Lovász-Stein Theorem

Seminar Extremal Graph Theory

Robin Link

20.12.2021

Advisors:
Prof. Dr. Maria Axenovich & Dr. Alexander Neal Riasanovsky

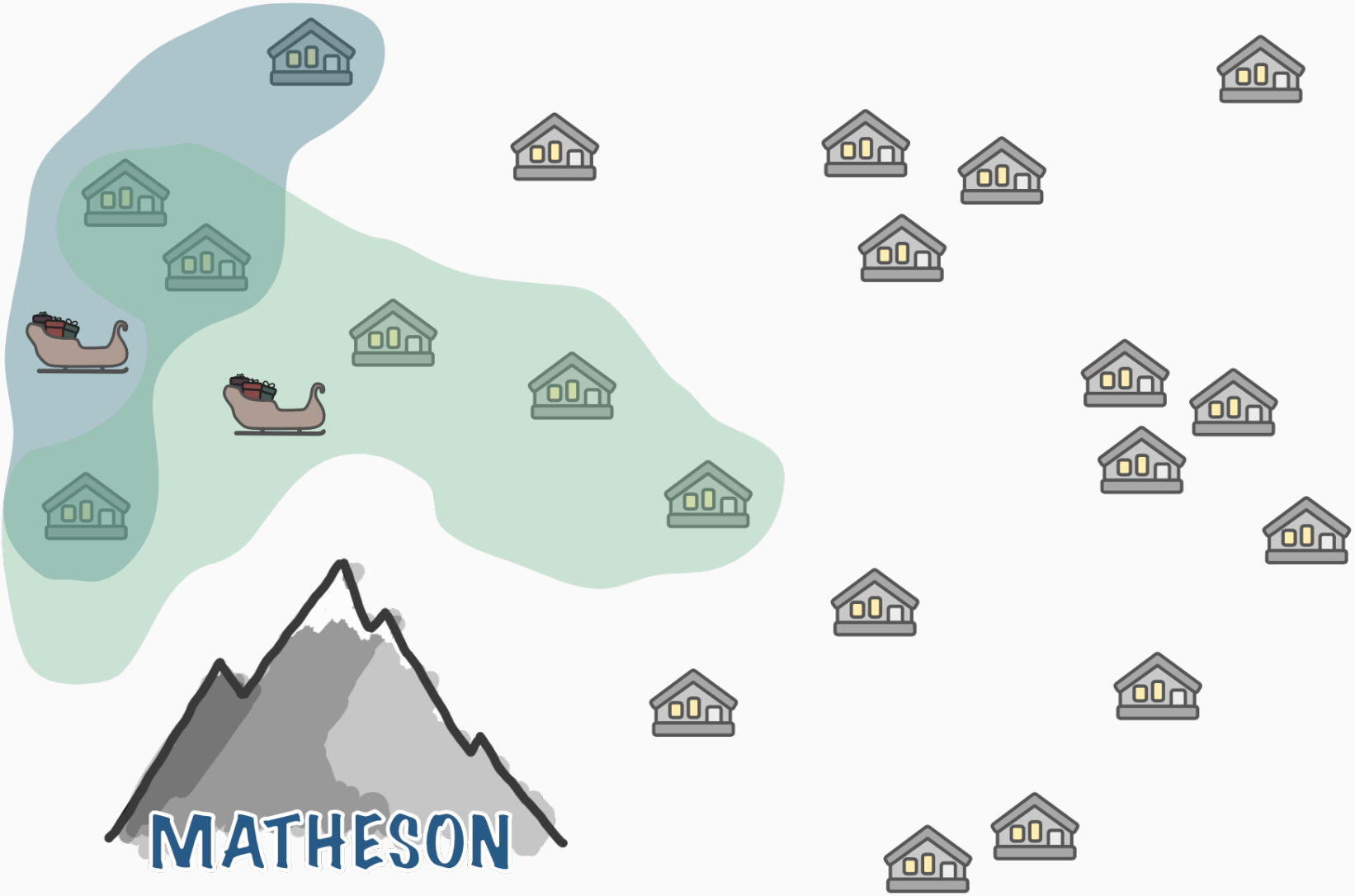
Lovász-Stein Theorem

Motivation

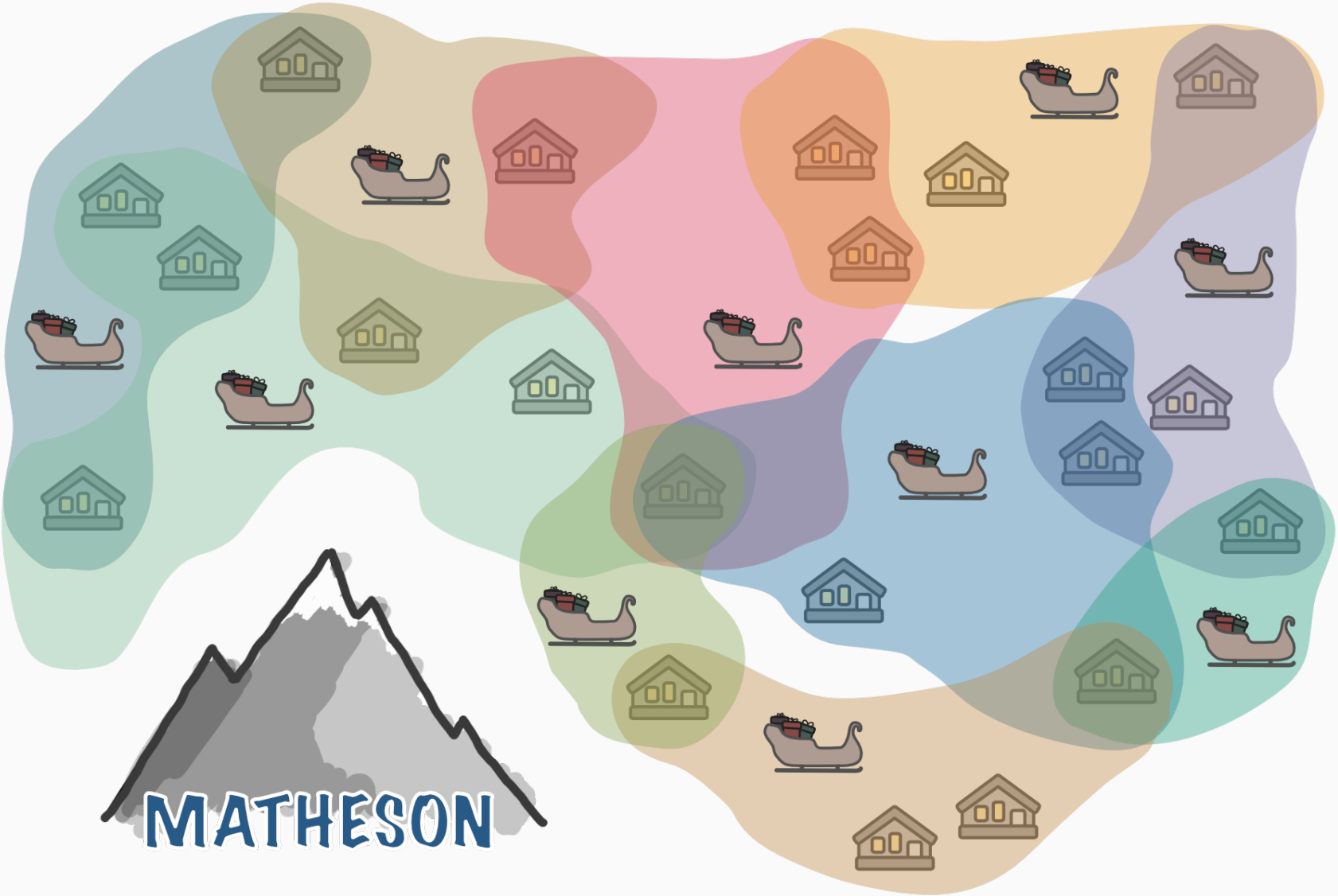
Motivation

Motivation

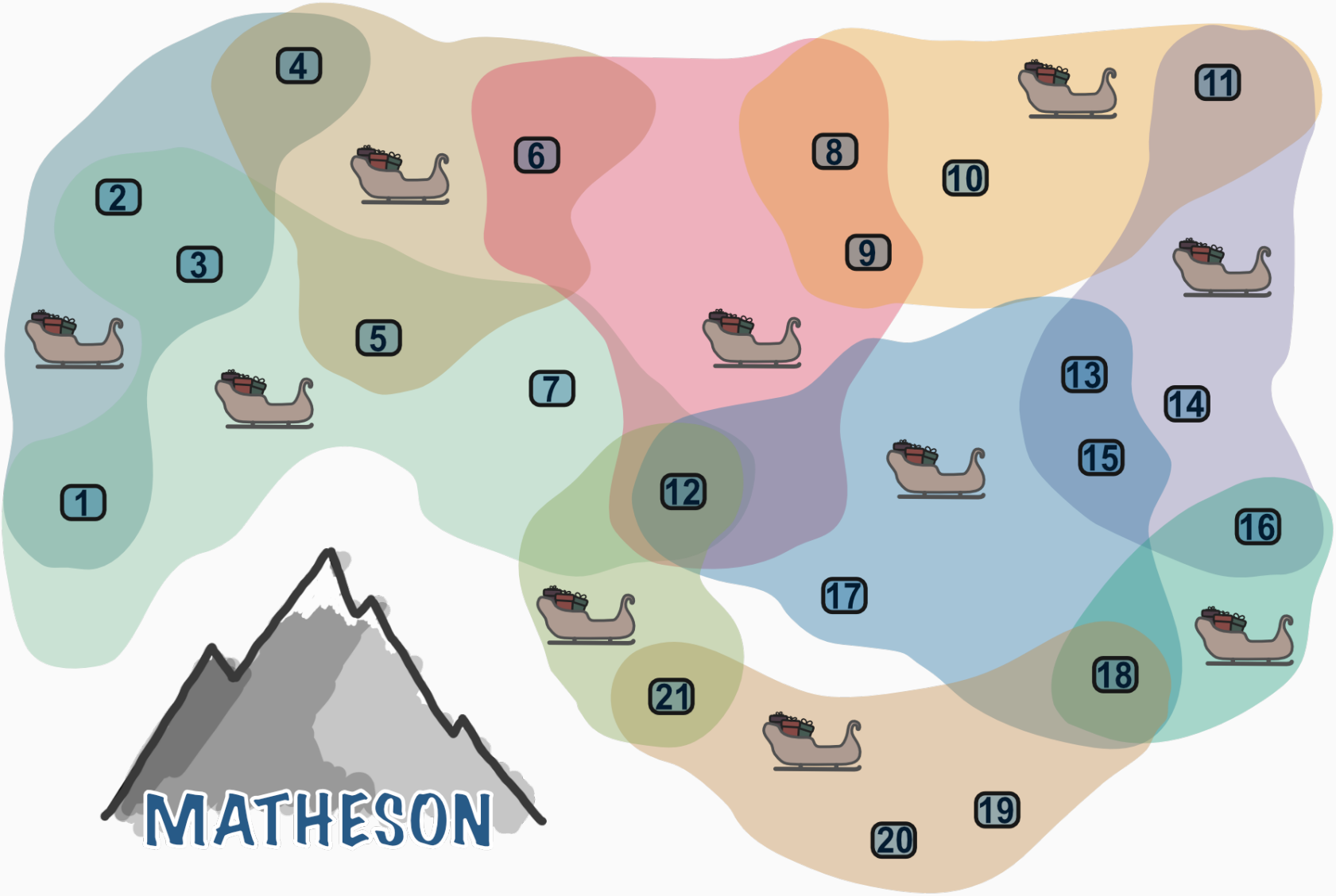
Motivation



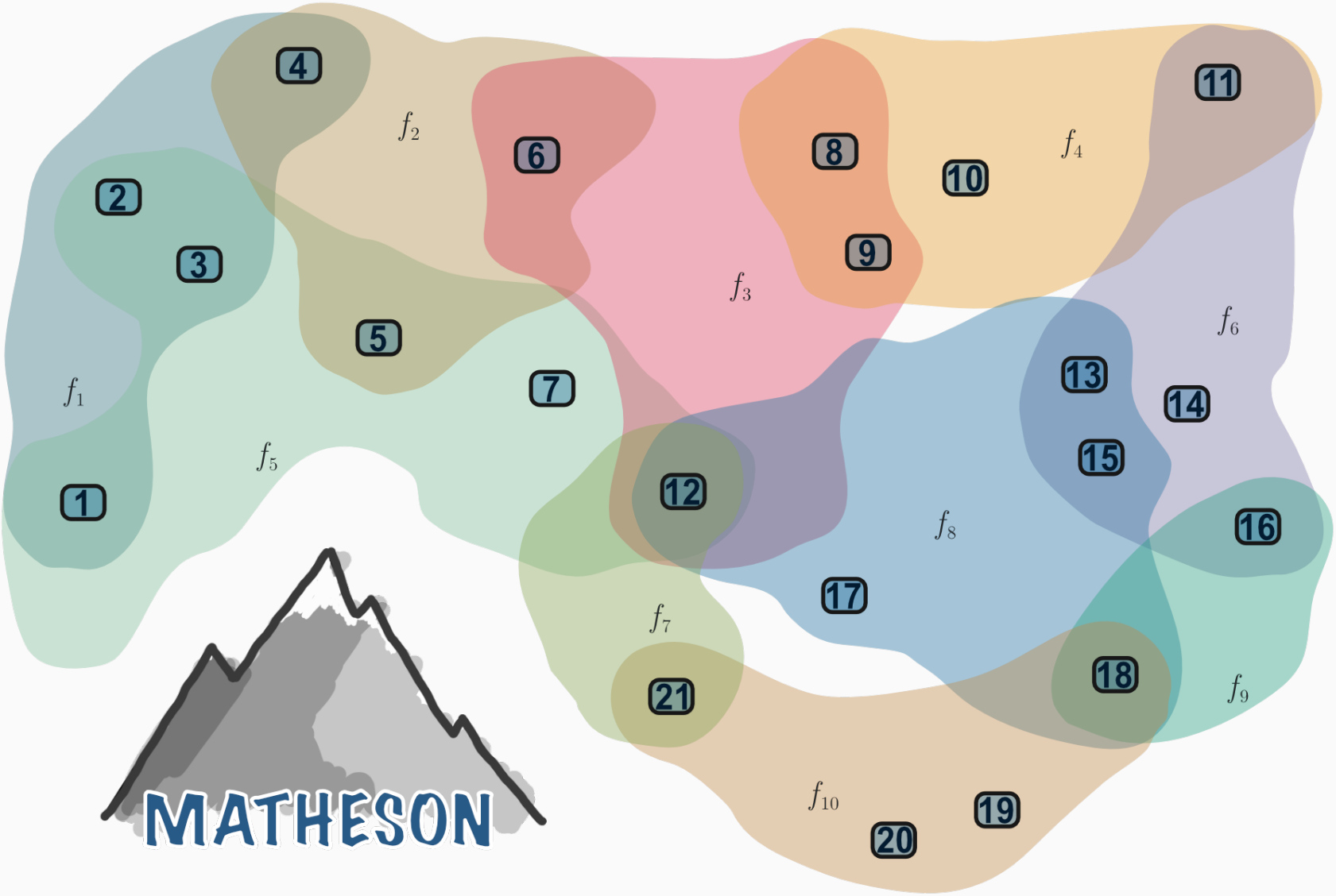
Motivation



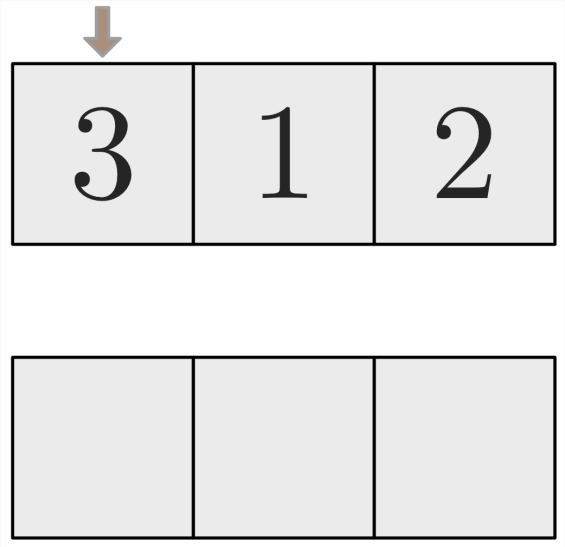
Motivation



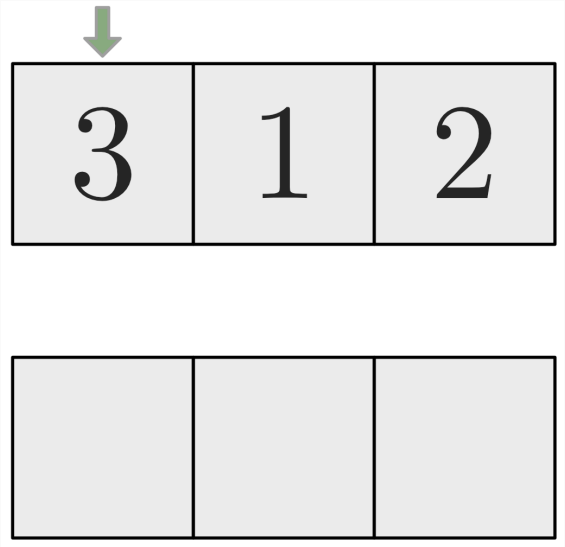
Motivation


4
f_2
6
8
10
f_4
11
2
3
9
f_3
f_6
5
7
13
14
f_1
f_5
15
12
f_8
16
1
17
f_7
18
21
f_9
f_{10}
19
20
MATHESON

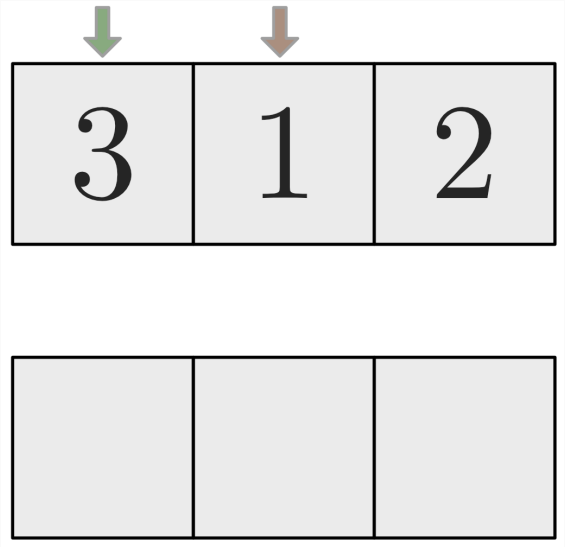
Greedy Sorting



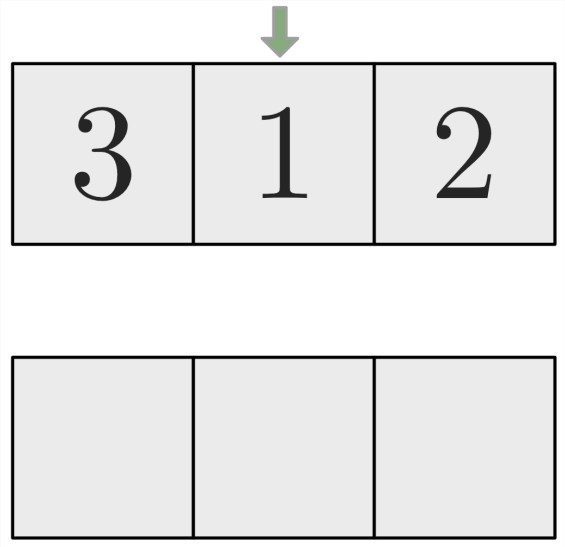
Greedy Sorting



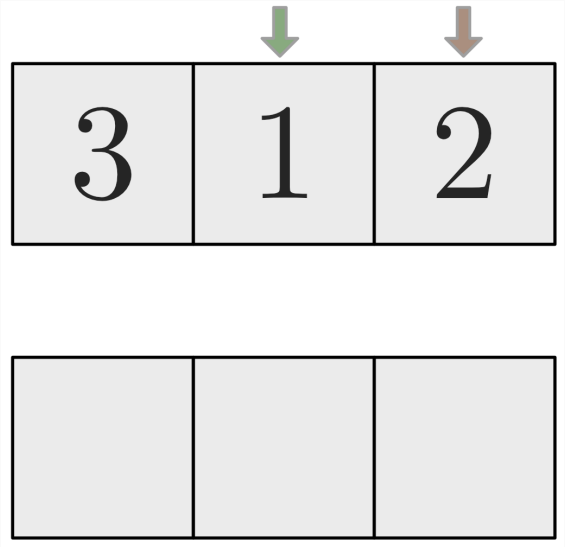
Greedy Sorting



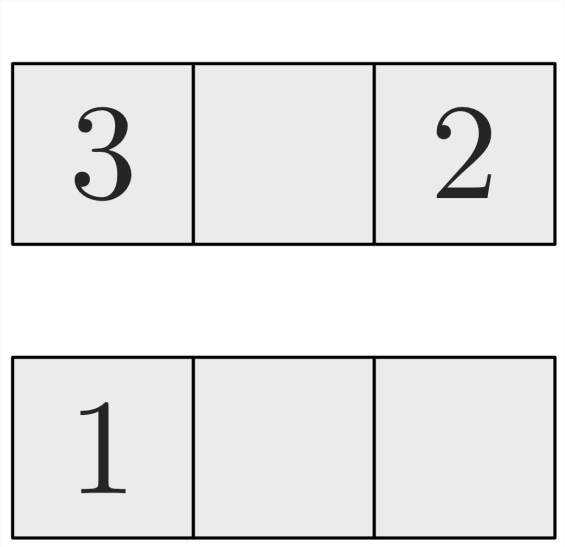
Greedy Sorting



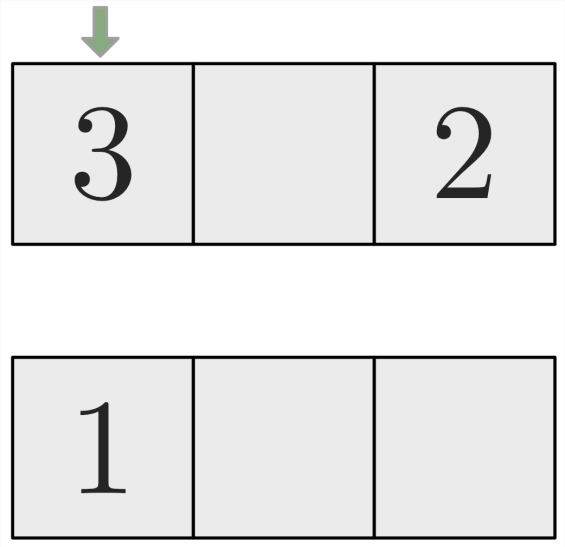
Greedy Sorting



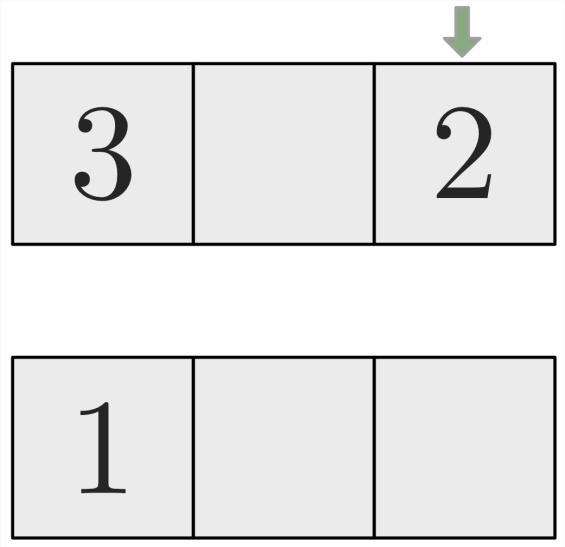
Greedy Sorting



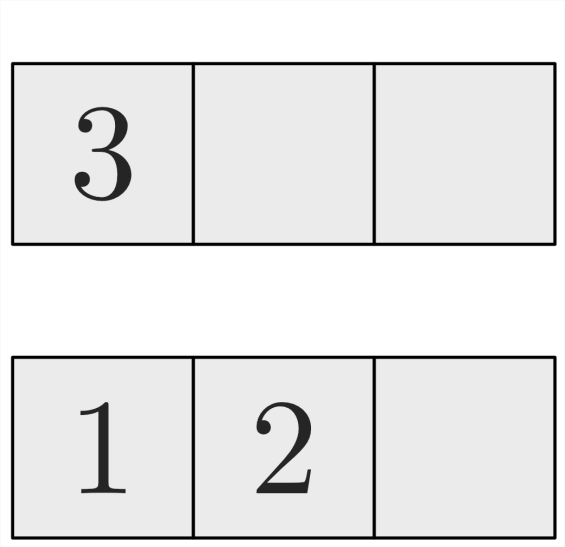
Greedy Sorting



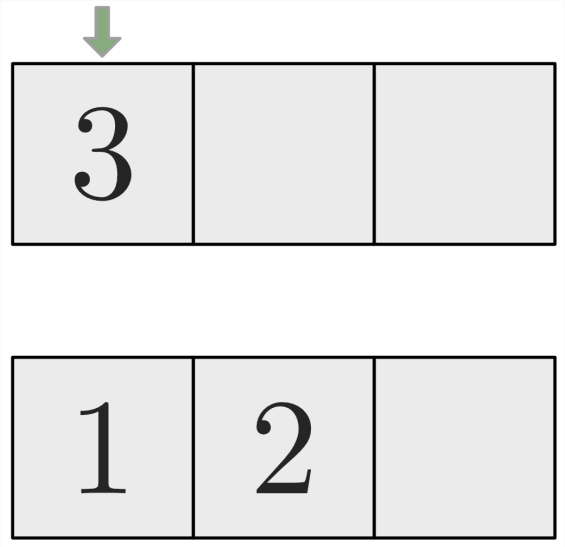
Greedy Sorting



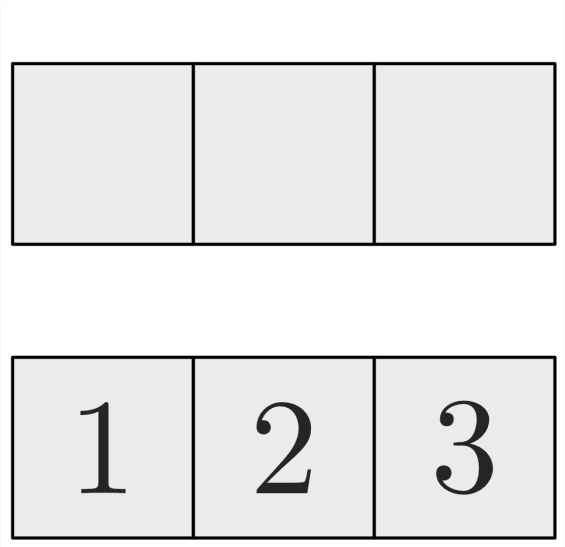
Greedy Sorting



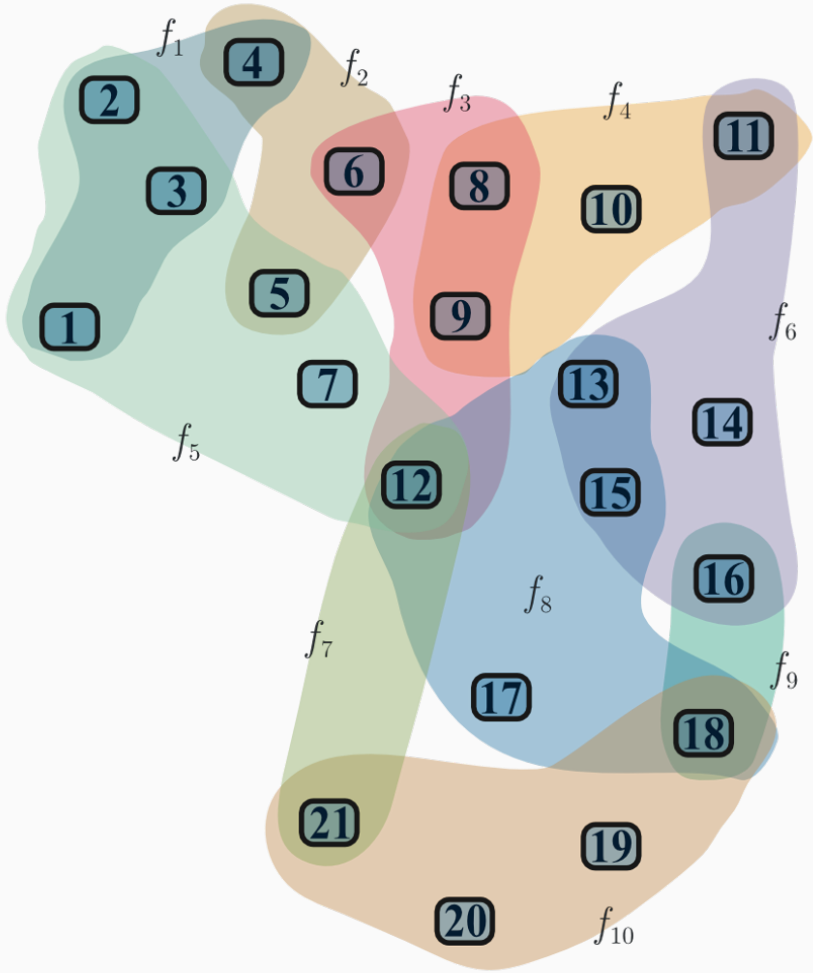
Greedy Sorting



Greedy Sorting

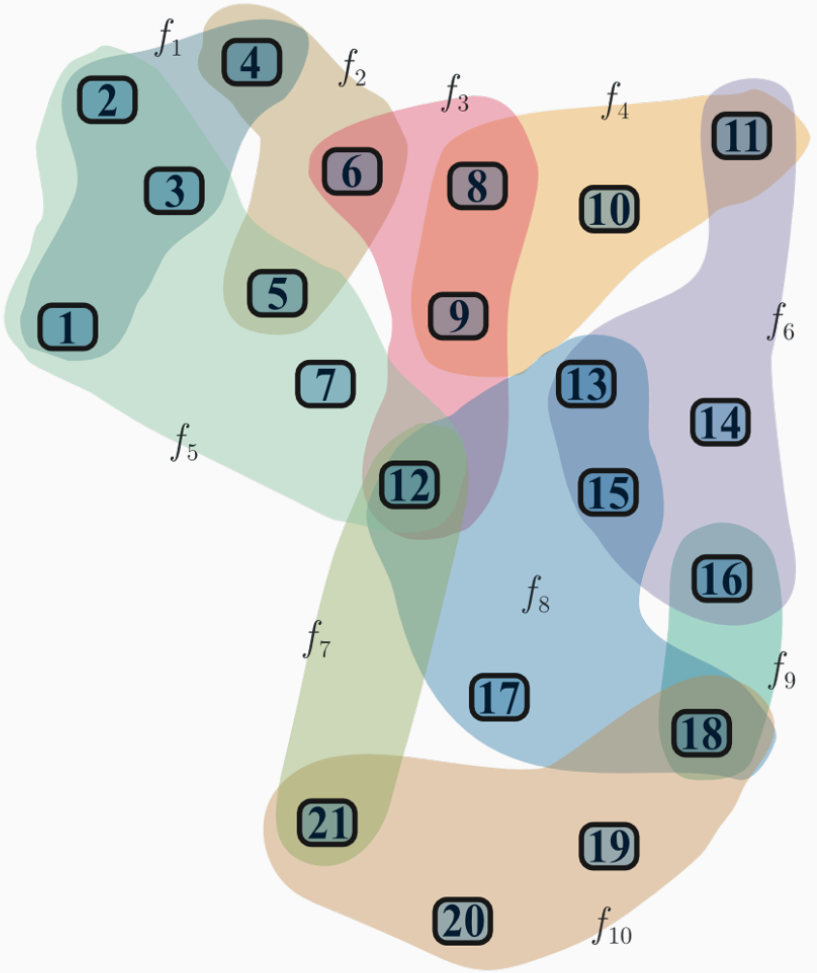


Lovász-Stein Algorithm



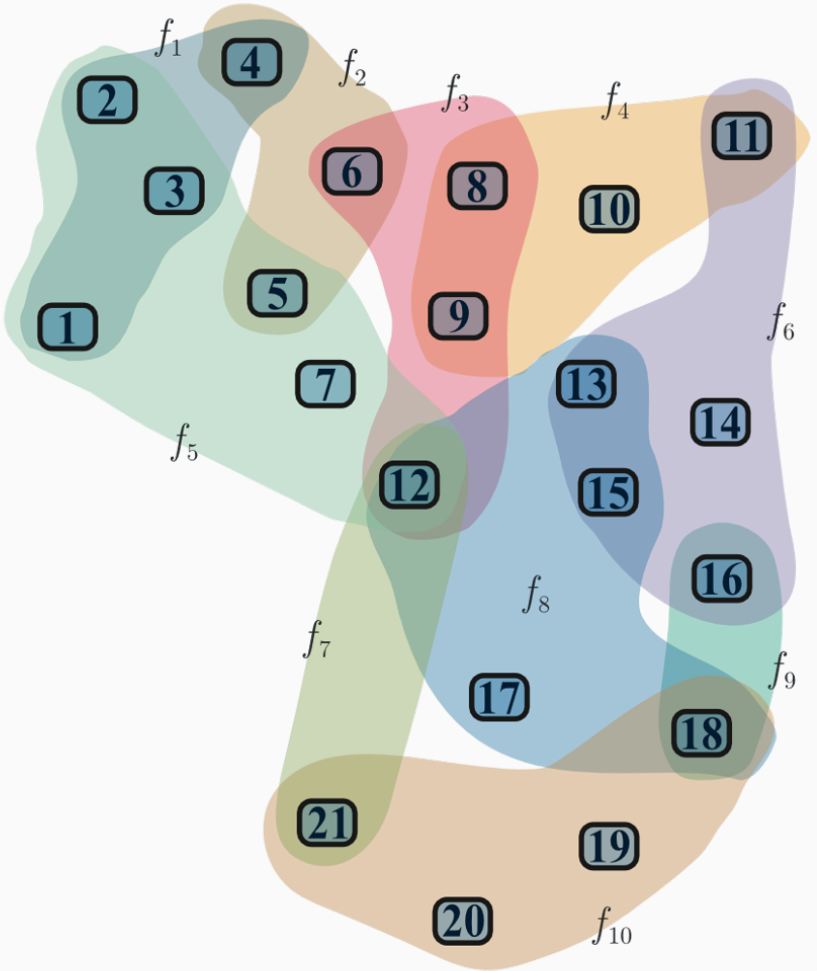
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



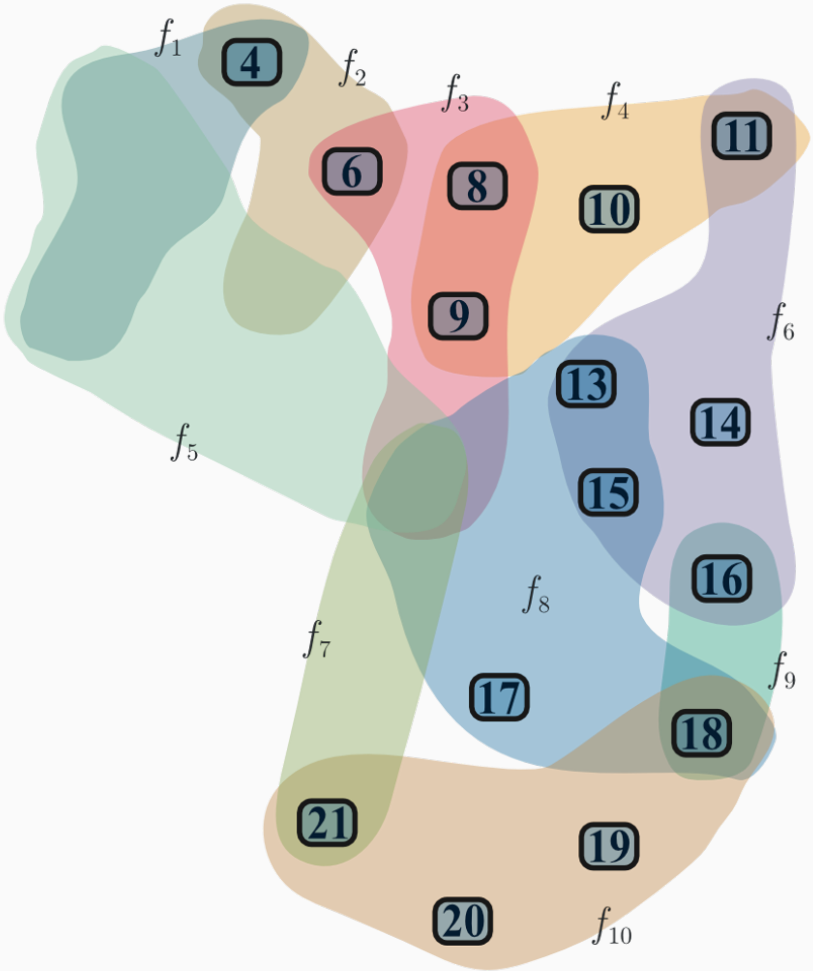
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



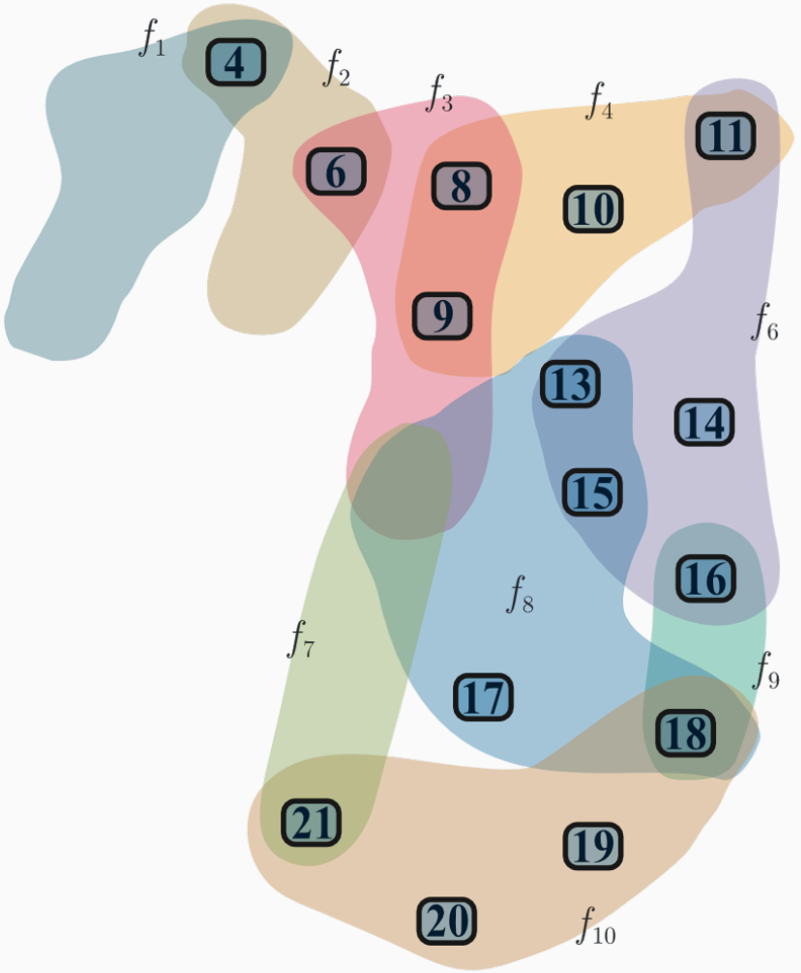
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



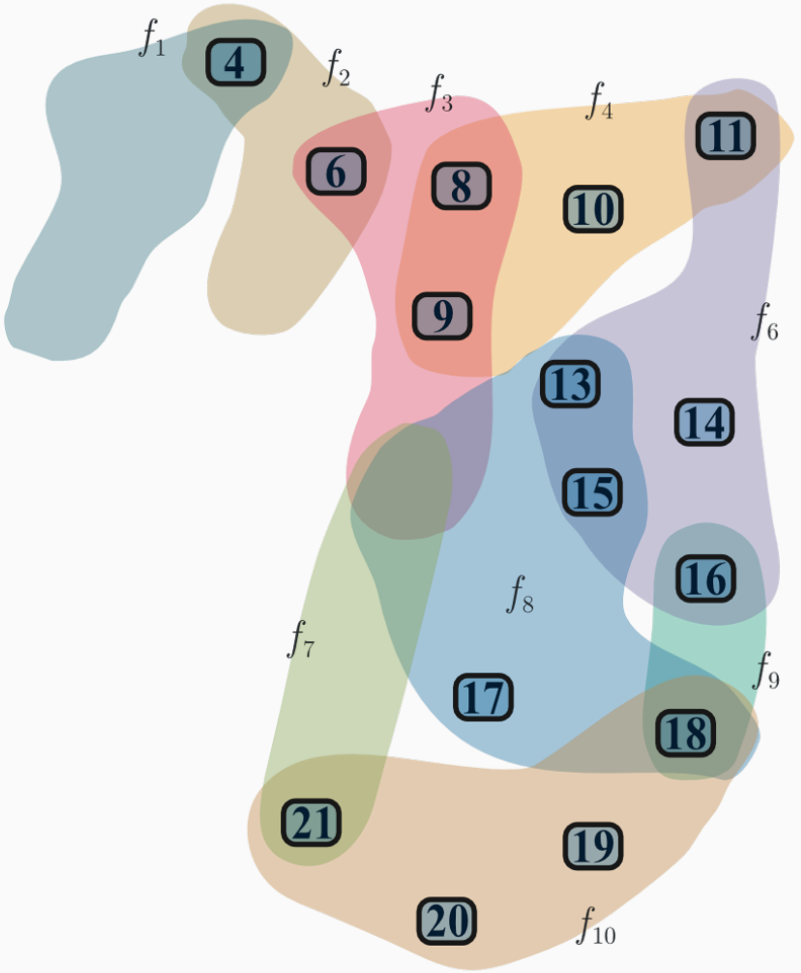
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



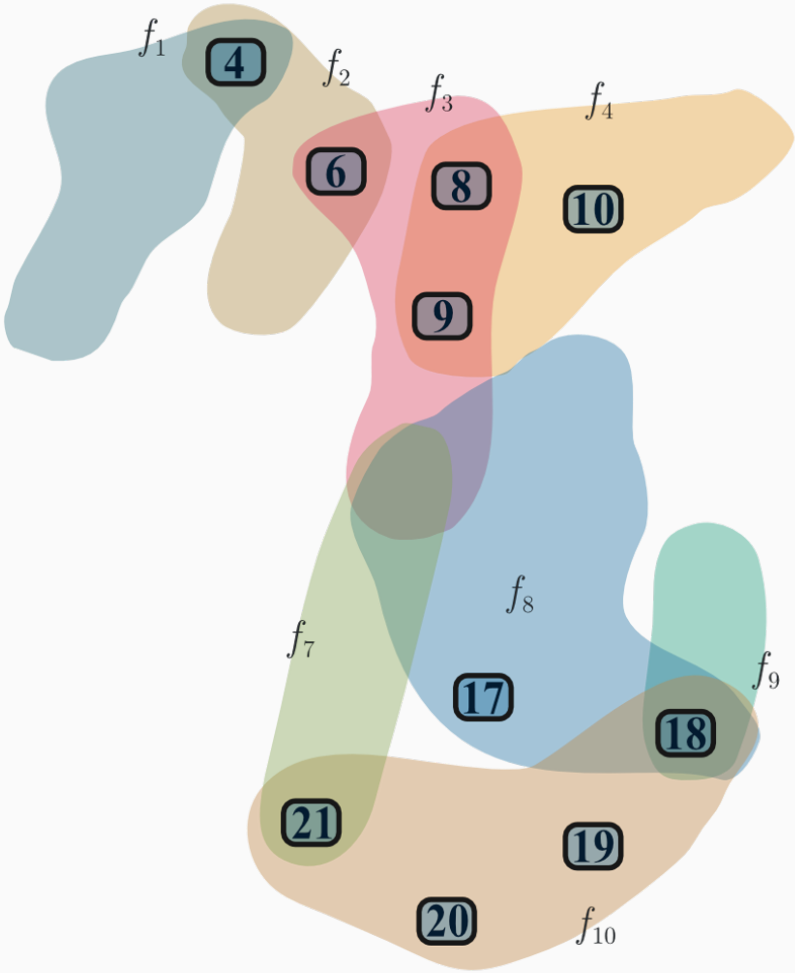
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



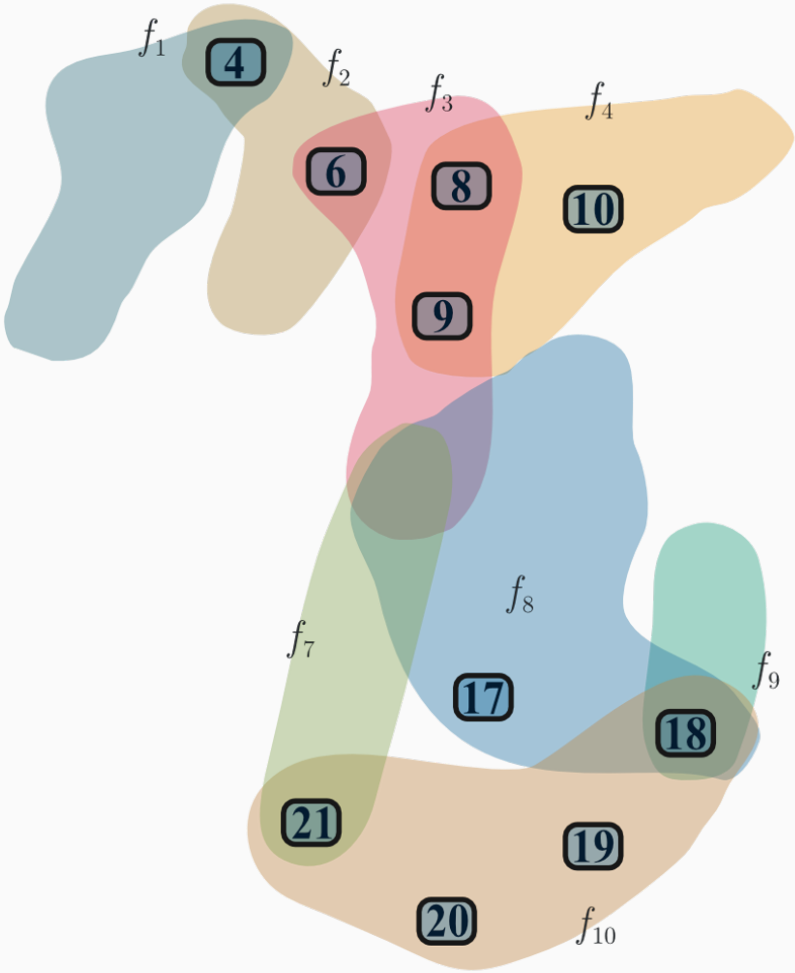
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



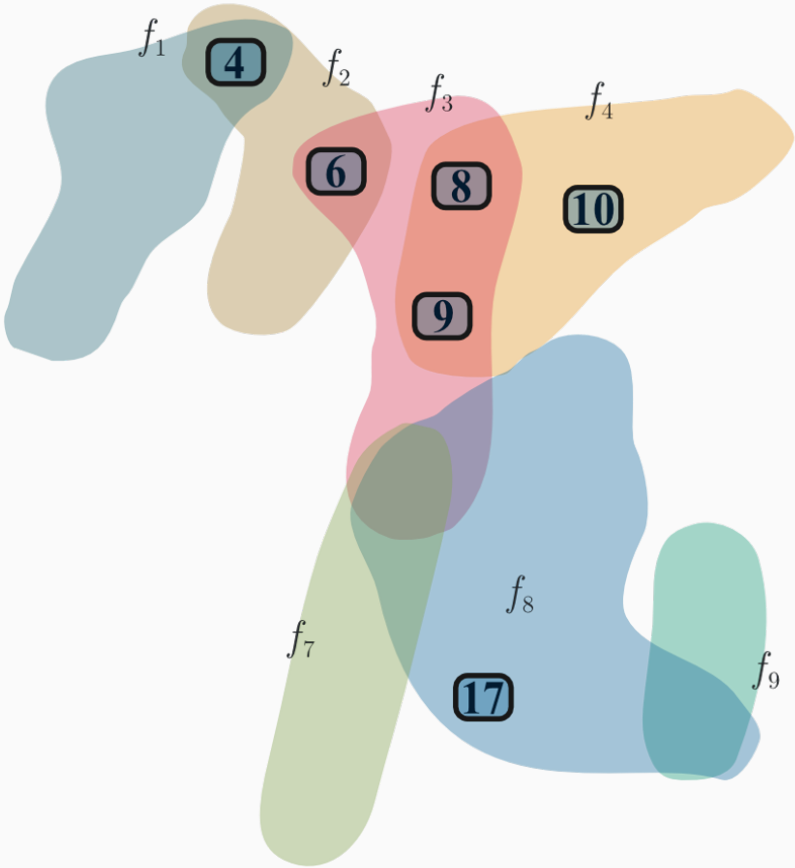
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



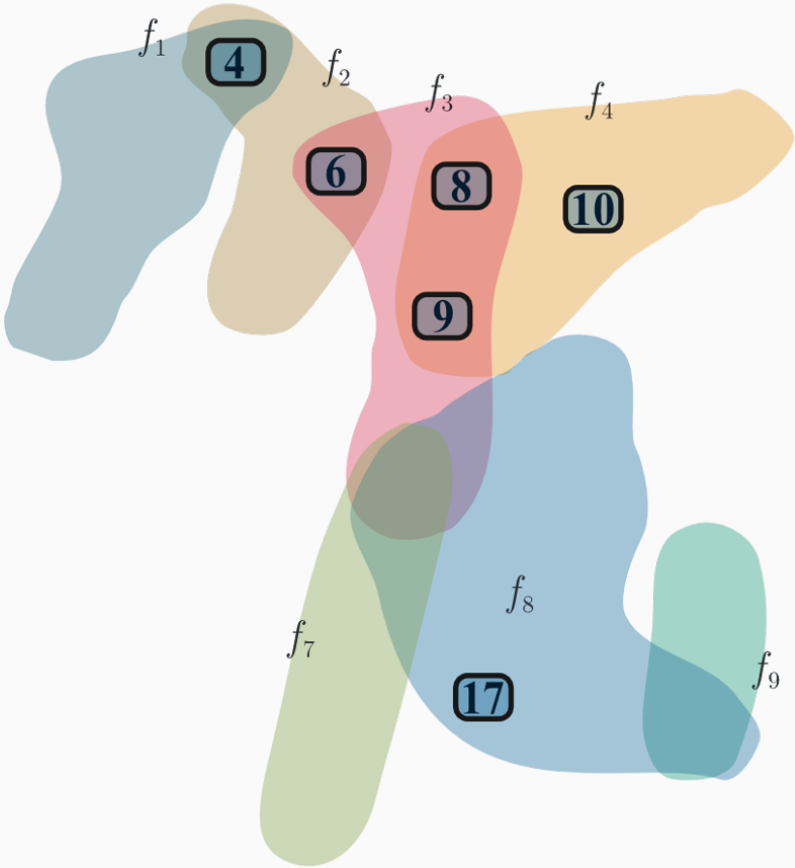
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



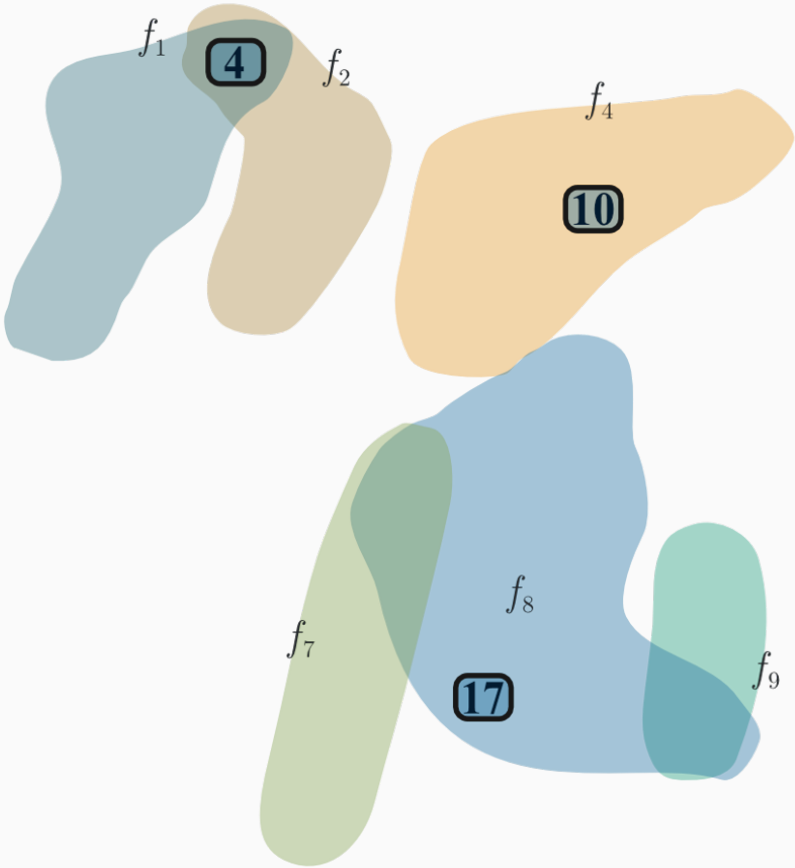
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



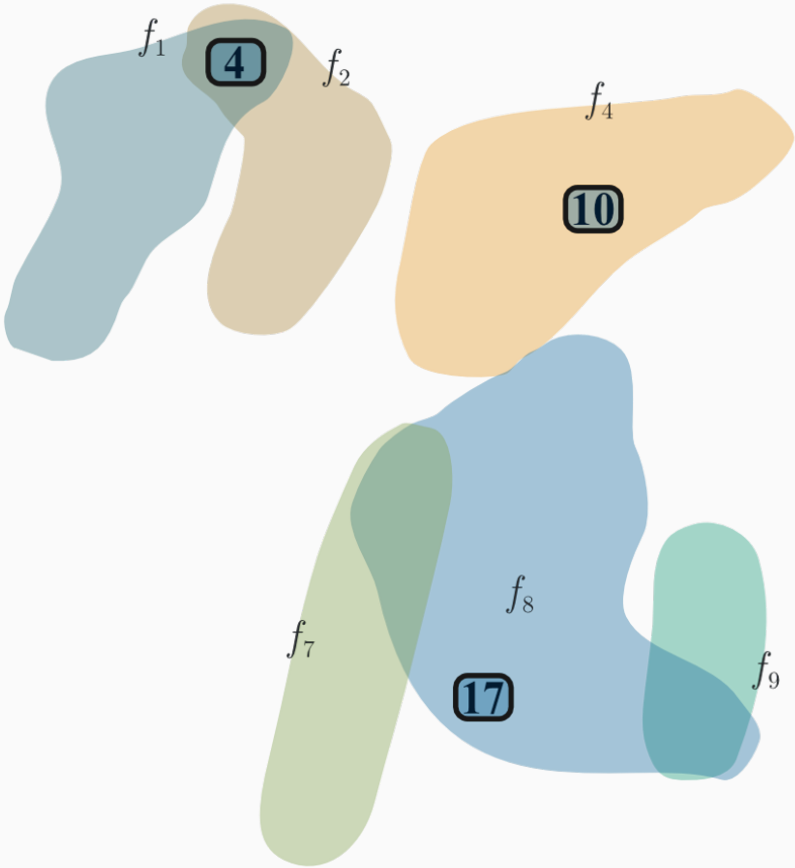
Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1



Lovász-Stein Algorithm

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
1	1				1					
2	1				1					
3	1				1					
4	1	1								
5		1			1					
6		1	1							
7					1					
8			1	1						
9			1	1						
10					1					
11				1		1				
12			1		1		1	1		
13						1		1		
14						1				
15						1		1		
16						1			1	
17								1		
18								1	1	1
19										1
20										1
21							1			1

Memos

Definitions:

Memos

Definitions:

- K is the size of the covering found by LSA

Memos

Definitions:

- K is the size of the covering found by LSA
- K_i is the number of edges found in step i

Memos

Definitions:

- K is the size of the covering found by LSA
- K_i is the number of edges found in step i
- a is an upper bound for number of 1's in a column

Memos

Definitions:

- K is the size of the covering found by LSA
- K_i is the number of edges found in step i
- a is an upper bound for number of 1's in a column
- v is a lower bound for the number of 1's in a row

Memos

Definitions:

- K is the size of the covering found by LSA
- K_i is the number of edges found in step i
- a is an upper bound for number of 1's in a column
- v is a lower bound for the number of 1's in a row

Lemmas:

(1) $Cov(\mathcal{G}) \leq \sum_{n=1}^{a} K_i$

Memos

Definitions:

- K is the size of the covering found by LSA
- K_i is the number of edges found in step i
- a is an upper bound for number of 1's in a column
- v is a lower bound for the number of 1's in a row

Lemmas:

(1) $Cov(\mathcal{G}) \leq \sum_{n=1}^{a} K_i$

(2) $K_i = \frac{k_{i+1}+k_i}{i}$

Memos

Definitions:

- K is the size of the covering found by LSA
- K_i is the number of edges found in step i
- a is an upper bound for number of 1's in a column
- v is a lower bound for the number of 1's in a row

Lemmas:

(1) $Cov(\mathcal{G}) \leq \sum_{n=1}^{a} K_i$

(2) $K_i = \frac{k_{i+1}+k_i}{i}$

(3) $k_i \leq \frac{(i-1)M}{v}$

Lovász-Stein-Theorem

Lovász-Stein Theorem (1974)

If each member of $\mathcal{F}$ has at most a elements, and each point belongs to at least v of the sets in $\mathcal{F}$, then

$$Cov(\mathcal{G}) \leq \frac{|\mathcal{F}|}{v}(1 + \ln(a)).$$

Hashing

Hash Functions Motivation

How fast can $x \overset{?}{\in} M$ be decided by an algorithm?

Hash Functions Motivation

How fast can $x \stackrel{?}{\in} M$ be decided by an algorithm?

Some ideas

(1) For each $m \in M$ check $x \stackrel{?}{=} m$.

Hash Functions Motivation

How fast can $x \stackrel{?}{\in} M$ be decided by an algorithm?

Some ideas

(1) For each $m \in M$ check $x \stackrel{?}{=} m$.

(2) If M is sortable, all elements can be stored in a binary tree of height $\log(|M|)$. Traverse the tree, checking $x \stackrel{?}{=} m$ for all visited elements m.

Hash Functions Motivation

How fast can $x \stackrel{?}{\in} M$ be decided by an algorithm?

Some ideas

(1) For each $m \in M$ check $x \stackrel{?}{=} m$.

(2) If M is sortable, all elements can be stored in a binary tree of height $\log(|M|)$. Traverse the tree, checking $x \stackrel{?}{=} m$ for all visited elements m.

We get a linear-time algorithm for (1) and a logarithmic-time algorithm for (2).

Hash Functions Motivation

How fast can $x \stackrel{?}{\in} M$ be decided by an algorithm?

Some ideas

(1) For each $m \in M$ check $x \stackrel{?}{=} m$.

(2) If M is sortable, all elements can be stored in a binary tree of height $\log(|M|)$. Traverse the tree, checking $x \stackrel{?}{=} m$ for all visited elements m.

We get a linear-time algorithm for (1) and a logarithmic-time algorithm for (2).

Question: How close can we get to constant-time?

Hash Functions

Idea:

- Use a function $h : M \to [n]$,

Hash Functions

Idea:

- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,

Hash Functions

Idea:

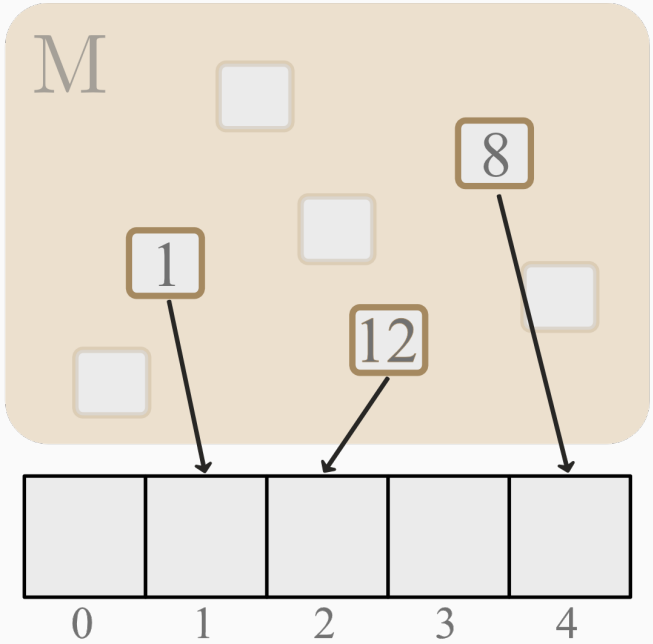
- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,
- to map elements of M to positions $1, \ldots, n$.

Hash Functions

Idea:

- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,
- to map elements of M to positions $1, \ldots, n$.

Example ($h(x) = x \bmod 5$):

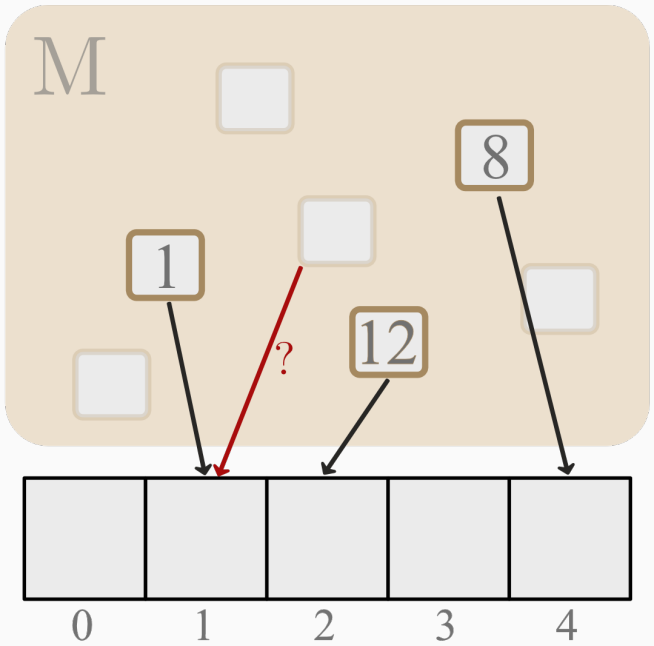


Hash Functions

Idea:

- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,
- to map elements of M to positions $1, \ldots, n$.

Example ($h(x) = x \bmod 5$):



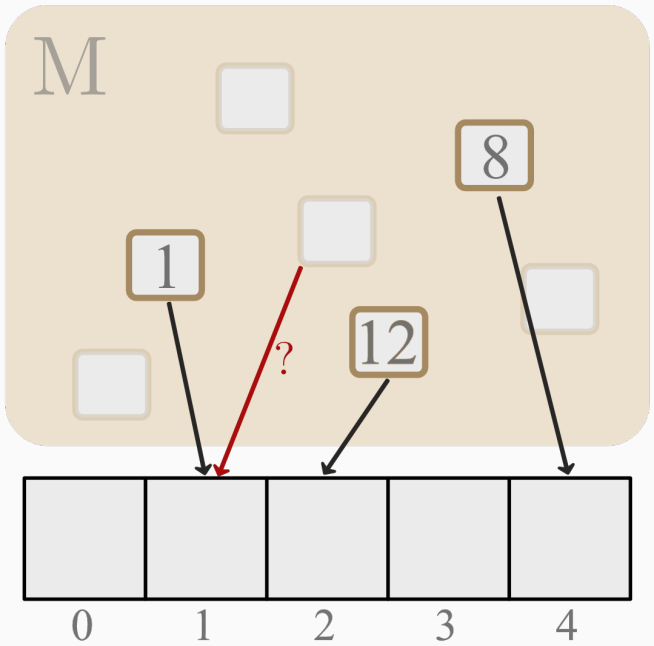
- What happens if two items get mapped to the same position?

Hash Functions

Idea:

- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,
- to map elements of M to positions $1, \ldots, n$.

Example ($h(x) = x \bmod 5$):



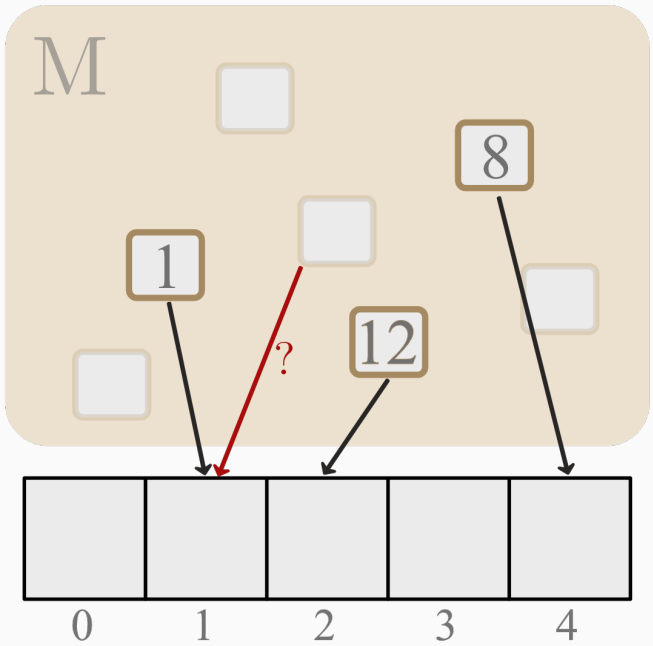
- What happens if two items get mapped to the same position?
- Time-complexity?

Hash Functions

Idea:

- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,
- to map elements of M to positions $1, \ldots, n$.

Example ($h(x) = x \bmod 5$):



- What happens if two items get mapped to the same position?
- Time-complexity?

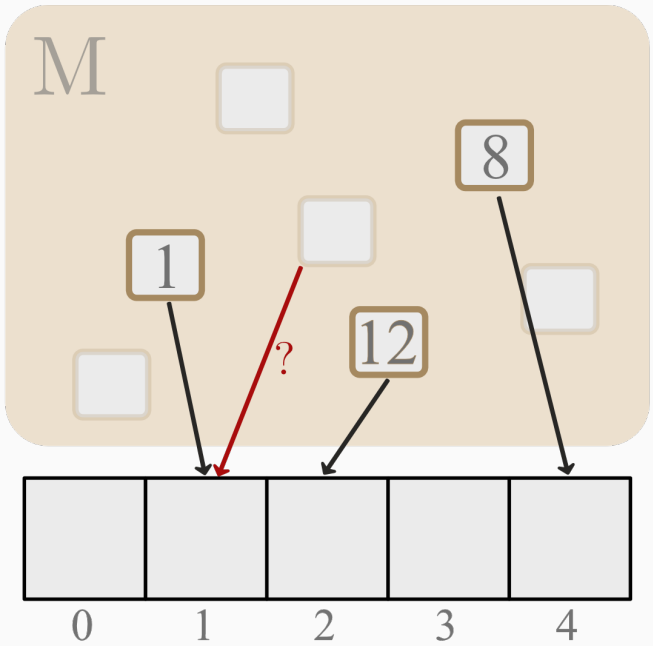
$\to$ Hashing with linear probing on sufficiently large arrays results in *expected constant time*.

Hash Functions

Idea:

- Use a function $h : M \to [n]$,
- that can be evaluated in $\mathcal{O}(1)$ time,
- to map elements of M to positions $1, \ldots, n$.

Example ($h(x) = x \bmod 5$):



- What happens if two items get mapped to the same position?
- Time-complexity?

$\to$ Hashing with linear probing on sufficiently large arrays results in *expected constant time*.

Question: Can we get to constant-time?

Perfect Hash Functions

Let $\mathcal{F}$ be a family of M-many functions $f : X \to Y$, where $|X| = n$, $|Y| = m$.

Perfect Hash Functions

Let $\mathcal{F}$ be a family of M-many functions $f : X \to Y$, where $|X| = n$, $|Y| = m$.

Definition. The family $\mathcal{F}$ is a perfect hashing family (PHF), if $\forall C \subseteq X, |C| = w : \exists f \in \mathcal{F} : f|_C$ is injective.

Perfect Hash Functions

Let $\mathcal{F}$ be a family of M-many functions $f : X \to Y$, where $|X| = n$, $|Y| = m$.

Definition. The family $\mathcal{F}$ is a perfect hashing family (PHF), if $\forall C \subseteq X, |C| = w : \exists f \in \mathcal{F} : f|_C$ is injective.

Theorem 2 (2011)

There exists a perfect hashing family with

$$M \leq \frac{n^w}{w!\binom{n}{w}}\left(1 + \ln\binom{m}{w}\right).$$

Thank You!

And that's a good place to stop.

Questions?

References i

Dameng Deng, Yuan Zhang, P. Li, and G. H. van Rees.
The stein-lovasz theorem and its applications to some combinatorial arrays.
JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing, 77, 05 2011.

Stasys Jukna.
Extremal Combinatorics : With Applications in Computer Science.
Texts in Theoretical Computer Science. An EATCS SeriesSpringerLink. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

References ii

Robin Link.
Lovász-stein presentation and algorithms.
https://github.com/RearrangedLetters/LovaszSteinCover, 2021.

The Manim Community Developers.
Manim – Mathematical Animation Framework, 12 2021.