Fixed Parameter Algorithms

Given a problem $P$, and some parameter $k$, the problem is FPT with parameter $k$ if it can be solved in time $f(k) \cdot \text{poly}(n)$ where $n$ is the instance size.

So if $n$ is graph size, $k$ is the vertex cover size then

- $n^k$ is not FPT
- $f(k) \cdot n^{100}$ is FPT
- $	ext{poly}(n, k)$ is definitely FPT.

Really a function of the parameter. E.g., graph coloring with parameter $k$, color set size is not believed to be solvable in time $n^k$ even. Clique or max clique size $k$ parameter, then it is believed that $2^k$ is close is best, but maybe some other parameter, we can do better. E.g., with parameter $k$ max degree $\Delta$, there is a $(2^k \cdot \Delta^2 n)$ time algo for clique.

OK. Several techniques. Also a lot of crossover with exact (fast) algorithms for NP-hard problems. Will see some of them in these 2 lectures.

1. Kernelization: apply simple "reduction" rules to get an instance that is of size $g(k)$, on which we can use a brute force algo to get poly(n, k) algo.

   - If $g(k) = \text{poly}(k)$ this is called kernelization.

   Notation: reduction rule gives $(I, k) \rightarrow (I', k')$ and is "safe" or "sound".

   - If $(I, k)$ is a YES instance $\Rightarrow (I', k')$ is a YES instance.

   denoted by

   $\text{K}(k = \text{oPT VEX COVER})$

Let's take a simple example. Vertex Cover. There are two simple reduction rules.

1. If $v$ is violated vertex, drop it: not in any VC.

   $(G, k) \Rightarrow (G - \text{violated vertex } v, k)$.

2. If $v$ has degree $> k$ then $v$ is in every VC.

   $(G, k) \Rightarrow (G - v, k - 1)$. 
This means: When no more reduction rules apply, graph is of min-degree 1 and max degree $\Delta = K$.

**Fact:** A vertex cover of such a graph can cover at most $K^2$ edges. (if size $K$)

So at end: if $G_{\text{final}}$, has more than $(K_{\text{final}})^2$ edges, $G = G_{\text{matchy}}$ with $K_{\text{matchy}}$ edges.
else $G = G_{\text{final}}$.

Follows that $(G, K) \iff (G', K_{\text{final}})$.
and $G'$ has at most $(K_{\text{final}})^2$ edges
and hence $\leq k^2$ vertices.

**Thm:** VC has a kernel of size $\leq 2k^2$ vertices ($k^2$ edges).

---

Can we do better? **Yes:** (and no)

**Thm:** VC has kernel of size $\leq 2k$ vertices.

**Pf:** Write LP for VC.

\[
\begin{align*}
\text{min } & \sum x_v \\
\text{st } & x_{uv} \geq 1 \quad \forall u, v \\
& x_{uv} \geq 0.
\end{align*}
\]

Thus [Newhauser-Porter] Any basic feasible solution to this LP has

(a) $x_v \in \{0, 1, 2, 3\}$ for all vertices
And if $x^*$ is optimal fractional solution
\[
V_0 = \{v : x_v = 3\}
\]
\[
V_2 = \{v : x_v = 2\}
\]
\[
V_1 = \{v : x_v = 1\}
\]

Then any optimal solution $x^*$ st $\sum x_{uv} \leq 2$.

**Note:** $\text{OPT} \leq 10^{\text{OPT}} \Rightarrow |V_0| \leq 20\text{OPT} = 2K$.

So reduction rule: drop $V_0$, pick $V_1$. $(G, K) \iff (G[V \cup V_1], K - 141)$. And size of new instance $\leq 2K$ vertices!
Kernelization is a good way to do things, and it's complete in some sense.

Thus, problem \( H \) with parameter \( k \) in \( FPT \) \( \Rightarrow \) it has a \( \text{polynomial-kernel} \).

**Proof:** one direction is immediate. other side: see it has \( f(k) n^c \) also, then

- when instance size \( \leq f(k) \) then output instance.
- else size \( \geq f(k) \) then run algorithm taking time \( f(k) n^c \leq n^{4k} \).

and output a trivial instance depending upon answer.

But that is not an interesting thing. We like poly size kernels. And that's the worst goal to aim for. But let's look at other techniques.

---

Aside: Sunflowers lemma \((\text{Erdos-Rado})\).

Minimize d-uniform hypergraph vertex cover (aka \( d \)-hitting set)
has kernel with at most \( d \cdot k^d \) sets.

---

### II. Bounding the Branching Factor in Exhaustive Search.

- **Observation:** in each solution, either a node is in \( O(k) \) or all its neighbors are in \( O(k) \).

  \[ \text{So } \text{alg}(G,k) = \min \left( \text{alg}(G-v, k-1), \text{alg}(G-N(v)-v, k-|N(v)|) \right) \]

And when max degree of nodes = 1, problem is trivial (graph is a matching).

Also, branch by picking highest degree vertex (degree \( \geq 2 \))

\[ T(k) \leq T(k-1) + T(k-2) + O(m) \]

\[ \text{if } |N(v)| \geq 2. \]

Solve to get \( T(k) \leq O(m) \cdot \phi^k \leq 1.618^k \) \ldots golden ratio.

- **Easy improvement:** what if graph has degree \( \geq 2 \). Paths and cycles. Much easier.

  So can stop when max degree = 2.

\[ T(k) \leq T(k-1) + T(k-2) + O(m) \]

\[ \text{since } T(k) \leq O(m) \cdot (1.46)^k. \]

Can combine with kernelization to get \((\text{time for kernelization}) + O(k) (1.46)^k\).
Branch-factor was studied for only SAT-problems exactly (DPLL procedure)

(Davis Putnam Logemann Loveland)

Exact \((1.33)^n\) ago for 2SAT:-

If there 3 variable clause then use 2SAT algo to check satisfiability.
Else \((e_1V e_2V e_3)\):

\[
\begin{align*}
\text{either } e_1 & \text{ is true or } \\
 e_1 \& e_2 & \text{ or } \\
 e_1 \& e_2 \& e_3 & \text{ }
\end{align*}
\]

\[
\text{number of variables } = \frac{T(n)}{T(n-1)+T(n-2)+T(n-3)}+O(nm). \\
\text{Total } = O(n^2). O(mn) \text{ is the real root of } x^2=x^3+x^4+1 \text{ } \\
x = 1.339...
\]

May improvements:

Monroe Speckenweghe '85 - Schöning '94

Patvin, Pudlak, Zane, Williams, etc.

Another (better) one: Suppose \(a^*\) is a satisfying assignment.

Then suppose \(|\{i \in a^* : i \leq n/2\}| \leq n/2\) (else start with the all \(1\) solution)

- start with all zero solution
- while current solution is not satisfying, pick unsat clause
- for each clause, with \(a^*\), for each \(i\) in that clause,

revisit depth \(\leq n/2\)

Total time \(\leq 3^{n/2}O(nm) = 1.732^nO(nm)\) better than above, but not best possible (by any means).

[Schöning '99]: Pick the first clause not sat by \(a^*\), and flip a random variable and repeat.

for depth \(T = O(n)\).

Claim: \(|P| \leq \frac{e^{O(3/4)^n}}{\sqrt{n}} \Rightarrow \text{repeat } O(\sqrt{n})\text{ poly}(n) \text{ times. } = (1.33)^n\)

[PP2] deals with unit clauses: random permutation on variables, random "default" assignment.

If current variable \(i\) is in some unit clause, sat that clause (if no way to sat all unit
Else set current var to the default value,

simplify
and move to next variable in random order. \((1.58)^n\) etc.
Color Coding: A powerful yet simple technique to obtain algos via randomization. Eg. easy algorithm to find paths of length \( k \) in time \( O(k!) \) and \( O(2^k) \). See HWS.

Inclusion Exclusion:

Want to find TSP on a given metric. Trivial soln: \( n! \) ploy \( (n) \).

\[
\text{TSP(ends at } t, \text{visits vertices } U) = \min_x \{ \text{TSP(ends at } x, \text{visits vertices } U \cup \{t\}) + d(x,t) \} \]

Starts at \( s \).

Takes time \((\text{and space})\) \( O(2^n \cdot \text{poly}(n)) \). Actually space \( O(2^n \cdot n) \) time \( 2^n N \times n \) possible lookup.

Suppose now we want to count # of solutions (Hamilton cycles).

Let \( A_u \) be the set of all \( n \)-length walks that pass through vertex \( u \in V \). \( \text{Note that } |\text{HAM}| = \bigcap_{u \in V} |A_u| = \sum_{x \in V} (-1)^x |\bigcap_{u \in x} \overline{A_u}| \) \( (\text{of length } n) \).

\[
\Rightarrow \text{binary problem: } |\bigcap_{u \in x} \overline{A_u}| = \# \text{ closed walks } x \text{ in } (V \setminus x) \text{ (starting from } x) \text{.}
\]

Can be computed using "matrix multiplication" type ideas.

\[
\text{Compute } (A(V \setminus x))^n.
\]

\Rightarrow \text{compute all simpler problems in time } O(n^5 \cdot 2^n)

but now space polynomial!!
Vertex cover: $G$ is a graph. We have $A = \text{approx VC size} \leq 2K$.
- Guess how $C^* \setminus A$ looks like. (call this $G_A$).
- $R_A = \text{rest of } A = A \setminus G_A$.
  - If $G[R_A]$ has an edge, fail for this $G_A$.
  - Else $N(R_A)$ must be all in the $C^*$. (Else some edge not covered).

$\Rightarrow$ if $|N(R_A)| + |G_A| > K$ fail for this $G_A$.

$\Rightarrow$ if $\exists \ J \subset G_A$ s.t $|G_A| + |N(R_A)| \leq K$, this is a VC of size $K$.
else fail.

Time: $O^*(2^K)$.

Much worse than previous ideas, can we improve? Yes. Iterative compression.

Let $G_i = G[V_i, \ldots, V_i]$.

Sps we have a VC of $G_i$, called $A_i$, of size $K$.
Then get a VC $A_{i+1}$ of $G_{i+1}$ of size $K_{i+1}$.

Now use the algo from previous part to get VC $C_{i+1}$ of size $K_{i+1}$.
In time $O^*(2|A_{i+1}|) = O^*(2^{K_{i+1}})$.

$\Rightarrow$ whole thing takes $O^*(2^K)$ time! 😊.

Loss is a factor $n^2$ in the runtime.

BTW: very general, but why did it work? We had a soln $A$, we said that we wanted to extend $G_A$ to another soln of size $K$, but it had to be disjoint from $R = A \setminus G_A$.
This gave a lot of structure, made it trivial in the VC case. But this disjointness requirement makes it easy in other cases too.
Longest path in $O(k^k)$ time

Want a polynomial $P(x) \neq 0$ iff $G$ has a $k$-path.

For every $k$-walk $v_1 = v, v_2, \ldots, v_k = v'$ in the graph, we have a monomial $\prod_{i=1}^{k-1} x_{v_i, v_{i+1}} \prod_{i=1}^{k} y_{v_i, v'}$ for edges, $\sum_{v_i}$ for vertices.

Note: for a path, the monomial is multilinear (each variable has degree 1)

So instead of checking for existence of some monomial, want to check existence of multilinear monomial.

Idea: Label vertices with a unique label. Now define polynomial over labeled walks.

$W = \text{walk}$

$l = \text{labeling } [kJ] \to [kJ] \text{ bijection.}$

$\forall e = (u, v) \in E, \ x_{uv}$

$\forall v \in V, \ \forall \alpha \in [kJ], \ y_{v, \alpha}$

$P(\bar{x}, \bar{y}) = \sum \sum_{\text{walks } W} \prod_{i=1}^{k-1} x_{v_i, v_{i+1}} \prod_{i=1}^{k} y_{v_i, v'}, l(\alpha).$

$I \in \{1 \ldots k\}, \ \text{Vars } = (v_1 \ldots v_k) \text{ bijection } \equiv \text{def.}$

$= \sum_{W} \sum_{l} \text{MONO}^k_{W, \alpha} (\bar{x}, \bar{y})$

View as a multivariate polynomial over $GF(2^{16} \times 2^k)$.

\[ P(\bar{x}, \bar{y}) = \sum \sum_{\text{paths } P} \text{MONO}^k_{P, \alpha} (\bar{x}, \bar{y}). \]

Main Lemma:\n
**note:** not sum over walks, but over paths.

We will see its proof soon, but for now let's use it.

*Crucial Fact*:

$GF(2^k)$ has characteristic 2 (i.e., $a + a = 0$ for $a \in GF(2^k)$)

*has a correspondence with t-bit vectors, but the operations are based on irreducible polynomials.*

See algebraic code.
Note that for two different paths $P, P'$ \(\text{MONO}(x,y)\) cannot cancel \(\text{MONO}_{P,P'}(x,y)\) for any $x, y$. (Since the paths are just different).

\[\Rightarrow\text{Corollary: } P \text{ is non-zero polynomial } \iff \exists \text{ a k-path in } G.\]

How to compute $P$ at $(x_i, y)$ efficiently?

**Evaluation lemma:** Given assignments to $x_i, y$, $P(x)$ can be calculated in time $O(2^k \cdot km)$ field operations.

**Thm:** An algorithm for LONGEST path in time $O^*(2^k)$.

**Pf:** Want to check if $P(x)$ is non-zero polynomial.
(by Corollary $\Rightarrow G$ has a k-path).
Use Schwartz-Zippel, evaluation lemma, says takes time $O^*(2^k)$ to evaluate it at random points. And if $P(x) \neq 0$ then with this will give non-zero value.

Now to prove the Lemmas:

1. **Proof of Main Lemma:** Let $W$ be a walk (non-path) and $\mathcal{E}$ bijection.
   Say $v$ is repeated, the first one. Flip the labels of these two locations, to get $\mathcal{E}'$, make $\text{MONO}_{W} = \text{MONO}_{W'}$. Hence they'll cancel each other out (by $\mathcal{E}$. $GF(2^k)$ has characteristic 2).

2. **Proof of Evaluation Lemma:**
Claim 1: \( w(W, x, y) \)

\[
\sum_{w, e} \text{Mono}(x, y) = \sum_{x \in [2]} \sum_{e: \text{maps from } [x] \to x} \text{Mono}(x, y)
\]

**Proof:** Let \( U \) be a set of all maps from \([2] \) to \([2] \).

\( A_i = \text{set of maps that map at least one thing to } i \)

\( w(x, y) = \text{Mono}(x, y) \)

\[
\text{RHS} = w(\cap A_i) = \sum_{x \in [2]} \sum_{e: \text{maps from } [x] \to x} (-1)^x \cdot w(\cap \overline{A}_i)
\]

\[
= \sum_{x \in [2]} (-1)^x \sum_{e: \text{maps from } [x] \to x} \text{Mono}(x, y)
\]

\[
= \sum_{x \in [2]} \sum_{e: \text{maps from } [x] \to x} \text{Mono}(x, y).
\]

\[
\Rightarrow P(x, y) = \sum_{\text{walk } W} \sum_{x \in [2]} \sum_{e: \text{maps from } [x] \to x} \text{Mono}(x, y).
\]

\[
= \sum_{x \in [2]} P(x, y) = 2^k \text{ polynomials of the form } \sum_{\text{walks } e} \sum_{x \in [2]} \text{Mono}(x, y).
\]

Claim 2: \( P(x, y) \) can be computed in time \( \text{poly}(n, k) \).

**Proof:** Just a simple DP on the labels of the walk.

Length 1: \[
\sum_{y, \ell(\cdot)} \text{labels from } [x] \to x
\]

\[
M(y, i) = \sum_{x \in [2]} \text{labels for } k \cdot \ell(i, x).
\]

Time: \( O(2^k n) \).