The Closest Substring problem with small distances

Dániel Marx

dmarx@informatik.hu-berlin.de

Humboldt-Universität zu Berlin

July 25, 2005

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 $s_1, \ldots, s_k$ 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 $d(s, s'_i) \leq d$ 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 parameter	$\|\Sigma\|$ is unbounded
d	?	?	W[1]-hard
k	W[1]-hard	W[1]-hard	W[1]-hard
d,k	?	?	W[1]-hard
L	FPT	FPT	W[1]-hard
d,k,L	FPT	FPT	W[1]-hard

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

Closest Substring—Results

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

Maximum Independent Set has an $f(t)\cdot n^{o(t)}$ algorithm
$\Downarrow$
n variable 3-SAT can be solved in $2^{o(n)}$ time
$\Updownarrow$
FPT=M[1]

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.

MAXIMUM INDEPENDENT SET has an $f(t) \cdot n^{o(t)}$ algorithm

$$\Downarrow$$

n variable 3-SAT can be solved in $2^{o(n)}$ time

$$\Updownarrow$$

FPT=M[1]

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 $\epsilon = \frac{1}{d+1}$, 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].

Corollary: Closest Substring has no $f(\epsilon) \cdot n^{o(\log \epsilon)}$ time PTAS, unless FPT=M[1].

What's next?

- $f_1(d,k) \cdot n^{O(\log d)}$ time algorithm
- Some results on hypergraphs
- $f_2(d,k) \cdot n^{O(\log \log k)}$ time algorithm
- Sketch of the completeness proof
- Conclusions

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.

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

First algorithm

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

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.

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.

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:

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

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:

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

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.

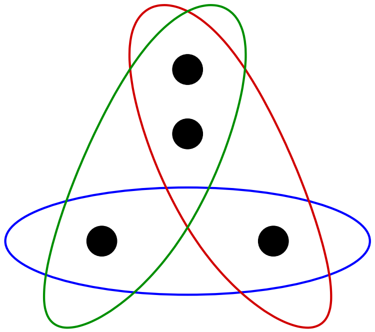
(Since every s'_i differs from s on at most d positions, the $O(\log d)$ strings will agree on all but at most $O(d \log d)$ positions.)

(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.

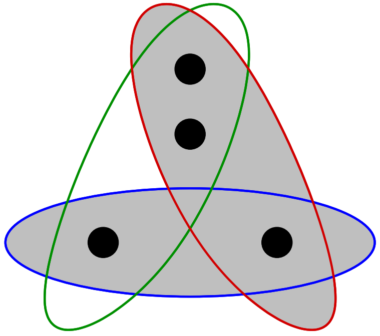


(Fractional) edge covering

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.



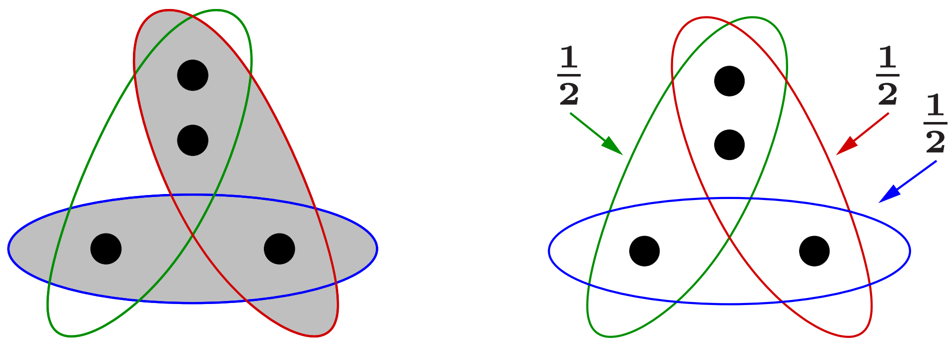
$$\varrho(H) = 2$$

(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.

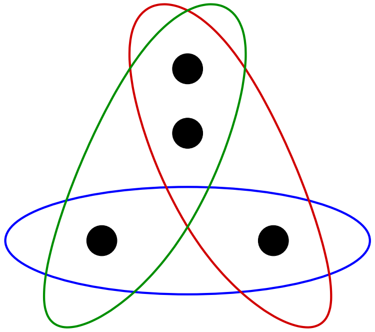


$\varrho(H) = 2$ $\quad\quad$ $\varrho^*(H) = 1.5$

(Fractional) stable sets

A **stable set** is a subset of the vertices such that every edge contains at most one selected vertex.
$\alpha(H)$: size of the largest stable set.

A **fractional stable set** is a weight assignment to the vertices such that the weight covered by each edge is at most 1.
$\alpha^*(H)$: largest total weight of a fractional stable set.



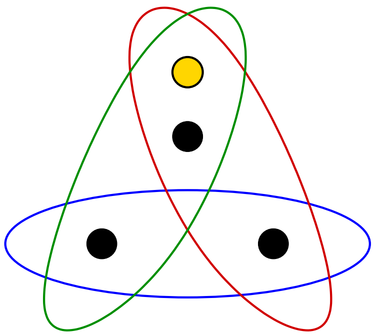
(Fractional) stable sets

A **stable set** is a subset of the vertices such that every edge contains at most one selected vertex.

$\alpha(H)$: size of the largest stable set.

A **fractional stable set** is a weight assignment to the vertices such that the weight covered by each edge is at most 1.

$\alpha^*(H)$: largest total weight of a fractional stable set.

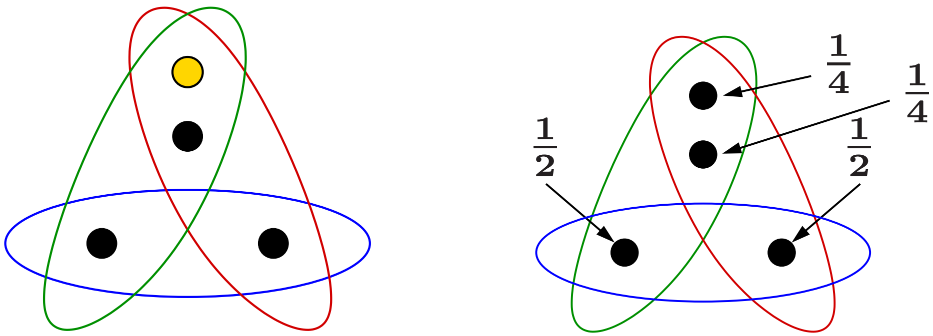


$\alpha(H) = 1$

(Fractional) stable sets

A **stable set** is a subset of the vertices such that every edge contains at most one selected vertex.
$\alpha(H)$: size of the largest stable set.

A **fractional stable set** is a weight assignment to the vertices such that the weight covered by each edge is at most 1.
$\alpha^*(H)$: largest total weight of a fractional stable set.

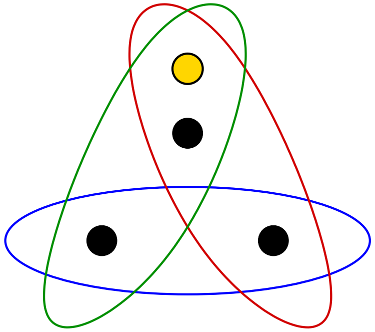


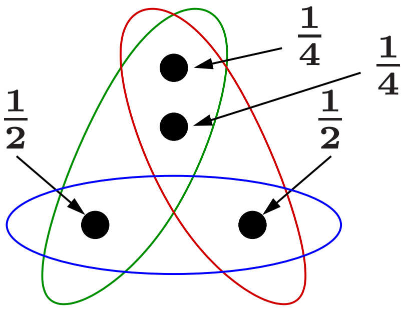
$\alpha(H) = 1$ $\qquad$ $\alpha^*(H) = 1.5$

(Fractional) stable sets

A **stable set** is a subset of the vertices such that every edge contains at most one selected vertex.
$\alpha(H)$: size of the largest stable set.

A **fractional stable set** is a weight assignment to the vertices such that the weight covered by each edge is at most 1.
$\alpha^*(H)$: largest total weight of a fractional stable set.



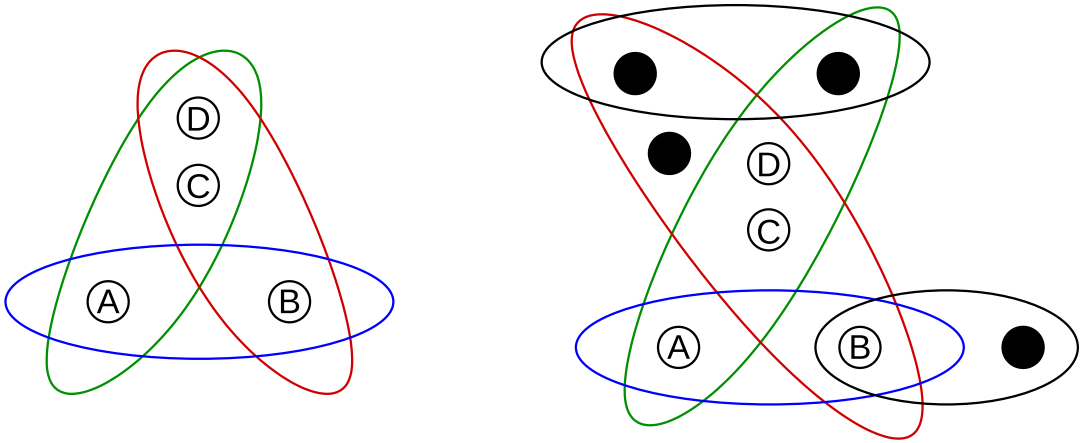


$$\alpha(H) = 1 \qquad \alpha^*(H) = 1.5$$

By linear programming duality: $\overset{\color{red}1}{\alpha(H)} \leq \overset{\color{red}1.5}{\alpha^*(H)} = \overset{\color{red}1.5}{\varrho^*(H)} \leq \overset{\color{red}2}{\varrho(H)}$

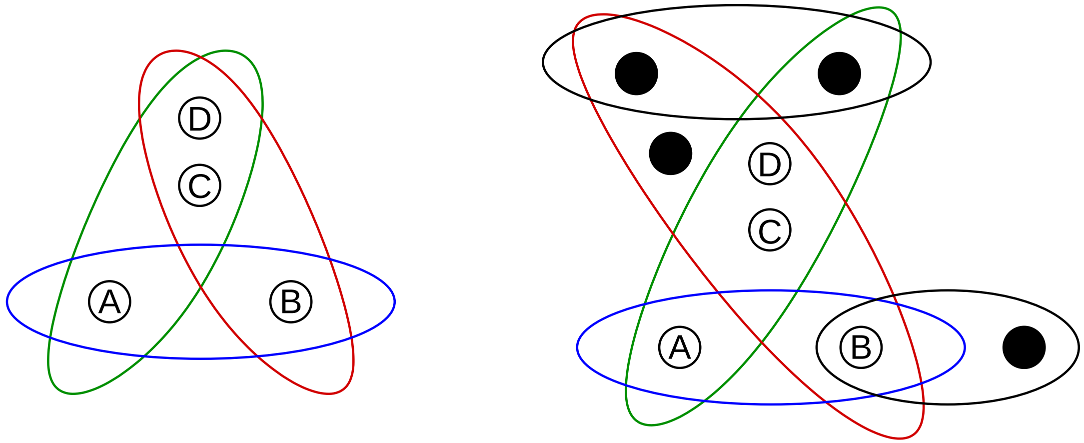
Finding subhypergraphs

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

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 $E_2 \cap X = \pi(E_1)$.

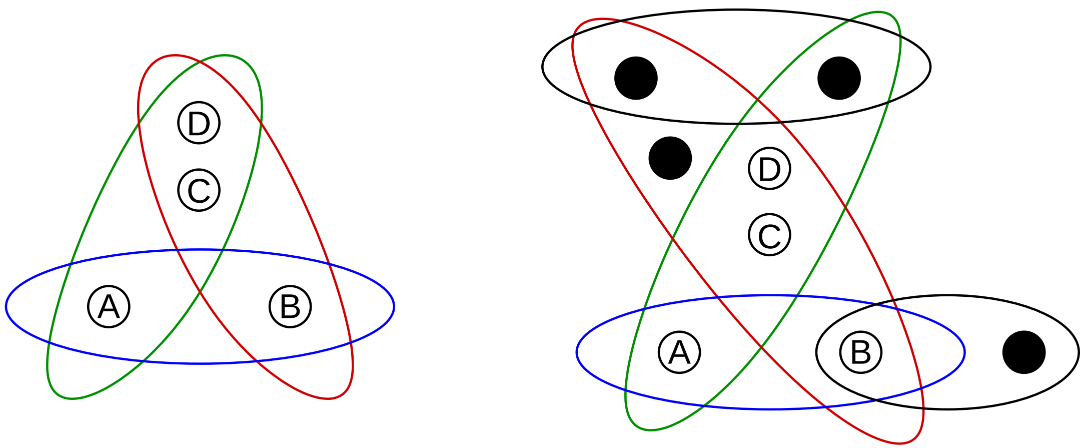


We would like to enumerate all the places where H_1 appears in H_2. Assume that H_2 has m edges and each has size at most ℓ.

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

Finding subhypergraphs

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.

Finding subhypergraphs

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

We want to turn this result into an algorithm (proof is based on Shearer's Lemma, not algorithmic).

Finding subhypergraphs

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

We want to turn this result into an algorithm (proof is based on Shearer's Lemma, not algorithmic).

Algorithm: Let $\{1, 2, \ldots, r\}$ be the vertices of H_1, and let $H_1^{(i)}$ be the induced subhypergraph of H_1 on $\{1, 2, \ldots, i\}$. For $i = 1, 2, \ldots, r$, the algorithm enumerates the list L_i of all the places where $H_1^{(i)}$ appears in H_2.

- L_1 is trivial.
- L_{i+1} is easy to construct based on L_i.
- Since $\varrho^*(H_1^{(i)}) \leq \varrho^*(H_1)$, the list L_i cannot be too large.

Lemma: We can enumerate in $f(\ell, \varrho^*(H_1)) \cdot m^{O(\varrho^*(H_1))}$ time all the places where H_1 appears in H_2.

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:	$k, L, d, \|\Sigma\|$
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

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

Proof of the lemma

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.

Proof:

- Consider a minimal solution.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s'_5	1	0	0	1	1	1	0	0	0	0
s	0	1	1	1	1	1	0	0	0	0

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k-1$ edges of H.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s'_5	1	0	0	1	1	1	0	0	0	0
s	0	1	1	1	1	1	0	0	0	0

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k - 1$ edges of H.
- P: the positions where s'_1 and s differ.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s'_5	1	0	0	1	1	1	0	0	0	0
s	0	1	1	1	1	1	0	0	0	0
				P						

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k-1$ edges of H.
- P: the positions where s'_1 and s differ.
- Restrict the $k-1$ edges to $P \Rightarrow H_0$.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s'_5	1	0	0	1	1	1	0	0	0	0
s	0	1	1	1	1	1	0	0	0	0
				P						

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k-1$ edges of H.
- P: the positions where s'_1 and s differ.
- Restrict the $k-1$ edges to $P \Rightarrow H_0$.
- Claim: H_0 has the half-covering property.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s'_5	1	0	0	1	1	1	0	0	0	0
s	0	1	1	1	1	1	0	0	0	0
				P						

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k-1$ edges of H.
- P: the positions where s'_1 and s differ.
- Restrict the $k-1$ edges to $P \Rightarrow H_0$.
- Claim: H_0 has the half-covering property.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s	0	1	1	1	1	1	0	0	0	0
				P						

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k-1$ edges of H.
- P: the positions where s'_1 and s differ.
- Restrict the $k-1$ edges to $P \Rightarrow H_0$.
- Claim: H_0 has the half-covering property.
- If half-covering is violated for $R \subseteq P \ldots$

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s	0	1	1	1	1	1	0	0	0	0
					R					

Proof of the lemma

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.

Proof:

- Consider a minimal solution.
- The solution gives $k-1$ edges of H.
- P: the positions where s'_1 and s differ.
- Restrict the $k-1$ edges to $P \Rightarrow H_0$.
- Claim: H_0 has the half-covering property.
- If half-covering is violated for $R \subseteq P \ldots$
- $\ldots$ then we can change s on R.

s'_1	0	0	0	0	0	0	0	0	0	0
s'_2	0	1	1	1	1	0	0	1	0	0
s'_3	0	1	0	0	0	1	1	0	0	0
s'_4	0	0	1	1	0	1	0	0	1	0
s	0	1	1	1	0	0	0	0	0	0
					R					

The reduction

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

The reduction is based on the proof of previous weaker result:

Theorem: [Fellows, Gramm, Niedermeier, 2002] CLOSEST SUBSTRING is W[1]-hard with parameter k.

The reduction

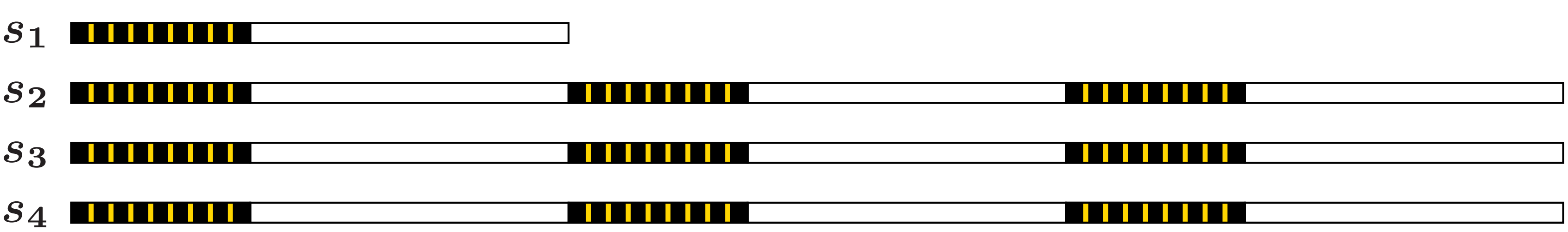
Theorem: CLOSEST SUBSTRING is W[1]-hard with parameters k and d.

The reduction is based on the proof of previous weaker result:

Theorem: [Fellows, Gramm, Niedermeier, 2002] CLOSEST SUBSTRING is W[1]-hard with parameter k.

Idea 1: Every string s_i is divided into blocks of length L. We ensure that s'_i is one complete block of s_i.

How: Each block starts with the front tag $(1^x0)^y$, and there is a special string having only one block.



The reduction

Reduction from MAXIMUM INDEPENDENT SET.

Idea 2: The center string (and each block) is divided into k segments of length n. We ensure that each segment contains exactly one symbol "1" and these k symbols describe an independent set of size k.

How: string $s_{i,j}$ ensures that vertex v_i and v_j are not connected. The blocks of $s_{i,j}$ contain 1's only in segments i and j, and there is a block for each valid combination.

Dirty trick to ensure that there is at least one "1" in each segment, but this requires large d.

The reduction

New idea: Instead of k segments of size n,

- vertex v_1 is described by a segment of size n
- vertex v_2 is described by 2 segments of size $n^{1/2}$
- vertex v_3 is described by 4 segments of size $n^{1/4}$
- $\ldots$

$\Rightarrow$ we have $2^t - 1$ segments.

For each subset S of the segments, there is a string that makes it impossible that there is no “1” in S, but there is at least one in every other segment.

$\Rightarrow k = 2^{2^{O(k)}}$

Conclusions

- Complete parameterized analysis of CLOSEST SUBSTRING.
- 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?