

4 Fixed-Parameter Algorithms

- 4.1 Fixed-Parameter Tractability
- 4.2 Depth-Bounded Exhaustive Search I
- 4.3 Problem Kernels
- 4.4 Depth-Bounded Search II: Planar Independent Set
- 4.5 Depth-Bounded Search III: Closest String
- 4.6 Linear Recurrences & Better Vertex Cover
- 4.7 Interleaving

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 $k = \text{MaxInt}(x)$
 - ▶ investigate influence of k (and n) on running time
- ~> 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
 - ▶ $k \geq 3$ 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 Σ^*) is a mapping $\kappa : \Sigma^* \rightarrow \mathbb{N}$ that is polytime computable. ◀

Definition 4.2 (Parameterized problem)

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

Definition 4.3 (Canonical Parameterizations)

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

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

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 $v : [n] \rightarrow \{0, 1\}$?



Definition 4.5 (p-Clique)

Given: graph $G = (V, E)$ and $k \in \mathbb{N}$

Parameter: k


Question: $\exists V' \subset V : |V'| \geq k \wedge \forall u, v \in V' : \{u, v\} \in E$?



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 $p\text{-}U$ 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:

- ▶ $p\text{-CLIQUE}$
- ▶ $p\text{-VERTEX-COVER}$
- ▶ $p\text{-GRAPH-COLORING}$
- ▶ ...

Naming convention for other parameters:

$p\text{-}clause\text{-CNF-SAT}$: 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

↪ $O(2^k \cdot n)$ running time

▶ p -CLIQUE

- ▶ k threshold (clique size); n vertices, m edges in graph

↪ $\binom{n}{k}$ candidates to check, each takes time $O(k^2)$ to check

↪ Total time $O(n^k \cdot k^2)$

▶ p -VERTEXCOVER

- ▶ k threshold (VC size); n vertices, m edges in graph

↪ $\binom{n}{k}$ candidates to check, each takes time $O(m)$ to check

↪ Total time $O(n^k \cdot m)$

▶ p -GRAPHCOLORING

- ▶ k threshold (#colors); n vertices, m edges in graph

↪ k^n candidates to check, each takes time $O(m)$

↪ Total time $O(k^n \cdot m)$

FPT Running Time

Definition 4.7 (fpt-algorithm)

Let κ be a parameterization for Σ^* .

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 $x \in \Sigma^*$ with $\kappa(x) = k$ is at most

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

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

```
1 procedure bruteForceSat( $\varphi, \mathcal{X} = \{x_1, \dots, x_k\}$ )
2   if  $k == 0$ 
3     if  $\varphi == \text{true}$  return  $\emptyset$  else UNSATISFIABLE
4   for value in  $\{\text{true}, \text{false}\}$  do
5      $A := \{x_1 \mapsto \text{value}\}$ 
6      $\psi := \varphi[x_1/\text{value}]$  // Substitute value for  $x_1$ 
7      $B := \text{bruteForceSat}(\psi, \{x_2, \dots, x_k\})$ 
8     if  $B \neq \text{UNSATISFIABLE}$ 
9       return  $A \cup B$ 
```

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 \rightarrow FPT)

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

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

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

g weakly increasing: $n \leq m \rightarrow g(n) \leq g(m)$

g unbounded: $\forall t \exists n : g(n) \geq t$

Proof:

Aren't we all FPT? – Proof

Proof (cont.):



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 $f(k)$
- ▶ 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

```
1 procedure simpleFptVertexCover( $G = (V, E), k$ ):
2   if  $E == \emptyset$  then return  $\emptyset$ 
3   if  $k == 0$  then return NOT_POSSIBLE // truncate search
4   Choose  $\{v, w\} \in E$  (arbitrarily)
5   for  $u$  in  $\{v, w\}$  do:
6      $G_u := (V \setminus \{u\}, E \setminus \{\{u, x\} \in E\})$  // Remove  $u$  from  $G$ 
7      $C_u := \text{simpleFptVertexCover}(G_u, k - 1)$ 
8   if  $C_v == \text{NOT\_POSSIBLE}$  then return  $C_w \cup \{w\}$ 
9   if  $C_w == \text{NOT\_POSSIBLE}$  then return  $C_v \cup \{v\}$ 
10  if  $|C_v| \leq |C_w|$  then return  $C_v \cup \{v\}$  else return  $C_w \cup \{w\}$ 
```

- ▶ Does not need explicit checks of solution candidates!
- ▶ runs in time $O(2^k(n + m)) \rightsquigarrow$ 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 !

```
1 procedure vertexCoverBfs( $G = (V, E)$ )
2   for  $k := 0, 1, \dots, |V|$  do
3      $C := \text{simpleFptVertexCover}(G, k)$ 
4     if  $C \neq \text{NOT\_POSSIBLE}$  return  $C$ 
```

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

↪ 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
$$\frac{a_1 \vee a_2 \vee \dots \vee x, \quad b_1 \vee b_2 \vee \dots \vee \neg x}{a_1 \vee a_2 \vee \dots \vee b_1 \vee b_2 \vee \dots}$$
 - ▶ 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 $K : \Sigma^* \rightarrow \Sigma^*$ 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 $\forall v \in V : 2 \leq \deg(v) \leq k$.

(Note that the rule might reduce the parameter k).

In the resulting graph, any VC of size $\leq k$ covers $\leq k^2$ edges.

If $m > k^2$, we output a trivial No-instance (e. g., a K_{k+1} a complete graph on $k + 1$ vertices).

If $m \leq k^2$, then the input size is now bounded by $g(k) = 2k^2$. ■

FPT iff Kernelization

Theorem 4.13 (FPT \leftrightarrow 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
 $p\text{-MAX-SAT} \in \text{FPT}$

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: $\exists V' \subset V : |V'| \geq k \wedge \forall u, v \in V' : \{u, v\} \notin E$?



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

```
1 procedure planarIndependentSet( $G = (V, E)$ ,  $k$ ):
2   if  $k == 0$  then return  $\emptyset$ 
3   if  $k > |V|$  then return NOT_POSSIBLE // truncate search
4   Choose  $v \in V$  with minimal degree; let  $w_1, \dots, w_d$  be  $v$ 's neighbors
5   // By planarity, we know  $d \leq 5$ .
6   for  $u$  in  $\{v, w_1, \dots, w_d\}$  do
7      $D := \{u\} \cup N(u)$ 
8      $G_u := (V \setminus D, E \setminus \{\{x, y\} \in E : x \in D\})$  // Delete  $u$  and its neighbors
9      $I_u := \{u\} \cup \text{planarIndependentSet}(G_u, k - 1)$ 
10  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, \dots, 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 $i = 1, \dots, m$? 

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 \rightarrow 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

```
1 procedure closestStringFpt( $s, d$ ):  
2   if  $d < 0$  then return NOT_POSSIBLE  
3   if  $d_H(s, s_i) > k + d$  for an  $i \in \{1, \dots, m\}$  then  
4     return NOT_POSSIBLE  
5   if  $d_H(s, s_i) \leq k$  for all  $i = 1, \dots, m$  then return  $s$   
6   Choose  $i \in \{1, \dots, m\}$  arbitrarily with  $d_H(s, s_i) > k$   
7    $P := \{p : s[p] \neq s_i[p]\}$   
8   Choose arbitrary  $P' \subseteq P$  with  $|P'| = k + 1$   
9   for  $p$  in  $P'$  do  
10     $s' := s$   
11     $s'[p] := s_i[p]$   
12     $s_{ret} := \text{closestStringFpt}(s', d - 1)$   
13    if  $s_{ret} \neq \text{NOT\_POSSIBLE}$  then return  $s_{ret}$   
14  return NOT_POSSIBLE
```

► initial call $\text{closestStringFpt}(s_1, k)$

Too Much Dirt

Lemma 4.23 (Pair Too Different \rightarrow No)

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



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

```
1 procedure betterFptVertexCover( $G = (V, E), k$ ):  
2   if  $E = \emptyset$  then return  $\emptyset$   
3   if  $k = 0$  then return NOT_POSSIBLE // truncate search  
4   if all node have degree  $\leq 2$  then  
5     Find connected components of  $G$   
6     for each component  $G_i$  do  
7       Fill  $C_i$  by picking every other node,  
8       starting with the neighbor of a degree-one node if one exists  
9      $C := \bigcup C_i$   
10    if  $|C| \leq k$  then return  $C$  else return NOT_POSSIBLE  
11    Choose  $v$  with maximal degree, let  $w_1, \dots, w_d$  be its neighbors //  $d \geq 3$   
12    For  $D$  in  $\{\{v\}, \{w_1, \dots, w_d\}\}$  do:  
13       $G_D := (V \setminus D, E \setminus \{\{x, y\} \in E : x \in D\})$  // Remove  $D$  from  $G$   
14       $C_D := D \cup \text{betterFptVertexCover}(G_D, k - |D|)$   
15    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 $T_n = O(B_n n^2)$

$$B_0 = 1, B_1 = 1, B_2 = 1$$

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

Solving Linear Recurrences

Solving Linear Recurrences – Result

Theorem 4.26 (Linear Recurrences)

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

The solution to the *homogeneous linear recurrence equation*

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

is always given by

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

where we sum over all roots z_{ℓ} of multiplicity μ_{ℓ} of the so-called *characteristic polynomial* $z^d - z^{d-d_1} - z^{d-d_2} \dots - z^{d-d_i}$.

The d coefficients $c_{\ell,j}$ are determined by the d initial values T_0, T_1, \dots, 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 $T_n = O(B_n n^2)$

$$B_0 = 1, B_1 = 1, B_2 = 1$$

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

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

```
1 procedure simpleFptVertexCover( $G = (V, E)$ ,  $k$ ):
2   if  $E == \emptyset$  then return  $\emptyset$ 
3   if  $k == 0$  then return NOT_POSSIBLE
4   // nothing
5   // new
6   // on
7   // this
8   // side
9   Choose  $\{v, w\} \in E$  (arbitrarily)
10  for  $u$  in  $\{v, w\}$  do:
11     $G_u := G[V \setminus \{u\}]$ 
12     $C_u := \text{simpleFptVertexCover}(G_u, k - 1)$ 
13  if  $C_v == \text{NOT\_POSSIBLE}$  then return  $C_w \cup \{w\}$ 
14  if  $C_w == \text{NOT\_POSSIBLE}$  then return  $C_v \cup \{v\}$ 
15  if  $|C_v| \leq |C_w|$  then
16    return  $C_v \cup \{v\}$ 
17  else
18    return  $C_w \cup \{w\}$ 
```

```
1 procedure simpleInterleavedVC( $G = (V, E)$ ,  $k$ ):
2   if  $E == \emptyset$  then return  $\emptyset$ 
3   if  $k == 0$  then return NOT_POSSIBLE
4    $C := \emptyset$ 
5   while  $\exists v \in V : \deg(v) > k$ 
6      $G := G[V \setminus \{v\}]$  // Remove  $v$ 
7      $C := C \cup \{v\}$ 
8      $k := k - 1$ 
9   Choose  $\{v, w\} \in E$  (arbitrarily)
10  for  $u$  in  $\{v, w\}$  do:
11     $G_u := G[V \setminus \{u\}]$ 
12     $C_u := \text{C} \cup \text{simpleInterleavedVC}(G_u, k - 1)$ 
13  if  $C_v == \text{NOT\_POSSIBLE}$  then return  $C_w \cup \{w\}$ 
14  if  $C_w == \text{NOT\_POSSIBLE}$  then return  $C_v \cup \{v\}$ 
15  if  $|C_v| \leq |C_w|$  then
16    return  $C_v \cup \{v\}$ 
17  else
18    return  $C_w \cup \{w\}$ 
```

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

$\deg > k$ 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 $\vec{d} = (d_1, \dots, d_i)$
(i. e., parameter in subproblems $k - d_1, \dots, k - d_i$)
 - ▶ 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 $k = \kappa(x)$:

$$O\left(p(n) + r(q(k)) \cdot \alpha^k\right)$$

With Interleaving

Generic interleaving:

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

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

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

Compare to non-interleaved version:

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

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 $\text{deg} > k$ 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 $d_1, \dots, d_i \in \mathbb{N}$ and $d = \max d_j$.

Consider the *inhomogeneous linear recurrence equation*

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

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

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

▶ number sequence $(a_n)_{n \geq 0}$

▶ recurrence equation

▶ closed form for a_n

▶ *asymptotic* approximation
 $a_n = z_0^{-n} n^{\alpha-1} (1 \pm O(n^{-1}))$

Generating Function Land

▶ (ordinary) generating function $A(z) = \sum_{n \geq 0} a_n z^n$

▶ (functional) equation for $A(z)$

↓ solve, simplify (e. g., partial fractions)

↪ closed form for $A(z)$

▶ exact coefficients $[z^n]A(z)$

OR approximate $A(z)$
near its *dominant singularity*

↪ *singular expansion* at $z = z_0$
 $A(z) = f(z) \pm O((1 - z/z_0)^{-\alpha})$

←
transfer thms

O-Transfer

Theorem 4.30 (Transfer-Theorem of Singularity Analysis)

Assume $f(z)$ is Δ -analytic and admits the singular expansion

$$f(z) = g(z) \pm O((1-z)^{-\alpha}) \quad (z \rightarrow 1)$$

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

$$[z^n]f(z) = [z^n]g(z) \pm O(n^{\alpha-1}) \quad (n \rightarrow \infty).$$



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.

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

- ▶ other powers α in $1/(1 - z)^\alpha$:

$$[z^n] \frac{1}{(1 - \frac{z}{z_0})^\alpha} = \frac{z_0^{-n} n^{\alpha-1}}{\Gamma(\alpha)} \left(1 \pm O(n^{-1})\right) \quad (n \rightarrow \infty) \quad \begin{array}{l} -\alpha \notin \mathbb{N}_0 \\ z_0 > 0 \end{array}$$

- ▶ 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: $O(1.4656^k + n + m)$

► 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