4 Fixed-Parameter Algorithms

14 May 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version 2025-06-04 14:42 H

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

$\rightsquigarrow$ 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
  - $k \leq 2 \quad \rightsquigarrow \quad$ 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 $\kappa : \Sigma^\star \to \mathbb{N}$ 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

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

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.

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

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

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.

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

Proof:

Aren't we all FPT? – Proof

Proof (cont.):

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!

- 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 $f(k)$

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

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 $E == \emptyset$ **then return** $\emptyset$
    if $k == 0$ **then return** `NOT_POSSIBLE` *// truncate search*
    Choose $\{v, w\} \in E$ (arbitrarily)
    for u in $\{v, w\}$ **do**:
        $G_u := \big(V \setminus \{u\}, E \setminus \{\{u, x\} \in E\}\big)$ *// Remove u from G*
        $C_u :=$ simpleFptVertexCover(G_u, $k - 1$)
    if $C_v ==$ `NOT_POSSIBLE` **then return** $C_w \cup \{w\}$
    if $C_w ==$ `NOT_POSSIBLE` **then return** $C_v \cup \{v\}$
    if $|C_v| \le |C_w|$ **then return** $C_v \cup \{v\}$ **else return** $C_w \cup \{w\}$

- 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

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
```

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

⇝ 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
- 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) $|x'| \leq g(k)$ 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 $O(k^2)$.

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:

FPT iff Kernelization [2]

Proof (cont.):

Max-SAT Kernel

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:

Max-SAT Kernel [2]

Proof (cont.):

Max-SAT Kernel [3]

Proof (cont.):

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: 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 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. ◀

Depth-Bounded Search for Planar Independent Set

```
procedure planarIndependentSet(G = (V, E), k):
    if k == 0 then return ∅
    if k > |V| then return NOT_POSSIBLE // truncate search
    Choose v ∈ V with minimal degree; let w_1, ..., w_d be v's neighbors
    // By planarity, we know d ≤ 5.
    for u in {v, w_1, ..., w_d} do
        D := {u} ∪ N(u)
        G_u := (V \ D, E \ {{x, y} ∈ E : x ∈ D}) // Delete u and its neighbors
        I_u := {u} ∪ planarIndependentSet(G_u, k − 1)
    return largest I_u or NOT_POSSIBLE if none exists
```

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 $d_H(s, s_i) \leq k$ holds for all $i = 1, \ldots, m$?

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. ◀

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 ∈ {1, ..., m} then
        return NOT_POSSIBLE
    if d_H(s, s_i) ≤ k for all i = 1, ..., m then return s
    Choose i ∈ {1, ..., m} arbitrarily with d_H(s, s_i) > k
        P := {p : s[p] ≠ s_i[p]}
        Choose arbitrary P' ⊆ 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 ≠ NOT_POSSIBLE then return s_ret
    return NOT_POSSIBLE
```

- initial call closestStringFpt(s_1, k)

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 $\max_{1 \le i \le m} d_H(s, s_i) \le k$. ◀

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 $O(mL + mk \cdot k^k)$. ◀

- 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?

Idea: Enlarge base case with "easy inputs"

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

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

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

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

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

Solving Linear Recurrences

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 $T_n = O(B_n n^2)$

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

$\rightsquigarrow$ $\vec{d} = (1,3)$; 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
  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

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

$\mathrm{deg} > k$ 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?

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
  - 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 $O(\alpha^k)$.

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

$$O\Big(p(n) + r\big(q(k)\big) \cdot \alpha^k\Big)$$

With Interleaving

Generic interleaving:

```
if |I| > c · 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), . . . , (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:

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

Compare to non-interleaved version:

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

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 $n + m = O(k^2)$
- 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

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***

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

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 $f_n = O((z-\varepsilon)^n)$
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

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 $z = z_0$ $A(z) = f(z) \pm O((1 - z/z_0)^{-\alpha})$

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.

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

- other powers α in $1/(1-z)^\alpha$:

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

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

- 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