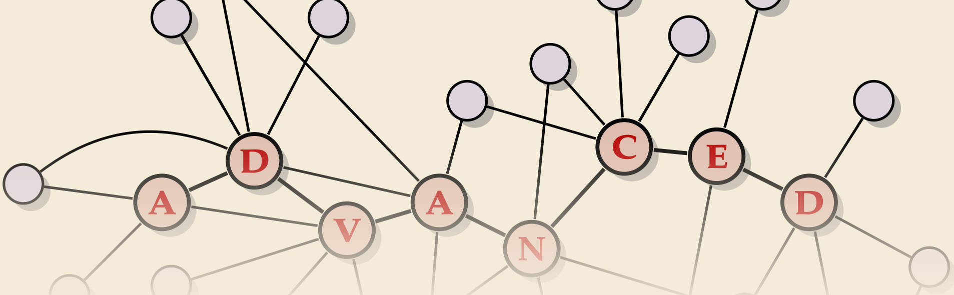


4 Fixed-Parameter Algorithms

14 May 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-05-26 23:23`

Outline

4 Fixed-Parameter Algorithms

Philosophy of FPT

- **Goal:** Principled theory for studying complexity based on two dimensions:
  input size $n = |x|$ (encoding length) and *some additional parameter k*
  - generalize ideas from $k = MaxInt(x)$
  - investigate influence of k (and n) on running time

Philosophy of FPT

- **Goal:** Principled theory for studying complexity based on two dimensions: input size $n = |x|$ (encoding length) and *some additional parameter k*
  - generalize ideas from $k = MaxInt(x)$
  - investigate influence of k (and n) on running time

$\rightsquigarrow$ Try to find a parameter k such that
(1) the problem can be solved efficiently as long as k is small, and
(2) practical instances have small values of k (even where n gets big).

Motivation: Satisfiability

Consider Satisfiability of CNF formula *the drosophila melanogaster of complexity theory*

- general worst case: NP-complete
- k = #literals per clause
  - $k \leq 2 \rightsquigarrow$ in P   2SAT
  - $k \geq 3$ NP-complete

$a \to b \equiv \neg a \vee b$

$x_i \vee \neg x_j \equiv x_j \to x_i$

$\equiv \neg x_i \to \neg x_j$

Motivation: Satisfiability

Consider Satisfiability of CNF formula *the drosophila melanogaster of complexity theory*

- general worst case: NP-complete
- k = #literals per clause
  - $k \leq 2 \rightsquigarrow$ in P
  - $k \geq 3$ NP-complete
- k = #variables
  - $O(2^k \cdot n)$ time possible (try all assignments)
- k = #clauses?
- k = #literals?
- k = #ones in satisfying assignment
- k = structural property of formula
- for Max-SAT, k = #optimal clauses to satisfy

Parameters

Definition 4.1 (Parameterization)

Let Σ a (finite) alphabet. A *parameterization* (of $\Sigma^\star$) is a mapping $\kappa : \Sigma^\star \longrightarrow \mathbb{N}$ that is polytime computable. ◀

Definition 4.2 (Parameterized problem)

A *parameterized (decision) problem* is a pair (L, κ) of a language $L \subset \Sigma^\star$ and a parameterization κ of $\Sigma^\star$. ◀

Definition 4.3 (Canonical Parameterizations)

We can often specify a parameterized problem conveniently as a language of *pairs* $L \subset \Sigma^\star \times \mathbb{N}$ with

$$(x, k) \in L \ \wedge \ (x, k') \in L \quad \rightarrow \quad k = k'$$

using the *canonical parameterization* $\kappa(x, k) = k$. ◀

Examples

As before: Typically leave encoding implicit.

Definition 4.4 (p-variables-SAT)

Given: formula boolean ϕ (same as before)
Parameter: number of variables
Question: Is there a satisfying assignment $v : [n] \to \{0,1\}$? ◀

Definition 4.5 (p-Clique)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$
Parameter: k
Question: $\exists V' \subset V \quad : \quad |V'| \geq k \ \wedge \ \forall u, v \in V' : \{u, v\} \in E$? ◀

Canonical Parameterization

Definition 4.6 (Canonically Parameterized Optimization Problems)

Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ be an optimization problem.
Then p-U denotes the *(canonically) parameterized (decision) problem* given by the threshold problem $Lang_U$. ◀

Recall: $Lang_U$ is the set of pairs (x, k) of all instances $x \in L_I$ that have solutions that are weakly "better" than k.

Canonical Parameterization

Definition 4.6 (Canonically Parameterized Optimization Problems)

Let $U = (\Sigma_I, \Sigma_O, L, L_I, M, cost, goal)$ be an optimization problem.
Then p-U denotes the *(canonically) parameterized (decision) problem* given by the threshold problem $Lang_U$. ◀

Recall: $Lang_U$ is the set of pairs (x, k) of all instances $x \in L_I$ that have solutions that are weakly "better" than k.

Examples:

- p-Clique
- p-Vertex-Cover
- p-Graph-Coloring
- ...

Naming convention for other parameters:
p-*clause*-CNF-SAT: CNF-SAT with parameter "number of *clauses*"

4.1 Fixed-Parameter Tractability

Exemplary Running Times of Parameterized Problems

- p-*variables*-SAT (consider simplest brute-force methods for problems)
  - k variables, n length of formula
  - $\rightsquigarrow$ $O(2^k \cdot n)$ running time

Exemplary Running Times of Parameterized Problems

(consider simplest brute-force methods for problems)

- *p-variables*-SAT
  - k variables, n length of formula
  - $\rightsquigarrow$ $O(2^k \cdot n)$ running time
- p-Clique
  - k threshold (clique size); n vertices, m edges in graph
  - $\rightsquigarrow$ $\binom{n}{k}$ candidates to check, each takes time $O(k^2)$ to check
  - $\rightsquigarrow$ Total time $O(n^k \cdot k^2)$

$$\binom{n}{k} = \frac{\overbrace{n(n-1)(n-2)\cdots(n-k+1)}^{k}}{k!}$$

$$\sim \frac{n^k}{k!}$$

Exemplary Running Times of Parameterized Problems

(consider simplest brute-force methods for problems)

- p-*variables*-SAT
  - k variables, n length of formula
  - $\rightsquigarrow$ $O(2^k \cdot n)$ running time
- p-Clique
  - k threshold (clique size); n vertices, m edges in graph
  - $\rightsquigarrow$ $\binom{n}{k}$ candidates to check, each takes time $O(k^2)$ to check
  - $\rightsquigarrow$ Total time $O(n^k \cdot k^2)$
- p-VertexCover
  - k threshold (VC size); n vertices, m edges in graph
  - $\rightsquigarrow$ $\binom{n}{k}$ candidates to check, each takes time $O(m)$ to check
  - $\rightsquigarrow$ Total time $O(n^k \cdot m)$

Exemplary Running Times of Parameterized Problems

(consider simplest brute-force methods for problems)

- p-*variables*-SAT
  - k variables, n length of formula
  - $\leadsto$ $O(2^k \cdot n)$ running time

$k = O(\log n)$ $\leadsto$ poly time

- p-Clique
  - k threshold (clique size); n vertices, m edges in graph
  - $\leadsto$ $\binom{n}{k}$ candidates to check, each takes time $O(k^2)$ to check
  - $\leadsto$ Total time $O(n^k \cdot k^2)$
- p-VertexCover
  - k threshold (VC size); n vertices, m edges in graph
  - $\leadsto$ $\binom{n}{k}$ candidates to check, each takes time $O(m)$ to check
  - $\leadsto$ Total time $O(n^k \cdot m)$

$k = O(1)$ $\leadsto$ poly time

- p-GraphColoring
  - k threshold (#colors); n vertices, m edges in graph
  - $\leadsto$ k^n candidates to check, each takes time $O(m)$
  - $\leadsto$ Total time $O(k^n \cdot m)$

$k = 1$ $\leadsto$ polynomial

$k = 3$ NP-hard

FPT Running Time

Definition 4.7 (fpt-algorithm)

Let κ be a parameterization for $\Sigma^\star$.
A (deterministic) algorithm A (with input alphabet Σ) is a *fixed-parameter tractable algorithm (fpt-algorithm)* w.r.t. κ if its running time on $x \in \Sigma^\star$ with $\kappa(x) = k$ is at most

only depends on k

polynomial

$$f(k) \cdot p(|x|) = O\big(f(k) \cdot |x|^c\big)$$

where p is a polynomial of degree c and f is an **arbitrary** computable function. ◀

FPT Running Time

Definition 4.7 (fpt-algorithm)

Let κ be a parameterization for $\Sigma^\star$.
A (deterministic) algorithm A (with input alphabet Σ) is a *fixed-parameter tractable algorithm (fpt-algorithm)* w.r.t. κ if its running time on $x \in \Sigma^\star$ with $\kappa(x) = k$ is at most

$$f(k) \cdot p(|x|) \quad = \quad O\big(f(k) \cdot |x|^c\big)$$

where p is a polynomial of degree c and f is an **arbitrary** computable function. ◀

Definition 4.8 (FPT)

A parameterized problem (L, κ) is *fixed-parameter tractable* if there is an fpt-algorithm that decides it.
The complexity class of all such problems is denoted by FPT. ◀

Intuitively, FPT plays the role of P.

A First FPT Example

Theorem 4.9 (p-variables-SAT is FPT)

p-variables-SAT $\in$ FPT.

A First FPT Example

Theorem 4.9 (p-variables-SAT is FPT)

p-variables-SAT $\in$ FPT.

Proof:

Suffices to use brute force satisfiability for *p-variables*-SAT

```
procedure bruteForceSat(φ, 𝒳 = {x_1, ..., x_k})
    if k == 0
        if φ == true return ∅ else UNSATISFIABLE
    for value in {true, false} do
        A := {x_1 ↦ value}
        ψ := φ[x_1/value] // Substitute value for x_1
        B := bruteForceSat(ψ, {x_2, ..., x_k})
        if B ≠ UNSATISFIABLE
            return A ∪ B
```

. . . but #variables not usually small

A First FPT Example

Theorem 4.9 (p-variables-SAT is FPT)

p-variables-SAT $\in$ FPT.

Proof:

Suffices to use brute force satisfiability for *p-variables*-SAT

```
procedure bruteForceSat(φ, 𝒳 = {x_1, ..., x_k})
    if k == 0
        if φ == true return ∅ else UNSATISFIABLE
    for value in {true, false} do
        A := {x_1 ↦ value}
        ψ := φ[x_1/value] // Substitute value for x_1
        B := bruteForceSat(ψ, {x_2, ..., x_k})
        if B ≠ UNSATISFIABLE
            return A ∪ B
```

$f(k)$

Worst case running time: $O(2^k n)$ for $n = |\varphi|$.

2^k recursive calls;

base case needs time $O(|\phi|)$ to check whether formula evaluates to *true* ∎

A First FPT Example

Theorem 4.9 (p-variables-SAT is FPT)

p-variables-SAT $\in$ FPT.

Proof:

Suffices to use brute force satisfiability for *p-variables*-SAT

```
procedure bruteForceSat(φ, 𝒳 = {x_1, ..., x_k})
    if k == 0
        if φ == true return ∅ else UNSATISFIABLE
    for value in {true, false} do
        A := {x_1 ↦ value}
        ψ := φ[x_1/value] // Substitute value for x_1
        B := bruteForceSat(ψ, {x_2, ..., x_k})
        if B ≠ UNSATISFIABLE
            return A ∪ B
```

Worst case running time: $O(2^k\, n)$ for $n = |\varphi|$.

2^k recursive calls;

base case needs time $O(|\phi|)$ to check whether formula evaluates to *true* ∎

. . . but #variables not usually small

Aren't we all FPT?

Theorem 4.10 (k never decreases → FPT)

Let $g : \mathbb{N} \to \mathbb{N}$ weakly increasing, unbounded and computable, and κ a parameterization with

$$\forall x \in \Sigma^\star : \kappa(x) \geq g(|x|).$$

Then $(L, \kappa) \in \mathsf{FPT}$ for *any* decidable L.

$g(x) = \log \log |x|$
possible

g weakly increasing: $\quad n \leq m \ \to\ g(n) \leq g(m)$
g unbounded: $\quad \forall t\ \exists n\ :\ g(n) \geq t$

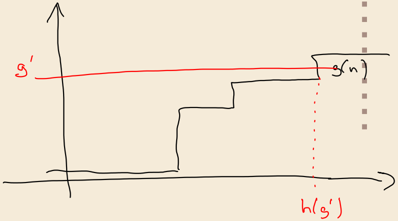
Proof: L decidable $\leadsto$ $\exists$ algorithm to decide L in time $\leq T(|x|)$

w.l.o.g. T weakly increasing

$T(|x|) \geq |x|$

Idea: "hide" $T(|x|)$ in $f(k)$

("g'") $h(g') = \max\{n' \in \mathbb{N} : g(n') \leq g'\} \cup \{1\}$



Aren't we all FPT? – Proof

Proof (cont.):

(1) g weakly incr. & unbounded $\Rightarrow$ h well-defined

(2) h weakly increasing

(3) g computable $\Rightarrow$ h computable

(4) $h(g(n)) \geq n$

time to decide whether $x \in \Sigma^*$ is in L

$n = |x|$

$k = \kappa(x) \geq g(n)$

$$\leq T(n) \underset{\substack{T \text{ incr.} \\ (4)}}{\leq} T(h(g(n))) \underset{T,\, h \text{ incr.}}{\leq} T(h(k)) =: f(k)$$

■

Back to “sensible” parameters

$\rightsquigarrow$ always check if parameter is reasonable (can be expected to be small)

- if not, FPT might not even mean in NP!

Back to “sensible” parameters

- ⇝ always check if parameter is reasonable (can be expected to be small)
  - if not, FPT might not even mean in NP!
- but now, for some positive examples!

4.2 Depth-Bounded Exhaustive Search I

FPT Design Pattern

- The simplest FPT algorithms use exhaustive search
- but with a search tree bounded by $f(k)$

FPT Design Pattern

- The simplest FPT algorithms use exhaustive search
- but with a search tree bounded by $f(k)$

- bruteforceSat was a typical example!
- does this work on other problems?

Depth-Bounded Search for Vertex Cover

Let's try p-VERTEXCOVER. brute force $\binom{n}{k} \cdot \text{poly}(n) = \Theta(n^k \text{poly}(n)) \neq$ fpt running time

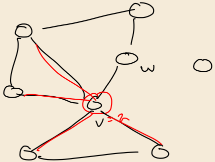
Key insight: for every edge $\{v, w\}$, any vertex cover must contain v or w

Depth-Bounded Search for Vertex Cover

Let's try p-VertexCover.

Key insight: for every edge $\{v, w\}$, any vertex cover must contain v or w

```
procedure simpleFptVertexCover(G = (V, E), k):
    if E == ∅ then return ∅
    if k == 0 then return NOT_POSSIBLE   // truncate search
    Choose {v, w} ∈ E (arbitrarily)
    for u in {v, w} do:
        G_u := (V \ {u}, E \ {{u, x} ∈ E})   // Remove u from G
        C_u := simpleFptVertexCover(G_u, k − 1)
    if C_v == NOT_POSSIBLE then return C_w ∪ {w}
    if C_w == NOT_POSSIBLE then return C_v ∪ {v}
    if |C_v| ≤ |C_w| then return C_v ∪ {v} else return C_w ∪ {w}
```



Depth-Bounded Search for Vertex Cover

Let's try p-VertexCover.

Key insight: for every edge $\{v, w\}$, any vertex cover must contain v or w

```
procedure simpleFptVertexCover(G = (V, E), k):
    if E == ∅ then return ∅
    if k == 0 then return NOT_POSSIBLE   // truncate search
    Choose {v, w} ∈ E (arbitrarily)
    for u in {v, w} do:
        G_u := (V \ {u}, E \ {{u, x} ∈ E})   // Remove u from G
        C_u := simpleFptVertexCover(G_u, k − 1)
    if C_v == NOT_POSSIBLE then return C_w ∪ {w}
    if C_w == NOT_POSSIBLE then return C_v ∪ {v}
    if |C_v| ≤ |C_w| then return C_v ∪ {v} else return C_w ∪ {w}
```

- Does not need explicit checks of solution candidates!
- runs in time $O\big(2^k(n+m)\big)$ $\rightsquigarrow$ fpt-algorithm for p-Vertex-Cover $\in$ FPT

Guessing the parameter

- Note: Previous algorithm only uses k to *truncate* branches.

$\rightsquigarrow$ We can *guess* a k and it still works

Guessing the parameter

- Note: Previous algorithm only uses k to *truncate* branches.

⇝ We can *guess* a k and it still works

⇝ Try all k!

```
procedure vertexCoverBfs(G = (V, E))
    for k := 0, 1, ..., |V| do
        C := simpleFptVertexCover(G, k)
        if C ≠ NOT_POSSIBLE return C
```

Guessing the parameter

- Note: Previous algorithm only uses k to *truncate* branches.

$\rightsquigarrow$ We can *guess* a k and it still works

$\rightsquigarrow$ Try all k!

```
procedure vertexCoverBfs(G = (V, E))
    for k := 0, 1, ..., |V| do
        C := simpleFptVertexCover(G, k)
        if C ≠ NOT_POSSIBLE return C
```

- Running time: $\displaystyle\sum_{k'=0}^{k} O(2^{k'}(n+m)) = O(2^{k}(n+m))$

$\rightsquigarrow$ For exponentially growing cost, trying all values up to k costs only constant factor more

4.3 Problem Kernels

Preprocessing

- Second key fpt technique are *reduction rules*
- **Idea:** Reduce the size of the instance (in polytime) without changing its outcome

Preprocessing

- Second key fpt technique are *reduction rules*
- **Idea:** Reduce the size of the instance (in polytime) without changing its outcome
- Trivial example for SAT:

  $\boxed{\text{If a CNF formula contains a single-literal clause } \{x\} \text{ resp. } \{\neg x\}, \text{ set } x \text{ to } \textit{true} \text{ resp. } \textit{false} \text{ and remove the clause.}}$

  - doesn't do anything in the worst case ...

Preprocessing

- Second key fpt technique are *reduction rules*
- **Idea:** Reduce the size of the instance (in polytime) without changing its outcome

- Trivial example for SAT:

  > If a CNF formula contains a single-literal clause $\{x\}$ resp. $\{\neg x\}$, set x to *true* resp. *false* and remove the clause.

  - doesn't do anything in the worst case . . .
  - special case of resolution calculus rule $\dfrac{a_1 \vee a_2 \vee \cdots \vee x, \quad b_1 \vee b_2 \vee \cdots \vee \neg x}{a_1 \vee a_2 \vee \cdots \vee b_1 \vee b_2 \vee \cdots}$
  - basis of practical SAT solvers

Preprocessing

- Second key fpt technique are *reduction rules*
- **Idea:** Reduce the size of the instance (in polytime)
  without changing its outcome

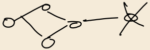
- Trivial example for SAT:

  $\boxed{\text{If a CNF formula contains a single-literal clause } \{x\} \text{ resp. } \{\neg x\}, \text{ set } x \text{ to } \textit{true} \text{ resp. } \textit{false} \text{ and remove the clause.}}$

  - doesn't do anything in the worst case ...
  - special case of resolution calculus rule $$\frac{a_1 \vee a_2 \vee \cdots \vee x, \quad b_1 \vee b_2 \vee \cdots \vee \neg x}{a_1 \vee a_2 \vee \cdots \vee b_1 \vee b_2 \vee \cdots}$$
  - basis of practical SAT solvers

- Trivial example for VertexCover

  $\boxed{\text{Remove vertices of degree } 0 \text{ or } 1.}$



(never needed as part of optimal VC)

Preprocessing

- Second key fpt technique are *reduction rules*
- **Idea:** Reduce the size of the instance (in polytime)
  without changing its outcome
- Trivial example for SAT:

  > If a CNF formula contains a single-literal clause $\{x\}$ resp. $\{\neg x\}$, set x to *true* resp. *false* and remove the clause.

  - doesn't do anything in the worst case . . .
  - special case of resolution calculus rule $\dfrac{a_1 \vee a_2 \vee \cdots \vee x, \quad b_1 \vee b_2 \vee \cdots \vee \neg x}{a_1 \vee a_2 \vee \cdots \vee b_1 \vee b_2 \vee \cdots}$
  - basis of practical SAT solvers
- Trivial example for VertexCover

  > Remove vertices of degree 0 or 1.

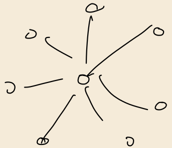
  (never needed as part of optimal VC)
- Here: reduction rules that provably shrink an instance to size $g(k)$

Buss's Reduction Rule for VC

- Given a p-VertexCover instance (G, k)

"deg > k" Rule: If G contains vertex v of degree $\deg(v) > k$, include v in potential solution and remove it from the graph.

- Can apply this simultaneously to degree $> k$ vertices.
- Either rule applies, or all vertices bounded degree(!)



Kernels

Definition 4.11 (Kernelization)

Let (L, κ) be a parameterized problem. A function $K : \Sigma^\star \to \Sigma^\star$ is *kernelization* of L w.r.t. κ if it maps any $x \in L$ to an instance $x' = K(x)$ with $k' = \kappa(x')$ so that

1. (self-reduction) $x \in L \iff x' \in L$
2. (polytime) K is computable in polytime.
3. (kernel-size) $|x'| \le g(k)$ for some computable function g

We call x' the *(problem) kernel* of x and g the *size of the problem kernel*. ◀

Buss's Kernel

Buss's Reduction for Vertex Cover: (repeatedly apply until no more changes)

- $\deg > k$ rule
- Remove degree 0 and 1 vertices

Theorem 4.12 (Buss's Reduction is Kernelization)

Buss' reduction yields a kernelization for p-VERTEX-COVER with kernel size $O(k^2)$. ◀

Buss's Kernel

Buss' rule for deg > k and 0/1 deg.

Buss's Reduction for Vertex Cover: (repeatedly apply until no more changes)

- deg $> k$ rule
- Remove degree 0 and 1 vertices

Theorem 4.12 (Buss's Reduction is Kernelization)

Buss' reduction yields a kernelization for p-VERTEX-COVER with kernel size $O(k^2)$.

Proof:

After repeatedly applying Buss's rule as well as the isolated/leaf rule until neither applies further, we have $\forall v \in V : 2 \leq \deg(v) \leq k$.
(Note that the rule might reduce the parameter k).

Buss's Kernel

Buss's Reduction for Vertex Cover: (repeatedly apply until no more changes)

- $\deg > k$ rule
- Remove degree 0 and 1 vertices

Theorem 4.12 (Buss's Reduction is Kernelization)

Buss' reduction yields a kernelization for p-Vertex-Cover with kernel size $O(k^2)$. ◀

Proof:

After repeatedly applying Buss's rule as well as the isolated/leaf rule until neither applies further, we have $\forall v \in V : 2 \leq \deg(v) \leq k$.
(Note that the rule might reduce the parameter k).
In the resulting graph, any VC of size $\leq k$ covers $\leq k^2$ edges.

Buss's Kernel

Buss's Reduction for Vertex Cover: (repeatedly apply until no more changes)

- $\deg > k$ rule
- Remove degree 0 and 1 vertices

Theorem 4.12 (Buss's Reduction is Kernelization)

Buss' reduction yields a kernelization for p-Vertex-Cover with kernel size $O(k^2)$. ◀

Proof:

After repeatedly applying Buss's rule as well as the isolated/leaf rule until neither applies further, we have $\forall v \in V : 2 \le \deg(v) \le k$.
(Note that the rule might reduce the parameter k).
In the resulting graph, any VC of size $\le k$ covers $\le k^2$ edges.
If $m > k^2$, we output a trivial No-instance (e. g., a K_{k+1} a complete graph on $k + 1$ vertices).

Buss's Kernel

Buss's Reduction for Vertex Cover: (repeatedly apply until no more changes)

- $\deg > k$ rule
- Remove degree 0 and 1 vertices

Theorem 4.12 (Buss's Reduction is Kernelization)
Buss' reduction yields a kernelization for p-VERTEX-COVER with kernel size $O(k^2)$.

Proof:
After repeatedly applying Buss's rule as well as the isolated/leaf rule until neither applies further, we have $\forall v \in V : 2 \le \deg(v) \le k$.
(Note that the rule might reduce the parameter k).
In the resulting graph, any VC of size $\le k$ covers $\le k^2$ edges.
If $m > k^2$, we output a trivial No-instance (e. g., a K_{k+1} a complete graph on $k+1$ vertices).
If $m \le k^2$, then the input size is now bounded by $g(k) = 2k^2$. ∎

FPT iff Kernelization

Theorem 4.13 (FPT ↔ kernel)

A computable, parameterized problem (L, κ) is fixed-parameter tractable if and only if there is a kernelization for L w.r.t. κ.

Proof:

"$\Leftarrow$" kernelization K for (L, κ) given,

L has decider A of running time $T(n)$ (w.l.o.g. weakly increasing)

(1) $x \in \Sigma^*$ to check $x \in L$ $\quad k = \kappa(x) \quad n = |x|$

compute $K(x) = x'$ poly time

$|x'| \leq g(k)$

(2) run A on x'

time $T(|x'|) \underset{\text{incr.}}{\leq} T(g(k)) =: f(k)$

$\Rightarrow$ algorithm for (L, κ) w/ time $O(f(k) + \text{poly}(n))$

FPT iff Kernelization [2]

Proof (cont.):

"$\Rightarrow$" Given fpt-algorithm A for (L, x) with time $\leq f(k) \cdot n^c$

(1) Simulate A for $\leq n^{c+1}$ steps (poly time)

(2) • If A terminated

if output Yes : output trivial Yes-instance

if " No - - - No - "

• otherwise $n^{c+1} \leq f(k) n^c \Rightarrow n \leq f(k)$

$\Rightarrow$ output original input

Max-SAT Kernel

k = # clauses to satisfy

Theorem 4.14 (Kernel for Max-SAT)

p-Max-SAT has a problem kernel of size $O(k^2)$ which can be constructed in linear time. ◀

Proof:

$$(x \vee y \vee \bar{z}) \wedge (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee z)$$

$$\rightsquigarrow \{\{x, y, \bar{z}\}, \{x, y, \bar{z}\}, \{\bar{x}, z\}, \{\bar{y}, z\}\}$$

assumption: each variables occurs at most once per clause
($x \vee \bar{x} \rightsquigarrow$ delete clause)

m = #clauses (n total input size # literals in all clauses)

Case 1: $k \le \lfloor \frac{m}{2} \rfloor$ (output Yes)

pick arbitrary assignment A of all variables

under A, ℓ clauses are satisfied $\ell \ge k$ ✓

if $\ell < k \le \lfloor \frac{m}{2} \rfloor$ $\rightarrow$ then $\bar{A}$ (inverse assignment) satisfies $m - \ell \ge \lfloor \frac{m}{2} \rfloor \ge k$ 21

Max-SAT Kernel [2]

Proof (cont.):

Case $k > \lfloor \frac{m}{2} \rfloor \implies k > \frac{m}{2} \implies \boxed{m < 2k}$

$\implies$ few clauses but they could be big

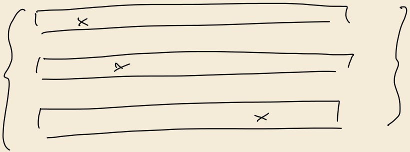
consider $F_L = \{ \text{clauses } C : C \text{ has} \geq k \text{ literals} \}$

$F_S = \{ \text{clauses } C : \text{——} < k \text{ ——} \}$

$|F_L| = \boxed{L \geq k}$

$\implies$ Yes instance

k



each has k variables

can pick unique variable per clause to satisfy it

$\boxed{L < k}$ consider $(F_S, k - L)$ if that is a Yes instance

$\iff (F, k)$ is a Yes instance

Max-SAT Kernel [3]

Proof (cont.): "$\Rightarrow$" If A satisfies $k-L$ clauses in F_S

each of the clauses contains a true literal $\leadsto$ $k-L$

$\leadsto$ A only "fixes" $k-L$ variables

$\Rightarrow$ for L long clauses can find L unique variables that are not fixed in A

"$\Leftarrow$" trivial.

$\Rightarrow$ reduced problem to $(F_S, k-L)$

at most $m-L \leq m$ clauses each has $< k$ literals

$< 2k$

$\Rightarrow$ # literals $= O(k^2)$ (encoding $O(k^2 \log k)$)

Corollary 4.15

p-Max-SAT $\in$ FPT

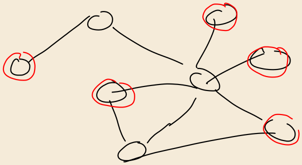
4.4 Depth-Bounded Search II: Planar Independent Set

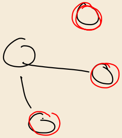
Deeper results (towards more shallow trees)

- Our previous examples of depth-bounded search were basically brute force
- Here we will see two more examples that exploit the problem structure in more interesting ways

Independent Set on Planar Graphs

~~Recall~~ We will see: general problem p-Independent-Set is $\mathcal{W}[1]$-hard.





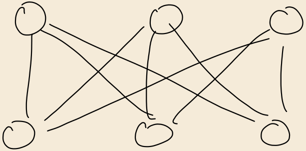
Definition 4.16 (p-Planar-Independent-Set)

Given: a *planar* graph $G = (V, E)$ and $k \in \mathbb{N}$
Parameter: k
Question: $\exists V' \subset V \ : \ |V'| \geq k \ \wedge \ \forall u, v \in V' : \{u, v\} \notin E$?

planar graph G :

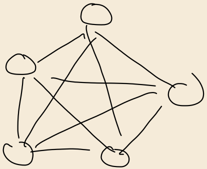
$\exists$ embedding (placement) of vertices in $\mathbb{R}^2$

and a drawing of edges without crossings



$K_{3,3}$ not planar

K_5 not planar



Independent Set on Planar Graphs

Recall: general problem p-Independent-Set is $\mathcal{W}[1]$-hard.

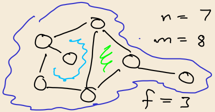
Definition 4.16 (p-Planar-Independent-Set)

Given: a *planar* graph $G = (V, E)$ and $k \in \mathbb{N}$
Parameter: k
Question: $\exists V' \subset V \quad : \quad |V'| \geq k \ \wedge \ \forall u, v \in V' : \{u, v\} \notin E$? ◀

Theorem 4.17 (Depth-Bounded Search for Planar Independent Set)

p-Planar-Independent-Set is in FPT and can be solved in time $O(6^k n)$. ◀

Elementary Knowledge on Planar Graphs



Theorem 4.18 (Euler's formula)

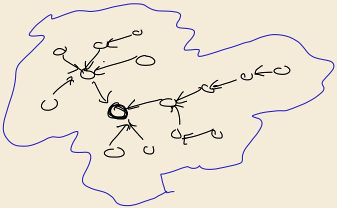
In any finite, connected planar graph G with n nodes, m edges, f faces holds $n - m + f = 2$. ◀

$7 - 8 + 3 = 2$

Proof idea: Induction on f

IB $f = 1$ $\Rightarrow$ G is a tree

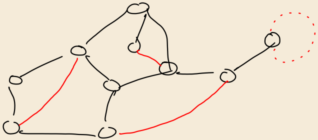
$\Rightarrow$ $n = m + 1$



faces = regions of $\mathbb{R}^2$

IS "add a new face"

$m{+}{+}$ $f{+}{+}$



Elementary Knowledge on Planar Graphs

Theorem 4.18 (Euler's formula)

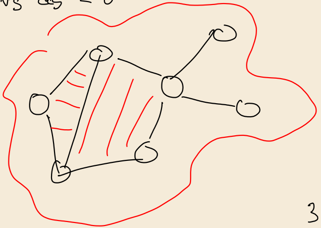
In any finite, connected planar graph G with n nodes, m edges f holds $n - m + f = 2$.

Corollary 4.19

A simple planar graph G on $n \geq 3$ nodes has $m \leq 3n - 6$ edges.
The average degree in G is < 6.

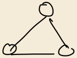
$\sum_v \deg(v) = 2m \le 6n - 12$

avg deg < 6



↯ not simple

↯ not simple



simple $\Rightarrow$ every face is delimited by ≥ 3 edges

$3f$ double counts each edge at most twice

$3f \le 2m$

$\| $

$3(2 - n + m) = 6 - 3n + 3m \quad \Big| -2m + 3n - 6$

$m \le 3n - 6$

avg deg < 6 $\Rightarrow$ $\boxed{\text{in any planar graph, } \exists v : \deg(v) \leq 5}$

"degeneracy" $d = 5$

always find vertex of degree $\leq d$ in G

and in any induced subgraph

induced subgraph

$G = (V, E)$ $\quad G[V'] = \left(V', \{\{u,v\} : u,v \in V', \{u,v\} \in E\}\right)$

$V' \subseteq V$

Depth-Bounded Search for Planar Independent Set

procedure planarIndependentSet($G = (V, E), k$):

  if $k == 0$ **then return** $\emptyset$

  if $k > |V|$ **then return** `NOT_POSSIBLE` // *truncate search*

  Choose $v \in V$ with minimal degree; let $w_1, \ldots, w_d$ be v's neighbors

  // *By planarity, we know* $d \le 5$.

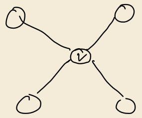
  for u in $\{v, w_1, \ldots, w_d\}$ **do**

    $D := \{u\} \cup N(u)$ — neighbors of u $G_u = G[V \setminus D]$

    $G_u := (V \setminus D, E \setminus \{\{x, y\} \in E : x \in D\})$ // *Delete* u *and its neighbors*

    $I_u := \{u\} \cup$ planarIndependentSet($G_u, k - 1$)

  return largest I_u or `NOT_POSSIBLE` **if** none exists



any maximal indep. set — can't add more vertices to this set

(if none of v's neighbors in the set, could add v)

≤ 6 recursive calls

in w.c. recurse until $k = 0$

$\Rightarrow 6^k$ recursive calls in total

each take $\Theta(n+m) = \Theta(n)$

$\Rightarrow$ total time $O(6^k \cdot n)$

Summary Planar Independent Set

- Note: INDEPENDENTSET is NP-hard on planar graphs even with vertex degrees at most 3
- planarIndependentSet will often be faster than $O(6^k n)$
- works unchanged in $O((d+1)^k n)$ time for any degeneracy-d graph

every (induced) subgraph has vertex of degree at most d

4.5 Depth-Bounded Search III: Closest String

Closest String

Definition 4.20 (p-Closest-String)

Given: S set of m strings $s_1, s_2, \ldots, s_m$ of length L over alphabet Σ and a $k \in \mathbb{N}$.
Parameter: k
Question: Is there a string s for which $d_H(s, s_i) \le k$ holds for all $i = 1, \ldots, m$?

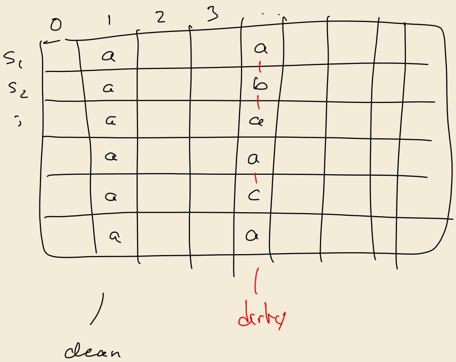
$d_H = 4$

mismatched positions

Dirty Columns

Definition 4.21 (Dirty Column)

A column of the $m \times L$ matrix corresponding to m strings of length L is called *dirty* if it contains at least 2 different symbols.



Dirty Columns

Definition 4.21 (Dirty Column)

A column of the $m \times L$ matrix corresponding to m strings of length L is called *dirty* if it contains at least 2 different symbols. ◀

Lemma 4.22 (Many Dirty Columns → No)

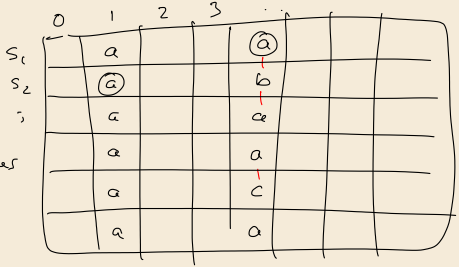
Let an instance to Closest-String with m strings of length L and parameter k be given. If the corresponding $m \times L$ matrix contains more than $m \cdot k$ dirty columns, then no solution for the given instance exists. ◀

If we have $> m \cdot k$ dirty cols

no matter what s,

one s_i must have $\geq k+1$ mismatches

↯



Depth-Bounded Search for Closest String

procedure closestStringFpt(s, d):
  if $d < 0$ **then return** `NOT_POSSIBLE`
  if $d_H(s, s_i) > k + d$ **for an** $i \in \{1, \ldots, m\}$ **then**
    return `NOT_POSSIBLE`
  if $d_H(s, s_i) \le k$ **for all** $i = 1, \ldots, m$ **then return** s
  Choose $i \in \{1, \ldots, m\}$ arbitrarily with $d_H(s, s_i) > k$
    $P := \{p : s[p] \neq s_i[p]\}$
    Choose arbitrary $P' \subseteq P$ with $|P'| = k + 1$
    for p in P' **do**
      $s' := s$
      $s'[p] := s_i[p]$
      $s_{ret} :=$ closestStringFpt($s', d - 1$)
      if $s_{ret} \neq$ `NOT_POSSIBLE` **then return** s_{ret}
  return `NOT_POSSIBLE`

next slide

- initial call closestStringFpt(s_1, k)

search space $(k+1)^k = O(k^k)$

$$\lim_{k \to \infty} \frac{(k+1)^k}{k^k} = \lim \left(\frac{k+1}{k}\right)^k = \lim \left(1 + \frac{1}{k}\right)^k = e = O(1)$$

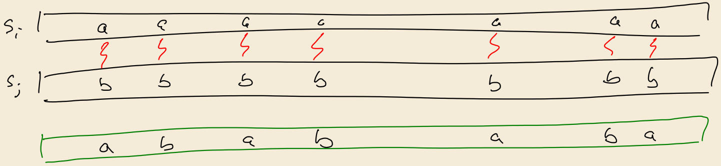
$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

Too Much Dirt

Lemma 4.23 (Pair Too Different → No)

Let $S = \{s_1, s_2, \ldots, s_m\}$ a set of strings and $k \in \mathbb{N}$. If there are $i, j \in \{1, \ldots, m\}$ with $d_H(s_i, s_j) > 2k$, then there is no string s with $\max_{1 \le i \le m} d_H(s, s_i) \le k$.

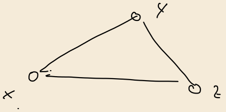


$2k+1$

s has distance $\geq k+1$ to either s_i or s_j

d_H is a metric

- $d_H(x,y) \geq 0$
- $d_H(x,x) = 0$
- "Δ-ineq." $\forall x,y,z$: $d_H(x,z) \leq d_H(x,y) + d_H(y,z)$



Depth-Bounded Search for Closest String

Theorem 4.24 (Search Tree for Closest String)

There is a search tree of size $O(k^k)$ for problem p-Closest-String.

Depth-Bounded Search for Closest String

Theorem 4.24 (Search Tree for Closest String)

There is a search tree of size $O(k^k)$ for problem p-Closest-String. ◀

Corollary 4.25 (Closest String is FPT)

p-Closest-String can be solved in time $O(mL + mk \cdot k^k)$. ◀

maybe can get down to $m \cdot k^k$

- preprocessing ($O(mL)$ time)
  - ignore any clean columns
  - reject if more than mk dirty columns

⇝ effective string length after preprocessing is $L' \leq mk$

- call closestStringFpt(s_1, k)
  - maintain $d_H(s, s_i)$ in an array
    - ⇝ checking any distance $d_H(s, s_i)$ takes $O(1)$ time
    - before and after recursive call, update array to reflect $d_H(s', s_i)$
      Single character changed, so update only needs to check single position
    - ⇝ Can maintain distances in $O(m)$ time per recursive call
  - P' can be computed in $O(mk)$ time

4.6 Linear Recurrences & Better Vertex Cover

A Better Algorithm for Vertex Cover

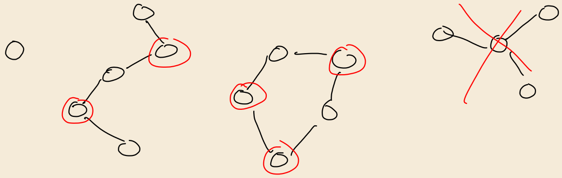
Recall: Branching on endpoints of k edges gives search space of size 2^k for Vertex-Cover.
Can we do better?

A Better Algorithm for Vertex Cover

Recall: Branching on endpoints of k edges gives search space of size 2^k for Vertex-Cover.
Can we do better?

Idea: Enlarge base case with "easy inputs"

Here: Consider graphs G with $\deg(v) \leq 2$ for all $v \in V(G)$.

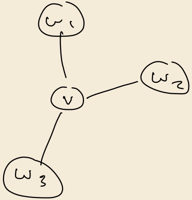


$\Rightarrow$ otherwise $\deg(v) \notin \{0, 1, k+1, k+2 \ldots\}$

$+ \; \exists u: \deg(u) \geq 3$

Depth-Bounded Search for Vertex Cover

```
procedure betterFptVertexCover(G = (V, E), k):
    if E = ∅ then return ∅
    if k = 0 then return NOT_POSSIBLE   // truncate search
    if all node have degree ≤ 2 then
        Find connected components of G
        for each component G_i do
            Fill C_i by picking every other node,
            starting with the neighbor of a degree-one node if one exists
        C := ⋃ C_i
        if |C| ≤ k then return C else return NOT_POSSIBLE
    Choose v with maximal degree, let w_1, ..., w_d be its neighbors // d ≥ 3
    For D in {{v}, {w_1, ..., w_d}} do:
        G_D := (V \ D, E \ {{x, y} ∈ E : x ∈ D})  // Remove D from G
        C_D := D ∪ betterFptVertexCover(G_u, k − |D|)
    return smallest C_D or NOT_POSSIBLE if none exists
```



recurse on (w_i) $k-1$

$k-3$

Depth-Bounded Search for Vertex Cover

```
procedure betterFptVertexCover(G = (V, E), k):
    if E = ∅ then return ∅
    if k = 0 then return NOT_POSSIBLE   // truncate search
    if all node have degree ≤ 2 then
        Find connected components of G
        for each component G_i do
            Fill C_i by picking every other node,
            starting with the neighbor of a degree-one node if one exists
        C := ⋃ C_i
        if |C| ≤ k then return C else return NOT_POSSIBLE
    Choose v with maximal degree, let w_1, ..., w_d be its neighbors // d ≥ 3
    For D in {{v}, {w_1, ..., w_d}} do:
        G_D := (V \ D, E \ {{x, y} ∈ E : x ∈ D})  // Remove D from G
        C_D := D ∪ betterFptVertexCover(G_u, k − |D|)
    return smallest C_D or NOT_POSSIBLE if none exists
```

How to analyze running time of betterFptVertexCover?

Analysis of betterFptVertexCover

worst case running time

- never have all degrees ≤ 2
- always need both recursive calls (until base case)
- ignore that graph gets smaller

$$T_0 = \Theta(1)$$
$$T_k = \Theta(\underbrace{|V|}_{n} + \underbrace{|E|}_{m}) + T_{k-3} + T_{k-1}$$

previously (simpleFptVertexCover)

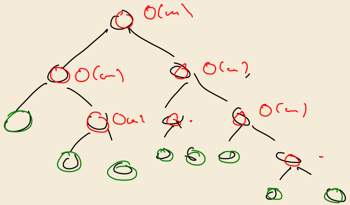
$T_0 = \Theta(1)$

$T_k = 2 \cdot T_{k-1} + \Theta(n+m)$

Analysis of betterFptVertexCover

worst case running time

- never have all degrees ≤ 2
- always need both recursive calls (until base case)
- ignore that graph gets smaller



$$T_0 = \Theta(1)$$
$$T_k = \Theta(|V| + |E|) + T_{k-3} + T_{k-1}$$

$T_k = \Theta(n+m) \cdot$ # non-base cases

$= \Theta(n+m)\,(\underbrace{\text{\# base cases}}_{\approx B_k} - 1)$

If we only number of base cases B_n, we obtain $T_n = O(B_n n^2)$

$$B_0 = 1,\ B_1 = 1,\ B_2 = 1$$
$$B_k = B_{k-3} + B_{k-1} \qquad (k \geq 3)$$

non-base case calls $\leq$ # base cases

$B_k = \Theta(1.46..^k)$

Solving Linear Recurrences

$B_2 = B_1 = B_0 = 1$

$B_k = B_{k-3} + B_{k-1} \quad (k \geq 3)$

$$B(z) = \sum_{k=0}^{\infty} B_k z^k$$

ordinary generation function of sequence $(B_k)_{k \geq 0}$

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\sum_{k=0}^{\infty} z^k = \frac{1}{1-z} \quad \text{— GF of } 1, 1, 1, \ldots$$

$$[z^k]\, B(z) := B_k \underset{a=0}{=} \frac{B^{(k)}(0)}{k!}$$

(not normally convenient to use)

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2} f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

approximate $f(x)$ by polynomial for x close to a

$$B(z) \overset{a=0}{=} B(0) + z \cdot B'(0) + z^2 \frac{B''(0)}{2} + z^3 \frac{B'''(0)}{3!} + \cdots$$

$$= \sum_{k=0}^{\infty} \frac{B^{(k)}(0)}{k!} z^k$$

$$B(z) = \sum_{k \geq 0} B_k \cdot z^k = 1 \cdot z^0 + 1 \cdot z^1 + 1 \cdot z^2 + \sum_{k \geq 3} \underbrace{B_k}_{B_{k-3} + B_{k-1}} \cdot z^k$$

$$= 1 + z + z^2 + \sum_{k\ge 3} B_{k-3} z^k + \sum_{k\ge 3} B_{k-1} z^k$$

$$= \text{—''—} + z^3 \sum_{k\ge 3} B_{k-3} z^{k-3} + z \sum_{k\ge 3} B_{k-1} z^{k-1}$$

$$= \text{—''—} + z^3 \underbrace{\sum_{k\ge 0} B_k z^k}_{B(z)} + z\left[\sum_{k\ge 2} B_k z^k + \underbrace{\left(B_1 z^1 + B_0 z^0 - \overset{1}{\overset{\shortparallel}{B_1}} z^1 - \overset{1}{\overset{\shortparallel}{B_0}} z^0\right)}_{0}\right]$$

$$B(z) \qquad \text{(second bracket term)}$$

$$B(z) = 1 + z + z^2 + z^3 B(z) + z B(z) - z^2 - z$$

$$= 1 + z^3 B(z) + z B(z)$$

$$B(z)\left(1 - z^3 - z\right) = 1 \qquad B(z) = \frac{1}{1 - z^3 - z}$$

$$B(z) = \frac{1}{1 - z^3 - z} = \frac{A}{z - z_0} + \frac{B}{z - z_1} + \frac{C}{z - z_2}$$

(all roots are distinct)

roots of $\uparrow$ z_0, z_1, z_2

Input interpretation

roots	$1 - z^3 - z = 0$

Results

$z \approx 0.68233$ $\leftarrow$

$z \approx -0.34116 - 1.16154\, i$

$z \approx -0.34116 + 1.16154\, i$

$$1 - z^3 - z = (z - z_0)(z - z_1)(z - z_2)$$

$$\frac{1}{0.6823} \approx 1.4655$$

$$\frac{A}{z - z_0}$$

$$\frac{1}{1 - cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$c = \frac{1}{z_0}$$

$$= \frac{A/z_0}{\frac{z}{z_0} - 1} = \frac{-A/z_0}{1 - \frac{z}{z_0}}$$

$$[z^k]\frac{A}{z - z_0} = -\frac{A}{z_0}[z^k]\frac{1}{1 - \frac{z}{z_0}}$$

$$= -\frac{A}{z_0} \cdot \left(\frac{1}{z_0}\right)^k$$

$$B_k = [z^k]B(z) = -\frac{A}{z_0}\left(\frac{1}{z_0}\right)^k - \frac{B}{z_1}\left(\frac{1}{z_1}\right)^k - \frac{C}{z_2}\left(\frac{1}{z_2}\right)^k = \Theta\left(\left|\frac{1}{z_0}\right|^k\right)$$

assuming $|z_0| < |z_1| < |z_2|$

Solving Linear Recurrences – Result

Theorem 4.26 (Linear Recurrences)

Let $d_1, \ldots, d_i \in \mathbb{N}$ and $d = \max d_j$.
The solution to the *homogeneous linear recurrence equation*

$$T_n \quad = \quad T_{n-d_1} + T_{n-d_2} + \cdots + T_{n-d_i}, \qquad (n \geq d)$$

is always given by

$$T_n \quad = \quad \sum_{\ell} \sum_{j=0}^{\mu_\ell - 1} c_{\ell,j}\, z_\ell^n\, n^j$$

where we sum over all roots z_ℓ of multiplicity μ_ℓ of the so-called *characteristic polynomial* $z^d - z^{d-d_1} - z^{d-d_2} \ldots - z^{d-d_i}$.
The d coefficients $c_{\ell,j}$ are determined by the d initial values $T_0, T_1, \ldots, T_{d-1}$. ◂

Corollary 4.27

$T_n = O(z_0^n n^d)$ for z_0 the root of the characteristic polynomial with *largest absolute value*. ◂

Analysis of betterFptVertexCover [2]

$T_0 = \Theta(1)$
$T_k = \Theta(|V| + |E|) + T_{k-3} + T_{k-1}$

If we only number of base cases B_n, we obtain $T_n = O(B_n n^2)$

$B_0 = 1,\ B_1 = 1,\ B_2 = 1$
$B_k = B_{k-3} + B_{k-1} \qquad (k \geq 3)$

Analysis of betterFptVertexCover [2]

$T_0 = \Theta(1)$
$T_k = \Theta(|V| + |E|) + T_{k-3} + T_{k-1}$

If we only number of base cases B_n, we obtain $T_n = O(B_n n^2)$

$B_0 = 1,\ \ B_1 = 1,\ \ B_2 = 1$
$B_k = B_{k-3} + B_{k-1} \qquad (k \geq 3)$

$\rightsquigarrow$ $\vec{d} = (1, 3)$; characteristic polynomial $z^3 - z^2 - 1$
roots at $z_0 \approx 1.4656$ and $z_{1,2} \approx -0.2328 \pm 0.7926i$

Analysis of betterFptVertexCover [2]

$T_0 = \Theta(1)$
$T_k = \Theta(|V| + |E|) + T_{k-3} + T_{k-1}$

If we only number of base cases B_n, we obtain $T_n = O(B_n n^2)$

$B_0 = 1,\ \ B_1 = 1,\ \ B_2 = 1$
$B_k = B_{k-3} + B_{k-1} \qquad (k \geq 3)$

$\rightsquigarrow$ $\vec{d} = (1, 3)$; characteristic polynomial $z^3 - z^2 - 1$
roots at $z_0 \approx 1.4656$ and $z_{1,2} \approx -0.2328 \pm 0.7926i$

Theorem 4.28 (Depth-Bounded Search for Vertex Cover)
p-Vertex-Cover can be solved in time $O(1.4656^k n^2)$. ◀

4.7 Interleaving

Motivation

Up to now, considered two-phase algorithms

1. Reduction to problem kernel
2. Solve kernel by depth-bounded exhaustive search

Idea: Apply kernelization *in each recursive step*.

(Extreme) Example: Vertex Cover with large-degree rule

- As a (slightly artificial) example, consider only using the simple reduction rule

  "deg > k" Rule: If G contains vertex v of degree $\deg(v) > k$, include v in potential solution and remove it from the graph.

- **Algorithm A:**
  1. Apply $\deg > k$ rule until saturation (only this rule)
  2. Call simpleFptVertexCover (recursively branch over arbitrary edge)

- **Algorithm B:** Same, interleaved:
  - Modified simpleFptVertexCover
  - Before choosing each new edge to branch on, apply $\deg > k$ rule.

SimpleFptVertexCover Interleaved

```
procedure simpleFptVertexCover(G = (V, E), k):
    if E == ∅ then return ∅
    if k == 0 then return NOT_POSSIBLE
    // nothing
    // new
    // on
    // this
    // side
    Choose {v, w} ∈ E (arbitrarily)
    for u in {v, w} do:
        G_u := G[V \ {u}]
        C_u := simpleFptVertexCover(G_u, k − 1)
    if C_v == NOT_POSSIBLE then return C_w ∪ {w}
    if C_w == NOT_POSSIBLE then return C_v ∪ {v}
    if |C_v| ≤ |C_w| then
        return C_v ∪ {v}
    else
        return C_w ∪ {w}
```

```
procedure simpleInterleavedVC(G = (V, E), k):
    if E == ∅ then return ∅
    if k == 0 then return NOT_POSSIBLE
    C := ∅
    while ∃v ∈ V : deg(v) > k
        G := G[V \ {v}] // Remove v
        C := C ∪ {v}
        k := k − 1
    Choose {v, w} ∈ E (arbitrarily)
    for u in {v, w} do:
        G_u := G[V \ {u}]
        C_u := C ∪ simpleInterleavedVC(G_u, k − 1)
    if C_v == NOT_POSSIBLE then return C_w ∪ {w}
    if C_w == NOT_POSSIBLE then return C_v ∪ {v}
    if |C_v| ≤ |C_w| then
        return C_v ∪ {v}
    else
        return C_w ∪ {w}
```

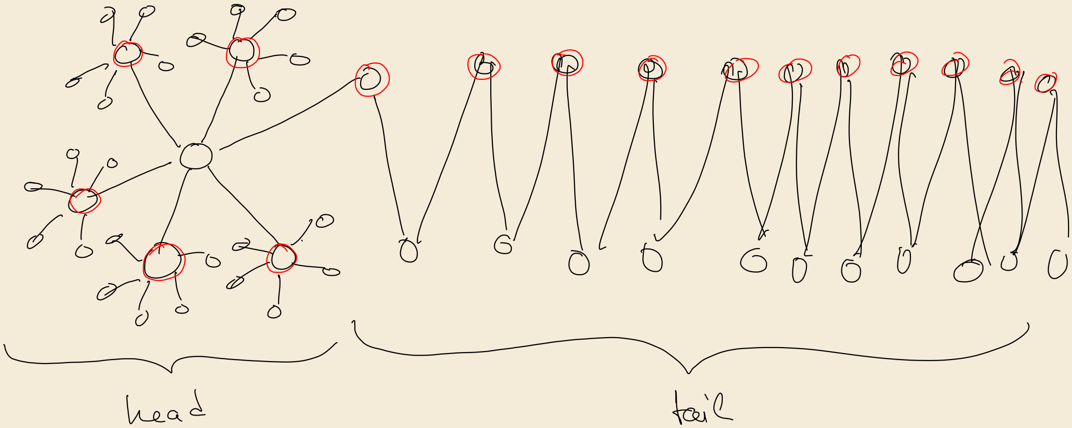
Comparison on Lollipop Flowers

input (G_k, k)

Consider family of graphs G_k "Lollipop Flowers":
"head" vertex with $k-2$ stars of $k-2$ leaves each attached + "tail" of $3k+1$ vertex path

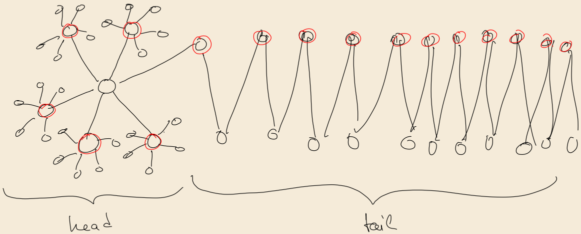
$k=7$

for more than $k=7$



Comparison on Lollipop Flowers

Consider family of graphs G_k “Lollipop Flowers”:
“head” vertex with $k-2$ stars of $k-2$ leaves each attached + “tail” of $3k+1$ vertex path



$n = |V(G_k)| \;=\; (k-2)(k-1)+1 \;+\; 3k+1 \;=\; k^2+4$

Algorithm A

deg $> k$ rule does nothing
search space remains 2^k
Answer No after exploring all branches

$\rightsquigarrow$ time $\Theta(2^k k^2)$

Algorithm B

initially same (no reduction)
after 2 edges removed from tail, parameter $k-2$
vertices in head have degree $k-1$
Output No (parameter 0, but tail edges left)

$\rightsquigarrow$ time $\Theta(k^2)$

Setting for Interleaving

Can we prove a general speedup?

Setting for Interleaving

Can we prove a general speedup?

Assumptions: (more restrictive than general kernelization!)

- K kernelization that
  - produces *kernel of size* $\leq q(k)$ for q a *polynomial*
  - in time $\leq p(n)$ for p a polynomial
- Branch in depth-bounded search tree (1, 3)
  - into i subproblems with branching vector $\vec{d} = (d_1, \ldots, d_i)$ (i. e., parameter in subproblems $k - d_1, \ldots, k - d_i$)
  - Branching is computed in time $\leq r(n)$ for r a polynomial

$\rightsquigarrow$ search space has size $O(\alpha^k)$.

Setting for Interleaving

Can we prove a general speedup?

Assumptions: (more restrictive than general kernelization!)

- K kernelization that
  - produces *kernel of size* $\le q(k)$ for q a *polynomial*
  - in time $\le p(n)$ for p a polynomial
- Branch in depth-bounded search tree
  - into i subproblems with branching vector $\vec{d} = (d_1, \ldots, d_i)$
    (i. e., parameter in subproblems $k - d_1, \ldots, k - d_i$)
  - Branching is computed in time $\le r(n)$ for r a polynomial

$\rightsquigarrow$ search space has size $O(\alpha^k)$.

$\rightsquigarrow$ Running time of two-phase approach on input x with $n = |x|$ and $k = \kappa(x)$:

$$O\Big(\overset{\text{kernelization}}{p(n)} + \overset{\text{branching}}{r}\big(\underset{\text{kernel}}{q(k)}\big) \cdot \alpha^k\Big)$$

With Interleaving

Generic interleaving: k is current parameter

if $|I| > c \cdot q(k)$ **then**
    $(I, k) := (I', k')$ where (I', k') forms a problem kernel // *Conditional Reduction*
end
replace (I, k) with $(I_1, k - d_1), (I_2, k - d_2), \ldots, (I_i, k - d_i)$ // *Branching*

$\rightsquigarrow$ Running time of interleaved approach on input x with $n = |x|$ and $k = \kappa(x)$ is at most T_k:

$$T_\ell \;=\; T_{\ell-d_1} + \cdots + T_{\ell-d_i} \;+\; p\big(q(\ell)\big) + r\big(q(\ell)\big)$$

Compare to non-interleaved version:

$$T_\ell \;=\; T_{\ell-d_1} + \cdots + T_{\ell-d_i} \;+\; r\big(q(k)\big)$$

Here the inhomogeneous term is constant w.r.t. ℓ, but depends on k
$\rightsquigarrow$ cannot ignore constant factors

Analysis of interleaved betterFptVertexCover [1]

Consider betterFptVertexCover from before, but with $\deg > k$ rule added.

- Initial call has unbounded n and m; after applying degree $0, 1, > k$ rules (in $O(n + m)$ time) size of graph $n + m = O(k^2)$

Analysis of interleaved betterFptVertexCover [1]

Consider betterFptVertexCover from before, but with $\deg > k$ rule added.

- Initial call has unbounded n and m; after applying degree $0, 1, > k$ rules (in $O(n+m)$ time) size of graph $n + m = O(k^2)$
- interleaving $\rightsquigarrow$ graph also bounded recursively (in terms of new k)

Analysis of interleaved betterFptVertexCover [1]

Consider betterFptVertexCover from before, but with $\deg > k$ rule added.

- Initial call has unbounded n and m; after applying degree $0, 1, > k$ rules (in $O(n+m)$ time) size of graph $n + m = O(k^2)$
- interleaving $\rightsquigarrow$ graph also bounded recursively (in terms of new k)
- Recursive worst-case time after first reduction:
  $T_0 = \Theta(1)$
  $T_k = O(k^2) + T_{k-3} + T_{k-1}$

Inhomogenous Linear Recurrences

$(T_k)_{k\geq 0} \quad \rightsquigarrow \quad GF$

$T_k = k^2 + T_{k-1} + T_{k-3}$

$$T(z) = \sum_{k\geq 0} T_k z^k = 1 + z + z^2 + \sum_{k\geq 3} (k^2 + T_{k-1} + T_{k-3}) z^k$$

$$= z(T(z) - \ldots) + z^3 T(z) + \underbrace{\sum_{k\geq 3} k^2 z^k}_{\frac{P(z)}{Q(z)}}$$

$$\sum_{k\geq 0} z^k = \frac{1}{1-z} \quad \Big| \frac{d}{dz}$$

$$\sum_{k\geq 1} k \cdot z^{k-1} = -(1-z)^{-2} \cdot (-1) = \frac{1}{1-z^2} \quad \Big| \frac{d}{dz}$$

$$\sum_{k\geq 2} k(k-1) z^{k-2} = -2(1-z)^{-3} \cdot (-1)$$

Inhomogenous Linear Recurrences Summary

Theorem 4.29 (Linear Recurrences II)

Let $d_1, \ldots, d_i \in \mathbb{N}$ and $d = \max d_j$.

Consider the *inhomogeneous linear recurrence equation*

$$T_n \quad = \quad T_{n-d_1} + T_{n-d_2} + \cdots + T_{n-d_i} + f_n, \qquad (n \geq d)$$

with $(f_n)_{n \in \mathbb{R}_{>0}}$ a known sequence of positive numbers, satisfying $f_n = O(n^c)$ and d initial values $T_0, \ldots, T_{d-1} \in \mathbb{R}_{>0}$.

Let z_0 be the root with largest absolute value of $z^d - \sum_{j=1}^{i} z^{d-d_j}$ and assume $\cancel{f_n = O((z_0 - \varepsilon)^n)}$ for some fixed $\varepsilon > 0$.

Then $T_n = O(T_n^0)$ where T_n^0 is defined as T_n with $f_n \equiv 0$.

A Little Excursion: Singularity Analysis

General strategy: use generating functions for asymptotic approximations

A Little Excursion: Singularity Analysis

General strategy: use generating functions for asymptotic approximations

Sequence Land		**Generating Function Land**
▶ number sequence $(a_n)_{n\geq 0}$	$\longrightarrow$	▶ (ordinary) generating function $A(z) = \sum_{n\geq 0} a_n z^n$
▶ recurrence equation	$\longrightarrow$	▶ (functional) equation for $A(z)$
		$\downarrow$ solve, simplify (e. g., partial fractions)
		$\rightsquigarrow$ closed form for $A(z)$
▶ closed form for a_n	$\longleftarrow$	▶ exact coefficients $[z^n]A(z)$

A Little Excursion: Singularity Analysis

General strategy: use generating functions for asymptotic approximations

Sequence Land		**Generating Function Land**
▶ number sequence $(a_n)_{n\geq 0}$	$\longrightarrow$	▶ (ordinary) generating function $A(z) = \sum_{n\geq 0} a_n z^n$
▶ recurrence equation	$\longrightarrow$	▶ (functional) equation for $A(z)$
		$\downarrow$ solve, simplify (e. g., partial fractions)
		$\rightsquigarrow$ closed form for $A(z)$
▶ closed form for a_n	$\longleftarrow$	▶ exact coefficients $[z^n]A(z)$
		OR approximate $A(z)$ near its *dominant singularity*
▶ *asymptotic* approximation $a_n = z_0^{-n} n^{\alpha-1}(1 \pm O(n^{-1}))$	$\xleftarrow[\text{transfer thms}]{}$	$\rightsquigarrow$ *singular expansion* at $z = z_0$ $A(z) = f(z) \pm O((1 - z/z_0)^{-\alpha})$ $z \to z_0$

O-Transfer

Theorem 4.30 (Transfer-Theorem of Singularity Analysis)
Assume $f(z)$ is Δ-analytic and admits the singular expansion

$$f(z) \quad = \quad g(z) \;\pm\; O\big((1-z)^{-\alpha}\big) \qquad (z \to 1)$$

with $\alpha \in \mathbb{R}$. Then

$$[z^n]\,f(z) \quad = \quad [z^n]g(z) \;\pm\; O\big(n^{\alpha-1}\big) \qquad (n \to \infty).$$

◀

Possible Extensions

- (constant) coefficients $c_j \cdot T_{n-d_j}$ in recurrence
  $\rightsquigarrow$ different characteristic polynomial, same ideas
- *any* recurrence that leads to a representation of the generating function as a *singular expansion* around the dominant singularity.

$$\begin{aligned} f(z) &= c(1-z/z_0)^{-m} \pm O((1-z/z_0)^{-m+1}) \qquad (z \to z_0) \\ \rightsquigarrow [z^n]\, f(z) &= \frac{c}{(m-1)!} z_0^{-n} n^{m-1} \cdot \Big(1 \pm O(n^{-1})\Big) \qquad (n \to \infty) \end{aligned}$$

- other powers α in $1/(1-z)^\alpha$:

$$[z^n]\frac{1}{(1-\frac{z}{z_0})^\alpha} = \frac{z_0^{-n} n^{\alpha-1}}{\Gamma(\alpha)}\Big(1 \pm O(n^{-1})\Big) \qquad (n \to \infty) \qquad \begin{array}{c} -\alpha \notin \mathbb{N}_0 \\ z_0 > 0 \end{array}$$

- much more! $\rightsquigarrow$ *analytic combinatorics*

Analysis of interleaved betterFptVertexCover [2]

- $T_0 = \Theta(1)$
  $T_k = O(k^2) + T_{k-3} + T_{k-1}$

$\rightsquigarrow$ $T_k = O(1.4656^k)$ (same characteristic polynomial)

- Total time: $O(1.4656^k + n + m)$

Analysis of interleaved betterFptVertexCover [2]

- $T_0 = \Theta(1)$
  $T_k = O(k^2) + T_{k-3} + T_{k-1}$

$\rightsquigarrow$ $T_k = O(1.4656^k)$ (same characteristic polynomial)

- Total time: $O(1.4656^k + n + m)$

- The current record is $O(1.2738^k + kn)$ time

Summary

- Strategies for fpt algorithms
  - Use parameter to bound depth of exhaustive search
  - Use problem specific reduction rules to shrink input $\rightsquigarrow$ kernel(ization)s
- analysis of exact exponential searches often reduces to linear recurrences
  - generating functions!
- more clever branching reduces exponent of search space
- interleaving kernelization and exhaustive search improves polynomial parts