Exact Algorithms
Lecture 8: Structural Parameters

16 February 2021 Lecturer: Michael Lampis

Summary: This lecture covers the following:

- The concept of structural parameters.
- Structurally parameterized algorithms and lower bounds for Coloring.

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 computable 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 todays 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 Independent Set, Min Vertex Cover, Max Clique, Min Dominating Set, Min Coloring.*

Proof: Easy exercise! $\square$

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 $fvs(G)$ 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 $S_0 \subseteq S$ that will be included in the independent set. (By "guess" we mean that the following steps will be repeated for each $S_0 \subseteq S$ and the best solution among them will be selected). If S_0 is not an independent set we can reject it, so in the remainder we assume S_0 is independent. We delete from the graph $S \setminus S_0$, since we have decided these vertices will not be in the maximum independent set. We also delete S_0 and all vertices connected to S_0. 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 $2^k n^{O(1)}$. $\square$

We now have a general setting: let C be a class of graphs where a problem can be solved efficiently. We define the class $C + kv$ as the class of graphs which belong to C if we delete at most k vertices. Example: if T is the class of all acyclic graphs, then $T + kv$ is the class of graphs G with $fvs(G) \leq k$. Example: if N is the class of empty (that is, edgeless) graphs, then $N + kv$ 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 $fvs \leq k$ 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 $2^{o(fvs(G))}$ algorithm, since any graph has $fvs(G) \leq n$, 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 B be the class of bipartite graphs. Then 3-Coloring is NP-complete on the class $B + 3v$.*

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 $B + kv$, 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 $C + kv$). But, as we will see there are other, more clever ways to measure a graphs structure.

3 Coloring parameterized by fvs and vertex cover

Consider the following: we are given a graph G and a set $S \subseteq V(G)$ such that $G - S$ is acyclic and $|S| = k$. Is this type of information useful for deciding, say, whether G is 3-colorable? The answer is yes, and indeed we can solve this problem in 3^k time (more or less). In this section we review how this can be done, and why it is impossible to do better under the SETH.

Before we proceed, let us consider an intermediate problem:

Definition 3. *In List Coloring we are given a graph $G = (V, E)$ and for each $v \in V$ we are given a list $L(v) \subseteq \mathbb{N}$. We are asked if there exists a proper coloring of G (which gives distinct colors to the endpoints of all edges) which also satisfies $\forall v\ c(v) \in L(v)$, that is, for each vertex we select a color from its list.*

Observe that List Coloring is a harder problem than normal Coloring. Indeed, G is 3-colorable if and only if we can find a list coloring of G if all the lists are equal to $\{1, 2, 3\}$. This shows that regular coloring is only a special case of list coloring.

Before we present the basic algorithm, let us however point out that list coloring remains tractable on trees.

Theorem 4. *List Coloring can be solved in polynomial time on trees.*

Proof: The algorithm is simple: if there exists a $v \in V$ such that $|L(v)| = 1$, then we don't really have a choice for v: we assign to it the only possible color. We then update the lists of all neighbors of v, removing the color we used for v (as it can no longer be used on these vertices). We repeat in this way until one of the following happens: (i) a vertex has an empty list, in which case we return NO (ii) all vertices have a list of size at least 2.

We now claim that in a forest, if all lists have size at least 2 then the answer is always YES. Indeed, if a component is a single vertex we can of course always color it. Otherwise, we select a leaf of a component and (by induction) list-color the rest of the component. The coloring can always be extended to the leaf, as it has 1 colored neighbor but a list of size 2. □

Now the algorithm for coloring parameterized by fvs is not too hard to obtain:

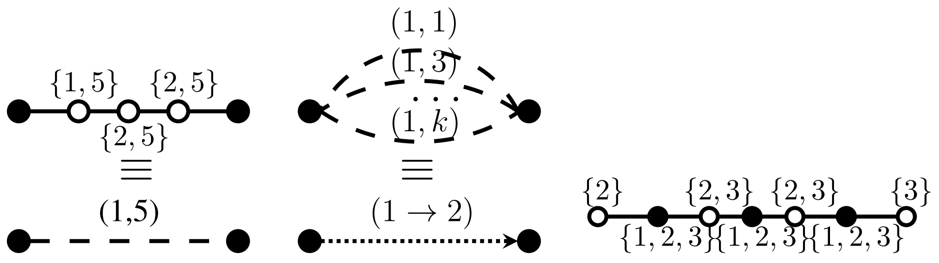

Figure 1: Basic gadgets, where empty vertices are internal and solid vertices are endpoints that will be connected to the rest of the graph. On the left, a weak edge that forbids the combination $(1, 5)$ on its endpoints. In the middle, an implication that forces color 2 on the right if color 1 is used on the left. On the right an OR gadget: one of the solid vertices must take color 1.

Theorem 5. *Given a graph G and a feedback vertex set of size k, for all $c \geq 3$, it is possible to decide if G is c-colorable in time $c^k n^{O(1)}$.*

Proof: Guess a coloring of S (there are c^k choices). For each such coloring we give all vertices of $V \setminus S$ the list $\{1, \ldots, c\}$, and then update lists according to the colors used by the neighbors of each vertex in S. We then use the algorithm for List Coloring on trees to check if the coloring of S can be extended to all of G. □

Interestingly, even though the algorithm above is not too hard, it looks like it is optimal!

Theorem 6. *Given a graph G and a feedback vertex set of size k, for all $c \geq 3$, if it is possible to decide if G is c-colorable in time $(c - \epsilon)^k n^{O(1)}$, then the SETH is false.*

Proof:

We will prove the theorem for $c = 4$ only, but the proof is similar for other cases (though a bit more complicated when the number of colors is not a power of 2). We perform a reduction from SAT. Given a SAT instance with n variables, we will construct a graph G with $fvs = n/2 + O(1)$ and G will be 4-colorable if and only if ϕ is satisfiable. Hence, if there is a way to decide this in $(4 - \epsilon)^k = (2 - \epsilon')^n$, we would disprove the SETH.

We will construct an instance of List Coloring where all lists are subsets of $\{1, 2, 3, 4\}$. This can then be reduced to 4-coloring as follows: we introduce four "palette" vertices p_1, p_2, p_3, p_4 connected in a clique; for each $v \in V$, for each $i \notin L(v)$, we connect v to p_i. It is not hard to see that a List Coloring of the old graph is a 4-coloring of the new graph and vice-versa.

For our construction we will need the gadgets shown in Figure 1 (where they are described for a number of colors c that could be more than 4). The idea of the first gadget is that it rules out only one combination of colors in its endpoints (this particular example rules out the combination $(1, 5)$) but has no effect if the endpoints are colored in any other way. Using this we can construct an implication gadget (if v has color 1 then u must have color 2), as well as an OR gadget (last gadget of the figure) which ensures that at least one of the vertices of the path received color 1.

Armed with these gadgets we do the following: we construct n vertices $y_1, \ldots, y_{n/2}$; each of these vertices represents two variables of ϕ, in the sense that the color of y_i tells us the assignment to the variables x_{2i-1}, x_{2i}. Note that there are 4 colors available for y_i and four assignments for the pair x_{2i-1}, x_{2i}, so we have a one-to-one correspondence.

Finally, for each clause of ϕ we construct an OR gadget, where the vertices that may take 1 represent literals of the clause (meaning: at least one literal must be set to True). We add appropriate incompatibility gadgets to make sure that the literal we set to True "agrees" with the coloring of the y_i vertices. $\square$

Interestingly, even though we used List Coloring as a stepping stone, List Coloring is a truly harder problem.

Theorem 7. *List Coloring is W[1]-hard parameterized by the vertex cover of the input graph, even on split graphs.*

Proof: Reduction from k-Independent Set. Given graph $G = (V, E)$ we construct a split graph $G' = (A, B, E')$ where A is a clique and has size k and the vertices of A have lists $\{1, \ldots, n\}$, where $n = |V|$. The idea is that the coloring of the clique represents the k vertices of the independent set (which must be distinct since A is a clique). Now, for each $uv \in E$ we need to make sure that it's not possible to use colors u and v in A. So, for each pair of vertices of A, we give them a common neighbor whose list is $\{u, v\}$. This completes the construction. $\square$