# Exact Algorithms
# Lecture 7: The (S)ETH and Implications

2 February 2021 Lecturer: Michael Lampis

Summary: This lecture covers the following:

- Overview of the ETH and SETH, sparsification lemma.
- Implication ETH $\rightarrow$ $k$-Clique cannot be solved in $n^{o(k)}$
- Implication SETH $\rightarrow$ $k$-DS cannot be solved in $n^{ck}$ for $c < 1$.
- Faster algorithm for $k$-Clique, showing that the latter bound cannot be proven for this problem.
- An exotic problem ($k$-DS on tournaments).

## 1 The ETH – How hard is 3-SAT?

One could view the whole theory of NP-completeness as more or less equivalent to the following statement: 3-SAT is hard. One believes that P$\neq$NP if and only if one believes that 3-SAT is a problem that does not admit a polynomial time algorithm. But then this raises the question: what is the fastest algorithm that we can design for 3-SAT? And more generally, how fast can we hope to solve any NP-hard problem? Furthermore, what's so special about 3-SAT? What about 4-SAT, or SAT in general?

The same mathematical intuition that has led to P$\neq$NP being an accepted assumption has thus led to the following hypothesis:

**Definition 1.** *Exponential Time Hypothesis: There exists a constant $c > 1$ such that no algorithm can solve 3-SAT on $n$ variables in time $c^n$.*

The ETH plays in the theorey of exponential-time algorithms the same role that the P$\neq$NP assumption plays in the theory of polynomial-time algorithms: it gives us a specific starting point from which, through reductions, we can prove that algorithms for various problems cannot exist.

Is this assumption plausible? A first observation is that if the ETH is true then P$\neq$NP. Therefore, if we accept this, we are accepting a stronger assumption (which is bad). However, after considerable effort has been devoted to designing a fast 3-SAT algorithm, the state of the art has been stuck in $c^n$ for a long time. It therefore seems reasonable to accept the ETH as a working hypothesis. At the very least, when we prove that the existence of an algorithm would contradict the ETH, we are showing that designing such an algorithm requires a breakthrough in SAT-solving.

## 1.1 Exponential Time in what? – Thinking things through

The previous section touches on a topic that is often crucial when considering exponential time algorithms: what is the main variable for which we measure the running time? In the domain of polynomial time algorithm, such question are often overlooked. For example, consider the case of a graph with $n$ vertices. An algorithm that runs in time polynomial in $n$, also runs in time polynomial in the number of edges $m$ (since $m < n^2$) and vice-versa. Similarly, if we have a polynomial-time algorithm for 3-SAT, it would not matter if the running time is polynomial in the number of variables or clauses, since the two are polynomially related. Nevertheless, the difference between a $2^n$ and a $2^{n^2}$ algorithm is huge. We cannot afford to be so careless when speaking about exponential time algorithms.

This raises the question: why did we define the ETH as we did? What about the following version:

**Definition 2.** *Exponential Time Hypothesis(2): There exists a constant $c > 1$ such that no algorithm can solve 3-SAT on $n$ variables and $m$ clauses in time $c^{n+m}$.*

In other words, the ETH2 claims that 3-SAT needs time exponential in **the whole input**. Observe that this is a *stronger* assumption. We have ETH2 $\rightarrow$ ETH $\rightarrow$ P$\neq$NP. To see this, notice that an algorithm that runs in time $c^n$ for any $c$, also runs in $c^{n+m}$ for any $c$.

Things are now starting to look suspicious. Is the ETH2 more useful in proving lower bounds? Is it plausible as a hypothesis? Let us first answer the first question:

**Theorem 1.** *If the ETH2 is true, there exists $c > 1$ such that MaxIS on graphs with $n$ vertices cannot be solved in time $c^n$.*

**Proof:**

The proof is the same as in the previous lecture. The difference is that now we observe that the number of vertices $N$ of the new graph is $O(n+m)$. Therefore, if we could solve the new instance in time $c^N$ for any $c$, with could also solve 3-SAT in time $c^{n+m}$ for any $c$.

□

This is nice, because now this gives a "tight" answer to what is the correct running time needed to solve MaxIS: it is $c^n$ for some constant $c$ (of course, we don't know what the correct constant is! At least, however, we get a bound on the type of function that is correct). If we had assumed the original version of the ETH, our reduction would "only" have proved that there is no $2^{o(n^{1/3})}$ algorithm for Max-IS (why?).

The question then is whether we should believe this, just because it matches the best known algorithmic result for MaxIS. After all, in order to obtain this lower bound we assumed the ETH2, a complexity assumption that is stronger than the ETH.

Actually, the two are equivalent. This was shown in a paper by Impagliazzo, Paturi and Zane (2001).

**Theorem 2.** *The ETH is true if and only if the ETH2 is true.*

**Proof Sketch:**

One direction is easy. To see the direction ETH $\rightarrow$ ETH2 we can prove the contrapositive: if ETH2 is false then ETH is false. Suppose then that for all $c > 1$ we have a $c^{n+m}$ algorithm: we will obtain also a $c^n$ algorithm.

The main observation is the following: if $m = O(n)$ we are done: the $c^{n+m}$ algorithm run in time $c_2^n$ for some (slightly larger) $c_2$ that only depends on $c$. So the question is how to deal with instances with many clauses.

The proof now follows from a sparsification lemma, which proceed thourgh branching to produce $c^n$ instances, such that all of them have few clauses, and the whole formula was satisfiable if one of the new instances are satisfiable. The (complicated) details are beyond the scope of this class and can be found in the paper of IPZ.

□

Thanks to the Sparsification Lemma of the above theorem we know that in fact the ETH and the ETH2 are equivalent! This allows us to obtain strong lower bounds on the running time of any algorithm solving various classical problems (3-Coloring, Vertex Cover, Dominating Set,...). All of them in fact need time $c^n$, where $n$ is the number of vertices of the input graph, and this can simply be derived by a more careful reading of their NP-completeness proof.

So can we conclude that everything that is NP-hard, in fact needs time $c^n$ to solve exactly? Not quite. Consider the case of Planar Max IS.

**Theorem 3.** *If the ETH is true, there exists $c > 1$ such that no algorithm can solve Max IS on planar graphs in time $c^{\sqrt{n}}$. Furthermore, there exists an algorithm for this problem running in $2^{O(\sqrt{n})}$.*

## 1.2 Think things through – Strong ETH

As we have seen, the ETH is still fundamentally a qualitative (and not a quantitative) statement: it simply says that *some* constant $c$ exists, so that the complexity of 3-SAT is $c^n$. What is that constant?

There do exist algorithms for 3-SAT that do better than $2^n$, by improving the base of the exponential. For example, consider the following branching algorithm: take a clause $(l_1 \vee l_2 \vee l_3)$ and branch over all its possible satisfying assignments. Each branch settles the value of three variables. Repeat this until all clauses have size 2 (in this case the problem can now be decided in polynomial time).

What is the complexity of this algorithm? A simple recursive formula is $T(n) \leq 7T(n-3)$ since there are 7 satisfying assignments for a clause of size 3. This gives $T(n) < 7^{n/3}$. This is an improvement over $2^n = 8^{n/3}$. Great!

Can we do this trick for 4-SAT? Of course. The running time is now $T(n) < 15^{n/4}$, since each 4-clause has 15 satisfying assignments. This is still better than $2^n$, but by a smaller margin. What about 5-SAT? 1000-SAT?

The algorithm we saw above is of course not the best known branching algorithm. The point here is that improving upon the trivial $2^n$ becomes harder and harder as the length of the clauses allowed becomes longer. In other words, the ETH states that there exists a constant $c_3 > 1$ such that $c_3^n$ is the correct running time for 3-SAT. If the ETH is true, what can we say about $c_4$, $c_5$, and

all the corresponding constants for larger clause sizes? One unsurprising observation is that things do not get easier as we increase the size of the clauses.

**Theorem 4.** *For all constant $k$ we have $c_k \leq c_{k+1}$.*

**Proof:**

Take an instance of $k$-SAT and add a new variable $y$. For each clause we make to copies. In the first copy we add $y$ and in the second $\neg y$. It is not hard to see that the new instance is satisfiable if the old instance was, while the number of variables is essentially the same.

□

What is somewhat more surprising is that if any of these problems needs exponential time, then the ETH is true.

**Theorem 5.** *If $c_k > 1$ for some constant $k$, then $c_3 > 1$.*

**Proof Sketch:**

Given an instance of $k$-SAT with $n$ variables, we first apply the sparsification lemma of the previous lecture, which produces some instances of the same problem with the property that $m = O(n)$. Now we can apply a standard trick: for a clause $C$ with length more than three, we introduce a new variable $y$. If $C = (l_1 \vee l_2 \vee l_3 \vee \ldots \vee l_k)$ we replace with two clauses $C_1 = (l_1 \vee l_2 \vee y) \wedge (\neg y \vee l_3 \vee \cdots \vee l_k)$. Repeating this produces an equivalent instance of 3-SAT.

How many new variables did we add? For a clause of size $k$ we add at most $k$ new variables, but $k$ is a constant. Therefore, we added at most $O(m)$ new variables. But $O(m) = O(n)$ therefore the total number of variables in the new instance is $O(n)$. Thus, if we have a $2^{o(n)}$ algorithm for the new instance we could get one for $k$-SAT.

□

The above theorem is somewhat reassuring in that it shows that there is nothing special about size 3: all constant sizes need exponential time. However, we still have no idea what the correct constants are.

Motivated by the above, and the idea that $c_k$ should be an increasing function we can now make the following conjecture:

**Definition 3.** *Strong Exponential Time Hypothesis: For any $\epsilon > 0$ there exists a $k$ such that $c_k > 2 - \epsilon$. In other words, $\lim_{k\to\infty} c_k = 2$. Therefore, there is no $(2-\epsilon)^n$ algorithm for* SAT.

Is the SETH true? Unlike the ETH, the SETH is mostly considered wide open. In particular, it is still considered quite plausible that there is a $1.99^n$ algorithm for SAT while the ETH is true. Nevertheless, lower bounds based on the SETH can still be useful, because they show that improving upon an algorithm would require improving on the fastest known satisfiability algorithm.

# 2 ETH implications for FPT problems

We will use the following version of the ETH:

**Definition 4** (ETH)**.** *The Exponential Time Hypothesis states the following: there exists no algorithm which, given a 3-SAT instance with $n$ variables and $m$ clauses decides if the instance is satisfiable in time $2^{o(n+m)}$.*

Recall that in the previous section we saw a slightly different (more careful) statement of the ETH, saying that there exists a $c$ such that no algorithm can achieve $c^{n+m}$ complexity. The two statements are almost equivalent (though it's an open problem whether they are exactly equivalent).

The ETH already tells us something about the complexity of the best FPT algorithm for VERTEX COVER.

**Theorem 6.** *If there exists an algorithm that decides if a graph on $n$ vertices has vertex cover at most $k$ in time $2^{o(k)}n^{O(1)}$ then the ETH is false.*

**Proof:**

Suppose that such an algorithm exists: we use it to compute the optimal vertex cover of any graph by repeatedly increasing $k$ (i.e. run it for $k = 1$, $k = 2$, ..., $k = n$). The total running time is $2^{o(n)}n^{O(1)}$. As we saw in the previous lecture, this contradicts the ETH.

□

Because of the simple observation of the previous theorem we can infer that most natural graph optimization problems (CLIQUE,VERTEX COVER,DOMINATING SET), even if they have an FPT algorithm, that algorithm must have at least an exponential $f(k)$. Intuitively, this makes sense: the simplest parameterization of any problem is to set $k = n$. If we know that for this "easiest" parameterization we cannot get $f(k)$ to be sub-exponential, it follows that we should not be able to do this for more ambitious parameterizations.

## 2.1 Clique is hard under ETH

As seen in the previous lecture, if we accept that Clique has no FPT algorithms, then we can infer that a number of other problems also have no FPT algorithms. We say that such problems are W-hard. But why should we believe that Clique has no FPT algorithm in the first place? One answer is that many people have tried to obtain better than $n^k$ algorithms for Clique, without success. However, Clique is not a problem that has been studied as intensively as SAT. Can we infer something about the hardness of Clique from the ETH?

The answer is yes. It turns out that if one accepts the ETH then there is no FPT algorithm for Clique. In fact, one can say something even stronger.

**Theorem 7.** *For any function $f$ we have the following: If there exists an algorithm for $k$-Clique that runs in time $f(k)n^{o(k)}$ then the ETH is false.*

**Proof:**

Fix some function $f : \mathbb{N} \to \mathbb{N}$ and suppose without loss of generality that this function is strictly increasing (this is a natural requirement for the function that describes an algorithm's running time). Let $f^{-1}(k)$ be its inverse function.

Fix some value $k$. Take an instance $\phi$ of 3-SAT on $n$ variables and $m = O(n)$ clauses. In particular, we can assume that $n > f(k)$ or equivalently $k < f^{-1}(n)$, because we can take a sufficiently large instance of 3-SAT as the starting point of our reduction.

We now do the following: partition the clauses of $\phi$ into $k$ sets $C_1, \ldots, C_k$. For each $C_i$, let $X_i$ be the set of variables of $\phi$ apearing in the clauses of $C_i$. We have $|C_i| \leq \frac{m}{k} = O(n/k)$. Therefore, $|X_i| = O(n/k)$, since all clauses have size at most 3.

We now construct a graph $G(V, E)$. For each $i \in \{1, \ldots, k\}$, for each truth assignment to the variables of $X_i$ we check if this assignment satisfies all the clauses of $C_i$. If yes, we construct a vertex $u$ in the graph $G$ corresponding to this assignment. Observe that the total number of vertices constructed is at most $k2^{|X_i|} = k2^{O(n/k)}$.

Now, in the graph $G(V, E)$ every vertex corresponds to some partial assignment to the variables of $\phi$. We add an edge between $u, v$ if and only if the partial assignment corresponding to these two vertices are *compatible*, that is, if they give the same value to all their common variables.

We claim that $G$ has a clique of size $k$ if and only if $\phi$ is satisfiable. Suppose $\phi$ is satisfiable. From a global satisfying assignment we can infer an assignment to the variables of each $X_i$. This assignment satisfies all of $C_i$, therefore, it is represented by a vertex in $G$. Clearly, these partial assignments are all compatible, so their vertices must form a clique.

For the converse direction: suppose there is a clique of size $k$ in $G$. We can partition $V$ into $k$ sets $V_1, \ldots, V_k$ such that $V_i$ contains the partial assignments to $X_i$. Clearly, each $V_i$ is an independent set (why?), so if a clique of size $k$ exists it must pick one vertex from each $V_i$. This gives an assignment for each vertex of each $X_i$, and thus an assignment for $\phi$ because all partial assignments are pair-wise compatible (if we have found a clique). To see that this assignment satisfies $\phi$ observe that is satisfies all of each $C_i$, because the vertices of $V_i$ represent partial assignments that satisfy $C_i$.

Finally, for the running time, suppose that we had an algorithm running in time $f(k)N^{o(k)}$ that decides if an $N$-vertex graph has a clique of size $k$. Running it on $G$ would take time $f(k)(k2^{O(n/k)})^{o(k)}$. As we initially selected $n$ large enough so that $f(k) < n$ the running time would be $2^{o(n)}$, contradicting the ETH.

□

What does the above theorem mean? One interpretation is that the trivial $n^k$ algorithm for Clique is in fact the optimal algorithm. However, note that the theorem states that there is no $n^{o(k)}$ algorithm, meaning that an algorithm that runs in time, say, $n^{0.1k}$ is still possible without violating the ETH? Could we hope to achieve such an algorithm?

The theorem below says maybe yes!

**Theorem 8.** *There is an algorithm that solves $k$-Clique in time $n^{\omega k/3}n^{O(1)}$, where $\omega$ is the matrix multiplication exponenet, that is, if we assume that the running time needed to calculate the product of two $n \times n$ matrices is $n^{\omega}$.*

**Proof:**

The algorithm is based on two parts: first, there is an algorithm which can find a triangle in a graph (a $K_3$) in time $n^{\omega}$. Then, we show how this algorithm can be used to find a $k$-clique.

We begin with the second part. Suppose that we are given a graph $G(V, E)$ and we want to find a clique on $k$ vertices. Without loss of generality, suppose $k$ is divisible by 3. We construct a new

graph $G'$ as follows: for each set $S \subseteq V$ of size $|S| = k/3$ that induces a clique on $G$ we construct a vertex $u_S$ in $G'$. We connect two vertices $u_S, u_{S'}$ if the sets $S, S'$ are disjoint and together they induce a clique of size $2k/3$ in $G$.

Observe now that the graph $G'$ contains a triangle if and only if the graph $G$ contains a clique of size $k$: if $S \subseteq V$ is such a clique of $G$, any partition of $S$ into three equal disjoint sets $S_1, S_2, S_3$ is a triangle $u_{S_1}, u_{S_2}, u_{S_3}$ in $G'$. In the converse direction, any triangle in $G'$ gives a clique in $G$ by taking the union of the three sets that correspond to its vertices.

Note also that $G'$ has at most $N = \binom{n}{k/3}$ vertices. It has therefore at most $n^{2k/3}$ edges, and it can be constructed in $n^{2k/3}$ time. If the claimed algorithm for finding triangles exists, we can run it in time $N^\omega = n^{\omega k/3}$ to find the clique of size $k$ in $G$.

For the remaining part, observe that if $A$ is the adjecency matric of $G$, then the matrix $A^3$ contains a non-zero value in position $(i, i)$ if and only if there exists a triangle that includes the vertex $i$. To see this, recall that for two matrices with non-negative elements $A, B$ we have $AB[i, j] > 0$ of and only if there exists $k$ such that $A[i, k] > 0$ and $B[k, j] > 0$. Therefore, $A^2$ contains a positive value in $(i, j)$ if and only if there exists a $k$ such that $A[i, k] = A[k, j] = 1$. Furthermore, $A^3 = AA^2$ contains a positive value in $(i, i)$ if and only if there exists $j$ such that $A[i, j] > 0$ and $A^2[j, i] > 0$. So, $A^3[i, i] > 0$ if and only if $i$ has a neighbor $(j)$ with whom he has a common neighbor.

□

We recall that the best currently known value of $\omega$ is slightly more than $2.3$, so this algorithm runs in time less than $n^{0.8k}$.

We conclude that we can say qualitatively what is the performance of the best parameterized algorithm for Clique, while still being unsure of the exact constants (under the ETH). We observe that the $n^{o(k)}$ lower bound we have shown for Clique immediately transfers to other W-hard problems. In particular, Dominating set also does not have $n^{o(k)}$ agorithms. However, as we see in the next section, there is reason to believe that Dominating Set is even harder.

## 2.2 Dominating Set and the SETH

Consider now the second most important W-hard problem: $k$-Dominating Set. Because of the reduction of the previous lecture, the hardness of $k$-Clique immediately carries over: there is no $n^{o(k)}$ algorithm for $k$-Dominating Set, unless the ETH is false. However, for this problem we can say a bit more.

**Theorem 9.** *If the SETH is true then there is no $n^{ck}$ algorithm for $k$-*DOMINATING SET *for any $c < 1$.*

**Proof:**

Take any SAT instance $\phi$ and partition the variables into $k$ groups of $n/k$ variables. We construct a graph $G(V, E)$. For each group of variables, for each assignment of values to these variables, we create a vertex in $V$, thus making a total $k2^{n/k}$ vertices. We add to $G$ one vertex for each clause of $\phi$. Finally, for each vertex that represents an assignment with add an edge connecting it to each clause that contains one of the literals the assignment sets to true. We also connect any two vertices representing assignments to the same group of variables.

If $\phi$ is satisfiable, we select one vertex from each group of vertices representing partial assignments, according to the satisfying assignment of $\phi$. This is a dominating set, since we selected one vertex from each clique, and we also dominate all clause vertices, because all clauses have a true literal. For the other direction, if there is a dominating set of size $k$ we select an assignment that agrees with the vertices this set selects from the cliques.

For the running time bound, observe that the new graph has size $k2^{n/k} + m \leq 2^{n/k}n^{O(1)}$. If there is a $N^{ck}$ algorithm for dominating set, the total running time will be $2^{cn}n^{O(1)}$, where $c < 1$. This would refute the SETH.

□

The above theorem essentially states that, if one believes the SETH, it follows that dominating set is in fact harder than clique, because as we saw there is in fact a $n^{ck}$ algorithm for clique where $c < 1$.

# 3 ETH lower bounds for "Easy" Problems

Recall that a tournament is a directed graph $G(V, E)$ such that for each pair of vertices $u, v \in V$ exactly one of the following is true: either $(u, v) \in E$ or $(v, u) \in E$.

Consider a version of dominating set on this class of graphs: a set $D \subseteq V$ is a dominating set if for all $u \in V \setminus D$ there exists $v \in D$ such that $(v, u) \in D$. Can this problem be solved in polynomial time? Almost!

**Theorem 10.** *There is an algorithm that finds the minimum dominating set of a tournament in time $n^{\log n+O(1)}$, where $n$ is the number of vertices of the tournament.*

**Proof:**

The proof rests on the following fact: for any tournament on $n$ vertices there exists a dominating set of size at most $\log n$. To see this, we note that a tournament has by definition $\binom{n}{2}$ arcs. If $d^+(u)$ denotes the out-degree of a vertex $u$ (that is, the number of arcs leaving from $u$) we have $\sum_{u \in V} d^+(u) = |E| = \frac{n(n-1)}{2}$. Therefore, there exists a vertex $u$ with out-degree at least $\frac{n-1}{2}$. Select it into the dominating set. We have now dominated all its out-neighbors, plus $u$ itself, so at least $\frac{n+1}{2} \geq \frac{n}{2}$ vertices. We now proceed by induction to find a dominating set of size at most $\log(n/2)$ in the tournament induced by the vertices not dominated by $u$. This set, together with $u$ is a dominating set of the whole graph.

Since the optimal solution has size at most $\log n$ a simple algorithm is to try out all sizes of at most this size. Checking if a set is dominating can of course be done in polynomial time.

□

The above algorithm runs in quasi-polynomial time. This implies, in particular, that the problem is probably not NP-hard.

**Theorem 11.** *If Dominating Set on Tournaments is NP-hard then the ETH is false.*

**Proof:**

Suppose that the problem is NP-hard, therefore there is a reduction which, given a 3-SAT instance $\phi$ produces a tournament $G(V, E)$ whose size tells us if $\phi$ is satisfiable. If $\phi$ has $n$ variables, because the reduction runs in polynomial time, we have that $|V| \leq n^c$ for some constant $c$.

Now, suppose we run the algorithm of the previous theorem on the tournament $G$. Its running time will be $N^{\log N+O(1)} = n^{c^2 \log n+O(1)} = 2^{O(\log^2 n)} = 2^{o(n)}$.

□

It seems that we can realistically hope for a polynomial time algorithm! Unfortunately, this is probably not the case.

**Theorem 12.** *If there exists an $n^{o(\log n)}$ algorithm for Dominating Set on tournaments then the ETH is false. Furthermore, if there exists an $n^{o(k)}$ algorithm for $k$-Dominating Set in tournaments, then the ETH is false.*

Before, we explain how this reduction works, let us state a useful graph theoretic fact about domination in tournaments.

**Theorem 13.** *For every $k > 1$ there exists a tournament of order at most $r_k = 2k^2 2^k + k$ that does not have any dominating set of size $k$.*

**Proof:**

The proof is by the so-called probabilistic method, one of the deepest and most surprising proof methods in discrete mathematics. The strategy is the following: we will construct a tournament on $r_k$ vertices **randomly**. Then we will prove that with some **positive** probability this tournament does not have a dominating set of size $k$. Then we are done! The reason we are done is that if the theorem was false, then **all** tournaments of order $r_k$ would have a dominating set of size $k$. Therefore the probability of randomly constructing one that does not would have been 0.

The construction simply takes $r_k$ vertices and, for each pair of vertices sets the direction of the arc in one directions with probability $1/2$. All choices are independent.

Fix a set of vertices $S$ with $|S| = k$. What is the probability that $S$ dominates a vertex $u \in V \setminus S$? The probability that $S$ does *not* dominate $u$ is exactly $(1/2)^k$ (all $k$ arcs connecting $u$ to $S$ must be leaving $u$). Thus, the probability that $S$ does dominate $u$ is $1 - (1/2)^k$.

What is the probability that $S$ is a dominating set? Here we observe that the probability that $S$ dominates $u$, and the probability that it dominates another vertex $v$ are independent. Therefore, the probability that $S$ is a dominating set is $(1 - (\frac{1}{2})^k)^{r_k - k}$. By using the fact that $1 - x \leq e^{-x}$ when $x \leq 1$ we conclude that the probability that $S$ is dominating is at most $e^{-2k^2}$.

Let us now upper bound the probability that there exists a dominating set. The probability of any specific set being dominating was upper bounded above by $e^{-2k^2}$. The number of candidate dominating sets is at most $\binom{r_k}{k}$, that is, there are at most $r_k^k = 2^{k^2+o(k^2)} \leq 2^{2k^2}$ sets $S$ of size $k$ in the tournament. Recall that if we have two random events $E_1, E_2$ we have that $P[E_1 \vee E_2] \leq P[E_1] + P[E_2]$ (this is called the union bound). We can then say that the probability that *any* set of size $k$ is dominating is at most $2^{2k^2} e^{-2k^2} < 1$. Therefore, the probability that no set of size $k$ is dominating is strictly positive.

□

We are now ready for the proof of Theorem 12.

**Proof of [:**Theorem 12]

We describe an FPT reduction from $k$-dominating set. Suppose that we have a tournament of size $r_k = O(k^2 2^k)$ which does not have any dominating set of size $k+1$. Call this tournament $T(V_T, E_T)$. We will build a construction that will include many copies of $T$. Such a tournament exists by the previous theorem.

We begin with an instance $G(V, E)$ of $k$-Dominating Set. Our constructed directed graph $G'$ will contain two sets of vertices $V_1, V_2$ and a special vertex $v$. We set $V_2 = V$. We also set $V_1 = V \times V_T$, in other words, for every vertex of $G$ we take a copy of $T$. For every arc $(v_1, v_2) \in E_T$ we include all possible arcs $((u_1, v_1), (u_2, v_2))$ in $G'$. In words, if there is an arc in $T$ we place arcs of the same direction between all copies of these vertices in $G'$. We add all possible arcs from $v$ to $V_2$ and all arcs from $V_1$ to $v$. For each vertex $u \in V_2$ and each vertex $(u', v_i) \in V_1$ we add the arc $(u, (u', v_i))$ if $u$ dominates $u'$ in $G$. Otherwise, we add an arc of the opposite direction. Finally, we add all missing arcs in an arbitrary direction to make the graph a tournament.

Suppose that the original graph has a dominating set of size $k$. We claim that selecting the same vertices from $V_2$ as well as $v$ is a dominating set of size $k+1$ in $G'$. Observe that $v$ dominates all of $V_2$, while, whenever $u$ dominates $u'$ in $G$, $u \in V_2$ dominates the copy of $T$ corresponding to $u'$. Therefore, all copies of $T$ in $V_1$ are dominated.

For the converse direction, suppose that a dominating set of size $k+1$ exists in $G'$. This set must contain some vertex outside $V_2$, otherwise $v$ is not dominated. It is therefore using at most $k$ vertices of $V_2$. We claim that these vertices must be a dominating set of $G$. For the sake of contradiction, suppose that they are not, and that they leave a vertex $u' \in V$ undominated. We look at the copy of $T$ corresponding to $u'$ in $V_1$. Its vertices are not dominated by the selected vertices of $V_2$, or by $v$, therefore they must be dominated by selected vertices in $V_1$. However, there are at most $k+1$ such selected vertices in $V_1$ and since $T$ does not have a dominating set of size $k+1$ we have a contradiction.

For the running time bound, the tournament we made has roughly $N = 2^k n$ and we set $k' = k+1$ vertices. So, if there existed a $N^{o(k')}$ algorithm to find its minimum dominating set we would get a running time of $2^{o(k^2)} n^{o(k)}$ for dominating set.

For the second running time bound, consider the reduction of Theorem 9, but set $k = \sqrt{n}$. We get that $n$-variable SAT can be (in $2^{\sqrt{n}} = 2^{o(n)}$ time) reduced to an instance of dominating set with $k = \sqrt{n}$ and $|V| = 2^{\sqrt{n}}$. Executing the reduction of this theorem gives an instance of dominating set on tournaments with roughly $N = 2^{O(\sqrt{n})}$ (and the reduction runs in time $2^{O(\sqrt{n})} = 2^{o(n)}$) if we are given the tournament $T$. So, if there is a $N^{o(\log N)}$ algorithm, this translates to $(2^{\sqrt{n}})^{o(\sqrt{n})} = 2^{o(n)}$.

□