Branching and Iterative Compression

Ariel Kulik

Seminar on Algorithms, Technion, Winter 18/19

Outline

Parameterized Complexity Basics

Definition

A **parameterized problem** *is a language* $L \subseteq \Sigma^* \times \mathbb{N}$*, when* Σ *is a fixed, finite alphabet.*
For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$*,* k *is called the* **parameter**

Parameterized Complexity Basics

Definition

A **parameterized problem** *is a language* $L \subseteq \Sigma^* \times \mathbb{N}$*, when* Σ *is a fixed, finite alphabet.*
For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$*,* k *is called the* **parameter**

Definition

We say that an algorithm $\mathcal{A}$ *is a* **parameterized algorithm** *for* $L \subseteq \Sigma^* \times \mathbb{N}$ *if:*

- *Given* $(x, k) \in \Sigma^* \times \mathbb{N}$ *it decide if* $(x, k) \in L$*.*
- *Its running time if bounded by* $f(k) \cdot |(x, k)|^c$ *for a computable* f *and a constant* c

If such algorithm exists we say that L *is* **fixed parameter tractable (FPT)**.

Vertex Cover

- Let G be a graph.

Vertex Cover

- Let G be a graph.
- A set $S \subseteq V(G)$ if called **vertex cover** of G if for every $(u, v) \in E(G)$ we have $u \in S$ or $v \in S$.

Vertex Cover

- Let G be a graph.
- A set $S \subseteq V(G)$ if called **vertex cover** of G if for every $(u, v) \in E(G)$ we have $u \in S$ or $v \in S$.
- We can define

$$\text{VC} = \{(G, k) | G \text{ has a vertex cover of size at most } k\}$$

- Example (on board)
- NP-hard, No $(2 - \epsilon)$-approximation under UGC

Basic algorithms for Vertex Cover

- Can easily solved in time $O(m \cdot n^k)$ - **not FPT**.
- Few Observations:
  1. If $E(G) = \emptyset$ and $k \geq 0$ this is a True instance.

Basic algorithms for Vertex Cover

- Can easily solved in time $O(m \cdot n^k)$ - **not FPT**.
- Few Observations:
  1. If $E(G) = \emptyset$ and $k \geq 0$ this is a True instance.
  2. If $E(G) \neq \emptyset$ and $k = 0$ this is a False instance.

Basic algorithms for Vertex Cover

- Can easily solved in time $O(m \cdot n^k)$ - **not FPT**.
- Few Observations:
  1. If $E(G) = \emptyset$ and $k \geq 0$ this is a True instance.
  2. If $E(G) \neq \emptyset$ and $k = 0$ this is a False instance.
  3. Let S be a vertex cover of G. For every $(u, v) \in E(G)$ we have $u \in S$ or $v \in S$.

Basic algorithms for Vertex Cover

- Can easily solved in time $O(m \cdot n^k)$ - **not FPT**.
- Few Observations:
  1. If $E(G) = \emptyset$ and $k \geq 0$ this is a True instance.
  2. If $E(G) \neq \emptyset$ and $k = 0$ this is a False instance.
  3. Let S be a vertex cover of G. For every $(u, v) \in E(G)$ we have $u \in S$ or $v \in S$.
- We can use these observations to define an algorithm

Basic algorithms for Vertex Cover

VC(G, k)

1. If $E(G) = \emptyset$ and $k \geq 0$ return **True**
2. If $E(G) \neq \emptyset$ and $k = 0$ return **False**.
3. Select an edge $(u, v) \in E(G)$
   return VC$(G \setminus \{u\}, k-1)$ or VC$(G \setminus \{v\}, k-1)$

Notation: $G \setminus U = (V(G) \setminus U, \{(u_1, u_2) \in E(G) | u_1, u_2 \notin U)$
Correctness ?
Run time - $2^k n^{O(1)}$- FPT

Outline

Branching Algorithms

The main idea- branching rule

Given $(I, k) \in \Sigma^* \times \mathbb{N}$, in polynomial time, either:

- Determine that $(I, k) \in L$ or $(I, k) \notin L$.
- Generate a new set of instances $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ such that

$$(I, k) \in L \iff \exists j, 1 \leq j \leq l : (I_j, k_j) \in L$$

The number of instances ℓ may depend on (I, k).

Branching Algorithms

The main idea- branching rule

Given $(I, k) \in \Sigma^* \times \mathbb{N}$, in polynomial time, either:

- Determine that $(I, k) \in L$ or $(I, k) \notin L$.
- Generate a new set of instances $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ such that

$$(I, k) \in L \iff \exists j, 1 \leq j \leq l : (I_j, k_j) \in L$$

The number of instances ℓ may depend on (I, k).

- The algorithm solves $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ recursively.

Branching Algorithms

The main idea- branching rule

Given $(I, k) \in \Sigma^* \times \mathbb{N}$, in polynomial time, either:

- Determine that $(I, k) \in L$ or $(I, k) \notin L$.
- Generate a new set of instances $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ such that

$$(I, k) \in L \iff \exists j, 1 \leq j \leq l : (I_j, k_j) \in L$$

The number of instances ℓ may depend on (I, k).

- The algorithm solves $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ recursively.
- We can build a *Branching Tree*: for the recursive calls, where $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ are all children of (I, k)

Branching Algorithms

The main idea- branching rule

Given $(I, k) \in \Sigma^* \times \mathbb{N}$, in polynomial time, either:

- Determine that $(I, k) \in L$ or $(I, k) \notin L$.
- Generate a new set of instances $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ such that

$$(I, k) \in L \iff \exists j, 1 \leq j \leq l : (I_j, k_j) \in L$$

The number of instances ℓ may depend on (I, k).

- The algorithm solves $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ recursively.
- We can build a *Branching Tree*: for the recursive calls, where $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ are all children of (I, k)
- We would want the instances to be $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ *easier*- often $0 \leq k_j < k$.

A better algorithm for VC

Corollary

For any $v \in V(G)$*:*
$(G, k) \in VC$ *if and only if*
$(G \setminus \{v\}, k-1) \in VC$ **or** $(G \setminus N(v), k - |N(v)|) \in VC$

observation

If the maximum degree in G is 1, then we can decide if $(G, k) \in$ VC in polynomial time.

A better algorithm for VC- cont

VC2(G,k)

1. If $k < 0$ or ($k = 0$ and $E(G) \neq \emptyset$) return False
2. If the maximal degree in G is 1, solve $(G, k) \in$ VC and return True/False accordingly.
3. Pick a vertex $v \in V(G)$ with maximal degree. Return **VC2**$(G \setminus \{v\}, k-1)$ or **VC2**$(G \setminus N(v), k - |N(v)|)$.

A better algorithm for VC- cont

VC2(G,k)

1. If $k < 0$ or ($k = 0$ and $E(G) \neq \emptyset$) return False
2. If the maximal degree in G is 1, solve $(G, k) \in$ VC and return True/False accordingly.
3. Pick a vertex $v \in V(G)$ with maximal degree. Return **VC2**$(G \setminus \{v\}, k-1)$ or **VC2**$(G \setminus N(v), k - |N(v)|)$.

- Correctness
- Run time: $\phi^k n^{O(1)}$, when $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$

Even better algorithm for VC

Observation

If G has maximal degree 2, we can decide if $(G, k) \in \text{VC}$ in polynomial time.

VC3(G,k)

1. If $k < 0$ or ($k = 0$ and $E(G) \neq \emptyset$) return False
2. If the maximal degree in G is 2, solve $(G, k) \in \text{VC}$ and return True/False accordingly.
3. Pick a vertex $v \in V(G)$ with maximal degree. return **VC3**$(G \setminus \{v\}, k-1)$ or **VC3**$(G \setminus N(v), k - |N(v)|)$.

Even better algorithm for VC

Observation

If G has maximal degree 2, we can decide if $(G, k) \in$ VC in polynomial time.

VC3(G,k)

1. If $k < 0$ or ($k = 0$ and $E(G) \neq \emptyset$) return False
2. If the maximal degree in G is 2, solve $(G, k) \in$ VC and return True/False accordingly.
3. Pick a vertex $v \in V(G)$ with maximal degree. return **VC3**$(G \setminus \{v\}, k-1)$ or **VC3**$(G \setminus N(v), k - |N(v)|)$.

- Correctness- same as before
- $T(k) = T(k-1) + T(k-3) \rightarrow$ run time $1.4646^k n^{O(1)}$

Vertex Cover - Summary

- We got a parameterized algorithm for VC with run time $1.618^k n^{O(1)}$ (or $1.4646^k n^{O(1)}$ if the time allowed).
- The best running time is due by Chen, Kanj and Xia. It is a branching algorithm that achieves $O(1.2738^k + kn)$.
- There is a theory on solving the running time (Section 3.2 in the book).
- Often **kernelization** is used before the algorithm to reduce the dependency in n (next week?)

Summary

- The main idea- split (I, k) into multiple easier instances $(I_1, k_1), \ldots (I_\ell, k_\ell)$:

$$(I, k) \in L \iff \exists j : (I_j, k_j) \in L$$

- Better rules can lead to better run times.
- Often need good understanding of the problem to get a branching rule.

Questions?

Lets take a breaks

Outline

The 3-Hitting Set problem

Hypergraph

An hypergraph is a pair $G = (V, E)$ such that V is a finite set and $E \subseteq 2^V \setminus \{\emptyset\}$.
The rank of G is $\max_{e \in E} |e|$.
An hypergraph of rank 3 or less is 3-hypergraph.

The 3-Hitting Set problem

Hypergraph

An hypergraph is a pair $G = (V, E)$ such that V is a finite set and $E \subseteq 2^V \setminus \{\emptyset\}$.
The rank of G is $\max_{e \in E} |e|$.
An hypergraph of rank 3 or less is 3-hypergraph.

Hitting Set

We say that $S \subseteq V$ is an **hitting set** for an hypergraph $G = (V, E)$ if for every $e \in E$ it holds that $e \cap S \neq \emptyset$.

The 3-Hitting Set problem

Hypergraph

An hypergraph is a pair $G = (V, E)$ such that V is a finite set and $E \subseteq 2^V \setminus \{\emptyset\}$.
The rank of G is $\max_{e \in E} |e|$.
An hypergraph of rank 3 or less is 3-hypergraph.

Hitting Set

We say that $S \subseteq V$ is an **hitting set** for an hypergraph $G = (V, E)$ if for every $e \in E$ it holds that $e \cap S \neq \emptyset$.

3-Hitting Set Problem

Given a 3-hypergraph $G = (V, E)$ and k, decide if G has a hitting set of size at most k.

$$3HS = \{(G, k) | G \text{ has an hitting set of set at most k}\}$$

The 3-Hitting Set Problem - cont

- A generalization of vertex cover.

The 3-Hitting Set Problem - cont

- A generalization of vertex cover.
- Can be solved in time $3^k n^{O(1)}$ using branching.

The 3-Hitting Set Problem - cont

- A generalization of vertex cover.
- Can be solved in time $3^k n^{O(1)}$ using branching.
- We are going to show a $2.61^k n^{O(1)}$ algorithm using **iterative compression**.

Iterative Compression- Overview

3-hitting set

$\Uparrow$

3-hitting set Compression

$\Uparrow$

disjoint 3-hitting set

$\Uparrow$

solve disjoint 3-hitting set

Compression

3-HS compression problem

Given a 3-hypergraph G, an hitting set X of G, $|X| \leq k+1$, and k. Either find a hitting set Z of G of size at most k, or decide no such set exists.

Compression

3-HS compression problem

Given a 3-hypergraph G, an hitting set X of G, $|X| \leq k+1$, and k. Either find a hitting set Z of G of size at most k, or decide no such set exists.

Lemma

If 3-HS compression has an algorithm with run time $f(k)poly(n)$ then 3-HS has an algorithm with run time $f(k)poly(n)$.

From 3-HS compression to 3-HS

- Define $G[U] = (U \cap V(G), \{e \in E(G) | e \subseteq U\})$.

Iterative Compression for 3-HS

1. Let $V(G) = \{v_1, v_2, \ldots, v_k\}$, $V_i = \{v_1, v_2, \ldots, v_i\}$, $G_i = G[V_i]$.
2. Set $S_0 = \emptyset$ (we refer to this line as iteration 0)
3. For i from 1 to n:
   1. Solve 3-HS compression for (G_i, k) given the hitting set $S_{i-1} \cup v_i$.
   2. If got a solution, set S_i to be the solution. Otherwise return $(G, k) \notin$ 3HS
4. return S_n as a hitting set of size k for G.

Iterative Compression- Overview

3-hitting set

$\Uparrow$

3-hitting set Compression

$\Uparrow$

disjoint 3-hitting set

$\Uparrow$

solve disjoint 3-hitting set

Compression using disjoint

3HS- Compression

Input: 3-Hypergraph G, an hitting set X,$|X| \leq k+1$
Output: An hitting set Z, $|Z| \leq k$ of G, or decide $(G,k) \notin$ 3HS.

Compression using disjoint

3HS- Compression

Input: 3-Hypergraph G, an hitting set X,$|X| \leq k+1$
Output: An hitting set Z, $|Z| \leq k$ of G, or decide $(G,k) \notin$ 3HS.

framework

Assume such a set Z exists.
Guess $X_Z = X \cap Z$.
Try to find a an hitting set W of $G_X = G \setminus X_Z$, such that $W \cap X = \emptyset$, $|W| \leq k - |X_Z|$.
Use the fact that $X \setminus X_Z$ is an hitting set of G_X.

Disjoint 3-HS

Disjoint 3-HS

input: A 3-hypergraph G, an HS X of G and k.
output: An HS Z of G, $|Z| \leq k$, $Z \cap X = \emptyset$. Or decide that no such set exists.

Disjoint 3-HS

Disjoint 3-HS

input: A 3-hypergraph G, an HS X of G and k.
output: An HS Z of G, $|Z| \leq k$, $Z \cap X = \emptyset$. Or decide that no such set exists.

Lemma

If Disjoint 3-HS can be solved in time $\alpha^k \cdot poly(n)$ then 3-HS compression can be solved in time $(1+\alpha)^k \cdot poly(n)$.

Compression using Disjoint

3-HS Compression

1. For every $X_Z \subseteq X$:
   1. Solve disjoint 3-HS for $G \setminus X_Z$ with $X \setminus X_Z$ and parameter $k - |X_Z|$.
   2. If got a solution S, then return $S \cup X_Z$
2. If failed for every set X_Z, determine that $(G, k) \notin$ 3HS.

Iterative Compression- Overview

3-hitting set

$\Uparrow$

3-hitting set Compression

$\Uparrow$

disjoint 3-hitting set

$\Uparrow$

solve disjoint 3-hitting set

Solving the disjoint problem

Disjoint 3-HS

input: 3-hypergraph G, an HS X and k.

Solving the disjoint problem

Disjoint 3-HS

input: 3-hypergraph G, an HS X and k.

1. If there is $e \in E(G)$ such that $e \subseteq X$ return failure.

Solving the disjoint problem

Disjoint 3-HS

input: 3-hypergraph G, an HS X and k.

1. If there is $e \in E(G)$ such that $e \subseteq X$ return failure.
2. Let $U = \{v \in V(G) | \exists e \in E(G), e \setminus X = \{v\}\}$. If $|U| > k$ return failure.

Solving the disjoint problem

Disjoint 3-HS

input: 3-hypergraph G, an HS X and k.

1. If there is $e \in E(G)$ such that $e \subseteq X$ return failure.
2. Let $U = \{v \in V(G) | \exists e \in E(G), e \setminus X = \{v\}\}$. If $|U| > k$ return failure.
3. Let $G' = (V', E')$ where $V' = V(G) \setminus X \setminus U$ and $E' = \{(u, v) \in V' | \exists e \in E(G) : \{u, v\} = e \setminus X\}$.
   mypause
4. Find a vertex cover S of G' of size $k' = k - |U|$. If one exists, return $S \cup U$. Otherwise return failure.

Iterative compression for 3HS- recap

- We can solve Disjoint 3-HS in time $1.61^k \cdot \text{poly}(n)$ using a reduction to Vertex Cover.
- We solve 3-HS compression in time $(1 + 1.61)^k \cdot \text{poly}(n)$.
- We solve 3-HS in time $2.61^k \cdot \text{poly}(n)$.

Iterative compression-summary

- Two main ideas:
  - Solve a problem by iteratively solving the compression problem.
  - Solve the compression problem using the disjoint problem.
- The disjoint problem is often more simple than the original problem.
- We demonstrated the technique for 3-HS. It an be applied to other problems: feedback vertex set, directed feedback vertex set in tournaments, Odd cycle traversal, etc.

Questions?

Questions?

Thank You!