1 Final Notes on k-Path

Here are the fastest known k-PATH algorithms:

- [Wil09]: k-PATH (on directed or undirected graphs) is in randomized $O^\star(2^k)$ time.
- [Tsu19]: k-PATH (on directed or undirected graphs) in deterministic $O^\star(2.56^k)$ time.
- [BHKK17]: Undirected k-PATH is in randomized $O^\star(1.66^k)$ time.
- [BDH18]: For graphs with at most p distinct k-paths, can find one in $O^\star(p^2 \cdot 2^k)$ deterministic time.

These algorithms also use very different techniques from the ones we showed you! Check the references if you're interested. In general, one reference for finding the fastest algorithm for various parametrized problems is `http://fpt.wikidot.com/fpt-races`, but note it's not always up to date (for deterministic k-PATH for example).

The gap between the randomized and deterministic algorithms provokes the question: is there an inherent gap between randomized and deterministic algorithms? The answer is "probably not", given the very recent work [CT20] (which builds on a lot of prior work). Lijie and Roei show that, under plausible hypotheses, that every randomized algorithm running in $T(n)$ time can be deterministically simulated in $O(nT(n)^{1+\varepsilon})$ time, for every desired $\varepsilon > 0$.

2 Parameterized Complexity

We now turn to developing a theoretical understanding of what parametric problems do *not* have FPT algorithms, i.e., the complexity theory of parametric problems. In these notes, we'll only cover a few of the known results on this topic, picking results that are closest to other material from this class. The textbook (`https://www.mimuw.edu.pl/~malcin/book/parameterized-algorithms.pdf`) covers some more results in detail.

Recall the class FPT consists of the parametric problems solvable in time $f(k)\, n^c$, for some constant c and some computable f. This "extended" notion of tractability not only includes polynomial-time solvable problems but also appropriate parameterizations of some NP-hard problems. We have discussed various techniques for deriving FPT algorithms for problems, but so far we haven't said much about limitations on solving problems in an FPT way.

2.1 The class XP

What should be considered *intractable* in the parameterized world? Generally speaking, intractable problems should be those for which the running time dependence on the parameter is huge, such as $n^{f(k)}$ where $f(k) \geq k$. When the exponent of the polynomial depends on the parameter, this is a much worse dependence: such a bound would be intractable for large n and even moderately sized k. This motivates the following class:

Definition 2.1 *The class* XP *consists of parameteric problems solvable in $n^{f(k)}$ time, where f is a computable function.*

On the positive side, any problem $L \in \mathsf{XP}$ is indeed solvable in polynomial time, for every fixed value of k. To formalize this, let L be a parametric problem, and for any constant $k \in \mathbb{N}$ we define the *k-slice of L* to be

$$L_k = \{(x,k) \mid (x,k) \in L\}.$$

That is, L_k only includes the instances of L with parameter equal to exactly k.

Proposition 2.1 *If $L \in \mathsf{XP}$, then for all $k \in \mathbb{N}$, $L_k \in \mathsf{P}$.*

Proof. For every $k \in \mathbb{N}$, L_K is solvable in $n^{f(k)}$ time, where $f(K)$ is some fixed constant. Let A be an algorithm solving L in at most $n^{f(k)}$ time. Then the following algorithm B solves L_K in $n^{f(k)}$ time (which is polynomial for constant K):

$B(x)$: *Run $A(x,K)$ and output the answer.* □

On the negative side, the degree of the polynomial in the running time for solving L_K can obviously increase with K, hence we may consider the algorithm A for L to be inefficient.

Exercise: Is the converse of the proposition true? (For all parametric L, if for all $k \in \mathbb{N}$, $L_k \in P$, then is $L \in \mathsf{XP}$?) Prove your answer.

2.2 A Lower Bound for XP

So, is XP really an "intractable" complexity class? The following theorem confirms this intuition, in some sense:

Theorem 2.1 $\mathsf{FPT} \neq \mathsf{XP}$.

This separation may be viewed as an analogue of the time hierarchy theorem from complexity theory, with $\mathsf{FPT} \approx \mathsf{P}$ and $\mathsf{XP} \approx \mathsf{EXP}$.

Proof. We give an explicit problem $L \in \mathsf{XP} \setminus \mathsf{FPT}$. Define

$$SIM = \left\{(M,x,k) \mid \text{Turing machine } M \text{ accepts input } x \text{ in at most } (|M|+|x|)^{k+1} \text{ steps}\right\}.$$

(Note: You don't actually need to know what a Turing machine is here, to get this argument. If it bothers you, substitute "algorithm" for "Turing machine" in the below.) First, observe that $SIM \in \mathsf{XP}$: note the input length n is $O(|M|+|x|+|k|)$. using a universal Turing machine we can simulate any given Turing machine M on an input x with polynomial overhead, so L is solvable in $n^{O(k)}$ time.

Suppose $SIM \in \mathsf{FPT}$. We will derive a contradiction. This implies that SIM has a $g(k) \cdot n^c$ algorithm, for some computable g and constant $c \geq 1$. Then the $(c+1)$-th slice of SIM, SIM_{c+1}, is decidable in $g(c+1) \cdot n^c \leq O(n^c)$ time. Let A_{c+1} be a Turing machine for SIM_{c+1} running in this time.

Now let B be **any** Turing machine running in n^{c+1} time. We can now simulate B on n-bit inputs in only $O(n^c)$ time, as follows:

> C: On input x, run A_{c+1} on (B,x) and output its answer.

On input x, C runs in $O(|B|+|x|)^c \leq O(|x|^c)$ time, by assumption on A_{c+1}. But we have given a way to simulate *every* Turing machine running in n^{c+1}, that runs in only $O(n^c)$ time. Essentially we have derived that

$$\mathsf{TIME}\left(n^{c+1}\right) \subseteq \mathsf{TIME}\left(n^c\right),$$

which contradicts the time hierarchy theorem from complexity theory (recall the time hierarchy theorem says that $\mathsf{TIME}\left(n^c\right) \subsetneq \mathsf{TIME}\left(n^{c+1}\right)$). □

2.3 The W-hierarchy

There are many problems contained in XP that are not believed to be in FPT. In fact there is an infinite hierarchy of problems between FPT and XP (sort of like the polynomial hierarchy, but not quite). The most widely-studied such hierarchy is the so-called W*-hierarchy*. The *W* has traditionally stood for a circuit parameter called *Weft* but for our purposes, we will think of the *W* as standing for *Weight* – this is the *Weight-hierarchy*.

The W-hierarchy is defined with respect to various weighted satisfiability problems.

Definition 2.2 *Let $x \in \{0,1\}^n$. The* weight of x *is the number of bits set to* 1 *(that is, its L_1 norm).*

A *parameterized weighted satisfiability problem* has the following high-level form:

> *Given a logical expression E encoding a Boolean function, is there a satisfying assignment to E with weight equal to k?*

The W-hierarchy is obtained by considering parameterized weighted SAT problems over different classes of logical expressions.

The canonical problem for the W-hierarchy is the *Weighted Depth-t SAT* problem. Let $t \geq 1$ be a fixed constant.

> **Weighted Depth-t SAT (a.k.a. W-Depth-t-SAT)**: given an AND, OR, and NOT circuit with t layers of unbounded fan-in gates, followed by a bottom layer (closest to the inputs) with fan-in at most two, is there an input assignment x of weight k that satisfies C?

In circuit complexity language, Weighted Depth-t SAT is asking if a given "AC^0 circuit of depth $t+1$" with bottom fan-in two has a SAT assignment with exactly k variables set to true. Intuitively, this problem should get harder as the depth of the circuit increases. (Don't worry if you are uncomfortable with circuits; we'll only really deal with CNFs in these lecture notes!)

To understand the true difficulty of these SAT problems, we would like to set up a complexity class where these weighted SAT problems are "complete" for those classes, in the sense that if the weighted SAT problem is FPT, then all problems in the class are also FPT. To have a notion of "completeness" for a complexity class, we first need a notion of *reducibility*.

2.4 Reductions

What makes NP-completeness so powerful is that an NP-complete problem can be used to express *any* other NP problem Π, with only a polynomial-time reduction from Π. Fine-grained hardness is powerful because its reductions from A to B "preserve exponents". We want reductions between parametric problems that "preserve" their parametrized tractability. The generic form of FPT reducibility, defined below, fits the bill:

Definition 2.3 *We say that there is an* FPT-reduction *from A to B, which we denote $A \leq_{fpt} B$ if there is a computable function $g : \mathcal{N} \to \mathcal{N}$ and an oracle Turing Machine M^B (with oracle access to B) such that:*

- *For all (x,k), $M^B(x,k)$ accepts if and only if $(x,k) \in A$.*
- *The running time of M^B is FPT.*
- *For every query (y,k') to B, we have $k' \leq g(k)$.*

The last item says that all queries to B by $M^B(x,k)$ must *preserve the original parameter k*. All parameters appearing in all queries must solely be a function of the original parameter k in the input to A.

Our definition of FPT reduction is more general than normally considered: typically in the literature, people have considered many-one reductions from A to B (where there is at most one oracle query). However, since the algorithms we consider are all either deterministic or randomized, the more general oracle version seems more logical, as it still satisfies the usual properties one would want from a reducibility notion:

Proposition 2.2 *If $A \leq_{fpt} B$ and $B \leq_{fpt} C$, then $A \leq_{fpt} C$.*

Corollary 2.1 *If B is FPT and $A \leq_{fpt} B$, then A is FPT.*

Exercise: Prove these.

We now define the parametrized complexity class $\mathsf{W}[t]$, for integers $t \geq 1$.

Definition 2.4 *Let $t \geq 1$ be an integer. $\mathsf{W}[t]$ is the class of parametric problems that have an FPT-reduction to W-Depth-t-SAT.*

Exercise: Prove that for every t, $\mathsf{FPT} \subseteq W[t]$.

Note also that for every t, $\mathsf{W}[t] \subseteq W[t+1]$ (we are increasing the depth from t to $t+1$, which can only make the circuit more expressive).

There is one more interesting parameterized class that fits in the above inclusions:

Definition 2.5 *The class $\mathsf{W}[P] \subseteq \mathsf{XP}$ consists of the set of problems FPT-reducible to satisfying an* arbitrary *Boolean circuit consisting of any number of ANDs, ORs, and NOTs (as long as it is still directed and acyclic), while setting exactly k variables to true. This latter problem is called* Weighted Circuit-SAT.

By definition, $\mathsf{W}[t] \subseteq \mathsf{W}[P]$ for every $t \geq 1$, and $\mathsf{W}[P] \subseteq \mathsf{XP}$.

Note that the W-classes are all defined with respect to NP-complete problems (NP-hard variants on satisfiability), so this hierarchy could be seen as a parameterized refinement of the class NP (although, there *can* be even harder problems in the W-hierarchy, depending on how a problem is parameterized!). Note that:

Proposition 2.3 *If $\mathsf{P} = \mathsf{NP}$ then Weighted Circuit-SAT $\in \mathsf{P}$, hence is in FPT, hence $\mathsf{W}[P] = \mathsf{FPT}$.*

That is, separating $\mathsf{W}[P]$ from FPT is only harder than separating NP from P. An interesting open problem is to find interesting consequences of $\mathsf{W}[1] = \mathsf{W}[100]$.

3 W[1] and k-Clique

For the class $\mathsf{W}[1]$, it suffices to look at a simpler problem.

Weighted 2-SAT (a.k.a. W2SAT):
Given a 2CNF formula F and a parameter k, does F have a satisfying assignment of exactly k ones?

Theorem 3.1 *$\mathsf{W}[1]$ equals the class of parametric problems that have an FPT-reduction to W2SAT.*

We won't prove the above theorem in this class; it's a bit complicated, and it's not clear how instructive the proof is.

Conjecture 3.1 $\mathsf{FPT} \neq W[1]$.

Why do we think $\mathsf{W}[1]$ contain hard problems? Let's give some evidence. For one, k-Clique is in $\mathsf{W}[1]$.

Theorem 3.2 *k-Clique $\in W[1]$.*

Proof. We give an FPT reduction from k-Clique to W2SAT. Given a graph $G = (V, E)$ with vertices $\{1, \ldots, n\}$, take the variables $x_1, \ldots, x_n$ and create the 2CNF instance

$$F = \wedge_{(i,j) \notin E} (\neg x_i \vee \neg x_j).$$

We claim that F has a weight-k satisfying assignment if and only if G has a k-clique. If F has a satisfying assignment with $x_{i_1}, \ldots, x_{i_k}$ true, then among each clause $\neg x_i \vee \neg x_j$ in F, one of the variables must be false. Therefore, no pair $x_{i_j}, x_{i_{j'}}$ appears in F, so there is an edge between i_j and $i_{j'}$. Conversely, if we have a k-clique $i_1, \ldots, i_k$ then we can set x_{i_j} to be true for all $j = 1, \ldots, k$ and the other variables false, and this assignment will satisfy F. □

In fact there is an FPT reduction from W2SAT to k-Clique (omitted in these notes), so the two problems are FPT equivalent. We could have simply defined $\mathsf{W}[1]$ to be the class of problems that are FPT-reducible to k-Clique, instead!

3.1 k-Clique and ETH

That's all fine and good... but why is k-Clique hard? For one, the best known algorithms for k-Clique only run in $n^{\delta k}$ time for some constants $\delta > 0$. But we have never settled for this kind of explanation in fine-grained complexity: we always want to know if there are any unexpected *consequences* of actually having a faster k-clique algorithm. One thing we can show is that, if we could get an FPT algorithm for k-Clique, then much of fine-grained complexity would look very different! Perhaps the most compelling evidence that $\mathsf{W}[1] \neq \mathsf{FPT}$ is:

Theorem 3.3 *If k-Clique $\in \mathsf{FPT}$ then ETH is false.*

Proof. Recall in the Max-2-SAT problem, we are given a 2CNF formula F and an integer K we wish to find an assignment that satisfies at least K clauses. Max-2-SAT is NP-complete. We'll break the proof into two parts:

(1) k-Clique $\in \mathsf{FPT} \implies$ Max-2-SAT is in $O(2^{\varepsilon n})$ time, for all $\varepsilon > 0$.

(2) Max-2-SAT is in $O(2^{\varepsilon n})$ time for all $\varepsilon > 0 \implies$ 3-SAT is in $O(2^{\varepsilon n})$ time for all $\varepsilon > 0$.

Part (2) follows from standard stuff:

- Given a 3SAT instance, use the sparsification lemma so that the instance has $O(n)$ clauses, WLOG.
- Given a 3SAT instance F with $O(n)$ clauses, the standard proof that Max-2-SAT is NP-complete shows that we can reduce F to a Max-2SAT instance with at most $O(n)$ variables and $O(n)$ clauses.[1]

Part (1) is more interesting. Suppose k-Clique is in $f(k) \cdot n^c$ time. Let F be a 2CNF on n variables with $m \leq O(n^2)$ clauses. We want to solve Max-2-SAT on it. Let k be a parameter to be chosen later.

Partition the n variables of F (arbitrarily) into k groups $G_1, G_2, \ldots, G_k$, where each group has at most $n/k + 1$ variables.

Now we construct a k-Clique instance G. For each group G_i associated with some $n/k + 1$ variables, make a separate node in G for all possible $O(2^{n/k})$ Boolean assignments on these $n/k + 1$ variables, so our graph G will have G has

[1] If you haven't seen this proof before, it's pretty tricky: you replace every clause $(x \vee y \vee z)$ in your 3SAT instance by ten 2CNF clauses (along with a new variable appearing in some of these 10 clauses) such that every satisfying assignment to $(x \vee y \vee z)$ can be extended to satisfy 7 of the 10 clauses, and $x = 0, y = 0, z = 0$ can only be extended to satisfy at most 6 of the 10 clauses. It's just something you stumble across after trial-and-error (or after googling). I'm definitely not giving it as an exercise!

$O(k \cdot 2^{n/k})$ nodes. We'll make G complete k-partite, so it only has edges between distinct pairs of groups G_i and G_j with $i \neq j$.

Now we'll put some positive and negative weights on the edges and nodes (which will be removed later).

- For each node $i = 1, \ldots, k$, we put a weight on node $v \in G_i$ with weight $\ell \iff$ the partial assignment given by v satisfies exactly ℓ clauses.
- For each pair of groups $(i, j) \in [k]$, put an edge between $v_1 \in G_i, v_2 \in G_j$ with weight $-K \iff$ there are exactly K clauses $C_1, \ldots, C_K$ in F such that v_1 satisfies all C_i **and** v_2 also satisfies C_i.
  (Note: this latter condition can only happen if the clause $C_i = (x_j \vee x_k)$ has one variable in group G_i and the other variable in group G_j.)

Claim: There is a k-clique in G with total node and edge weight exactly $m^\star$ if and only if there is an assignment to the variables of F satisfying exactly $m^\star$ clauses.

Exercise: Convince yourself of the claim. The idea is that the node weights count up many satisfied clauses, but the sum of all node weights will overcount those clauses that are satisfied "twice", by variable assignments from two different groups. Those clauses are subtracted from the edge weights.

Here we are exploiting the fact that each clause has at most two variables; if clauses had three variables we'd have to worry about a clause's variables appearing among three different groups.

So far, we have reduced Max-2-SAT to finding a k-clique of maximum node and edge weight in a graph with $O(k2^{n/k})$ nodes. To get rid of the weights, we can simply "brute force" all possible values for the weights!

Suppose we want to know if F has a satisfying assignment of weight exactly $m^\star$. We try all $O(m)^{k+\binom{k}{2}}$ possible ways to assign weights in $\{0, 1, \ldots, m\}$ to k nodes, and weights in $\{0, -1, \ldots, -m\}$ to $\binom{k}{2}$ edges, such that the sum of weights equals $m^\star$. Think of it as going over all vectors $w = (W_1, \ldots, W_k, W_{1,2}, \ldots, W_{k-1,k})$ such that $\sum_i W_i - \sum_{i,j} W_{i,j} = m^\star$.

Make an unweighted graph G^w which only includes nodes and edges that obey these node and edge assignments: we only include node v_i in G_i if the weight of v is exactly W_i, and only include edge $(v_i, v_j) \in G_i \times G_j$ if it has weight exactly $-W_{i,j}$. Observe that there is a k-clique in G with total node and edge weight exactly $m^\star$ if and only if there is a w such that the unweighted graph G^w has a k-clique.

So our final algorithm for Max-2-SAT is as follows: for decreasing $m^\star = m, m-1, \ldots$ and for all vectors w as described above, we run our FPT algorithm for k-clique on G^w, in time $f(k) \cdot N^c \leq f(k) \cdot 2^{cn/k}$. As soon as we find a k-clique, we know that $m^\star$ is the maximum number of clauses in F that can be simultaneously satisfied.

The total running time of our algorithm is $m^{O(k^2)} \cdot f(k) \cdot 2^{cn/k}$. Setting k to be an arbitrarily large constant, this runtime will be $O(2^{\varepsilon n})$ for any desired $\varepsilon > 0$. □

In fact this proof also shows a stronger statement: if k-Clique is in $n^{k/\log\log\log\log k}$ time, then ETH is false. (We could substitute $\log\log\log\log k$ with any slowly growing unbounded function of k. As a result, $\mathsf{W}[1] \neq \mathsf{FPT}$ is widely believed.

Exercise: The clique problem and the vertex cover problem are both NP-complete, yet there seems to be a difference between them in the parameterized world: k-Clique is $\mathsf{W}[1]$-hard (every $\mathsf{W}[1]$ problem can be reduced to it) while k-vertex cover is FPT. How can this be? (Why doesn't the polynomial-time reduction from clique to vertex cover work?)

3.2 Weighted CNF-SAT?

There are other W[1] complete problems, but we won't cover them here. (An example: k-SUM with numbers in the range $[-n^{2k}, \ldots, n^{2k}]$ is also in W[1] [ALW14].) Instead, we'll look at problems that don't seem to be in W[1]. What about the Weighted CNF-SAT problem, where we are given a CNF and parameter k, and wish to know if there's a satisfying assignment of weight exactly k. Can we get an FPT reduction from Weighted CNF-SAT to Weighted 3-SAT?

We of course have a polynomial time reduction from Weighted CNF-SAT to weighted 3-SAT in the usual NP-completeness sense. However, this reduction works by introducing a new set of variables: for each clause C of the CNF-SAT instance $C = (x_1 \vee x_2 \vee \cdots \vee x_\ell)$, we replace it with a series of clauses (each of size 3):

$$(x_1 \ \vee \ x_2 \vee z_1), (\overline{z_1} \ \vee \ x_3 \vee z_2), (\overline{z_2} \ \vee \ x_4 \vee z_3), \ldots, (\overline{z_{\ell-3}} \vee x_{\ell-1} \vee x_\ell).$$

If the original CNF had a weight k assignment that set all clauses to true, then the new instance has $\Omega(\ell)$ variables set to true: the new is not just a function of k, so this reduction is not FPT with respect to the parameter k.

Of course, the above reasoning does not rule out an FPT reduction from Weighted CNF-SAT to Weighted 3-SAT: it just says that the standard reduction from CNF-SAT to 3-SAT does not provide an FPT reduction. There could in fact be an FPT reduction from Weighted CNF-SAT to Weighted 3-SAT—we will see later that this would imply that W[1] = W[2]. We do not know of any evidence against W[1] = W[2].

4 W[2] (Optional)

Recall the definition of the class W[2]: it is the set of parametric problems that are FPT-reducible to finding a weight-k SAT assignment to circuits with two layers of unbounded fan-in gates, and another layer of gates with fan-in at most 2 at the bottom level. It turns out that the most expressive types of these circuits are when there is a single AND gate at the top, a level of ORs in the middle, and AND as the bottom gates with fan-in 2. Moreover, when you have these two types of layers of unbounded fan-in, you can even get rid of the bottom gates that are restricted to fan-in 2. So, what we are left with is simply CNF-SAT.

Theorem 4.1 *The class* W[2] *is the class of problems that are FPT-reducible to Weighted CNF-SAT. In other words, Weighted CNF-SAT is* W[2]*-hard.*

Note that W[1] deals with problems having "bounded-width" constraints that are more like 3-SAT or 4-SAT, whereas W[2] deals with problems that have "unbounded-width" constraints. There do not seem to be many/any natural problems known to be complete for classes higher than W[2]. (If you find such a problem, let us know!) But for W[2], there is a natural complete problem (besides Weighted CNF-SAT) that does not seem to be in W[1]. In the k-Dominating Set problem, we are given a graph G and a parameter k, and we want to know if G contains a dominating set of size at most k. (This is a node set S such that every other node in the graph has an edge coming from S.)

Theorem 4.2 *k-Dominating Set is in* W[2].

Proof. Given a graph $G = (V, E)$, define the following formula over variables $\{x_u \mid u \in V\}$:

$$F = \bigwedge_{v \in V} \left(x_v \vee \bigvee_{u:(u,v)\in E} x_u \right).$$

Observe there is a weight-k SAT assignment to F iff there is a k-dominating set in G. □

Theorem 4.3 *There is an FPT reduction from Weighted CNF-SAT to k-Dominating Set. In other words, k-Dominating Set is* W[2]*-hard.*

Proof. The proof uses color-coding. Let $\{x_1, \ldots, x_n\}$ be the variables of the CNF-SAT instance F. Make n nodes, one for each of the variables, and *randomly partition* them into sets $G_1, \ldots, G_k$. (Imagine assigning each variable a random color in $\{1, \ldots, k\}$.) Suppose F has a weight-k SAT assignment A: we know that with probability at least $1/e^k$, this A is "colorful" in the sense that the k variables in A set to true each appear in distinct sets G_i. (As in the k-path algorithm which picks random colorings, we can derandomize this randomized coloring step by using an explicitly computable collection of $O^\star(c^k)$ different hash functions from $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$, in the following.)

For each $i = 1, \ldots, k$, put edges between all the nodes in G_i (forming cliques within each of the G_i) and add a new vertex w_i that has edges to all nodes in G_i (and no other edges). These edges ensure that every dominating set of size k must contain exactly one node from each G_i (or possibly a node w_i instead).

Next we make m extra nodes, one for each of the m clauses $C_1, \ldots, C_m$ of F. We put an edge from the node associated with variable x_j to a node associated with clause C_i if and only if either x_j appears positively in C_i or there is another variable in the same partition as x_j which appears negatively in C_i. (That is, we put an edge from variable x_j to clause C_i if and only if the partial assignment corresponding to "$x_j := 1$ and all other variables with the same color as x_j are set to 0" satisfies C_i.)

Now, every k-dominating set must choose exactly one vertex from each partition, and a weight-k satisfying assignment to all clauses is obtained by setting the variables associated with nodes in the dominating set to true, and all other variables to false. Conversely, a "colorful" weight-k satisfying assignment A to F corresponds to a dominating set where we choose the k variables x_i set to true in A as the nodes of our dominating set. □

4.1 k-Dominating Set and SETH

How quickly can k-Dominating Set be solved? The fastest known algorithms only achieve $n^{k+o(1)}$ for sufficiently large k [EG04]. Note this is a slower algorithm than what is known for the W[1]-complete problem k-Clique, where we know how to solve it in time about $n^{0.8k}$. Could k-Dominating Set be solved in a similar way? It turns out that would refute SETH!

Recall that SETH says: For all $\delta \in (0, 1)$, there exists $k \geq 3$ such that k-SAT requires $\geq 2^{\delta n}$ time.

Theorem 4.4 ([PW10]) *If for some constant $k \geq 3$ and some constant $e > 0$, k-dominating set is in $O(n^{k-\varepsilon})$ time, then SETH is false.*

We will give a fine-grained reduction from SAT to k-Dominating Set, which will be a bit similar to our reduction of Weighted CNF-SAT to k-Dominating Set.

Proof. Let F be a CNF formula, and let V be the set of variables. Instead of randomly partitioning V as above, we simply arbitrarily partition V into k parts $V_1, \ldots, V_k$ with at most $\frac{n}{k} + 1$ variables each. We create an instance of k-dominating set. Make a node for *every partial assignment* to the variables in V_i, so there are $2^{n/k}$ different nodes for each V_i, letting S_i be the set of all partial assignment nodes for V_i. Add an extra node w_i for each part V_i, and make a clique out of $S_i \cup \{q_i\}$. As before, these cliques will force us to pick one node from each set for a k-dominating set (except now, our set of possible assignments is different). As before, we add nodes $c_1, \ldots, c_m$ for each of the m clauses, which are connected to a node for a partial assignment A_i (on variables from V_i) iff the assignment A_i satisfies the clause C_j (at least one literal in C_j is set true by some assigned variable in A_i). Call this graph G

> **Exercise:** Prove that, if A is a satisfying assignment to F, then $A = \{A_1 \cup \cdots \cup A_k\}$ (where $A_i \in S_i$ is the restriction of A to variables in V_i) is a k-dominating set in G.

Conversely, we claim that if Gh has a k-dominating set, then F is satisfiable. To see this, let D be the k-dominating set. D must contain a node from $S_i \cup \{w_i\}$ for all i, so D contains either some $A_i \in S_i$ or w_i. If $w_i \in D$, then swap w_i with any $A_i \in S_i$. Taking the union of all partial assignments A_i, you get a satisfying assignment: each "clause

node" c_j in the graph must be dominated, meaning that some partial assignment node in D satisfies the clause C_j. Therefore, if all clause nodes are dominated, then all clauses are satisfied by the partial assignments $\{A_i\}$.

Finally, suppose we have an $O(n^{k-\varepsilon})$ algorithm for k-dominating set. Our graph G has $m + k2^{n/k} + k$ nodes. Therefore CNF-SAT can be solved in $(m + k2^{n/k} + k)^{k-\varepsilon} \leq m^{k-\varepsilon}O(k2^{n/k})^{k-\varepsilon} \leq 2^{n(1-\varepsilon/k)} \cdot \text{poly}(m)$ time, so SETH is false! □

References

[ALW14] Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2014.

[BDH18] Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164. ACM, 2018.

[BHKK17] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017.

[CT20] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. *Electron. Colloquium Comput. Complex.*, 27:148, 2020.

[EG04] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

[PW10] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010.

[Tsu19] Dekel Tsur. Faster deterministic parameterized algorithm for *k*-path. *Theor. Comput. Sci.*, 790:96–104, 2019.

[Wil09] Ryan Williams. Finding paths of length k in $o^\star(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.