Closest substring problems with small distances

Dániel Marx

Humboldt-Universität zu Berlin

`dmarx@informatik.hu-berlin.de`

April 20, 2006

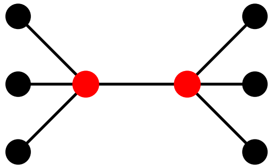
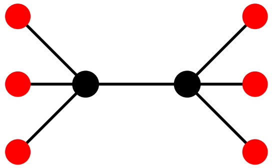
Department of Computer Science and Operations Research

Université de Montréal

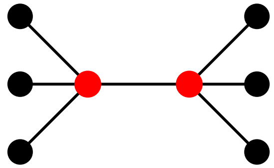
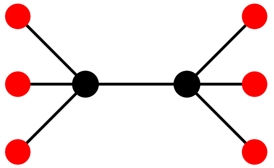
Overview

- Parameterized complexity
- The CLOSEST SUBSTRING problem
  - Complexity
  - First algorithm
  - Results on hypergraphs
  - Second algorithm
- The CONSENSUS PATTERNS problem

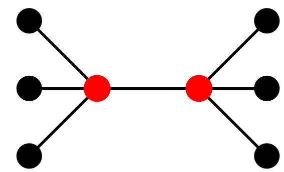
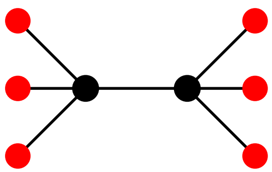
Parameterized complexity

Problem:	MINIMUM VERTEX COVER	MAXIMUM INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices? 	Is it possible to find k independent vertices? 
Complexity:	NP-complete	NP-complete

Parameterized complexity

	Minimum Vertex Cover	Maximum Independent Set
Problem:	MINIMUM VERTEX COVER	MAXIMUM INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete
Complete enumeration:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

Problem:	MINIMUM VERTEX COVER	MAXIMUM INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices? 	Is it possible to find k independent vertices? 
Complexity:	NP-complete	NP-complete
Complete enumeration:	$O(n^k)$ possibilities	$O(n^k)$ possibilities
	$O(2^k n^2)$ algorithm exists 	No $n^{o(k)}$ algorithm known 

Parameterized Complexity

Parameterized problem: input has a special part (usually an integer) called the **parameter.**

Parameterized Complexity

Parameterized problem: input has a special part (usually an integer) called the **parameter.**

- A parameterized problem is **fixed-parameter tractable (FPT)** if it has an $f(k) \cdot n^c$ time algorithm, where c is independent of k.

  Example: MINIMUM VERTEX COVER is solvable in $O(2^k \cdot n^2)$ time (or even in $O(1.2832^k k + k|V|)$ time!).

Parameterized Complexity

Parameterized problem: input has a special part (usually an integer) called the **parameter.**

- A parameterized problem is **fixed-parameter tractable (FPT)** if it has an $f(k) \cdot n^c$ time algorithm, where c is independent of k.

  Example: MINIMUM VERTEX COVER is solvable in $O(2^k \cdot n^2)$ time (or even in $O(1.2832^k k + k|V|)$ time!).

- A **W[1]-hard** problem is unlikely to be FPT. To show that a problem L is W[1]-hard, we have to give a **parameterized reduction** from a known W[1]-hard problem to L.

  Example: MAXIMUM INDEPENDENT SET is W[1]-hard, no $n^{o(k)}$ algorithm is known.

Parameterized Problems

For a large number of NP-hard problems, the parameterized version is fixed-parameter tractable. For some other problems, the parameterized version is W[1]-hard.

Fixed-parameter tractable problems:

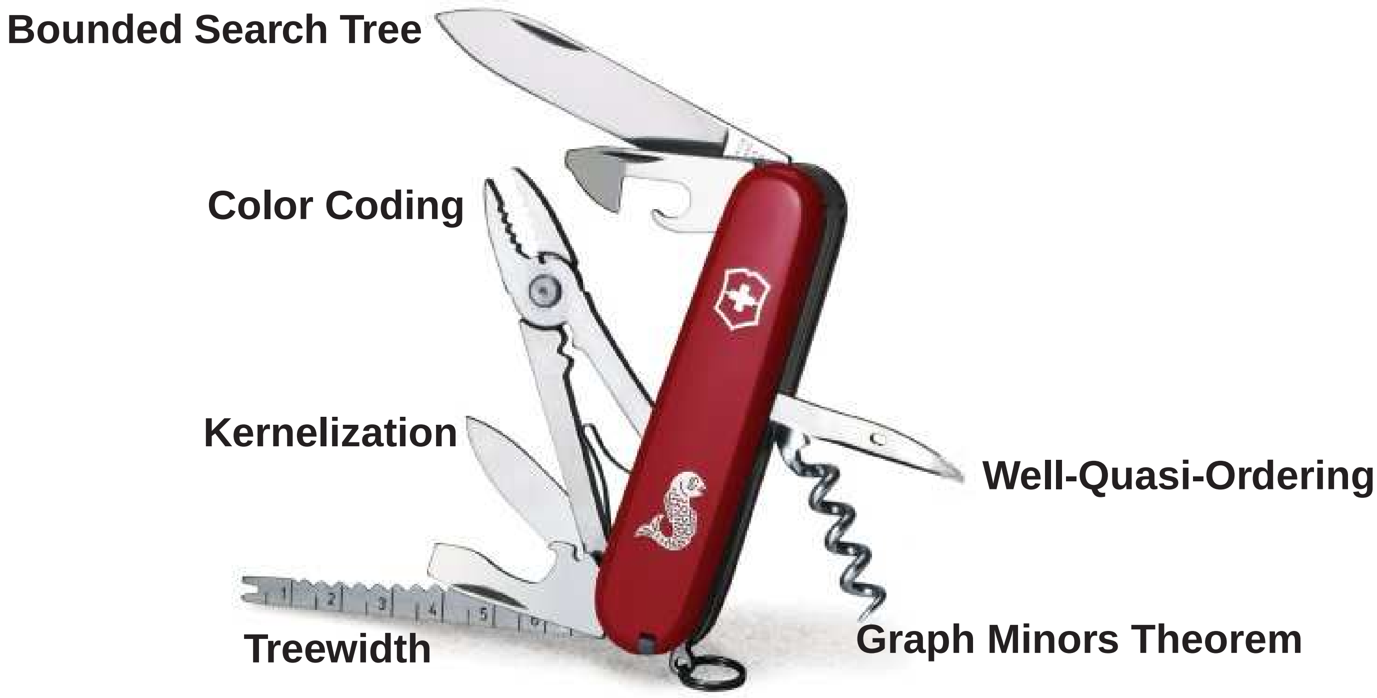
- MINIMUM VERTEX COVER
- LONGEST PATH
- DISJOINT TRIANGLES
- GRAPH GENUS
- ...

W[1]-hard problems:

- MAXIMUM INDEPENDENT SET
- MINIMUM DOMINATING SET
- LONGEST COMMON SUBSEQUENCE
- SET PACKING
- ...

Parameterized Complexity – Motivation

- Practical importance: efficient algorithms for small values of k.
- Powerful toolbox for designing FPT algorithms:



The Closest String problem

CLOSEST STRING	
Input:	Strings $s_1, \ldots, s_k$ of length L
Solution:	A string s of length L (center string)
Minimize:	$\max_{i=1}^{k} d(s, s_i)$

$d(w_1, w_2)$: the number of positions where w_1 and w_2 differ (Hamming distance).

Applications: computational biology (e.g., finding common ancestors)

Problem is NP-hard even with binary alphabet [Frances and Litman, 1997].

The Closest Substring problem

CLOSEST SUBSTRING

Input:	Strings $s_1, \ldots, s_k$, an integer L
Solution:	— string s of length L (center string),
	— a length L substring s'_i of s_i for every i
Minimize:	$\max_{i=1}^{k} d(s, s'_i)$

Remark: For a given s, it is easy to find the best s'_i for every i.

Applications: finding common patterns, drug design.

The Closest Substring problem

CLOSEST SUBSTRING

Input: Strings $s_1, \ldots, s_k$, an integer L

Solution: — string s of length L (center string),

— a length L substring s'_i of s_i for every i

Minimize: $\max_{i=1}^{k} d(s, s'_i)$

Remark: For a given s, it is easy to find the best s'_i for every i.

Applications: finding common patterns, drug design.

- Problem is NP-hard even with binary alphabet (CLOSEST STRING is the special case $|s_i| = L$.)
- CLOSEST SUBSTRING admits a PTAS [Li, Ma, & Wang, 2002]: for every $\epsilon > 0$ there is an $n^{O(1/\epsilon^4)}$ algorithm that produces a $(1+\epsilon)$-approximation.

Parameterized Closest Substring

CLOSEST SUBSTRING	
Input:	Strings $s_1, \dots, s_k$ over Σ, integers L and d
Possible parameters:	$k, L, d, \lvert\Sigma\rvert$
Find:	— string s of length L (center string),
	— a length L substring s'_i of s_i for every i
	such that $d(s, s'_i) \leq d$ for every i

Possible parameters:

- k: might be small
- d: might be small
- L: usually large
- $|\Sigma|$: usually a small constant

Closest Substring—Results

parameter	$\|\Sigma\|$ is constant	$\|\Sigma\|$ is unbounded
d	?	W[1]-hard
k	W[1]-hard	W[1]-hard
d,k	?	W[1]-hard
L	FPT	W[1]-hard
d,k,L	FPT	W[1]-hard

(Hardness results by [Fellows, Gramm, Niedermeier 2002].)

Closest Substring—Results

parameter	$\|\Sigma\|$ is constant	$\|\Sigma\|$ is unbounded
d	W[1]-hard	W[1]-hard
k	W[1]-hard	W[1]-hard
d,k	W[1]-hard	W[1]-hard
L	FPT	W[1]-hard
d,k,L	FPT	W[1]-hard

(Hardness results by [Fellows, Gramm, Niedermeier 2002].)

Theorem: [D.M.] CLOSEST SUBTRING is W[1]-hard with parameters k and d, even if $|\Sigma| = 2$. (In the rest of the talk, Σ is always $\{0, 1\}$.)

Hardness of Closest Substring

Theorem: [D.M.] Closest Substring is W[1]-hard with parameters k and d.

Proof by parameterized reduction from Maximum Independent Set.

Maximum Independent Set	$\Rightarrow$	Closest Substring
(G, t)		$k = 2^{2^{O(t)}}$ $d = 2^{O(t)}$

Corollary: No $f(k, d) \cdot n^c$ algorithm for Closest Substring unless FPT=W[1].

Hardness of Closest Substring

Theorem: [D.M.] CLOSEST SUBSTRING is W[1]-hard with parameters k and d.

Proof by parameterized reduction from MAXIMUM INDEPENDENT SET.

MAXIMUM INDEPENDENT SET		CLOSEST SUBSTRING
(G,t)	$\Rightarrow$	$k = 2^{2^{O(t)}}$
		$d = 2^{O(t)}$

Corollary: No $f(k,d) \cdot n^c$ algorithm for CLOSEST SUBSTRING unless FPT=W[1].

Corollary: No $f(k,d) \cdot n^{o(\log d)}$ or $f(k,d) \cdot n^{o(\log \log k)}$ algorithm for CLOSEST SUBSTRING unless MAXIMUM INDEPENDENT SET has an $f(t) \cdot n^{o(t)}$ algorithm.

Hardness of Closest Substring

Corollary: No $f(k,d) \cdot n^{o(\log d)}$ or $f(k,d) \cdot n^{o(\log \log k)}$ algorithm for CLOSEST SUBSTRING unless MAXIMUM INDEPENDENT SET has an $f(t) \cdot n^{o(t)}$ algorithm.

Hardness of Closest Substring

Corollary: No $f(k,d) \cdot n^{o(\log d)}$ or $f(k,d) \cdot n^{o(\log \log k)}$ algorithm for CLOSEST SUBSTRING unless MAXIMUM INDEPENDENT SET has an $f(t) \cdot n^{o(t)}$ algorithm.

The lower bound on the exponent of n is best possible:

Theorem: [D.M.] CLOSEST SUBSTRING can be solved in $f_1(d,k) \cdot n^{O(\log d)}$ time.

Theorem: [D.M.] CLOSEST SUBSTRING can be solved in $f_2(d,k) \cdot n^{O(\log \log k)}$ time.

Relation to approximability

PTAS: algorithm that produces a $(1+\epsilon)$-approximation in time $n^{f(\epsilon)}$.

EPTAS: (efficient PTAS) a PTAS with running time $f(\epsilon) \cdot n^{O(1)}$.

Observation: if $\epsilon = \frac{1}{2d}$, then a $(1+\epsilon)$-approximation algorithm can correctly decide whether the optimum is d or $d+1$
$\Rightarrow$ if an optimization problem has an EPTAS, then it is FPT.

Corollary: CLOSEST SUBSTRING has no EPTAS, unless FPT=W[1].

The first algorithm

Definition: A solution is a **minimal solution** if $\sum_{i=1}^{k} d(s, s'_i)$ is as small as possible (and $d(s, s'_i) \leq d$ for every i).

The first algorithm

Definition: A solution is a **minimal solution** if $\sum_{i=1}^{k} d(s, s'_i)$ is as small as possible (and $d(s, s'_i) \leq d$ for every i).

Definition: A set of length L strings $\mathcal{G}$ generates a length L string s if whenever the strings in $\mathcal{G}$ agree at the i-th position, then s has the same character at this position.

Example: $\mathcal{G}_1$ generates s but $\mathcal{G}_2$ does not.

$$
\begin{array}{c|c|c|c|c|c|c}
 & 1 & 1 & 0 & 1 & 0 & 1 \\
\mathcal{G}_1 & 0 & 1 & 0 & 1 & 1 & 1 \\
 & 1 & 1 & 0 & 0 & 1 & 1 \\
 & & & & & & \\
s & 1 & 1 & 0 & 1 & 0 & 1
\end{array}
\qquad\qquad
\begin{array}{c|c|c|c|c|c|c}
 & 1 & 1 & 0 & 1 & 1 & 1 \\
\mathcal{G}_2 & 0 & 1 & 0 & 1 & 1 & 1 \\
 & 1 & 1 & 0 & 0 & 1 & 1 \\
 & & & & & & \\
s & 1 & 1 & 0 & 1 & 0 & 1
\end{array}
$$

First algorithm

Let $\mathcal{S}$ be the set of all length L substrings of $s_1, \ldots, s_k$. Clearly, $|\mathcal{S}| \leq n$.

Lemma: If s is the center string of a minimal solution, then $\mathcal{S}$ has a subset $\mathcal{G}$ of size $O(\log d)$ that generates s, and the strings in $\mathcal{G}$ agree in all but at most $O(d \log d)$ positions.

First algorithm

Let $\mathcal{S}$ be the set of all length L substrings of $s_1, \ldots, s_k$. Clearly, $|\mathcal{S}| \le n$.

Lemma: If s is the center string of a minimal solution, then $\mathcal{S}$ has a subset $\mathcal{G}$ of size $O(\log d)$ that generates s, and the strings in $\mathcal{G}$ agree in all but at most $O(d \log d)$ positions.

Algorithm:

- Construct the set $\mathcal{S}$.
- Consider every subset $\mathcal{G} \subseteq \mathcal{S}$ of size $O(\log d)$.
- If there are at most $O(d \log d)$ positions in $\mathcal{G}$ where they disagree, then try every center string generated by $\mathcal{G}$.

Running time: $|\Sigma|^{O(d \log d)} \cdot n^{O(\log d)}$.

Proof of the lemma

Lemma: If s is the center string of a minimal solution, then $\mathcal{S}$ has a subset $\mathcal{G}$ of size $O(\log d)$ that generates s, and the strings in $\mathcal{G}$ agree in all but at most $O(d \log d)$ positions.

Proof: Let $(s, s'_1, \ldots, s'_k)$ be a minimal solution. We show that $\{s'_1, \ldots, s'_k\}$ has a $O(\log d)$ subset that generates s.

The **bad positions** of a set of strings are the positions where they agree, but s is different. Clearly, $\{s'_1\}$ has at most d bad positions.

We show that if a set of strings has p bad positions, then we can decrease the number of bad positions to $p/2$ by adding a string s'_i $\Rightarrow$ no bad position remains after adding $\log d$ strings.

Proof of the lemma (cont.)

Example: there are 4 bad positions:

	1	1	1	1	1	1	1	1	0
	0	1	1	1	1	0	0	1	0
	1	1	1	1	1	1	1	0	0
s	1	0	0	0	0	1	1	0	0

To make a bad position non-bad, we have to add a string that disagree with the previous strings at this position.

There is a string s'_i that disagree on at least half of the bad positions, otherwise we could change s to make $\sum_{i=1}^{k} d(s, s'_i)$ smaller.

Proof of the lemma (cont.)

Example: there are 4 bad positions:

$$
\begin{array}{c|c|cccc|cccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
 & & & & & & & & & \\
s & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}
\quad \Rightarrow \quad
\begin{array}{c|c|cc|cccccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
s'_i & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
 & & & & & & & & & \\
s & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}
$$

To make a bad position non-bad, we have to add a string that disagree with the previous strings at this position.

There is a string s'_i that disagree on at least half of the bad positions, otherwise we could change s to make $\sum_{i=1}^{k} d(s, s'_i)$ smaller.

Proof of the lemma (cont.)

Example: there are 4 bad positions:

$$\begin{array}{c|c|cccc|cccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
 & & & & & & & & & \\
s & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}
\quad \Rightarrow \quad
\begin{array}{c|c|cc|cccccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
s'_i & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
 & & & & & & & & & \\
s & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}$$

To make a bad position non-bad, we have to add a string that disagree with the previous strings at this position.

There is a string s'_i that disagree on at least half of the bad positions, otherwise we could change s to make $\sum_{i=1}^{k} d(s, s'_i)$ smaller.

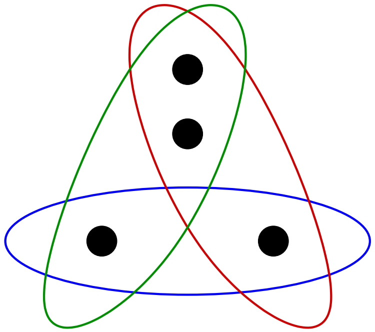
(Since every s'_i differs from s on at most d positions, the $O(\log d)$ strings will agree on all but at most $O(d \log d)$ positions.)

(Fractional) edge covering

Hypergraph: each edge is an arbitrary set of vertices.

An **edge cover** is a subset of the edges such that every vertex is covered by at least one edge.
$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.
$\varrho^*(H)$: smallest total weight of a fractional edge cover.

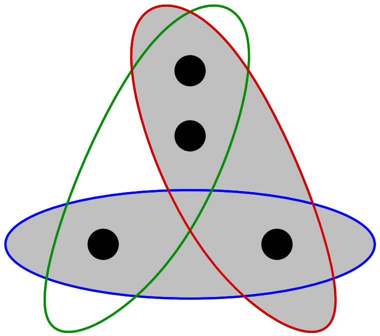


(Fractional) edge covering

Hypergraph: each edge is an arbitrary set of vertices.

An **edge cover** is a subset of the edges such that every vertex is covered by at least one edge.
$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.
$\varrho^*(H)$: smallest total weight of a fractional edge cover.



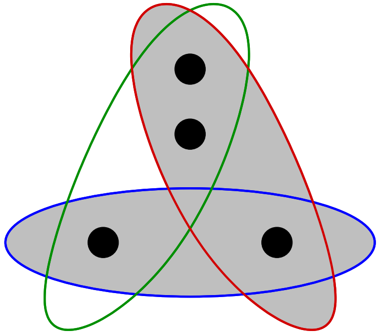
$\varrho(H) = 2$

(Fractional) edge covering

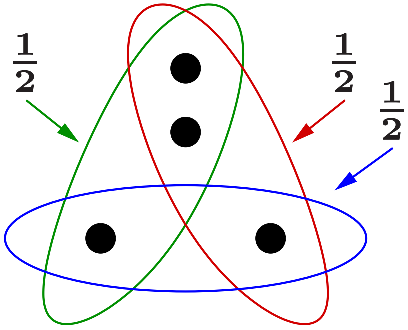
Hypergraph: each edge is an arbitrary set of vertices.

An **edge cover** is a subset of the edges such that every vertex is covered by at least one edge.
$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.
$\varrho^*(H)$: smallest total weight of a fractional edge cover.



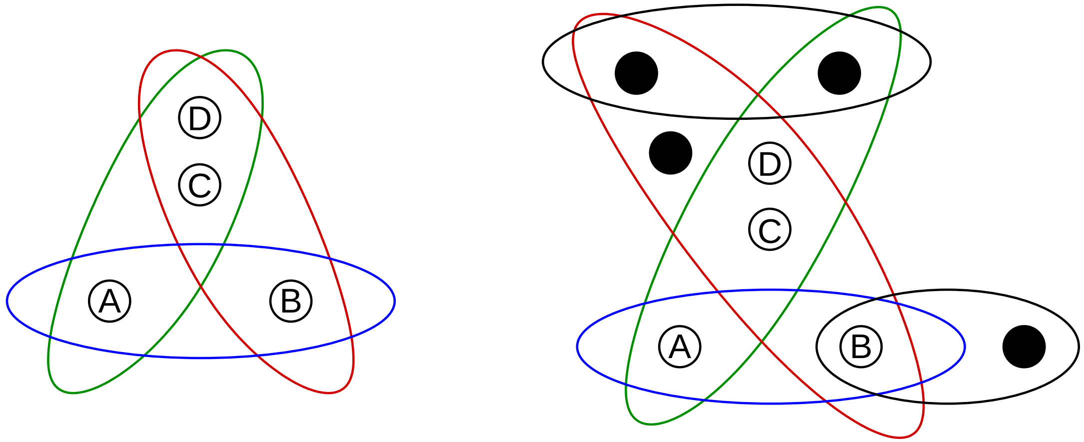
$\varrho(H) = 2$



$\varrho^*(H) = 1.5$

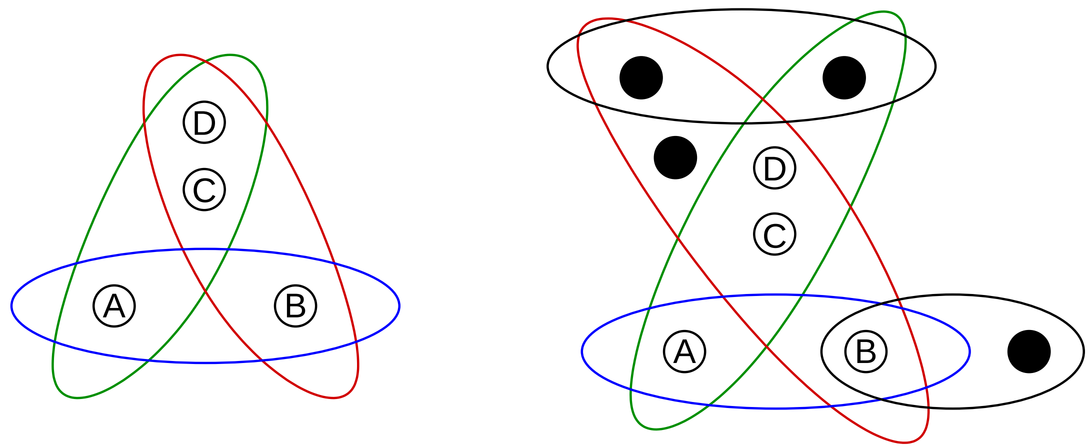
Finding subhypergraphs

Definition: Hypergraph H_1 appears in H_2 as **subhypergraph** at vertex set X, if there is a mapping π between X and the vertices of H_1 such that for each edge E_1 of H_1, there is an edge E_2 of H_2 with $E_2 \cap X = \pi(E_1)$.



Finding subhypergraphs

Definition: Hypergraph H_1 appears in H_2 as **subhypergraph** at vertex set X, if there is a mapping π between X and the vertices of H_1 such that for each edge E_1 of H_1, there is an edge E_2 of H_2 with $E_2 \cap X = \pi(E_1)$.

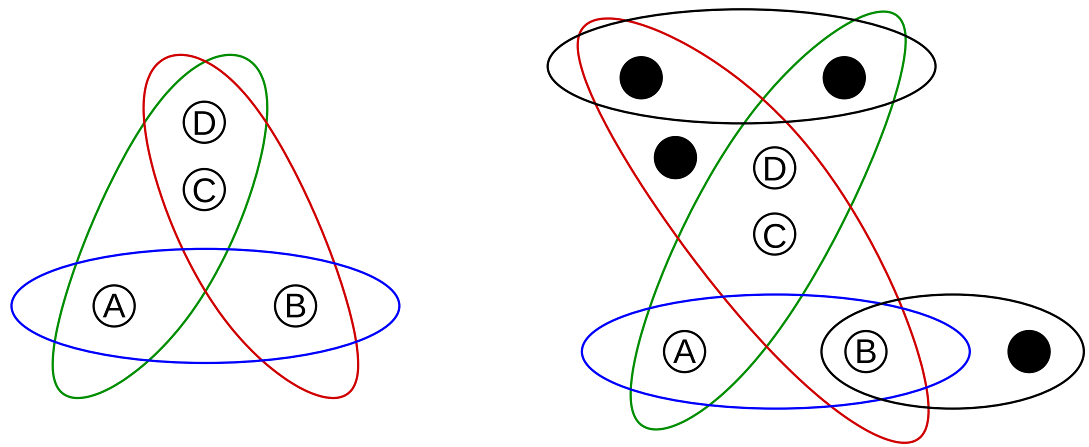


We would like to enumerate all the places where H_1 appears in H_2. Assume that H_2 has m edges and each has size at most ℓ.

Lemma: (easy) H_1 can appear in H_2 at max. $f(\ell, \varrho(H_1)) \cdot m^{\varrho(H_1)}$ places.

Finding subhypergraphs

Definition: Hypergraph H_1 appears in H_2 as **subhypergraph** at vertex set X, if there is a mapping π between X and the vertices of H_1 such that for each edge E_1 of H_1, there is an edge E_2 of H_2 with $E_2 \cap X = \pi(E_1)$.



We would like to enumerate all the places where H_1 appears in H_2. Assume that H_2 has m edges and each has size at most ℓ.

Lemma: (easy) H_1 can appear in H_2 at max. $f(\ell, \varrho(H_1)) \cdot m^{\varrho(H_1)}$ places.
Lemma: [follows from Friedgut and Kahn, 1998] H_1 can appear in H_2 at max. $f(\ell, \varrho^*(H_1)) \cdot m^{\varrho^*(H_1)}$ places.

Half-covering

Defintion: A hypergraph has the half-covering property if for every set X of vertices there is an edge Y with $|X \cap Y| > |X|/2$.

Lemma: If a hypergraph H with m edges has the half-covering property, then $\varrho^*(H) = O(\log \log m)$.

(The $O(\log \log m)$ is best possible.)

Proof: by probabilistic arguments.

Reminder

CLOSEST SUBSTRING	
Input:	Strings $s_1, \ldots, s_k$ over Σ, integers L and d
Possible parameters:	$k, L, d, \lvert\Sigma\rvert$
Find:	— string s of length L (center string),
	— a length L substring s'_i of s_i for every i
	such that $d(s, s'_i) \le d$ for every i

Goal: $f(k, d, \Sigma) \cdot n^{O(\log \log k)}$ running time.

The second algorithm

First step: guess the correct s'_1 ($\leq n$ possibilities).

Consider the set $\mathcal{S}$ of all length L substrings of $s_1, \ldots, s_k$. We turn $\mathcal{S}$ into a hypergraph H on vertices $\{1, 2, \ldots, L\}$: if a string in $\mathcal{S}$ differs from s'_1 on positions $P \subseteq \{1, 2, \ldots, L\}$, then let P be an edge of H.

The second algorithm

First step: guess the correct s_1' ($\leq n$ possibilities).

Consider the set $\mathcal{S}$ of all length L substrings of $s_1, \ldots, s_k$. We turn $\mathcal{S}$ into a hypergraph H on vertices $\{1, 2, \ldots, L\}$: if a string in $\mathcal{S}$ differs from s_1' on positions $P \subseteq \{1, 2, \ldots, L\}$, then let P be an edge of H.

Lemma: Assume that in a minimal solution s differs from s_1' on positions P. Then there is a hypergraph H_0 with at most d vertices and k edges having the half-covering property such that H_0 appears at P in H.

The second algorithm

First step: guess the correct s'_1 ($\leq n$ possibilities).

Consider the set $\mathcal{S}$ of all length L substrings of $s_1, \ldots, s_k$. We turn $\mathcal{S}$ into a hypergraph H on vertices $\{1, 2, \ldots, L\}$: if a string in $\mathcal{S}$ differs from s'_1 on positions $P \subseteq \{1, 2, \ldots, L\}$, then let P be an edge of H.

Lemma: Assume that in a minimal solution s differs from s'_1 on positions P. Then there is a hypergraph H_0 with at most d vertices and k edges having the half-covering property such that H_0 appears at P in H.

Algorithm: Consider every hypergraph H_0 as above and enumerate all the places where H_0 appears in H.

The second algorithm (cont.)

Algorithm:

- Construct the hypergraph H.
- Enumerate every hypergraph H_0 with at most d vertices and k edges (constant number).
- Check if H_0 has the half-covering property.
- If so, then enumerate every place P where H_0 appears in H.
  (max. $\approx n^{O(\varrho^*(H_0))} = n^{O(\log\log k)}$ places).
- For each place P, check if there is a good center string that differs from s_1' only at P.

Running time: $f(k, d, \Sigma) \cdot n^{O(\log\log k)}$.

Consensus Patterns

CONSENSUS PATTERNS

Input:	Strings $s_1, \ldots, s_k$ over Σ, integers L and D
Possible parameters:	$k, L, D, \|\Sigma\|$
Find:	— string s of length L (center string),
	— a length L substring s'_i of s_i for every i
	such that $\sum_{i=1}^{k} d(s, s'_i) \leq D$ for every i

Another natural parameter: $\delta = D/k$, the average distance.

Consensus Patterns —Results

parameter	$|\Sigma|$ is constant	$|\Sigma|$ is unbounded
δ	?	W[1]-hard
D	?	W[1]-hard
k	W[1]-hard	W[1]-hard
L	FPT	W[1]-hard

D: total distance

δ: average distance

Consensus Patterns —Results

parameter	$\lvert\Sigma\rvert$ is constant	$\lvert\Sigma\rvert$ is unbounded
δ	FPT	W[1]-hard
D	FPT	W[1]-hard
k	W[1]-hard	W[1]-hard
L	FPT	W[1]-hard

D: total distance

δ: average distance

Theorem: [D.M.] CONSENSUS PATTERNS is fixed-parameter tractable with parameter δ if Σ is bounded.

Algorithm for CONSENSUS PATTERNS

First step: guess the correct s'_1 ($\leq n$ possibilities).

Consider the set $\mathcal{S}$ of all length L substrings of $s_1, \ldots, s_k$. We turn $\mathcal{S}$ into a hypergraph H on vertices $\{1, 2, \ldots, L\}$: if a string in $\mathcal{S}$ differs from s'_1 on positions $P \subseteq \{1, 2, \ldots, L\}$, then let P be an edge of H.

Algorithm for CONSENSUS PATTERNS

First step: guess the correct s'_1 ($\leq n$ possibilities).

Consider the set $\mathcal{S}$ of all length L substrings of $s_1, \ldots, s_k$. We turn $\mathcal{S}$ into a hypergraph H on vertices $\{1, 2, \ldots, L\}$: if a string in $\mathcal{S}$ differs from s'_1 on positions $P \subseteq \{1, 2, \ldots, L\}$, then let P be an edge of H.

Lemma: Assume that in a minimal solution s differs from s'_1 on positions P. Then there is a hypergraph H_0 with at most δ and $\varrho^*(G) \leq 5$ such that H_0 appears at P in H.

Algorithm for CONSENSUS PATTERNS

First step: guess the correct s'_1 ($\leq n$ possibilities).

Consider the set $\mathcal{S}$ of all length L substrings of $s_1, \ldots, s_k$. We turn $\mathcal{S}$ into a hypergraph H on vertices $\{1, 2, \ldots, L\}$: if a string in $\mathcal{S}$ differs from s'_1 on positions $P \subseteq \{1, 2, \ldots, L\}$, then let P be an edge of H.

Lemma: Assume that in a minimal solution s differs from s'_1 on positions P. Then there is a hypergraph H_0 with at most δ and $\varrho^*(G) \leq 5$ such that H_0 appears at P in H.

Algorithm: Consider every hypergraph H_0 as above and enumerate all the places where H_0 appears in H.

As H_0 has constant fractional edge cover number, the search can be done in polynomial time!

Conclusions

- Complete parameterized analysis of CLOSEST SUBSTRING and CONSENSUS PATTERNS.
- Tight bounds for subexponential algorithms.
- “Weak” parameterized reduction $\Rightarrow$ subexponential algorithms?
- Subexponential algorithms $\Rightarrow$ proving optimality using parameterized complexity?
- Other applications of fractional edge cover number and finding hypergraphs?