6.890: Algorithmic Lower Bounds: Fun With Hardness Proofs Fall 2014

Lecture 13 — October 16, 2014

Prof. Erik Demaine *Scribe: Fermi Ma and Aahlad Gogineni*

1 Overview

Today we begin a new topic, parameterized complexity. This area was born around 1999. (See Parametrized Complexity by Downey, Fellows, Niedermeier and Rossmanith [DFNR98]) The general idea is to measure the complexity as a function of the problem size n and an additional parameter k.

2 General Idea

A parameter k is a function from the set of instances of a problem to $\mathbb{N}$.

As an example, in k-Vertex Cover, the question is: Is there a vertex cover of size $\leq k$.

In this problem k is a natural parameter because the decision problem asks if $OPT \leq k$ for different values of k. In addition, k is an input into the problem, so the parameter function just takes k and throws away the rest of the graph.

However, there are other examples where the parameter function can be very complicated or even computationally hard to determine.

Our goal in general is to get a very good dependence on n at the cost of a (possibly) bad dependence on k. In other words we try to achieve a polynomial dependence on the problem size n at the cost of an exponential dependence on the parameter k. (see: FPT) We now define:

$$XP = \{\text{parameterized problems solvable in } n^{f(k)}\}$$

This is considered a bad running time. However, we can define a class of problems with better running time:

$$FPT = \{\text{parameterized problems solvable in } f(k)n^{O(1)}\}$$

The reason why XP is not considered great is that the polynomial changes as k changes, whereas the polynomial *does not change* as k increases in FPT. FPT stands for fixed parameter tractable (when you fix the parameter k, you get a tractable problem). Note that problems in XP are also "tractable" when k is fixed, but the difference is that changing k greatly affects exactly how tractable the problem is.

FPT can also be defined as the set of parameterized problems solvable in $f(k) + n^{O(1)}$ time.

2.1 Example: k-Vertex Cover

Instance: Graph $G = (V, E)$, positive integer $k \leq |V|$. Question: Is there a vertex cover of size k for G?

This problem is in XP, because we can check each of the n^k vertex sets of size k in linear time giving us an $O(n^{k+1})$ running time.

We can show that this problem is in FPT by showing a run time of $2^k \cdot n$. The algorithm works as follows: We start with an edge, and include one of the vertices on one end of this edge. Then we delete all the edges coming out of that vertex, and then we repeat. We do this k times, and we check if we have a vertex cover. If not, we go back and try a different one. We note that the decision tree has depth k and branching factor 2, so we end up checking 2^k different things. Each check takes linear $O(n)$ time, so we get $2^k n$ time.

2.2 EPTAS

We define the class $EPTAS$ (Efficent $PTAS$) $\subseteq PTAS$ as the class of problems admitting polynomial time approximation schemes with running time $f(1/\epsilon) * n^{O(1)}$. Any problem in $EPTAS$ is in FPT w.r.t the parameter $1/\epsilon$. In addition the existence of an $EPTAS$ implies we're in FPT with respect to the natural parameter k.

Proof. Let the output of some $EPTAS$ for a problem w.r.t $\epsilon = 1/(2 * k)$ be L. The "length" (function being optimized) l of L is within a factor of $1 + \epsilon$ of the optimal (in this case, shortest) "length" (OPT) i.e. $l \leq (1+\epsilon)OPT$. Therefore $OPT \leq k \iff OPT(1+1/(2k)) \leq k(1+1/(2k)) = k + 1/2 \iff \lfloor OPT(1 + 1/(2k)) \rfloor \leq k \iff \lfloor l \rfloor \leq k$. □

$\therefore$ If a problem is not in FPT that implies it is not in $EPTAS$.

3 Parameterized Reductions

As usual, we'll show problems are hard via reductions. In general, we have a decision problem A with parameter k denoted (A, k), and we want to convert this into a decision problem B with parameter k', denoted (B, k'). This is going to look similar to Karp-style reductions, but with a few tweaks.

The reduction takes an instance x of A and applies a function f and gives an instance $x' = f(x)$ of B. It must satisfy the following requirements:

- The function f is computable in polynomial time. In particular x' has polynomial size w.r.t x.
- The reduction needs to be answer preserving, so x is a yes instance for A if and only if x' is a yes instance for B.
- The reduction needs to be parameter preserving So $k'(x') \leq g(k(x))$ for some $g : \mathbb{N} \to \mathbb{N}$. g needs to be a computable function. (g will be discussed further in the next lecture)

Then if $B \in FPT$, then $A \in FPT$. This means that we will show hardness (not being in FPT) of a problem B by reducing from a hard problem A that is known to not be in FPT.

3.1 Examples

Independent Set to Vertex Cover:

$f : (G, k) \to (G, n - k)$

Here $k' = n - k$. This is not a valid parameter preserving reduction since there is no function $g : \mathbb{N} \to \mathbb{N}$ such that $n - k \leq g(k) \forall n \in \mathbb{N}$.

Independent Set to Clique:

$f : (G, k) \to (\overline{G}, k)$

Here $k' = k$. This is a parameter preserving reduction since it can be shown that there exists a function of the parameter k in Independent set such that the parameter k' in Clique is bounded by it for all possible instances of Independent Set.
Note: The reduction from Independent Set to Clique is almost exactly the same.

Note: Vertex Cover is in FPT. Independent Set and Clique are not in FPT if the Exponential time Hypothesis is true.

3.2 k-step Nondeterministic Turing Machines (NTM)

A Turing machine is an infinite tape with a head. In this problem, we are given a Turing machine with $O(n)$ internal states, $O(n)$ internal lines of code, and an alphabet of size $O(n)$ (meaning that it can write any one of $O(n)$ different letters on each cell of the tape). We also assume that the Turing machine has $O(n)$ choices that are made nondeterministically, and that the tape is initially blank.

The question is: Does there exist a YES path of length k.

We expect that the best algorithm is $O(n^k)$, since we want to pick among $O(n)$ different options k times (on a deterministic machine).

For now, we will use this problem to define the class $W[1]$. A problem is in $W[1]$ if it can be reduced to a k-step Non-deterministic Turing machine, and a problem is $W[1]$-hard if it is at least as hard as this k-step NTM problem. (i.e. k-step nondeterministic Turing machine can be reduced to it). So this problem is our first $W[1]$-complete problem. (analogous to 3-SAT w.r.t NP)

3.3 Independent Set is $W[1]$-complete

We are going to give a reduction of this k-step NTM to Independent Set. Given a k-step NTM, we want to get an instance of Independent Set so that if there is an independent set of the right size, then there's a YES path of length $\leq k$.

The reduction is as follows: k-Step NDTM $\to$ Independent Set. Here $k' = k^2$. To set this up, we create a graph G with k^2 cliques. The clique (i, j) corresponds to memory cell i at time j. The vertices in each clique correspond to combinations of state of the Turing machine and the symbol on tape at the location and time specified by the clique. Therefore the size of the clique is $O(n^2)$ Since $k' = k^2$ any possible solution must have 1 vertex per clique. (Since there can be no more than 1 from any one clique there must be at least 1 from each clique.)

After we construct the cliques, we draw an edge between two vertices if and only if the two vertices correspond to two state, symbol, time, cell tuples that are mutually exclusive.

Then if we can solve the Independent Set problem, we can solve the Turing machine problem. So Independent Set is $W[1]$-hard.

It turns out that Independent Set is actually $W[1]$-complete. To show that, we give a reduction from Independent Set to k'-step nondeterministic Turing Machine (NTM) (here k' is going to be $O(k^2)$). To do this, we guess k vertices. For each pair of the vertices, we check that there is no edge between them. The k vertices are written on k cells of the tape.

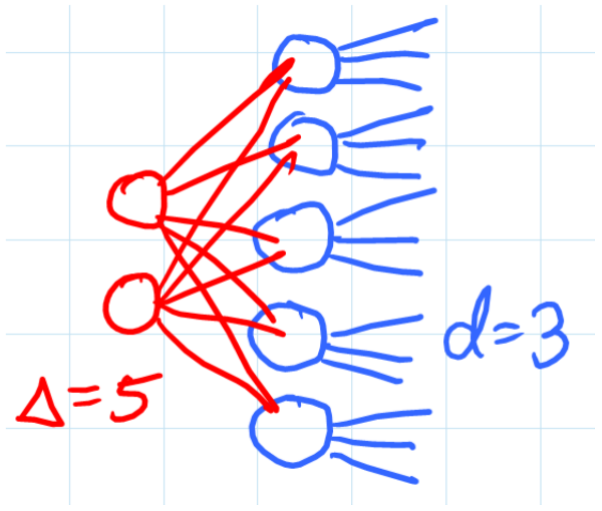
We store the entire graph inside the machine in $O(n)$ lines of code. We use the code as a lookup table in order to perform the checks. Note that $n' = O(|V| + |E|)$. We did not get into the details of this proof in class, so this is a high-level overview.

Note: By the earlier reduction from Clique to Independent set, Clique is also W[1]-Complete.

3.4 Clique in Regular Graphs

We demonstrate that Clique is W[1]-Hard when restricted to regular graphs. Let Δ be the maximum degree in some graph G. We are going to transform G into a Δ-regular graph. Recall that a regular graph is one where every node has the same degree. To do this, we make Δ copies of the original graph. Each vertex v goes to vertices $v_1, v_2, \ldots, v_\Delta$. In addition for each vertex v in the original graph of degree d construct $\Delta - d$ new vertices $v'_1, v'_2, \ldots, v'_{\Delta-d}$ and connect each of them to all of the $v_1, v_2, \ldots, v_\Delta$. Therefore each v'_i has degree Δ and each v_i has degree d (from the original graph) + $\Delta - d$ (from the edges to each v'_i) = Δ. We argue that this transformation preserves clique size and therefore $k' = k$.

In the case where $\Delta = 5$ and $d = 3$, the construction follows is illustrated in the image below.



Note: It follows that Δ-regular k-Independent Set is W[1]-Hard since there is a simple reduction from Clique to Independent Set (analogous to the reduction in the other direction done in 3.1).

3.5 Partial Vertex Cover:

Do there exist k vertices that cover l edges? We do a reduction from Δ-regular k-Independent Set. To do the reduction, we give the graph to Partial Vertex Cover with $l' = \Delta \cdot k$. The idea is that if we can choose k independent set vertices, each one covers Δ edges, so we cover $\Delta \cdot k$ vertices.

If we parameterize by k the problem is $W[1]$-complete, and if we parameterize by l, the problem is FPT. So we need to be careful about which parameter we are looking at.

3.6 Multicolored Clique

This is another version of clique that is $W[1]$-hard. This is an analogue of 3-Dimensional matching. The set of vertices $V = V_1 \cup \cdots \cup V_k$. The question is does there exist a $k - clique$ with 1 vertex per V_i. In other words, given a graph and a k-coloring of it, does there exist a k-clique in it.

Reduction from clique:

For each vertex v in the original graph, make k copies $v_1, v_2, \ldots, v_k$. (corresponding to the k colors) For each edge (v, w) we make an edge $(v_i, w_j) \forall i \neq j$. (creating a k-coloring) This is essentially the obvious reduction to do once we have k copies of a vertex. So if there exists a k-clique with 1 vertex per V_i, this corresponds to a clique of the same size in the original graph.

Reduction to clique: The problem instance is merely a restriction of clique in which any clique in the graph must be multicolored.

Therefore it is $W[1]$-Complete.

Note: Similarly, Multicolored Independent Set is $W[1]$-complete. (See reductions (both directions) from Independent Set to Clique in 3.1)

(These results were shown in papers by Pietrzak [Pie03] and Fellows, Hermelin, Rosamond, Vialette [BDFH09])

3.7 Shortest Common Supersequence

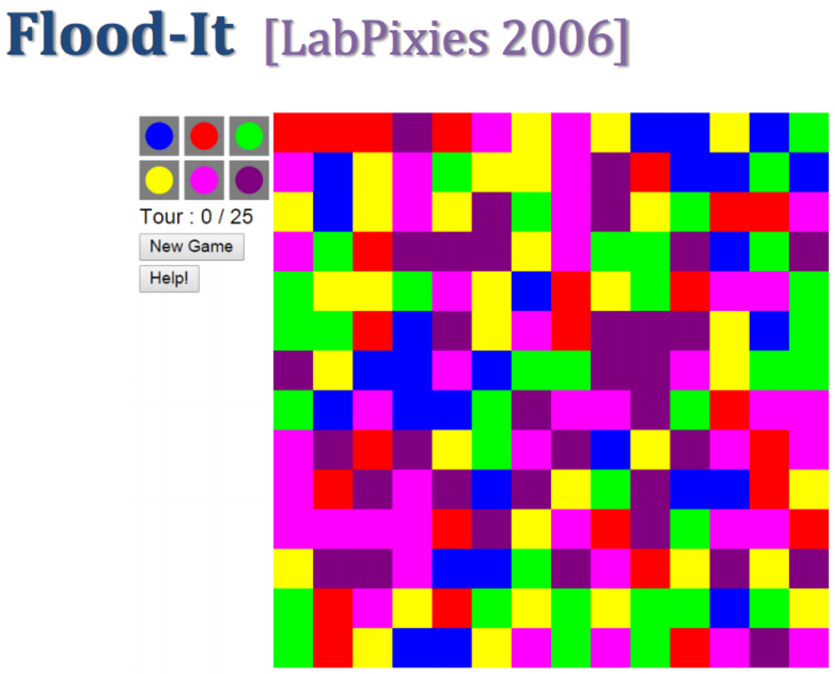
Given k strings, alphabet Σ. Given l. We want to find a string of length l that is a supersequence of all the input strings. This problem is motivated by applications to DNA analysis.

So maybe we are given a case where $k = 2$ where $ACGGCT$ is one string and another string $CAGGAT$, so the supersequence is $ACAGGACT$.

Pietrzak showed that this problem is $W[1]$-hard for $\Sigma = 2$ in [Pie03].

This problem can be used to show that some variants of Flood-It are hard.

3.8 Flood-It



In Flood-It, you control the color of the upper left-hand corner square. When you change its color, you change the color of all the nodes of the same current color connected to it and recursively all the nodes of the same current color connected to them.

Instead of the square grid graph from the game, we consider the version of Flood-It on trees. We take the shortest common subsequence problem, do a substitution where for each sequence each letter a is mapped to aS where S is a letter not present in any of the original sequences (this is to prevent multiple letters from being "flooded" by one color change) and then each letter of each sequence is mapped to a node. Each node is connected to the nodes corresponding to the previous and next letters of the same sequence. A new "root" node is created and the nodes corresponding to the first letter of each sequence are connected to it. The nodes are assigned colors as a function f of the letter they correspond to. A solution of length k' to the Flood-It instance consists of a sequence of colors, that would reach the end of each branch of the tree. By applying f^{-1} to this sequence we can find a supersequence of length k. This implies that a solution to an instance of Shortest Common Supersequence of length k exists if and only if the corresponding instance of Flood-It on Trees can be solved in k' or less moves where $k' \leq g(k)$ for some $g : \mathbb{N} \to \mathbb{N}$.

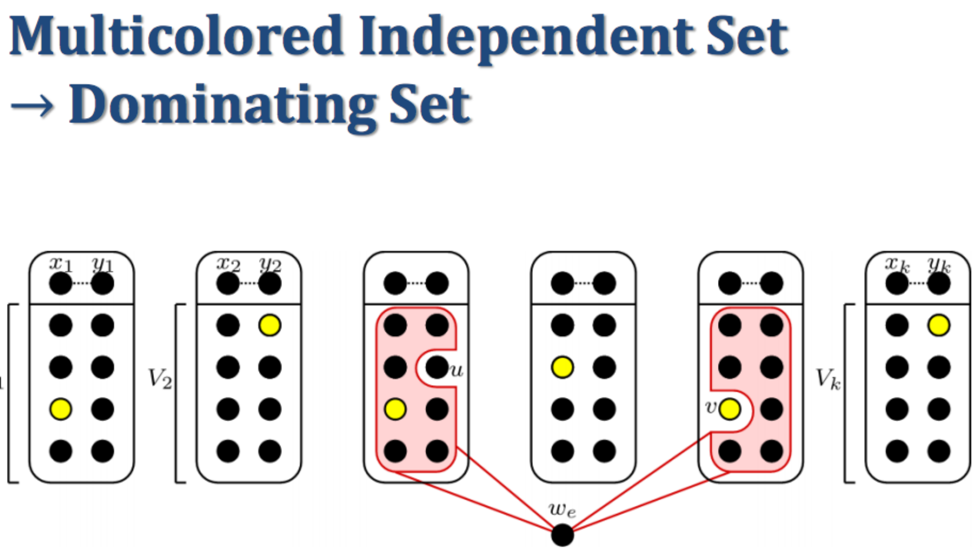
Therefore Flood-It on trees is W[1]-complete.

3.9 Dominating Set

We show that Dominating Set is $W[1]$-hard by reduction from Multicolored Independent Set. We map the set of vetices from the instance of Multicolored Independent Set as is to the instance of Dominating Set. We add edges to create a clique on each set of vertices of a particular color. We add two dummy vertices to each clique. For each edge $e = \{u, v\}$ in the original graph we create a new vertex w_e connected to all the vertices in the color classes of u and v except for u, v themselves. It can be shown that any solution to this instance of dominating set can be easily modified to a solution that includes exactly one vertex from each color class excluding the dummy vertices. Therefore we can assume that our solution is in that form.

The construction below shows that if we DON'T choose one of u or v, then we can cover this extra vertex w_e. So this simulates independence using dominating sets.

The following result from this reduction: Dominating set is $W[1]$-hard. (It is in fact $W[2]$-complete.) Dominating set $\notin FPT$ unless $FPT = W[2]$. The reverse reduction cannot be done unless $W[1] = W[2]$.



Multicolored Independent Set → Dominating Set

The reduction from Dominating set to Set Cover done in Lecture 11 preserves the parameter, therefore Set Cover is $W[2]$-complete.

3.10 Weighted Circuit SAT (Circuit Ones)

The "Ones" in Circuit Ones means that we want to get output 1 with k inputs set to 1. It's called minimum weighted since we are trying to minimize the Hamming distance to the all 0's set of input.

The question is can we satisfy the given acyclic Boolean circuit using only k 1's.

This problem defines the set $W[P]$. $W[P]$ is the set of all parameterized problems that can be reduced to weighted circuit SAT.

The depth of a circuit is the longest input-output path. The weft of a circuit is defined to be the maximum number of big gates on input to output path. A big gate is a gate with more than 2 inputs (in general, just needs to have more than a constant number of inputs).

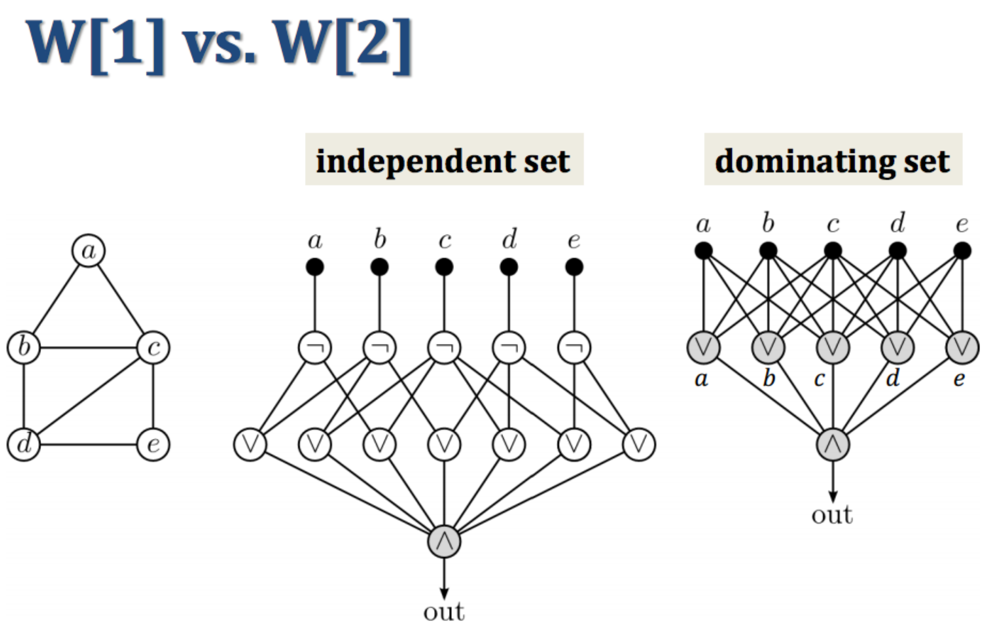
$W[t] = \{$ parameterized problems reducible to constant depth weft-t weighted Circuit SAT$\}$
(See paper by Buss and Islam [BI06] for related proofs and results)

Note that t is a fixed constant, and depth is always larger than weft.

It turns out you can write $FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq XP$. This inclusions are *all* strict if the exponential time hypothesis is true.

3.11 W[1] vs. W[2]

We can compare independent set to dominating set, as shown in the example below. We have two levels of big gates for dominating set, which shows that it is in $W[2]$. We have one level of big gates for independent set, so that is in $W[1]$.



Another example is weighted $O(1)$-SAT is $W[1]$ complete, while weighted CNF-SAT is $W[2]$ complete. 2-tape NTM is $W[2]$ complete.

Note: While boolean circuits of a higher weft can be converted to equivalent circuits of lower weft, this can lead to an exponential (in circuit size) blow up of the parameter. Therefore this doesn't constitute a parameterized reduction.

$W[*] = \cup W[i] \forall i \in \mathbb{N}$.

While "natural" problems in $W[1]$ and $W[2]$ are quite common, "natural" $W[3]$ problems do not seem to come up.

References

[BDFH09] Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[BI06] Jonathan F. Buss and Tarique Islam. Simplifying the weft hierarchy. *Theor. Comput. Sci.*, 351(3):303–313, 2006.

[DFNR98] Rod G. Downey, Michael R. Fellows, Rolf Niedermeier, and Peter Rossmanith. Parameterized complexity, 1998.

[Pie03] Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. pages 757–771, 2003.