of T. For each $x \in X$ we define T_x as the sub-tree of T that contains x and all its descendants (all vertices whose path to r goes through x). We define G_x as the subgraph of G induced by the set $\cup_{y \in T_x} y$, that is, all vertices that appear in the decomposition in x or below.

For every subset $S \subseteq x$ we store a value: the weight of the maximum independent set of G_x that contains all vertices of S and no vertices of $x \setminus S$. Note that no such set may exist (if S already induces some edges). In that case, we set the value corresponding to S to $-\infty$. Thus, for every node $x \in X$ we calculate a table of values $D_x : 2^x \to \mathbb{N}$.

Observe that if we can calculate all these tables we can solve the problem by looking for the largest value of D_r. We now proceed recursively, starting from the leaves of T. If $x \in X$ is a leaf, for each $S \subseteq x$ that is independent we set $D_x(S) = \sum_{v \in S} w(v)$.

Now, let $x \in X$ be an internal node and let $x_1, x_2, \ldots, x_c$ be its children. For each $S \subseteq x$ we set $D_x(S) = \sum_{v \in S} w(v) + \sum_{i=1}^{c} \left(D_{x_i}(S \cap x_i) - \sum_{v \in S \cap x_i} w(v) \right)$. Let us explain this: the maximum independent set in G_x that takes the vertices of S has weight that includes the vertices of S (this is the first sum), plus for each component we obtain if we remove x from the graph the maximum independent set of this component. Because the G_{x_i} graphs are disconnected if we remove x, the result is a valid independent set.

□