Generalised Pattern Matching Revisited

Bartłomiej Dudek[1] Paweł Gawrychowski[1]
Tatiana Starikovskaya[2]

[1]University of Wrocław, Poland
[2]DIENS, École normale supérieure, PSL Research University, France

March 20, 2020

Pattern Matching

Input: A text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

T : `a b b a b c`

P : `a b c`

Pattern Matching

Input: A text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$

.

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

T : `a b b a b c`

P : `a b c` $\rightarrow$ 1 mismatch

Pattern Matching

Input: A text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$

.

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

T : `a b b a b c`

P : `a b c` $\rightarrow$ 2 mismatches

Pattern Matching

Input: A text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

T : `a b b a b c`

P : `    a b c` $\rightarrow$ 3 mismatches

Pattern Matching

Input: A text $T \in \Sigma^n$, a pattern $P \in \Sigma^m$

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

T : `a b b a b c`

P : `      a b c` $\rightarrow$ match

Generalized Pattern Matching

Input: A text $T \in (\Sigma_T)^n$, a pattern $P \in (\Sigma_P)^m$, and a **matching relationship** $\subseteq \Sigma_T \times \Sigma_P$.

.

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

Generalized Pattern Matching

Input: A text $T \in (\Sigma_T)^n$, a pattern $P \in (\Sigma_P)^m$, and a **matching relationship** $\subseteq \Sigma_T \times \Sigma_P$.

Output (Reporting): All i such that $T[i, i+m-1]$ matches P.
Output (Counting): For each i, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

$\Sigma_T \setminus \Sigma_P$	A	C	G	T
A	✓			
C		✓		
G				✓
T				✓

Text:	C	A	G	T	
Patterns:	C	A	T		0 mismatches
		C	A	T	2 mismatches

Size of matching relationship

Parameters describing the matching relationship:

- $\mathcal{D}$ - maximum number of characters that match a fixed character
- $\mathcal{S}$ - number of matching pairs of characters
- $\mathcal{I}$ - number of intervals of matching characters from the pattern, $\mathcal{I} = \sum_{j \in [m]} |I(P[j])|$

$\Sigma_T \setminus \Sigma_P$	A	C	G	T
A	✓		✓	
B	✓	✓	✓	
C			✓	✓
D	✓			✓
E	✓	✓		✓
F	✓		✓	✓
G			✓	

In the example: $\mathcal{D} = 5$, $\mathcal{S} = 16$, $\mathcal{I} = 2 + 2 + 2 + 1 = 7$

Size of matching relationship

Parameters describing the matching relationship:

- $\mathcal{D}$ - maximum number of characters that match a fixed character
- $\mathcal{S}$ - number of matching pairs of characters
- $\mathcal{I}$ - number of intervals of matching characters from the pattern, $\mathcal{I} = \sum_{j \in [m]} |I(P[j])|$

$\Sigma_T \setminus \Sigma_P$	A	C	G	T
A	✓		✓	
B	✓	✓	✓	
C			✓	✓
D	✓			✓
E	✓	✓		✓
F	✓		✓	✓
G			✓	

In the example: $\mathcal{D} = 5$, $\mathcal{S} = 16$, $\mathcal{I} = 2 + 2 + 2 + 1 = 7$

Size of matching relationship

Parameters describing the matching relationship:

- $\mathcal{D}$ - maximum number of characters that match a fixed character
- $\mathcal{S}$ - number of matching pairs of characters
- $\mathcal{I}$ - number of intervals of matching characters from the pattern, $\mathcal{I} = \sum_{j \in [m]} |I(P[j])|$

$\Sigma_T \setminus \Sigma_P$	A	C	G	T
A	✓		✓	
B	✓	✓	✓	
C			✓	✓
D	✓			✓
E	✓	✓		✓
F	✓		✓	✓
G			✓	

In the example: $\mathcal{D} = 5$, $\mathcal{S} = 16$, $\mathcal{I} = 2 + 2 + 2 + 1 = 7$

History of the reporting variant

Time	Det./Rand.	Author
$\mathcal{O}(\mathcal{D} \log^2 n(\lvert\Sigma_P\rvert\mathcal{D} + n \log n \log m))$	Det.	Indyk '97
$\mathcal{O}(\mathcal{D}\, n \log^6 n)$	Det.	DGS '20
$\mathcal{O}(\mathcal{D}\, n \log n \log m)$	Rand.	Muthukrishnan '95
$\mathcal{O}(\mathcal{D}\, n \log n \log m)$	Rand.	DGS '20
$\mathcal{O}((\mathcal{S}\, m \log^2 m)^{1/3} n)$	Det.	M. and R. '95
$\mathcal{O}(\sqrt{\mathcal{S}}\, n \log^{7/2} n)$	Det.	DGS '20
$\mathcal{O}(\sqrt{\mathcal{S}}\, n \log m \sqrt{\log n})$	Rand.	DGS '20
$\Omega(\mathcal{S})$	Rand.	DGS '20
$\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3} n \sqrt{\log m})$	Det.	Muthukrishnan '95
$\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$	Det.	DGS '20

History of the reporting variant

Time	Det./Rand.	Author
$\tilde{\mathcal{O}}(\lvert\Sigma_P\rvert\mathcal{D}^2 + \mathcal{D}\,n)$	Det.	Indyk '97
$\tilde{\mathcal{O}}(\mathcal{D}\,n)$	Det.	DGS '20
$\tilde{\mathcal{O}}(\mathcal{D}\,n)$	Rand.	Muthukrishnan '95
$\tilde{\mathcal{O}}(\mathcal{D}\,n)$	Rand.	DGS '20
$\tilde{\mathcal{O}}((\mathcal{S}\,m)^{1/3} n)$	Det.	M. and R. '95
$\tilde{\mathcal{O}}(\sqrt{\mathcal{S}}\,n)$	Det.	DGS '20
$\tilde{\mathcal{O}}(\sqrt{\mathcal{S}}\,n)$	Rand.	DGS '20
$\Omega(\mathcal{S})$	Rand.	DGS '20
$\tilde{\mathcal{O}}(\mathcal{I} + (m\mathcal{I})^{1/3} n)$	Det.	Muthukrishnan '95
$\tilde{\mathcal{O}}(n\sqrt{\mathcal{I}})$	Det.	DGS '20

History of the counting variant

Time	Det./Rand.	Approx.	Author
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}^2 n \log^3 n)$	Det.	$(1-\varepsilon)$	Indyk '97
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n \log^6 n)$	Det.	$(1-\varepsilon)$	DGS '20
$\mathcal{O}(\mathcal{D}\, n \log n \log m)$	Rand.	$\log m$	Muthukrishnan '95
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n \log^3 n)$	Rand.	$(1-\varepsilon)$	Indyk '97
$\mathcal{O}(\varepsilon^{-1}\mathcal{D}\, n \log n \log m)$	Rand.	$(1-\varepsilon)$	DGS '20
$\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n \log^{7/2} n)$	Det.	$(1-\varepsilon)$	DGS '20
$\mathcal{O}(\sqrt{\varepsilon^{-1}\mathcal{S}}\, n \log m \sqrt{\log n})$	Rand.	$(1-\varepsilon)$	DGS '20
$\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3} n \sqrt{\log m})$	Det.	—	Muthukrishnan '95
$\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$	Det.	—	DGS '20

History of the counting variant

Time	Det./Rand.	Approx.	Author
$\tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D}^2 n)$	Det.	$(1-\varepsilon)$	Indyk '97
$\tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D}\, n)$	Det.	$(1-\varepsilon)$	DGS '20
$\tilde{\mathcal{O}}(\mathcal{D}\, n)$	Rand.	$\log m$	Muthukrishnan '95
$\tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D}\, n)$	Rand.	$(1-\varepsilon)$	Indyk '97
$\tilde{\mathcal{O}}(\varepsilon^{-1}\mathcal{D}\, n)$	Rand.	$(1-\varepsilon)$	DGS '20
$\tilde{\mathcal{O}}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n)$	Det.	$(1-\varepsilon)$	DGS '20
$\tilde{\mathcal{O}}(\sqrt{\varepsilon^{-1}\mathcal{S}}\, n)$	Rand.	$(1-\varepsilon)$	DGS '20
$\tilde{\mathcal{O}}(\mathcal{I} + (m\mathcal{I})^{1/3} n)$	Det.	—	Muthukrishnan '95
$\tilde{\mathcal{O}}(n\sqrt{\mathcal{I}})$	Det.	—	DGS '20

Reporting variant of GPM

Recall: $\mathcal{S}$ is the total number of matching pairs (✓)

Character $b \in \Sigma_P$ is *heavy* if it matches at least $\sqrt{\mathcal{S}}$ characters $a \in \Sigma_T$.

Light characters match $< \sqrt{\mathcal{S}}$ characters, so $\mathcal{D} = \sqrt{\mathcal{S}}$

Mismatches due to light characters

Run the $\tilde{O}(\mathcal{D}\, n)$-time algorithm for $\mathcal{D} = \sqrt{\mathcal{S}}$.

Reporting variant of GPM

Recall: $\mathcal{S}$ is the total number of matching pairs (✓)

Character $b \in \Sigma_P$ is *heavy* if it matches at least $\sqrt{\mathcal{S}}$ characters $a \in \Sigma_T$.

Light characters match $< \sqrt{\mathcal{S}}$ characters, so $\mathcal{D} = \sqrt{\mathcal{S}}$

Mismatches due to light characters

Run the $\tilde{\mathcal{O}}(\mathcal{D}\, n)$-time algorithm for $\mathcal{D} = \sqrt{\mathcal{S}}$.

Reporting variant of GPM

Recall: $\mathcal{S}$ is the total number of matching pairs (✓)

Character $b \in \Sigma_P$ is *heavy* if it matches at least $\sqrt{\mathcal{S}}$ characters $a \in \Sigma_T$.

Light characters match $< \sqrt{\mathcal{S}}$ characters, so $\mathcal{D} = \sqrt{\mathcal{S}}$

Mismatches due to light characters

Run the $\tilde{\mathcal{O}}(\mathcal{D}\, n)$-time algorithm for $\mathcal{D} = \sqrt{\mathcal{S}}$.

Mismatches due to heavy characters

Character $b \in \Sigma_P$ is *heavy* if it matches at least $\sqrt{\mathcal{S}}$ characters $a \in \Sigma_T$.

$\rightarrow$ there are at most $\sqrt{\mathcal{S}}$ of them

Create an instance of Pattern Matching with Don't Cares for every heavy character $b \in \Sigma_P$:

$$T_b[j] = \begin{cases} 0 & \text{if } T[j] \not\approx b, \\ ? & \text{otherwise.} \end{cases} \qquad P_b[j] = \begin{cases} 1 & \text{if } P[j] = b, \\ ? & \text{otherwise.} \end{cases}$$

[Clifford, Clifford, '07] Pattern Matching with Don't Cares can be solved in $\tilde{\mathcal{O}}(n)$ time.

$\tilde{\mathcal{O}}(\sqrt{\mathcal{S}}\, n)$ rand. time algorithm for reporting variant of GPM.

Mismatches due to heavy characters

Character $b \in \Sigma_P$ is *heavy* if it matches at least $\sqrt{\mathcal{S}}$ characters $a \in \Sigma_T$.

$\rightarrow$ there are at most $\sqrt{\mathcal{S}}$ of them

Create an instance of Pattern Matching with Don't Cares for every heavy character $b \in \Sigma_P$:

$$T_b[j] = \begin{cases} 0 & \text{if } T[j] \not\approx b, \\ ? & \text{otherwise.} \end{cases} \qquad P_b[j] = \begin{cases} 1 & \text{if } P[j] = b, \\ ? & \text{otherwise.} \end{cases}$$

Text:	b	a	c	b	a	c	$a \not\approx b$	?	0	?	?	0	?
Pattern:	b	b	a				$\overrightarrow{c \approx b}$	1	1	?			

[Clifford, Clifford, '07] Pattern Matching with Don't Cares can be solved in $\tilde{\mathcal{O}}(n)$ time.

$\tilde{\mathcal{O}}(\sqrt{\mathcal{S}}\, n)$ rand. time algorithm for reporting variant of GPM.

Mismatches due to heavy characters

Character $b \in \Sigma_P$ is *heavy* if it matches at least $\sqrt{\mathcal{S}}$ characters $a \in \Sigma_T$.

$\rightarrow$ there are at most $\sqrt{\mathcal{S}}$ of them

Create an instance of PATTERN MATCHING WITH DON'T CARES for every heavy character $b \in \Sigma_P$:

$$T_b[j] = \begin{cases} 0 & \text{if } T[j] \not\approx b, \\ ? & \text{otherwise.} \end{cases} \qquad P_b[j] = \begin{cases} 1 & \text{if } P[j] = b, \\ ? & \text{otherwise.} \end{cases}$$

Text:	b	a	c	b	a	c	$a \not\approx b$	?	0	?	?	0	?
Pattern:	b	b	a				$\xrightarrow{}$ $c \approx b$	1	1	?			

[Clifford, Clifford, '07] PATTERN MATCHING WITH DON'T CARES can be solved in $\tilde{\mathcal{O}}(n)$ time.

$\tilde{\mathcal{O}}(\sqrt{\mathcal{S}}\, n)$ rand. time algorithm for reporting variant of GPM.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. alg. for reporting variant of GPM

Use 2-wise independent hash function $h : \Sigma_T \rightarrow [\mathcal{D}]$
$h(x) \approx (a \cdot x + b) \mod \mathcal{D}$

Create a new matching relationship M':

$$M'[h(a), b] = \checkmark \quad \text{iff} \quad \exists_{a \in \Sigma_T} M[a, b] = \checkmark$$

Fact: If $M[a, b] \neq \checkmark$ then $Pr(M'[h(a), b] = \checkmark) \leq 1/2$

Eliminate every non-occurrence of P in T with prob. $\geq 1/2$ and use PATTERN MATCHING WITH DON'T CARES to solve GPM for M'.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. time algorithm for reporting variant of GPM.

Same complexity in [Muthukrishnan, '95].

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. alg. for reporting variant of GPM

Use 2-wise independent hash function $h : \Sigma_T \to [\mathcal{D}]$
$h(x) \approx (a \cdot x + b) \mod \mathcal{D}$

Create a new matching relationship M':

$$M'[h(a), b] = \checkmark \quad \text{iff} \quad \exists_{a \in \Sigma_T} M[a, b] = \checkmark$$

Fact: If $M[a, b] \neq \checkmark$ then $Pr(M'[h(a), b] = \checkmark) \leq 1/2$

Eliminate every non-occurrence of P in T with prob. $\geq 1/2$ and use Pattern Matching with Don't Cares to solve GPM for M'.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. time algorithm for reporting variant of GPM.

Same complexity in [Muthukrishnan, '95].

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. alg. for reporting variant of GPM

Use 2-wise independent hash function $h : \Sigma_T \to [\mathcal{D}]$
$h(x) \approx (a \cdot x + b) \mod \mathcal{D}$

Create a new matching relationship M':

$$M'[h(a), b] = \checkmark \quad \text{iff} \quad \exists_{a \in \Sigma_T} M[a, b] = \checkmark$$

Fact: If $M[a, b] \neq \checkmark$ then $Pr(M'[h(a), b] = \checkmark) \leq 1/2$

Eliminate every non-occurrence of P in T with prob. $\geq 1/2$ and use PATTERN MATCHING WITH DON'T CARES to solve GPM for M'.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. time algorithm for reporting variant of GPM.

Same complexity in [Muthukrishnan, '95].

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. alg. for reporting variant of GPM

Use 2-wise independent hash function $h : \Sigma_T \to [\mathcal{D}]$
$h(x) \approx (a \cdot x + b) \mod \mathcal{D}$

Create a new matching relationship M':

$$M'[h(a), b] = \checkmark \quad \text{iff} \quad \exists_{a \in \Sigma_T} M[a, b] = \checkmark$$

Fact: If $M[a, b] \neq \checkmark$ then $Pr(M'[h(a), b] = \checkmark) \leq 1/2$

Eliminate every non-occurrence of P in T with prob. $\geq 1/2$ and use Pattern Matching with Don't Cares to solve GPM for M'.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. time algorithm for reporting variant of GPM.

Same complexity in [Muthukrishnan, '95].

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. alg. for reporting variant of GPM

Use 2-wise independent hash function $h : \Sigma_T \to [\mathcal{D}]$
$h(x) \approx (a \cdot x + b) \mod \mathcal{D}$

Create a new matching relationship M':

$$M'[h(a), b] = \checkmark \quad \text{iff} \quad \exists_{a \in \Sigma_T} M[a, b] = \checkmark$$

Fact: If $M[a, b] \neq \checkmark$ then $Pr(M'[h(a), b] = \checkmark) \leq 1/2$

Eliminate every non-occurrence of P in T with prob. $\geq 1/2$ and use Pattern Matching with Don't Cares to solve GPM for M'.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. time algorithm for reporting variant of GPM.

Same complexity in [Muthukrishnan, '95].

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. alg. for reporting variant of GPM

Use 2-wise independent hash function $h : \Sigma_T \to [\mathcal{D}]$
$h(x) \approx (a \cdot x + b) \mod \mathcal{D}$

Create a new matching relationship M':

$$M'[h(a), b] = \checkmark \quad \text{iff} \quad \exists_{a \in \Sigma_T} M[a, b] = \checkmark$$

Fact: If $M[a, b] \neq \checkmark$ then $Pr(M'[h(a), b] = \checkmark) \leq 1/2$

Eliminate every non-occurrence of P in T with prob. $\geq 1/2$ and use PATTERN MATCHING WITH DON'T CARES to solve GPM for M'.

$\tilde{\mathcal{O}}(\mathcal{D}\, n)$ rand. time algorithm for reporting variant of GPM.

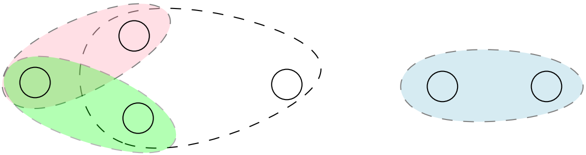
Same complexity in [Muthukrishnan, '95].

Discrepancy minimization

Discrepancy

Consider a family $\mathcal{F}$ of z sets $S_i \subseteq U$, $i \in [z]$. We call a function $\chi : U \to \{-1, +1\}$. The *discrepancy* of a set S_i is defined as $\chi(S_i) = \sum_{u \in S_i} \chi(u)$.

Goal: minimize $\max_{i \in [z]} |\chi(S_i)|$



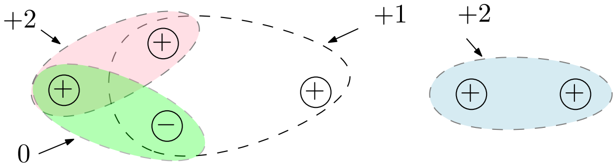
Random assignment χ gives $|\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log |U|})$ whp. ($|S_i| \leq k$)

Discrepancy minimization

Discrepancy

Consider a family $\mathcal{F}$ of z sets $S_i \subseteq U$, $i \in [z]$. We call a function $\chi : U \to \{-1, +1\}$. The *discrepancy* of a set S_i is defined as $\chi(S_i) = \sum_{u \in S_i} \chi(u)$.

Goal: minimize $\max_{i \in [z]} |\chi(S_i)|$



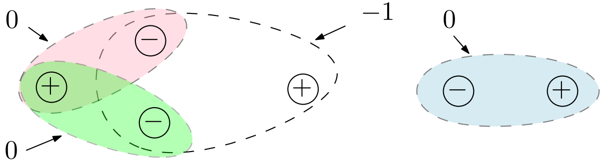
Random assignment χ gives $|\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log |U|})$ whp. ($|S_i| \leq k$)

Discrepancy minimization

Discrepancy

Consider a family $\mathcal{F}$ of z sets $S_i \subseteq U$, $i \in [z]$. We call a function $\chi : U \to \{-1, +1\}$. The *discrepancy* of a set S_i is defined as $\chi(S_i) = \sum_{u \in S_i} \chi(u)$.

Goal: minimize $\max_{i \in [z]} |\chi(S_i)|$



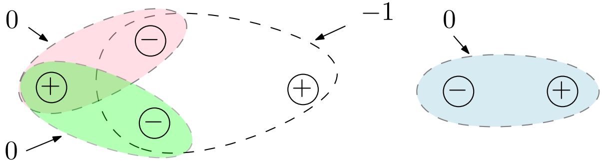
Random assignment χ gives $|\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log |U|})$ whp. ($|S_i| \leq k$)

Discrepancy minimization

Discrepancy

Consider a family $\mathcal{F}$ of z sets $S_i \subseteq U$, $i \in [z]$. We call a function $\chi : U \to \{-1, +1\}$. The *discrepancy* of a set S_i is defined as $\chi(S_i) = \sum_{u \in S_i} \chi(u)$.

Goal: minimize $\max_{i \in [z]} |\chi(S_i)|$



Random assignment χ gives $|\chi(S_i)| \le \mathcal{O}(\sqrt{k \log |U|})$ whp. ($|S_i| \le k$)

Derandomization

Derandomization [Chazelle, Beck, Fiala]

Assign $\chi(u_1), \chi(u_2), \ldots$ without ever backtracking.

ε satisfies $\log \frac{1+\varepsilon}{1-\varepsilon} = \Theta(\sqrt{\log(z)/k})$. Minimize $G = \sum_{i \in [z]} G_i$ where:

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$$

To assign $\chi(u_j)$ compare G^+ and G^- (note: $G^+ + G^- = 2G$).

Technical details:

1. compute ε with fixed basic operations
2. use only addition and multiplication
3. never operate on too small numbers
4. run the procedure efficiently

For a family of z sets of at most k elements we can find in deterministic $\mathcal{O}(zk \log z)$ time a colouring χ such that $\max_{i \in [z]} |\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log z})$

Derandomization

Derandomization [Chazelle, Beck, Fiala]

Assign $\chi(u_1), \chi(u_2), \ldots$ without ever backtracking.

ε satisfies $\log \frac{1+\varepsilon}{1-\varepsilon} = \Theta(\sqrt{\log(z)/k})$. Minimize $G = \sum_{i\in[z]} G_i$ where:

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$$

To assign $\chi(u_j)$ compare G^+ and G^- (note: $G^+ + G^- = 2G$).

Technical details:

1. compute ε with fixed basic operations
2. use only addition and multiplication
3. never operate on too small numbers
4. run the procedure efficiently

For a family of z sets of at most k elements we can find in deterministic $\mathcal{O}(zk\log z)$ time a colouring χ such that $\max_{i\in[z]} |\chi(S_i)| \le \mathcal{O}(\sqrt{k\log z})$

Derandomization

Derandomization [Chazelle, Beck, Fiala]

Assign $\chi(u_1), \chi(u_2), \ldots$ without ever backtracking.

ε satisfies $\log \frac{1+\varepsilon}{1-\varepsilon} = \Theta(\sqrt{\log(z)/k})$. Minimize $G = \sum_{i \in [z]} G_i$ where:

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$$

To assign $\chi(u_j)$ compare G^+ and G^- (note: $G^+ + G^- = 2G$).

Technical details:

1. compute ε with fixed basic operations
2. use only addition and multiplication
3. never operate on too small numbers
4. run the procedure efficiently

For a family of z sets of at most k elements we can find in deterministic $\mathcal{O}(zk \log z)$ time a colouring χ such that $\max_{i \in [z]} |\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log z})$

Derandomization

Derandomization [Chazelle, Beck, Fiala]

Assign $\chi(u_1), \chi(u_2), \ldots$ without ever backtracking.

ε satisfies $\log \frac{1+\varepsilon}{1-\varepsilon} = \Theta(\sqrt{\log(z)/k})$. Minimize $G = \sum_{i \in [z]} G_i$ where:

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$$

To assign $\chi(u_j)$ compare G^+ and G^- (note: $G^+ + G^- = 2G$).

Technical details:

1. compute ε with fixed basic operations
2. use only addition and multiplication
3. never operate on too small numbers
4. run the procedure efficiently

For a family of z sets of at most k elements we can find in deterministic $\mathcal{O}(zk \log z)$ time a colouring χ such that $\max_{i \in [z]} |\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log z})$

Derandomization

Derandomization [Chazelle, Beck, Fiala]

Assign $\chi(u_1), \chi(u_2), \ldots$ without ever backtracking.

ε satisfies $\log \frac{1+\varepsilon}{1-\varepsilon} = \Theta(\sqrt{\log(z)/k})$. Minimize $G = \sum_{i \in [z]} G_i$ where:

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$$

To assign $\chi(u_j)$ compare G^+ and G^- (note: $G^+ + G^- = 2G$).

Technical details:

1. compute ε with fixed basic operations
2. use only addition and multiplication
3. never operate on too small numbers
4. run the procedure efficiently

For a family of z sets of at most k elements we can find in deterministic $\mathcal{O}(zk \log z)$ time a colouring χ such that $\max_{i \in [z]} |\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log z})$

Derandomization

Derandomization [Chazelle, Beck, Fiala]

Assign $\chi(u_1), \chi(u_2), \dots$ without ever backtracking.

ε satisfies $\log \frac{1+\varepsilon}{1-\varepsilon} = \Theta(\sqrt{\log(z)/k})$. Minimize $G = \sum_{i \in [z]} G_i$ where:

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$$

To assign $\chi(u_j)$ compare G^+ and G^- (note: $G^+ + G^- = 2G$).

Technical details:

1. compute ε with fixed basic operations
2. use only addition and multiplication
3. never operate on too small numbers
4. run the procedure efficiently

For a family of z sets of at most k elements we can find in deterministic $\mathcal{O}(zk \log z)$ time a colouring χ such that $\max_{i \in [z]} |\chi(S_i)| \leq \mathcal{O}(\sqrt{k \log z})$

Superimposed codes

Superimposed code

Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of bitstrings $\{C_u, u \in U\}$ of length ℓ with w ones, is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$.

Superimposed codes

Superimposed code

Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of bitstrings $\{C_u, u \in U\}$ of length ℓ with w ones, is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$.

Code for $S_1 = \{2, 4, 5\}, \tau = 2, \ell = 8, w = 5$:

$$C_1 = 01101110$$
$$C_2 = 11110001$$
$$C_3 = 11001010$$
$$C_4 = 10110100$$
$$C_5 = 01100101$$

$$C_3 = 1100\underline{1}01\underline{1}0$$
$$\bigcup_{v \in S_1} C_v = 11110101$$

Superimposed codes

Superimposed code

Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of bitstrings $\{C_u, u \in U\}$ of length ℓ with w ones, is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$.

Code for $S_1 = \{2, 4, 5\}, \tau = 2, \ell = 8, w = 5$:

$$C_1 = 01101110$$
$$C_2 = 11110001$$
$$C_3 = 11001010$$
$$C_4 = 10110100$$
$$C_5 = 01100101$$

$$C_3 = 1100\underline{1}01\underline{0}$$
$$\bigcup_{v \in S_1} C_v = 11110101$$

[FOCS '97, Indyk]

Is there a deterministic algorithm computing in $\tilde{\mathcal{O}}((zk)/\varepsilon^{\mathcal{O}(1)})$-time an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code with $\ell = \tilde{\mathcal{O}}(k/\varepsilon^{\mathcal{O}(1)})$?

Improved data-dependent superimposed codes

Our approach:

1. recursively partition the universe using discrepancy minimization $\rightarrow$ after $\log k + \mathcal{O}(\log^* k)$ steps, parts X_c: $|X_c \cap S_i| \leq \mathcal{O}(\log z)$
2. combine it with a family of hash functions $h_p(u) = \mathrm{POL}(u) \bmod p$ for all $\Theta(\frac{2^d}{d})$ irreducible polynomials $p \in \mathcal{P}_d$ of degree $d = \Theta(\log \frac{t \log z}{\varepsilon})$

$$\rightarrow \text{ for } u \in X_c: \quad C_u = \{\mathrm{NUM}(h_p(u)) + 2^d \cdot \mathrm{NUM}(p) + 4^d \cdot c \mid p \in \mathcal{P}_d\}$$

$$= \{\underbrace{\boxed{h_p(u)}}_{d} + \underbrace{\boxed{p}}_{d} + \underbrace{\boxed{c}}_{k \cdot 2^{\mathcal{O}(\log^* k)}} \mid p \in \mathcal{P}_d\}$$

Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of weight $w = \tilde{\mathcal{O}}(\varepsilon^{-1})$ and length $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2} k)$ in deterministic $\tilde{\mathcal{O}}(\varepsilon^{-1}|U|)$ time and space.

Improved data-dependent superimposed codes

Our approach:

1. recursively partition the universe using discrepancy minimization $\rightarrow$ after $\log k + \mathcal{O}(\log^* k)$ steps, parts X_c: $|X_c \cap S_i| \leq \mathcal{O}(\log z)$
2. combine it with a family of hash functions $h_p(u) = \mathrm{POL}(u) \bmod p$ for all $\Theta(\frac{2^d}{d})$ irreducible polynomials $p \in \mathcal{P}_d$ of degree $d = \Theta(\log \frac{t \log z}{\varepsilon})$

$$\rightarrow \text{ for } u \in X_c : \; C_u = \{\mathrm{NUM}(h_p(u)) + 2^d \cdot \mathrm{NUM}(p) + 4^d \cdot c \mid p \in \mathcal{P}_d\}$$

$$= \{\underbrace{\boxed{h_p(u)}}_{d} + \underbrace{\boxed{\quad p \quad}}_{d} + \underbrace{\boxed{\quad c \quad}}_{k \cdot 2^{\mathcal{O}(\log^* k)}} \mid p \in \mathcal{P}_d\}$$

Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of weight $w = \tilde{\mathcal{O}}(\varepsilon^{-1})$ and length $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2}k)$ in deterministic $\tilde{\mathcal{O}}(\varepsilon^{-1}|U|)$ time and space.

Improved data-dependent superimposed codes

Our approach:

1. recursively partition the universe using discrepancy minimization $\rightarrow$ after $\log k + \mathcal{O}(\log^* k)$ steps, parts X_c: $|X_c \cap S_i| \leq \mathcal{O}(\log z)$
2. combine it with a family of hash functions $h_p(u) = \mathrm{POL}(u) \bmod p$ for all $\Theta(\frac{2^d}{d})$ irreducible polynomials $p \in \mathcal{P}_d$ of degree $d = \Theta(\log \frac{t \log z}{\varepsilon})$

$$\rightarrow \text{ for } u \in X_c: \; C_u = \{\mathrm{NUM}(h_p(u)) + 2^d \cdot \mathrm{NUM}(p) + 4^d \cdot c \mid p \in \mathcal{P}_d\}$$
$$= \{\underbrace{\boxed{h_p(u)}}_{d} + \underbrace{\boxed{\quad p \quad}}_{d} + \underbrace{\boxed{\quad c \quad}}_{k \cdot 2^{\mathcal{O}(\log^* k)}} \mid p \in \mathcal{P}_d\}$$

Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of weight $w = \tilde{\mathcal{O}}(\varepsilon^{-1})$ and length $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2}k)$ in deterministic $\tilde{\mathcal{O}}(\varepsilon^{-1}|U|)$ time and space.

Improved data-dependent superimposed codes

Our approach:

1. recursively partition the universe using discrepancy minimization $\rightarrow$ after $\log k + \mathcal{O}(\log^* k)$ steps, parts X_c: $|X_c \cap S_i| \leq \mathcal{O}(\log z)$
2. combine it with a family of hash functions $h_p(u) = \mathrm{POL}(u) \bmod p$ for all $\Theta(\frac{2^d}{d})$ irreducible polynomials $p \in \mathcal{P}_d$ of degree $d = \Theta(\log \frac{t \log z}{\varepsilon})$

$$\rightarrow \text{ for } u \in X_c: \; C_u = \{\mathrm{NUM}(h_p(u)) + 2^d \cdot \mathrm{NUM}(p) + 4^d \cdot c \mid p \in \mathcal{P}_d\}$$
$$= \{\underbrace{\boxed{h_p(u)}}_{d} + \underbrace{\boxed{\quad p \quad}}_{d} + \underbrace{\boxed{\quad c \quad}}_{k \cdot 2^{\mathcal{O}(\log^* k)}} \mid p \in \mathcal{P}_d\}$$

Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of weight $w = \tilde{\mathcal{O}}(\varepsilon^{-1})$ and length $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2}k)$ in deterministic $\tilde{\mathcal{O}}(\varepsilon^{-1}|U|)$ time and space.

Application of superimposed codes

Deterministic $(1-\varepsilon)$-approximation of counting variant of GPM

Construct $(\{S_b\}, (1-\varepsilon)w)$-superimposed code for universe Σ_T and sets S_b with $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D})$.
Create:

- text $T'[1, n\ell]$ by replacing characters from T with their codes
- pattern $P'[1, m\ell]$ where every b from P is replaced with $\bigcup_{a \in S_b} C_a$

Replace 1's in P' with ? and run $\tilde{\mathcal{O}}(n\ell)$-time algorithm for counting mismatches between T' and P'.

Deterministic $(1-\varepsilon)$-approximation in $\tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D}\, n)$ time.

Application of superimposed codes

Deterministic $(1-\varepsilon)$-approximation of counting variant of GPM

Construct $(\{S_b\},(1-\varepsilon)w)$-superimposed code for universe Σ_T and sets S_b with $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D})$.

Create:

- text $T'[1,n\ell]$ by replacing characters from T with their codes
- pattern $P'[1,m\ell]$ where every b from P is replaced with $\bigcup_{a\in S_b} C_a$

Replace 1's in P' with ? and run $\tilde{\mathcal{O}}(n\ell)$-time algorithm for counting mismatches between T' and P'.

$$
\begin{array}{cc}
T[1] \in S_{P[1]} & T[2] \notin S_{P[2]} \\
\underbrace{\boxed{\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ ? & ? & 0 & ? & 0 & ? \end{array}}}_{\ell} & \boxed{\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 & 0 \\ ? & ? & 0 & ? & 0 & ? \end{array}}
\end{array}
$$

$T[2]$ doesn't match $P[2]$
$[(1-\varepsilon)w, w]$ mismatches

Deterministic $(1-\varepsilon)$-approximation in $\tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D}\,n)$ time.

Application of superimposed codes

Deterministic $(1-\varepsilon)$-approximation of counting variant of GPM

Construct $(\{S_b\}, (1-\varepsilon)w)$-superimposed code for universe Σ_T and sets S_b with $\ell = \tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D})$.
Create:

- text $T'[1, n\ell]$ by replacing characters from T with their codes
- pattern $P'[1, m\ell]$ where every b from P is replaced with $\bigcup_{a \in S_b} C_a$

Replace 1's in P' with ? and run $\tilde{\mathcal{O}}(n\ell)$-time algorithm for counting mismatches between T' and P'.

$$\underbrace{\overset{T[1] \in S_{P[1]}}{\boxed{\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ ? & ? & 0 & ? & 0 & ? \end{array}}}}_{\ell} \; \overset{T[2] \notin S_{P[2]}}{\boxed{\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 & 0 \\ ? & ? & 0 & ? & 0 & ? \end{array}}}$$

$T[2]$ doesn't match $P[2]$
$[(1-\varepsilon)w, w]$ mismatches

Deterministic $(1-\varepsilon)$-approximation in $\tilde{\mathcal{O}}(\varepsilon^{-2}\mathcal{D}\, n)$ time.

The end: How to deal with parameter $\mathcal{I}$?

Let b be a parameter, $S = \{x_1, x_2, \ldots, x_\ell\}$ be a sequence of integers, and $s = \sum_{i \in [\ell]} x_i$. Then S can be partitioned into $\mathcal{O}(s/b + 1)$ ranges $S_1, S_2, \ldots$ such that, for every i, either $|S_i| = 1$ or $\sum_{x \in S_i} x \leq b$.

Approach:

- divide characters from Σ_T based on their counts
- use Subset Pattern Matching and Pattern Matching with Don't Cares

Det. algorithm for counting variant of GPM in $\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$.

In Threshold Pattern Matching $a \approx b$ if $|a - b| < \delta$, so $\mathcal{I} = m$.

The end: How to deal with parameter $\mathcal{I}$?

Let b be a parameter, $S = \{x_1, x_2, \ldots, x_\ell\}$ be a sequence of integers, and $s = \sum_{i \in [\ell]} x_i$. Then S can be partitioned into $\mathcal{O}(s/b + 1)$ ranges $S_1, S_2, \ldots$ such that, for every i, either $|S_i| = 1$ or $\sum_{x \in S_i} x \leq b$.

Approach:

- divide characters from Σ_T based on their counts
- use Subset Pattern Matching and Pattern Matching with Don't Cares

Det. algorithm for counting variant of GPM in $\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$.

In Threshold Pattern Matching $a \approx b$ if $|a - b| < \delta$, so $\mathcal{I} = m$.

The end: How to deal with parameter $\mathcal{I}$?

Let b be a parameter, $S = \{x_1, x_2, \ldots, x_\ell\}$ be a sequence of integers, and $s = \sum_{i \in [\ell]} x_i$. Then S can be partitioned into $\mathcal{O}(s/b + 1)$ ranges $S_1, S_2, \ldots$ such that, for every i, either $|S_i| = 1$ or $\sum_{x \in S_i} x \leq b$.

Approach:

- divide characters from Σ_T based on their counts
- use Subset Pattern Matching and Pattern Matching with Don't Cares

Det. algorithm for counting variant of GPM in $\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$.

In Threshold Pattern Matching $a \approx b$ if $|a - b| < \delta$, so $\mathcal{I} = m$.

The end: How to deal with parameter $\mathcal{I}$?

Let b be a parameter, $S = \{x_1, x_2, \ldots, x_\ell\}$ be a sequence of integers, and $s = \sum_{i\in[\ell]} x_i$. Then S can be partitioned into $\mathcal{O}(s/b + 1)$ ranges $S_1, S_2, \ldots$ such that, for every i, either $|S_i| = 1$ or $\sum_{x\in S_i} x \le b$.

Approach:

- divide characters from Σ_T based on their counts
- use Subset Pattern Matching and Pattern Matching with Don't Cares

Det. algorithm for counting variant of GPM in $\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$.

In Threshold Pattern Matching $a \approx b$ if $|a - b| < \delta$, so $\mathcal{I} = m$.