EMS-GT2: An Improved Exact Solution for the (l, d)-Planted Motif Problem

Mark Joseph D. Ronquillo
Proceso L. Fernandez PhD.

1. Context of the Study
2. Definition of Terms
3. EMS-GT
4. Additional Speedup Techniques
5. Methods
6. Results
7. Conclusion

Context of the Study

Context

Motif Finding

Given a set of DNA sequences, find common substrings in each sequence considering a number of allowed mutations

Context

(l, d)-planted motif search problem

*Given a set of **n** sequences with length of **m** each, find the planted motif of length **l** considering up to **d** mutations:*

(l, d)-planted motif problem:

Ex: ***n = 4 m = 30 l = 5 d = 2***

S1 taagctgctattctacggatagatactaca

S2 acacttgactatatataggatctaggatacat

S3 actagatatacctataggcacattgctgga

S4 tagagcacatagacctgacacatagtactt

Find the planted motif M: ***?***

(l, d)-planted motif problem:

Ex: ***n = 4   m = 30   l = 5   d = 2***

S1 taagctgcta**ttcta**cggatagatactaca

S2 a**cactt**gactatataggatctaggatacat

S3 actagatatacctataggcacat**tgctg**ga

S4 tagagcacatag**acctg**acacatagtactt

Find the planted motif M: ***tactg***

Definition of terms

Definition of Terms

***l*-mer:**

- a string of length l in the alphabet Σ

in this context we use Σ = {a, c, g, t}

Definition of Terms

motif:

- *l*-mer that occurs across different sequences (but with subject to mutations)

Definition of Terms

dH(x,y)**: hamming distance**

- the number of mismatch positions between two l-mers.

ex: dH(actg, gcta) = 2

Definition of Terms

d-neighbor of an l-mer x:

- is another *l*-mer ***y*** whose hamming distance with ***x*** is at most **d**.

Definition of Terms

N(x, d): ***d-neighborhood of an l-mer x:***

- is the set of all *l*-mers with at most *d* hamming distance value from *x*.

Definition of Terms

N(x, d): ***d-neighborhood of an l-mer x:***

Example: l = 4, d = 2

gatc

{

**a**a**a**c, **a**a**c**c, **a**a**g**c, **a**at**a**, **a**atc, **a**at**g**, **a**at**t**, **ac**tc, **ag**tc, **at**tc,

**c**a**a**c, **c**a**c**c, **c**a**g**c, **c**at**a**, **c**atc, **c**at**g**, **c**at**t**, **cc**tc, **cg**tc, **ct**tc,

_ga**aa**, ga**a**c, ga**ag**, ga**at**, ga**ca**, ga**c**c, ga**cg**, ga**ct**, ga**ga**,_

ga**g**c, ga**gg**, ga**gt**, ga**ta**, gatc, gat**g**, gat**t**, g**ca**c, g**cc**c,

_g**cg**c, g**c**t**a**, g**c**tc, g**c**t**g**, g**c**t**t**, g**ga**c, g**gc**c, g**gg**c, g**g**t**a**,_

_g**g**tc, g**g**t**g**, g**g**t**t**, g**ta**c, g**tc**c, g**tg**c, g**t**t**a**, g**t**tc, g**t**t**g**, g**t**t**t**,_

**t**a**a**c, **t**a**c**c, **t**a**g**c, **t**at**a**, **t**atc, **t**at**g**, **t**at**t**, **tc**tc, **tg**tc, **tt**tc

}

Definition of Terms

N(*S, d*)***: d-neighborhood of a sequence S***

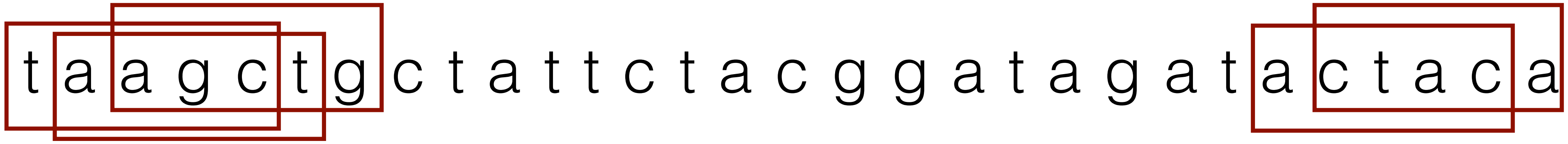
- is the set of all d-neighbors of all l-mers in sequence S.

- there are exactly *(m - l +1) l*-mers in sequence S of length m.

Definition of Terms

N*(S, d)****: d-neighborhood of a sequence S***

l = 5

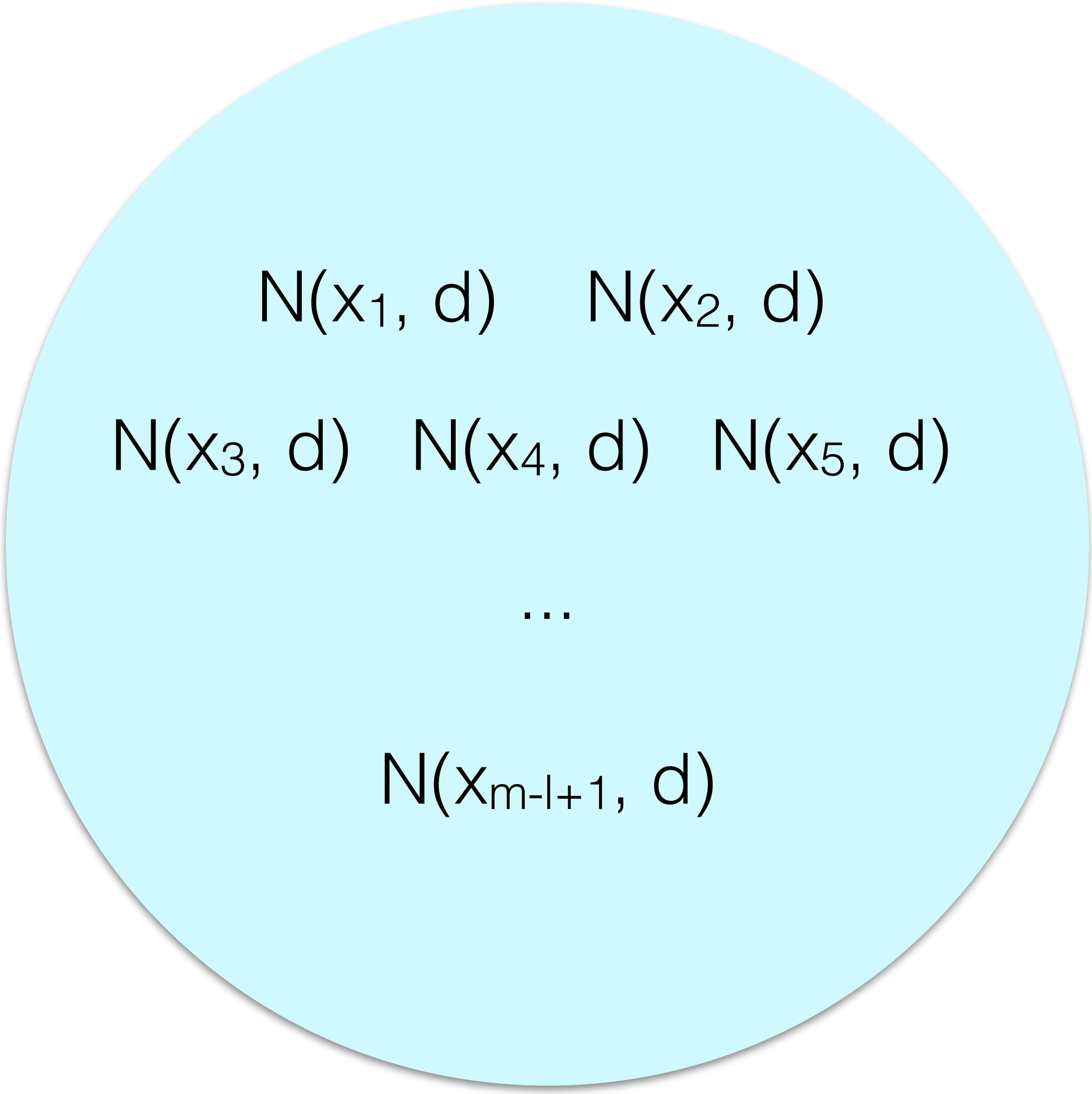


S1 taagctgctattctacggatagatactaca

Definition of Terms

N**(S, d)*: ***d*-neighborhood of a sequence *S

***N**(S, d)* =



$N(x_1, d)$ $N(x_2, d)$

$N(x_3, d)$ $N(x_4, d)$ $N(x_5, d)$

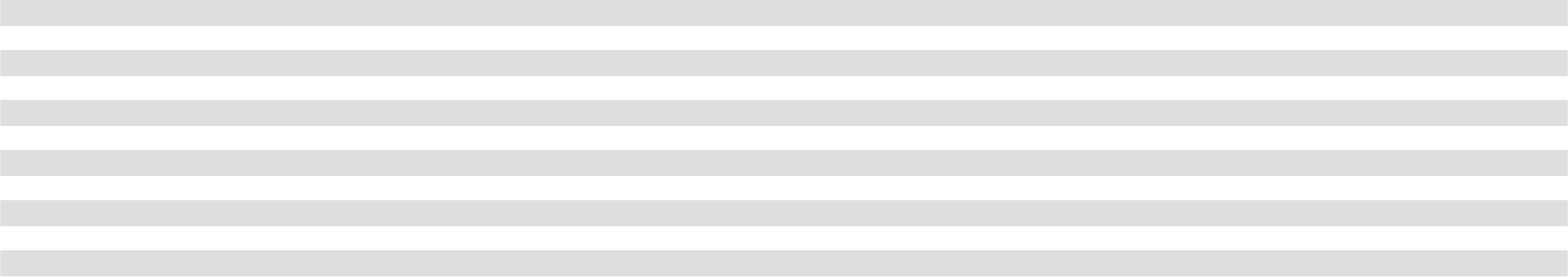
...

$N(x_{m-l+1}, d)$

EMS-GT Algorithm

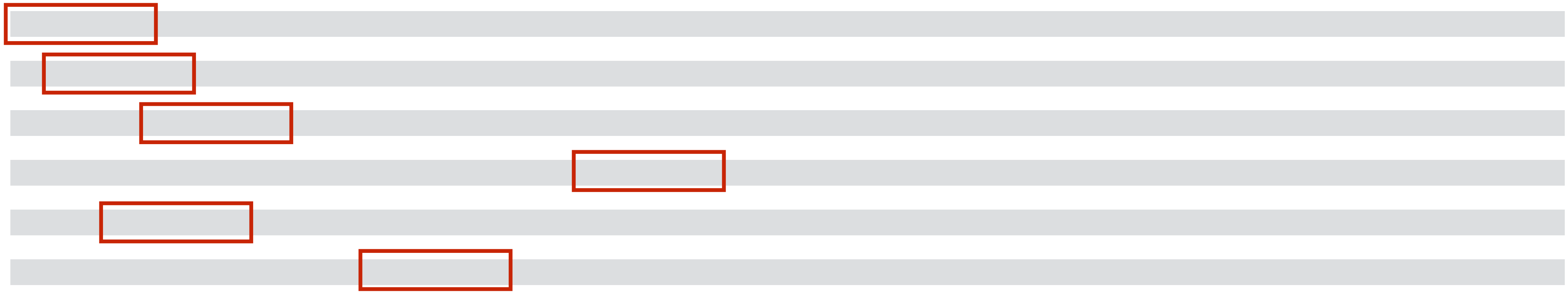
First Approach:

It checks every possible combination of positions in different sequences and tests if it is the correct position where the motif is planted.



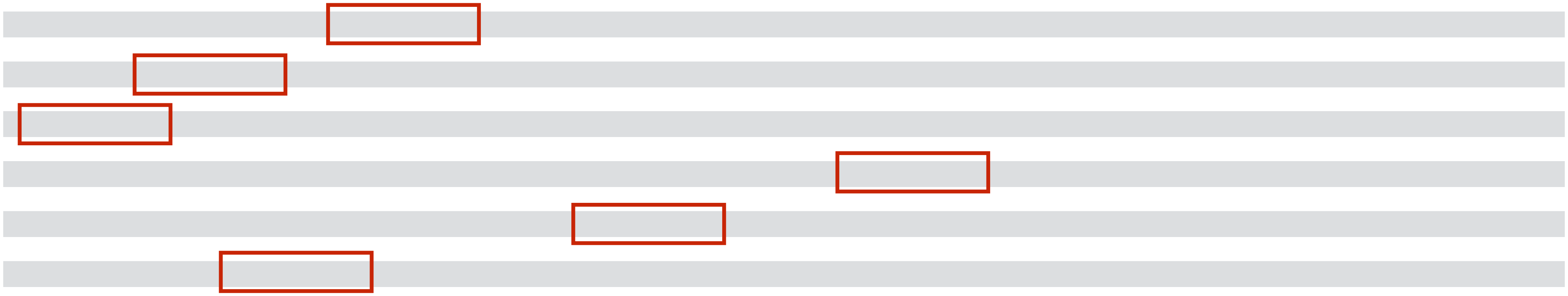
First Approach:

It checks every possible combination of positions in different sequences and tests if it is the correct position where the motif is planted.



First Approach:

It checks every possible combination of positions in different sequences and tests if it is the correct position where the motif is planted.



Second Approach:

Exhaustively searches all possible 4^l l-mers if it a motif.

4^l possible l-mers

Exact Motif Search - Generate and Test (EMS-GT)

EMS-GT is composed of 2 phases.

Generate Phase: quickly filters the search space by doing a set intersection between the d-neighborhood of the first n' sequences. Resulting set is the candidate motif set.

Test Phase: evaluates each candidate motif if it has at least one d-neighbor in the remaining (n - n') sequences.

EMS-GT: Demonstration:

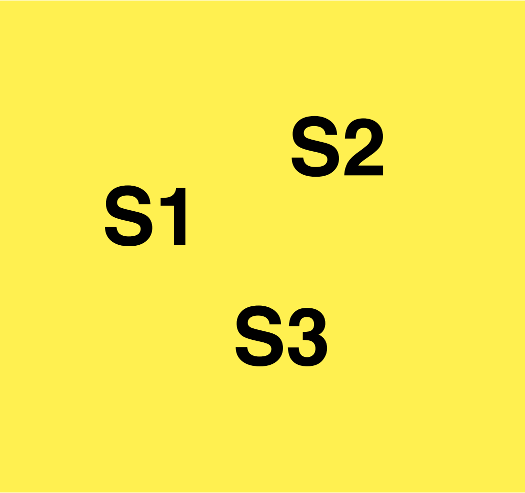
Parameters	$n = 5$	$m = 30$	$l = 5$	$d = 2$	$n' = 3$
Dataset	**S1**	**S2**	**S3**	**S4**	**S5**

EMS-GT: Demonstration:

Parameters $n = 5$ $m = 30$ $l = 5$ $d = 2$ $n' = 3$

Dataset **S1** **S2** **S3** **S4** **S5**

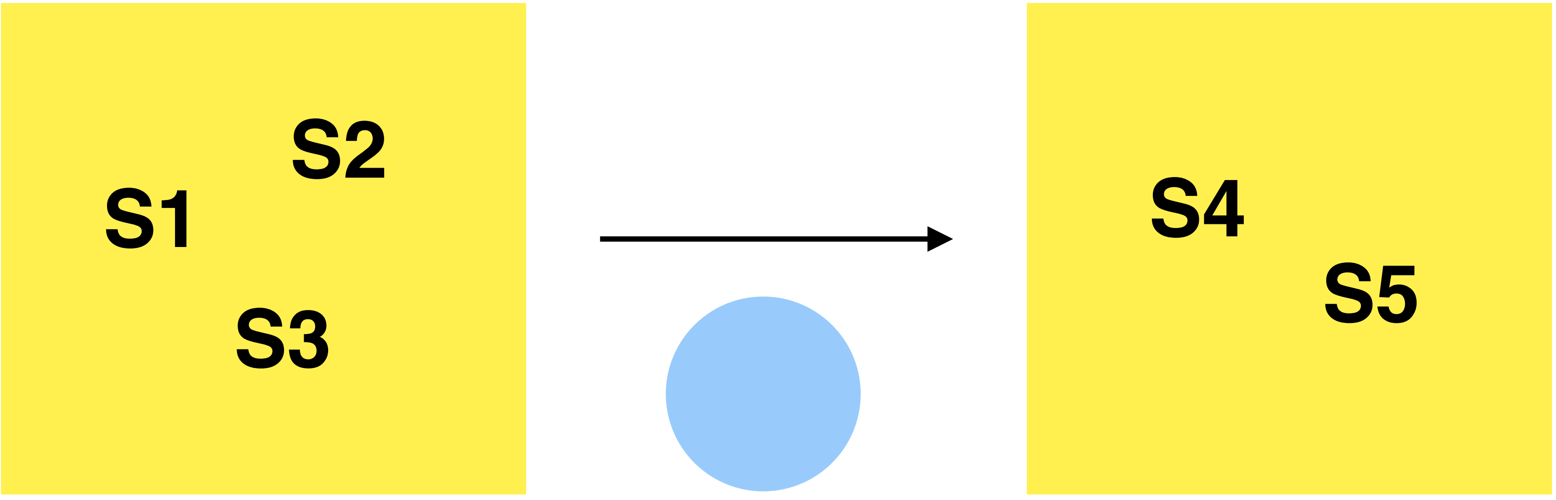
Generate



EMS-GT: Demonstration:

Parameters $n = 5$ $m = 30$ $l = 5$ $d = 2$ $n' = 3$

Dataset **S1 S2 S3 S4 S5**



Generate

S1 S2 S3

Test

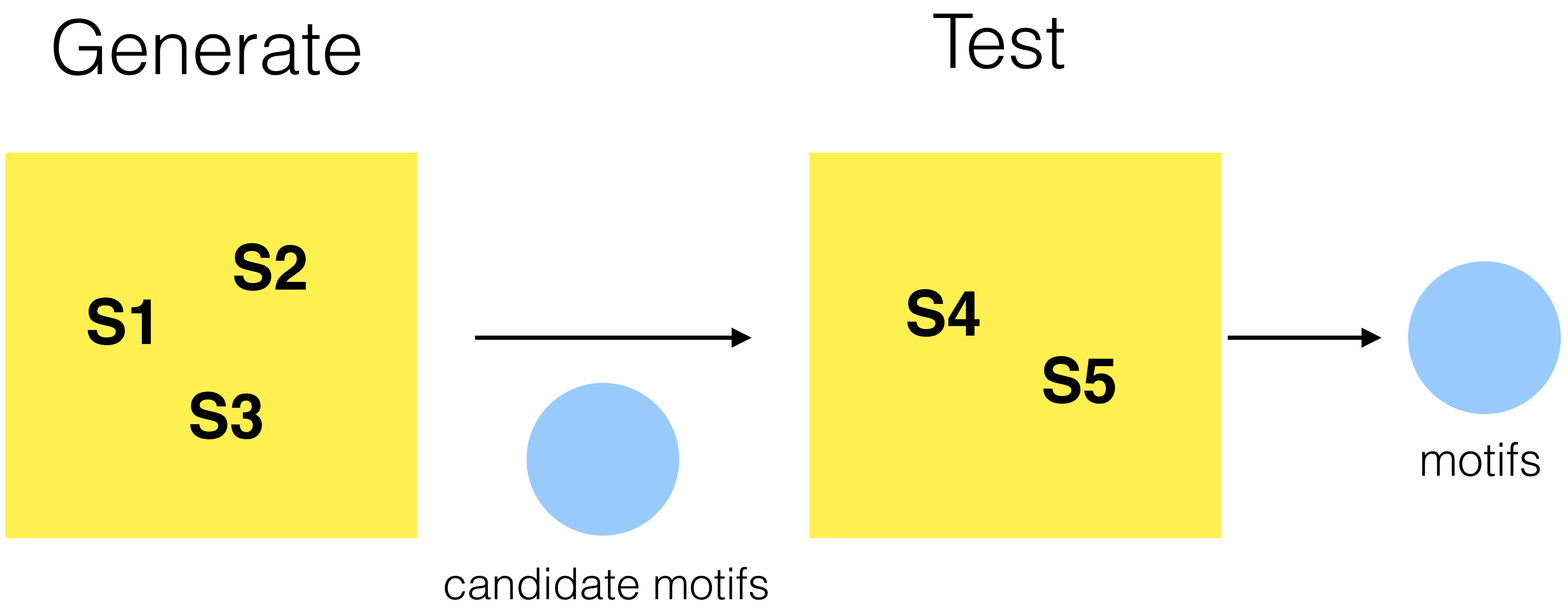
S4 S5

candidate motifs

EMS-GT: Demonstration:

Parameters $n = 5$ $m = 30$ $l = 5$ $d = 2$ $n' = 3$

Dataset **S1 S2 S3 S4 S5**



Generate

Test

S1 **S2** **S3**

candidate motifs

S4 **S5**

motifs

Generate Phase

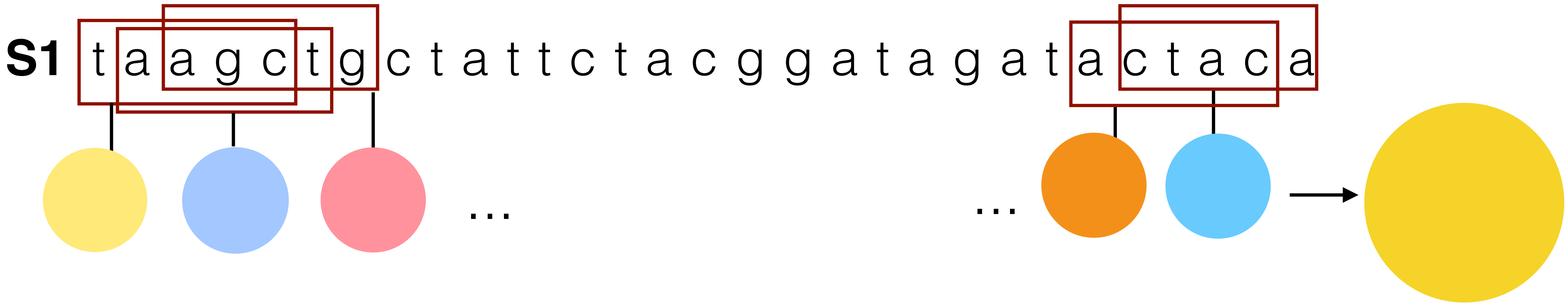
$n = 5$ $m = 30$ $l = 5$ $d = 2$ $n' = 3$

S1 t a a g c t g c t a t t c t a c g g a t a g a t a c t a c a

S2 a c a c t t g a c t a t a t a g g a t c t a g g a t a c a t

Generate Phase

***n* = 5 *m* = 30 *l* = 5 *d* = 2 *n'* = 3**

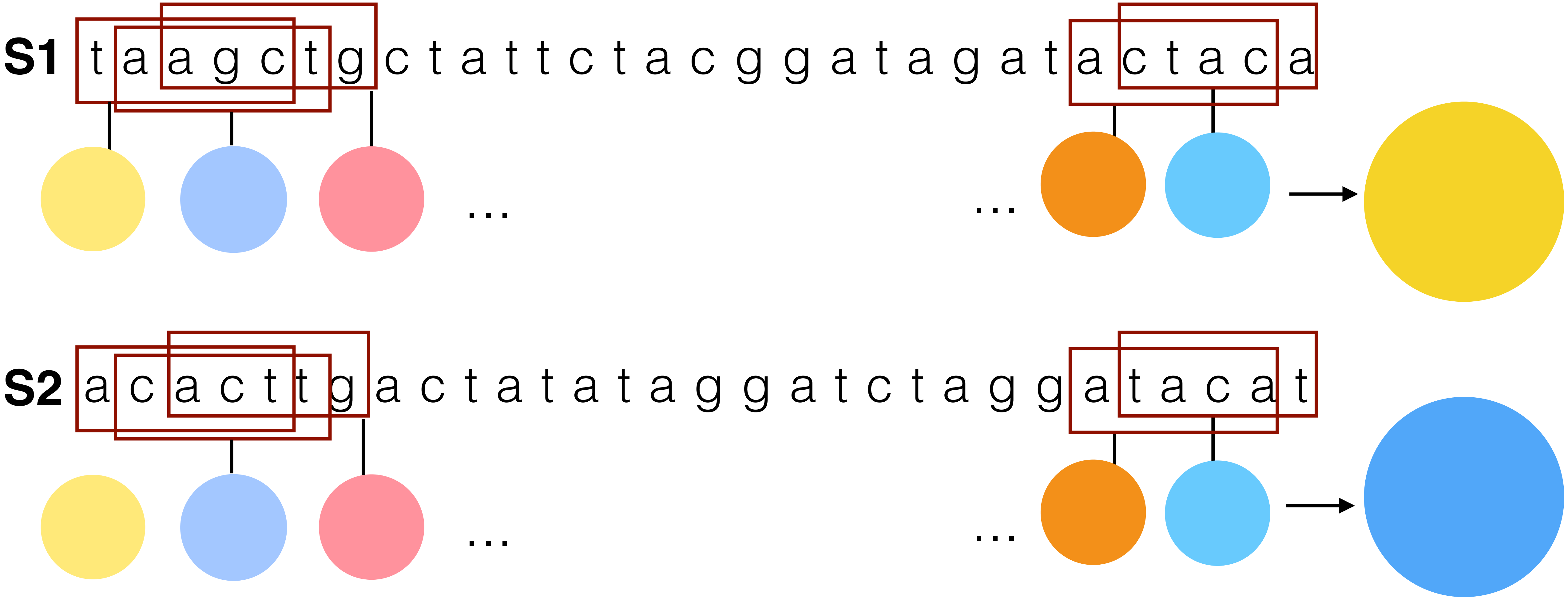


S1 t a a g c t g c t a t t c t a c g g a t a g a t a c t a c a

S2 a c a c t t g a c t a t a t a g g a t c t a g g a t a c a t

Generate Phase

***n* = 5 *m* = 30 *l* = 5 *d* = 2 *n'* = 3**

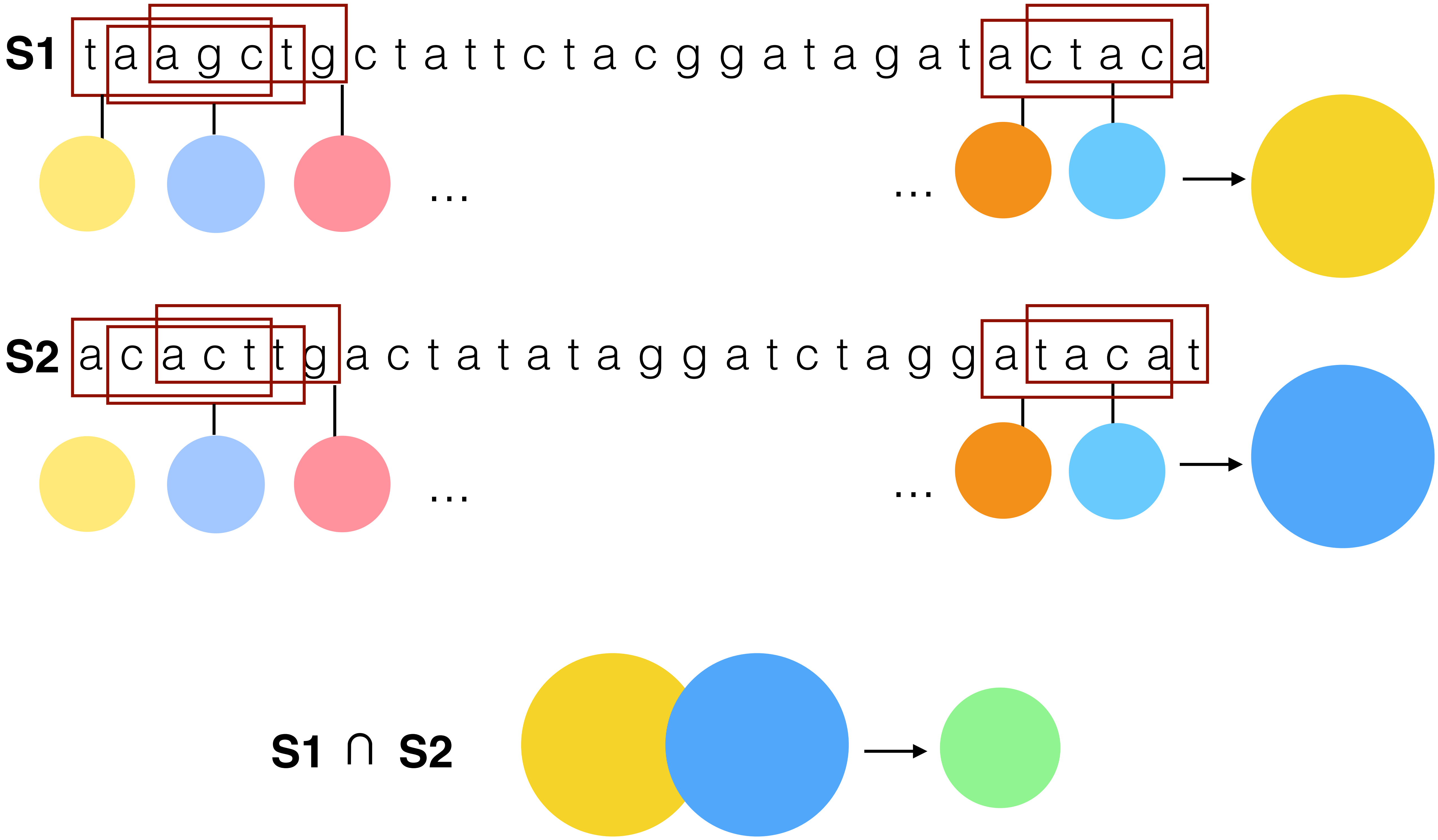


S1 taagctgctattctacggatagatactaca … …

S2 acacttgactatataggatctaggatacat … …

Generate Phase

***n* = 5 *m* = 30 *l* = 5 *d* = 2 *n'* = 3**



S1 taagctgctattctacggatagatactaca

…

…

S2 acacttgactatataggatctaggatacat

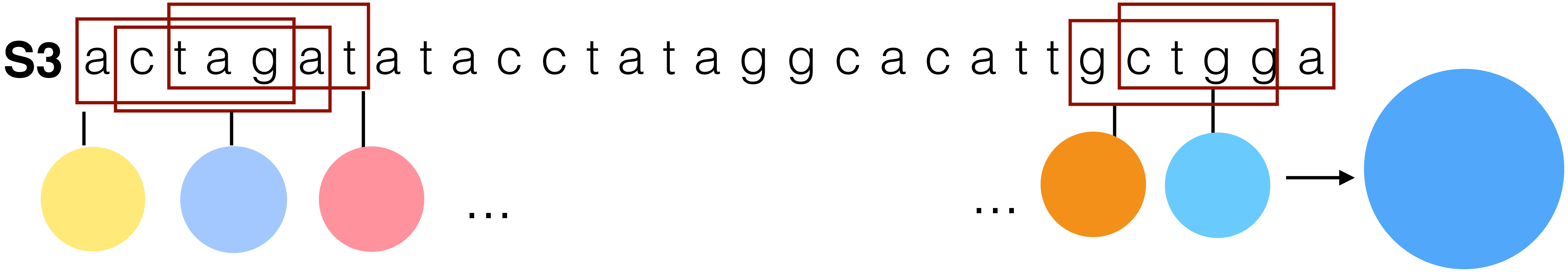
…

…

S1 ∩ S2

Generate Phase

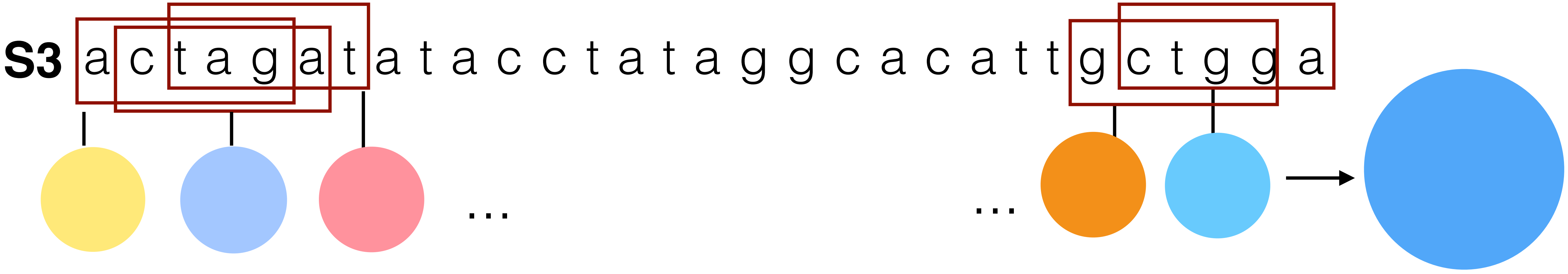
n = 5   m = 30   l = 5   d = 2   n' = 3



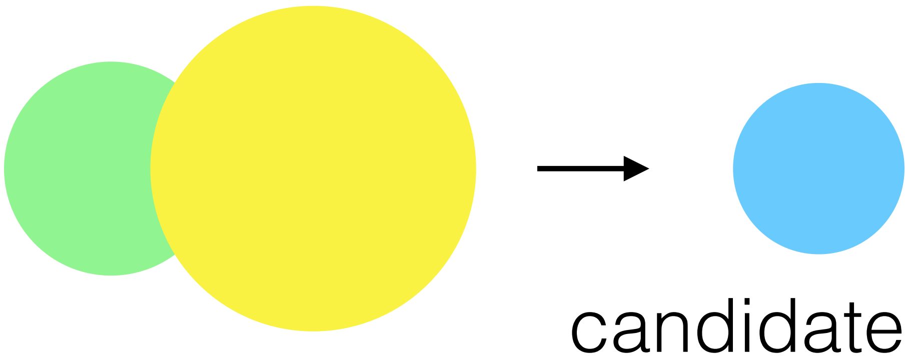
S3 a c t a g a t a t a c c t a t a g g c a c a t t g c t g g a

Generate Phase

***n* = 5 *m* = 30 *l* = 5 *d* = 2 *n'* = 3**



S1 ∩ S2 ∩ S3

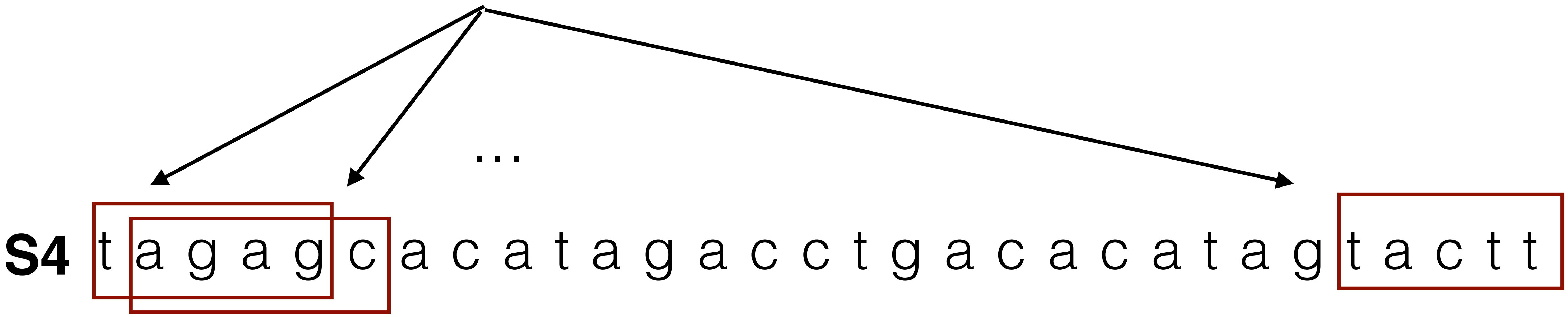


candidate motifs

EMS-GT: Demonstration: ***n = 5 m = 30 l = 5 d = 2 n' = 2***

Test Phase:

candidate motifs = {tagac, ccatg, tagtt}



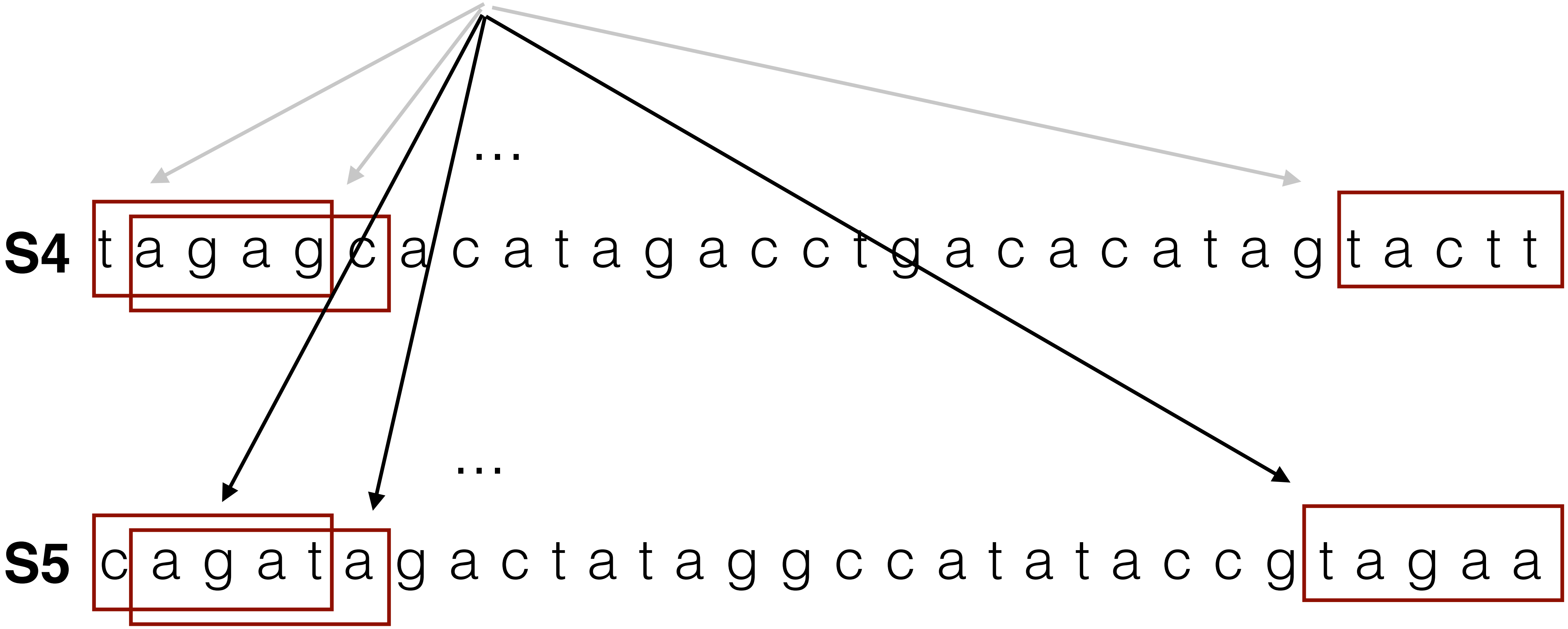
S4 t a g a g c a c a t a g a c c t g a c a c a t a g t a c t t

S5 c a g a t a g a c t a t a g g c c a t a t a c c g t a g a a

EMS-GT: Demonstration: ***n* = 5 *m* = 30 *l* = 5 *d* = 2 *n'* = 2**

Test Phase:

candidate motifs = {tagac, ccatg, tagtt}



...

S4 t a g a g c a c a t a g a c c t g a c a c a t a g t a c t t

...

S5 c a g a t a g a c t a t a g g c c a t a t a c c g t a g a a

Implementation

Implementation: Speedup Techniques

1. Integer Mapping of an *l*-mer

a = 00 | c = 01 | g = 10 | t =11

example:

tacgt is mapped to 1100011011

integer value = 795

Implementation: Speedup Techniques

2. Bit-based set representation and l-mer enumeration

Set Representation

0	0
1	1
2	1
3	0
⋮	
4^l	0

We know that there are ***4^l*** *number of possible l-mers given the DNA bases.*

A value of ***1*** *means it is in the set while* ***0*** *means otherwise*

Implementation: Speedup Techniques

2. Bit-based set representation and l-mer enumeration

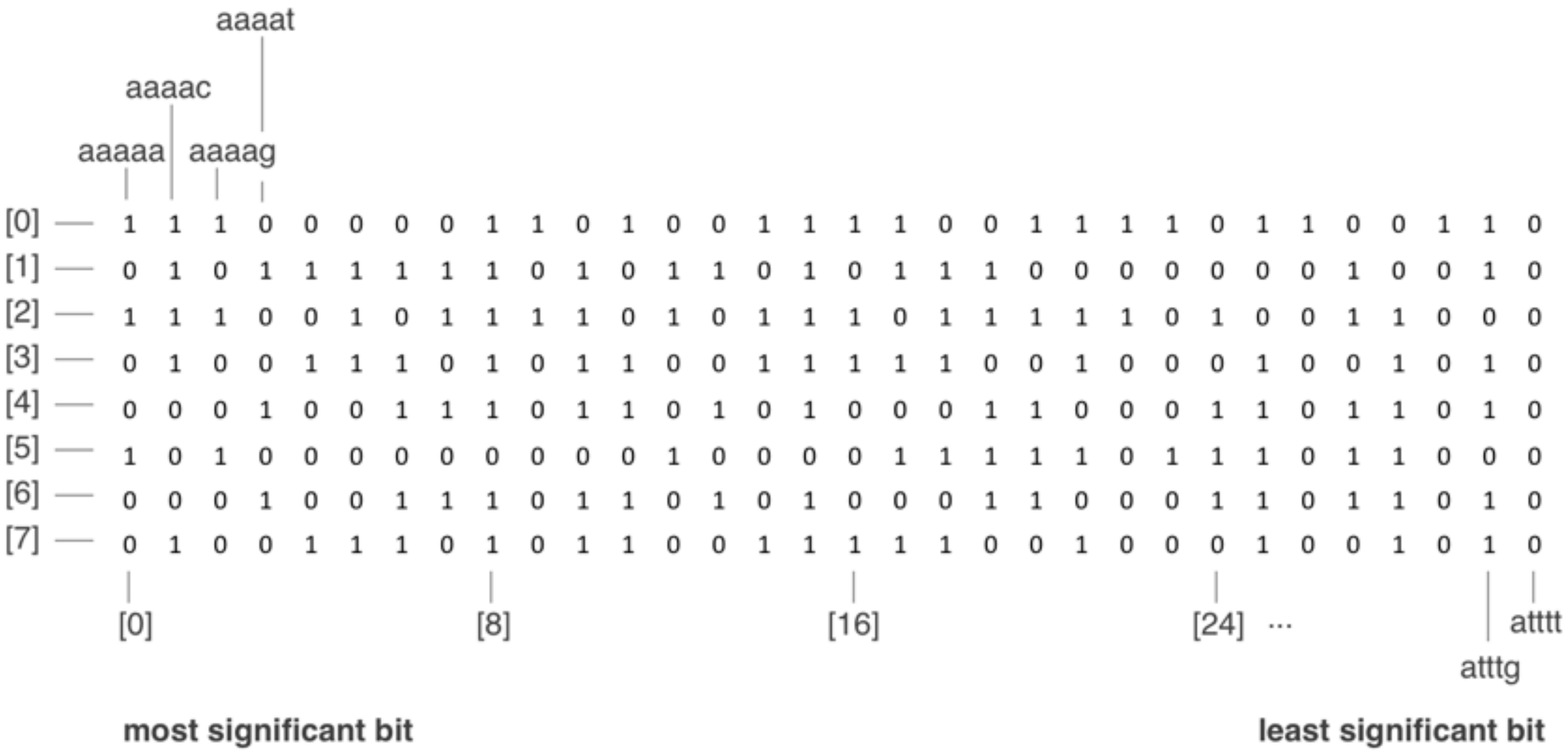
Set Representation

0	0
1	1
2	1
3	0
⋮	
4^l	0

*In this example we know that l-mers with value of **1** and **2** belong to the set.*

1 = AAAAC and 2 = AAAAG, if l = 5

3. Bit-array compression



	aaaaa	aaaac	aaaag	aaaat																												
[0]	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0
[1]	0	1	0	1	1	1	1	1	1	0	1	0	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	0
[2]	1	1	1	0	0	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	0	1	0	0	1	1	0	0	0
[3]	0	1	0	0	1	1	1	0	1	0	1	1	0	0	1	1	1	1	1	0	0	1	0	0	0	1	0	0	1	0	1	0
[4]	0	0	0	1	0	0	1	1	1	0	1	1	0	1	0	1	0	0	0	1	1	0	0	0	1	1	0	1	1	0	1	0
[5]	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	1	1	1	0	1	1	0	0	0
[6]	0	0	0	1	0	0	1	1	1	0	1	1	0	1	0	1	0	0	0	1	1	0	0	0	1	1	0	1	1	0	1	0
[7]	0	1	0	0	1	1	1	0	1	0	1	1	0	0	1	1	1	1	1	0	0	1	0	0	0	1	0	0	1	0	1	0
	[0]								[8]								[16]								[24]	···					atttg	atttt

most significant bit

least significant bit

Implementation: Speedup Techniques

3. Bit-array compression

** each element contains a 32-bit integer for bit flags instead of one.*

Implementation: Speedup Techniques

3. Bit-array compression

** the process of setting and checking the value:*

gacgt = 1000011011 = **539** in decimal

bit position = 539 mod 32 = 27;

array index = 539 / 32 = 16;

*The bit flag for **gacgt** is in the 27th least significant bit of the integer at array index 16.*

Implementation: Speedup Techniques

4. XOR-based Hamming distance computation

Example:

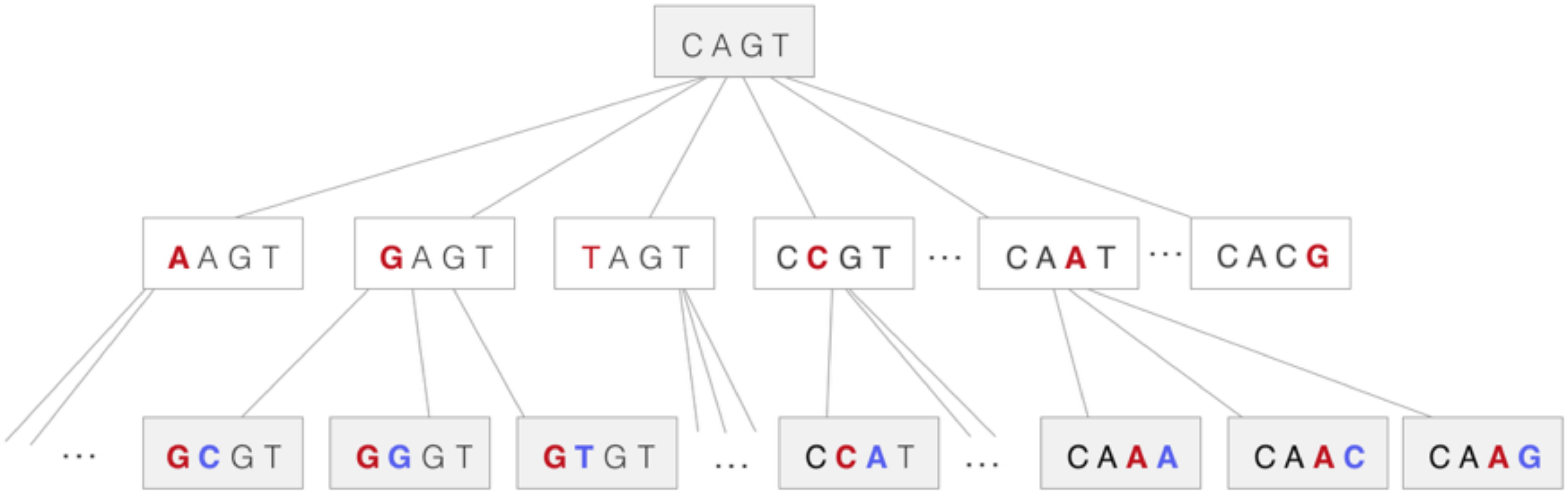
aacgt maps to 0000011011

tacgc maps to 1100011001

XOR produces 1100000010 *= 2 mismatches*

Implementation: Speedup Techniques

5. Recursive neighborhood generation



Implementation: Speedup Techniques

6. Block-based optimization for neighborhood generation

- *instead of setting one bit at a time in the neighborhood generation, we generate the neighborhood by blocks*

Implementation: Speedup Techniques

6. Block-based optimization for neighborhood generation

- *instead of setting one bit at a time in the neighborhood generation, we generate the neighborhood by blocks*

- *if we partition the bit-array N into* ***4^k*** *of bits, where $k < l$, each block will conform to one of* ***(k + 2) patterns.***

Implementation: Speedup Techniques

6. Block-based optimization for neighborhood generation

- instead of setting one bit at a time in the neighborhood generation, we generate the neighborhood by blocks

- if we partition the bit-array N into ***4^k*** *of bits, where k < l, each block will conform to one of* ***(k + 2) patterns.***

- we pre-computed these ***block patterns*** *and use it to set the bits in the bit-array by blocks.*

Implementation: Speedup Techniques

6. Block-based optimization for neighborhood generation

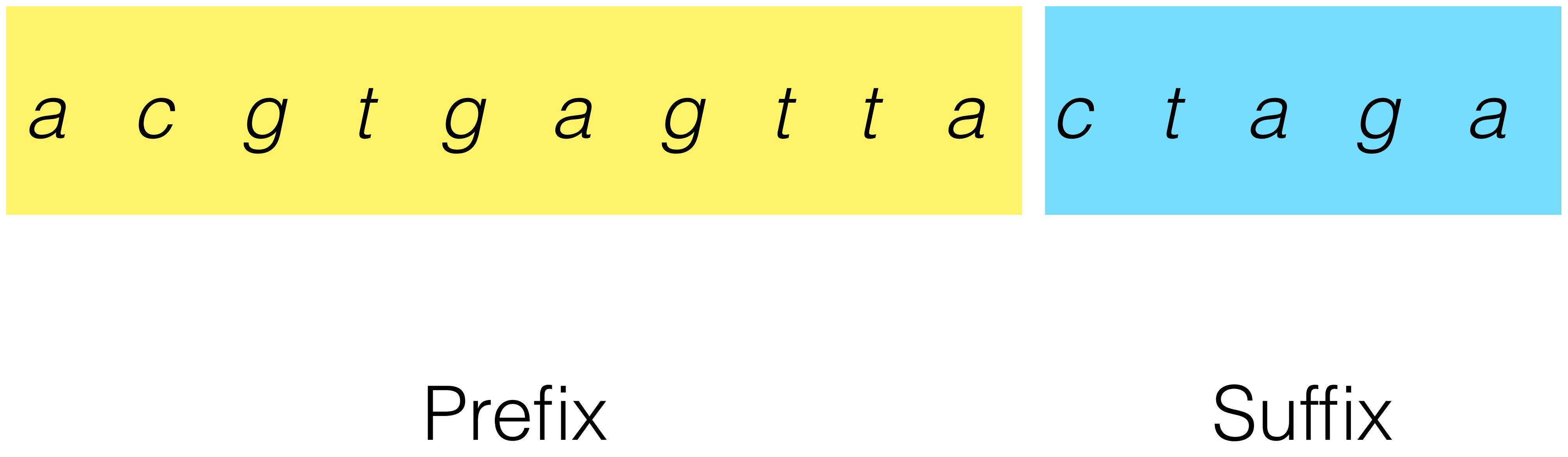
Lets say we want to generate the d-neighborhood of 15-mer

a c g t g a g t t a c t a g a

Implementation: Speedup Techniques

6. Block-based optimization for neighborhood generation

Lets say we want to generate the d-neighborhood of 15-mer



a c g t g a g t t a *c t a g a*

Prefix Suffix

Implementation: Speedup Techniques

6. Block-based optimization for neighborhood generation

Now instead of recursively generating all d-neighbor of length ***l****, we only consider up to (l-k) prefix of l. For each prefix, we apply the corresponding block pattern based on the remaining allowed mutations.*

Additional Speedup Techniques

1. Faster Candidate Motif Elimination through Block Processing.

1. Faster Candidate Motif Elimination through Block Processing.

Observations:

- The bit array that stores the candidate motifs is enumerated alphabetically.

1. Faster Candidate Motif Elimination through Block Processing.

Observations:

- The bit array that stores the candidate motifs is enumerated alphabetically.

- Given the alphabetical enumeration, *l*-mers near other do not differ that much.

1. Faster Candidate Motif Elimination through Block Processing.

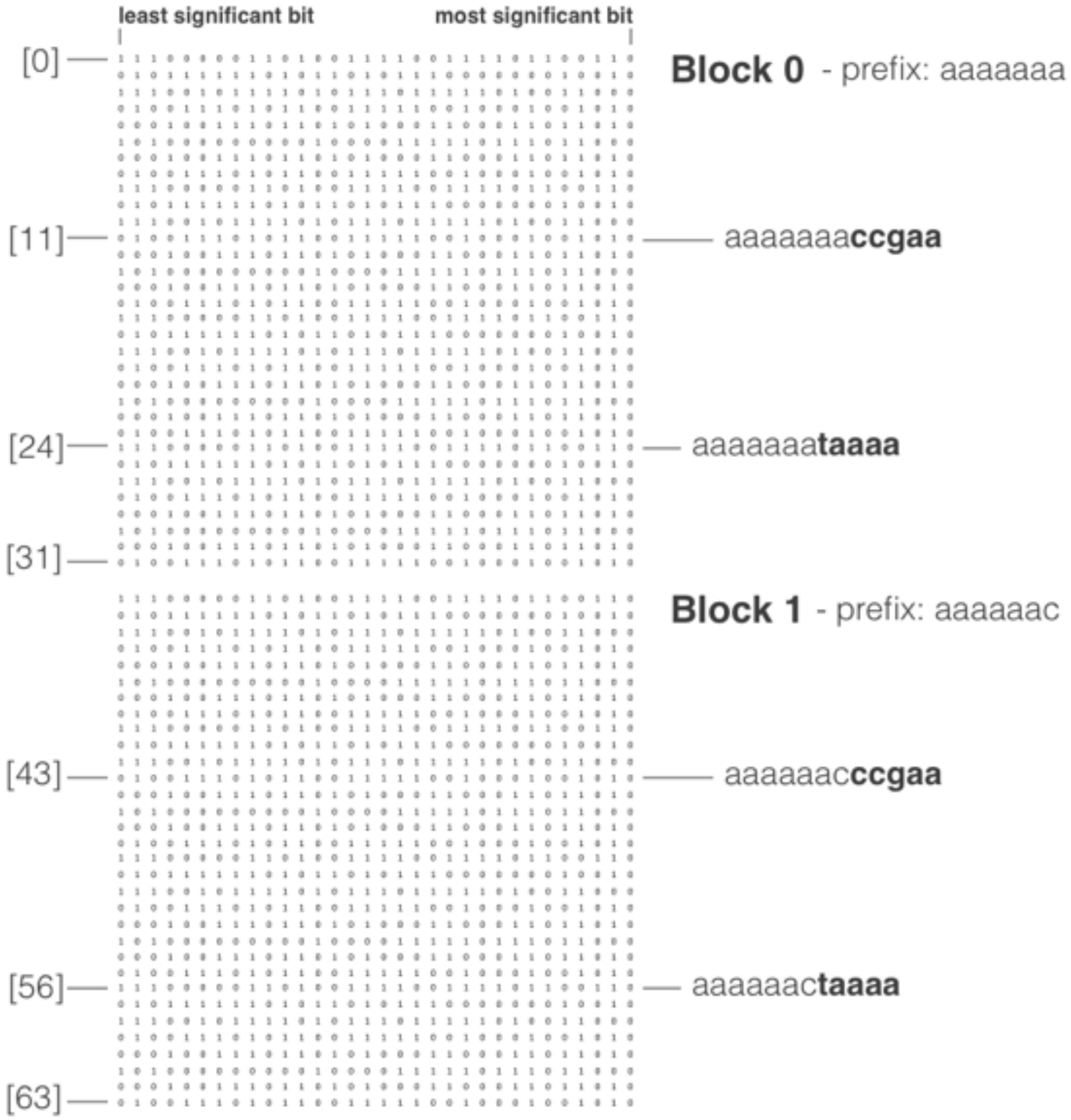
Observations:

- The bit array that stores the candidate motifs is enumerated alphabetically.

- Given the alphabetical enumeration, *l*-mers near other do not differ that much.

- In EMS-GT, each candidate motif testing is independent of other.

Faster Candidate Motif Elimination through Block Processing.



least significant bit

most significant bit

[0] — **Block 0** - prefix: aaaaaaa

[11] — aaaaaaa**ccgaa**

[24] — aaaaaaa**taaaa**

[31]

Block 1 - prefix: aaaaaac

[43] — aaaaaac**ccgaa**

[56] — aaaaaac**taaaa**

[63]

By grouping every 4^k bits into blocks, we can exploit some of the blocks' properties.

Faster Candidate Motif Elimination through Block Processing.

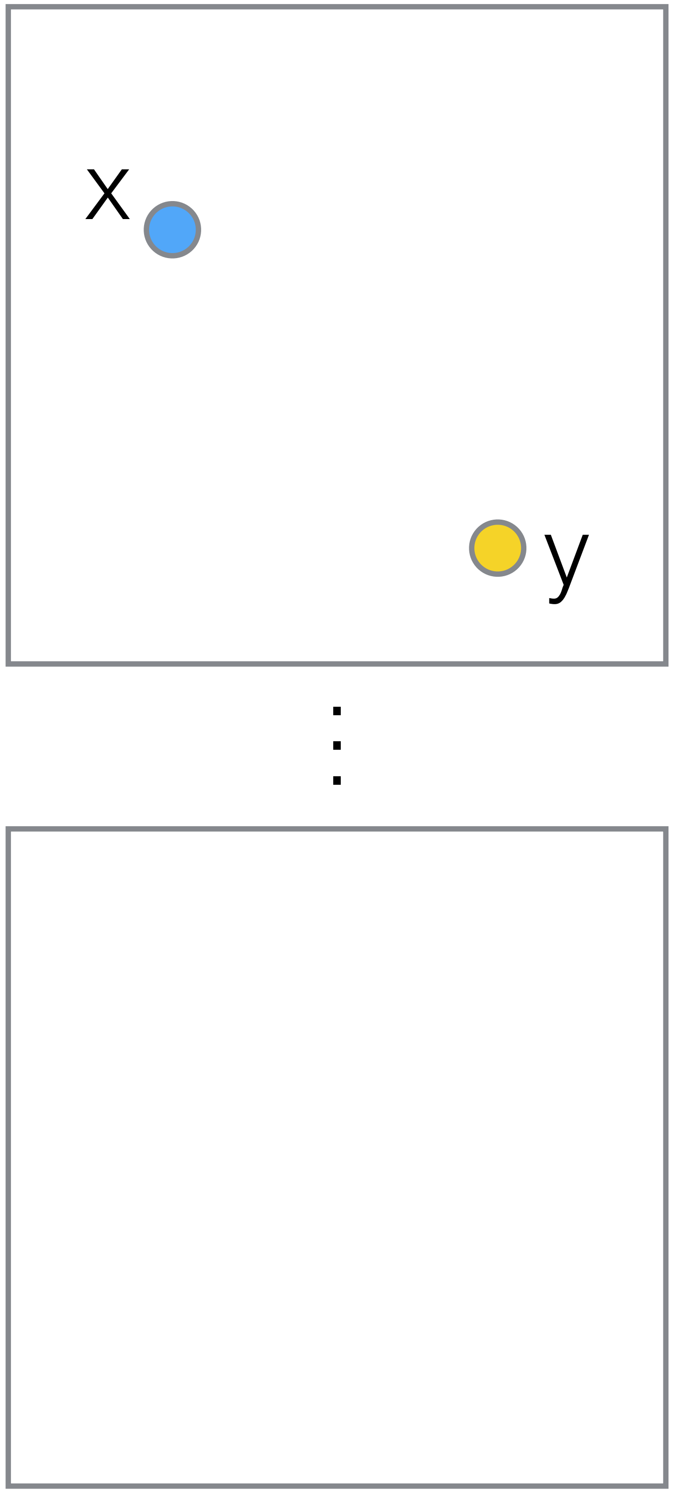
By grouping every 4^k bits into blocks, we can derive these properties:

- *Each l-mer in a block shares the same prefix string.*
- *For every block, the hamming distance of any two l-mers is at most k.*

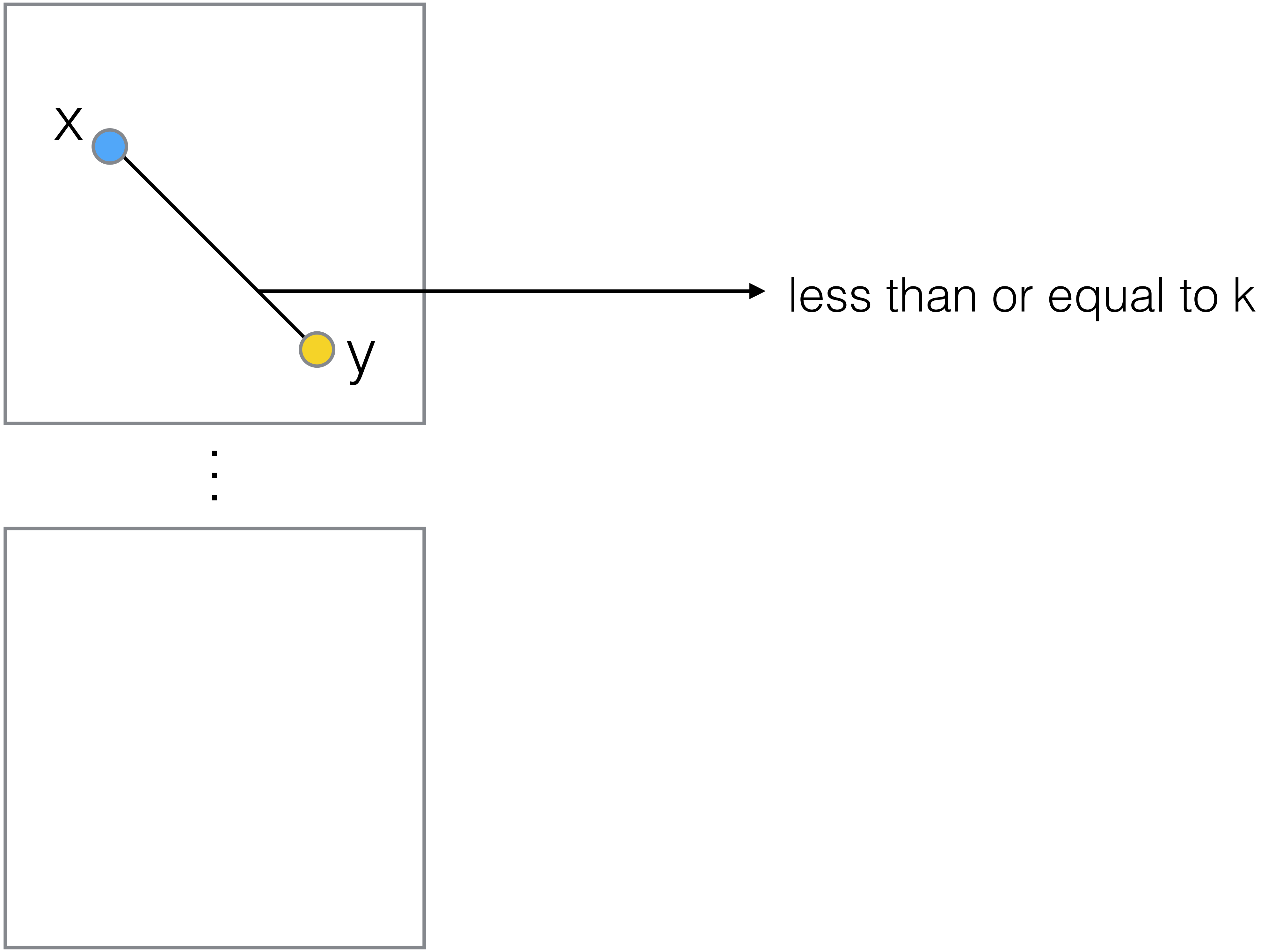
Faster Candidate Motif Elimination through Block Processing.

Theorem. *Let x and y be l-mers in a block in the search space containing 4^k l-mers. Let d be the number of allowed mutations in the problem instance. Let z be another l-mer. If $dH(x, z) > (k + d)$ then $dH(y, z) > d$, and therefore z is not in $N(y, d)$*

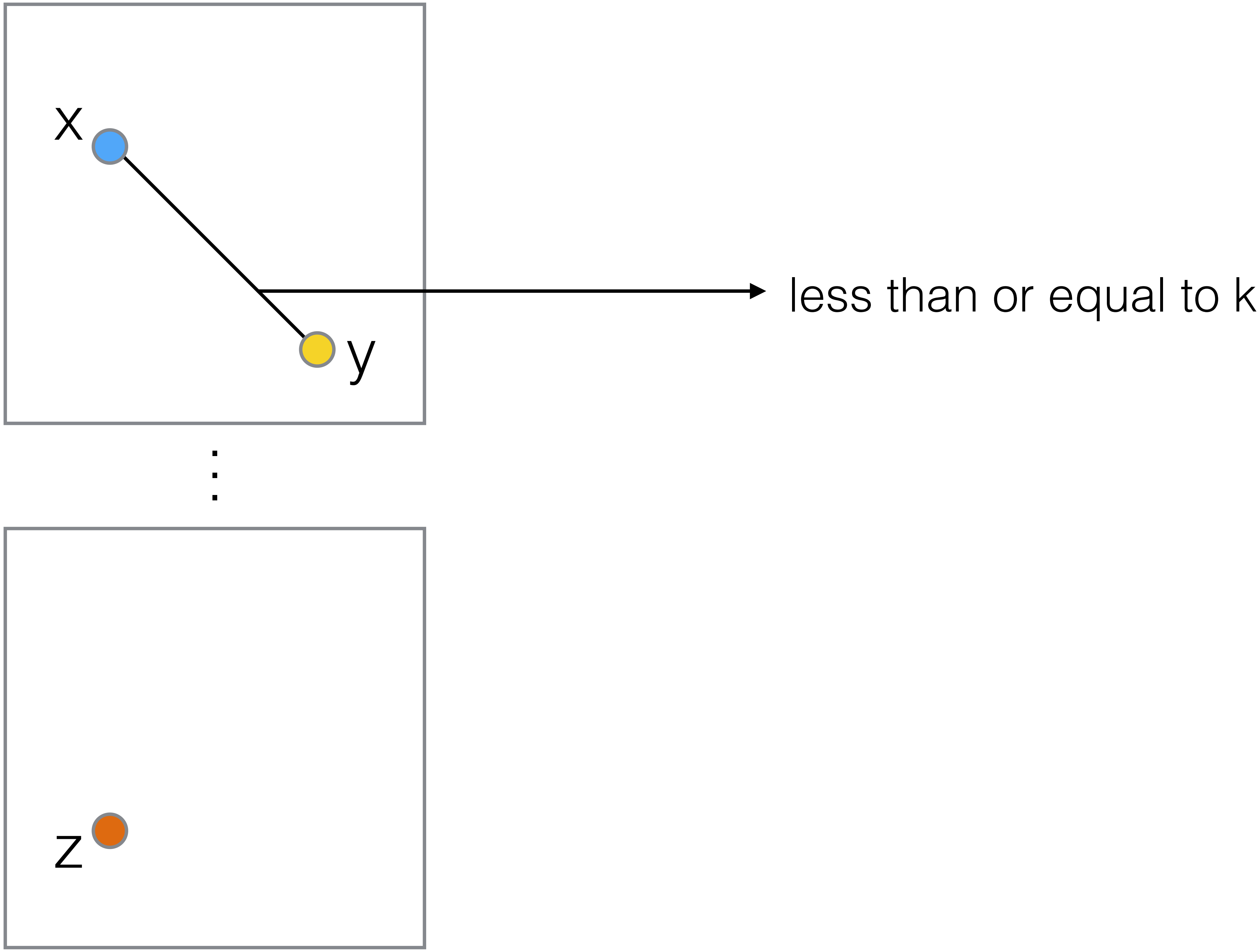
Faster Candidate Motif Elimination through Block Processing.



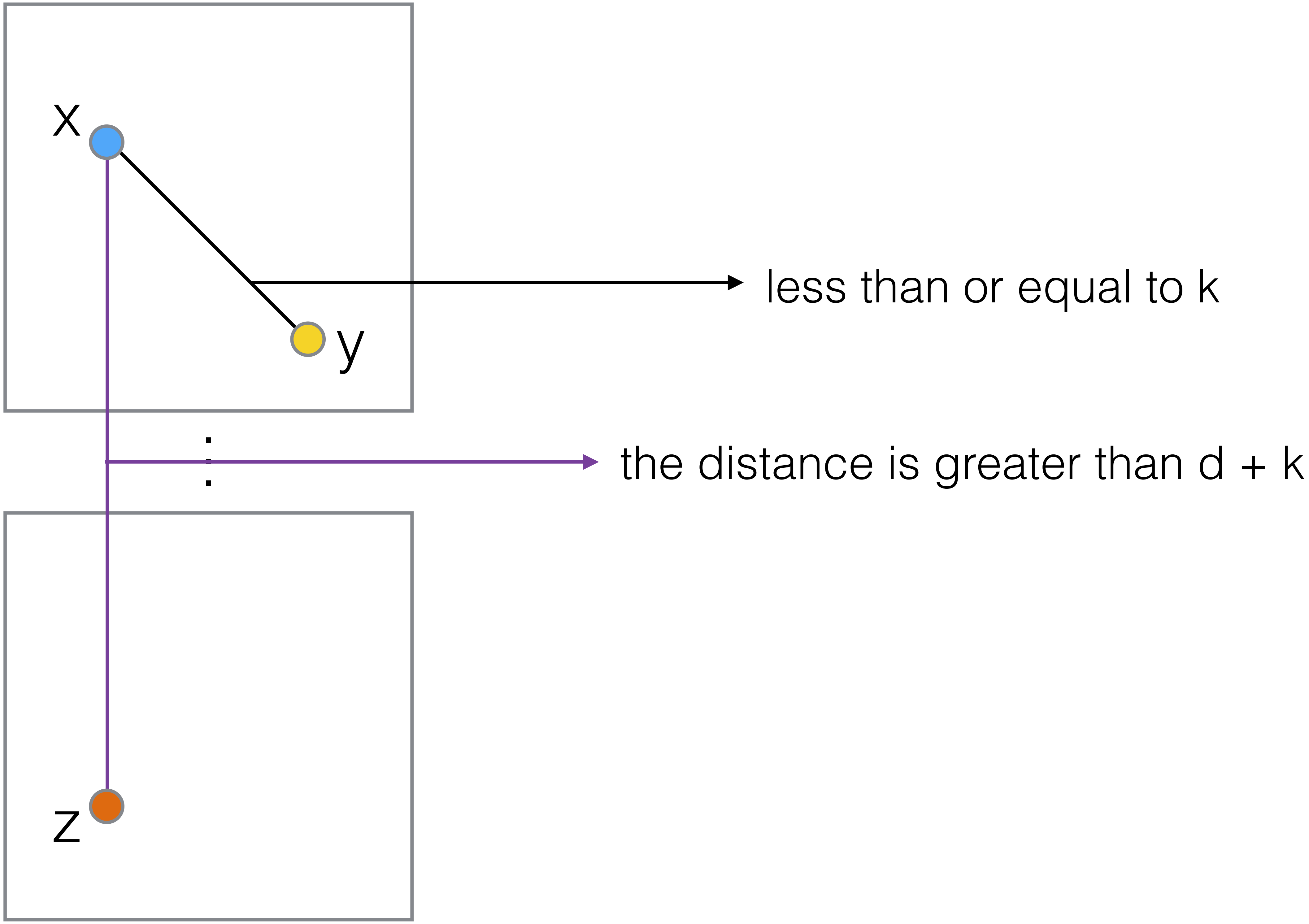
Faster Candidate Motif Elimination through Block Processing.



Faster Candidate Motif Elimination through Block Processing.



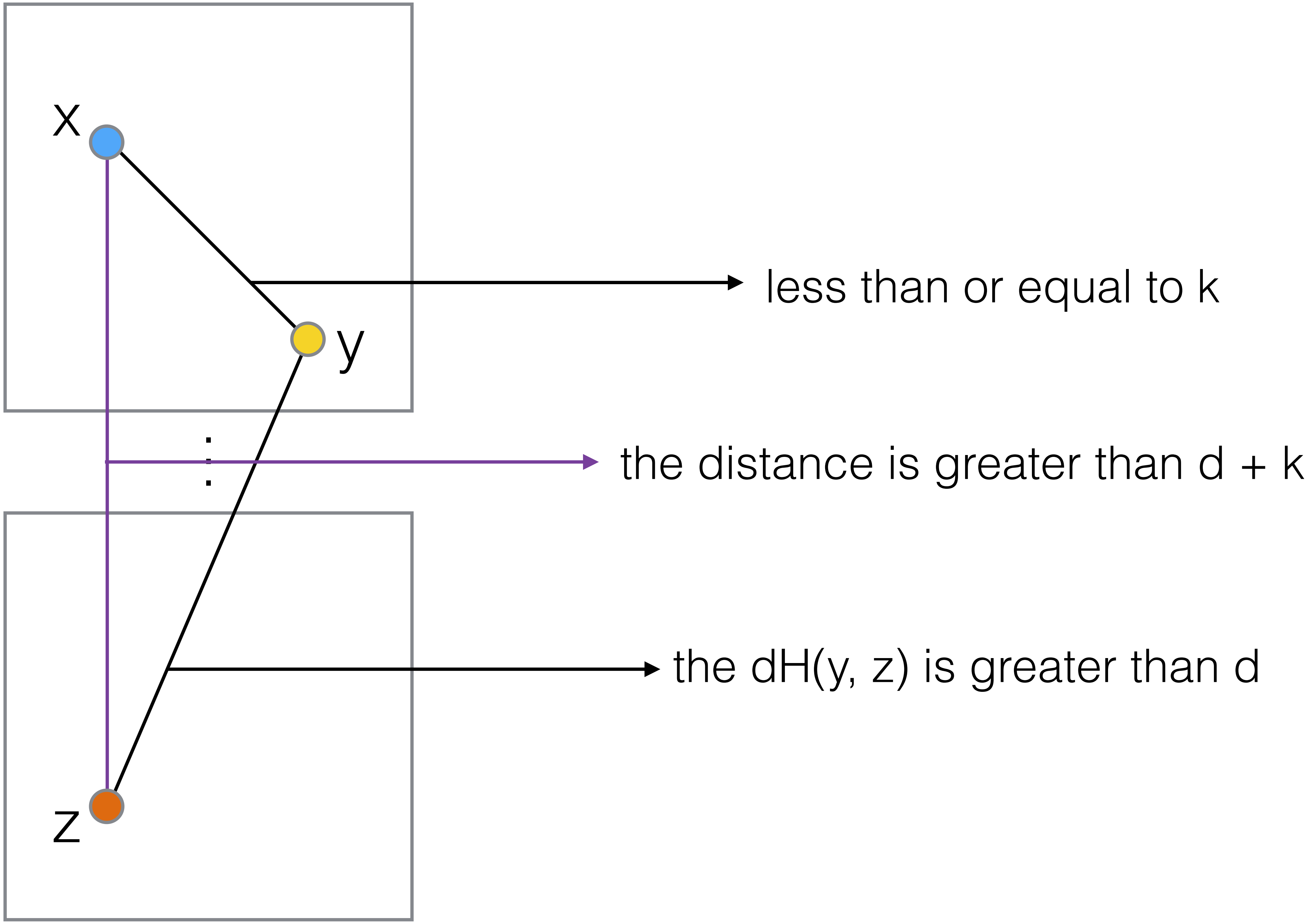
Faster Candidate Motif Elimination through Block Processing.



less than or equal to k

the distance is greater than d + k

Faster Candidate Motif Elimination through Block Processing.



less than or equal to k

the distance is greater than d + k

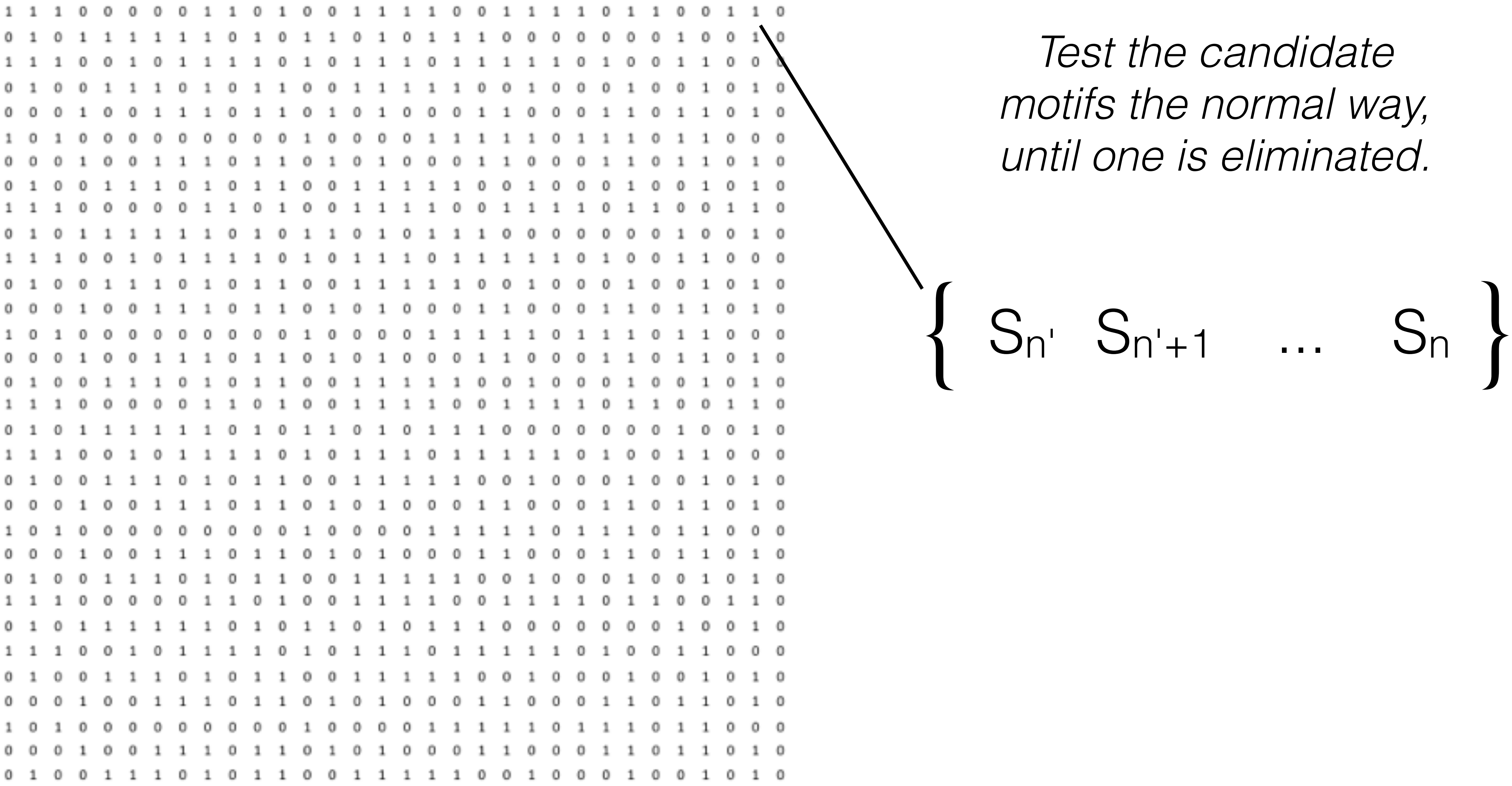
the dH(y, z) is greater than d

Faster Candidate Motif Elimination through Block Processing.

We used this theorem in the Test Phase by filtering some of the l-mers in a sequence S.

Faster Candidate Motif Elimination through Block Processing.

Improved Test Phase:



```
1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 1 1 0
0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0
1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 1 1 0
0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0
1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 1 1 0
0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0
1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 1 1 0
0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0
1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 0
0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
```

Test the candidate motifs the normal way, until one is eliminated.

$$\left\{ S_{n'} \quad S_{n'+1} \quad \dots \quad S_{n} \right\}$$

Faster Candidate Motif Elimination through Block Processing.

Improved Test Phase:

gagct

$\mathbf{S_{n''}}$ t a g a g c a c a t a g a c c t g a c a c a t a g t a c t t

↓ l-mers in sequence Sn''

{ tagag, agagc, gagca, agcac ... tactt }

Faster Candidate Motif Elimination through Block Processing.

Improved Test Phase:

gagct

$S_{n''}$ t a g a g c a c a t a g a c c t g a c a c a t a g t a c t t

↓ l-mers in sequence Sn''

{ tagag, ~~agagc,~~ gagca, ~~agcac~~ ... tactt }

we can filter some l-mers in the set of (m-l+1) l-mers of Sn''
where the candidate motif got eliminated

Faster Candidate Motif Elimination through Block Processing.

*We now use this **filtered** set of l-mers over the original sequence for testing the remaining candidate motifs in the block.*

2. Pre-computation of Mismatch values for Hamming distance Computation

2. Pre-computation of Mismatch values for Hamming distance Computation

Example:

aacgt maps to 0000011011
tacgc maps to 1100011001

XOR produces 1100000010 *= 2 mismatches*

2. Pre-computation of Mismatch values for Hamming distance Computation

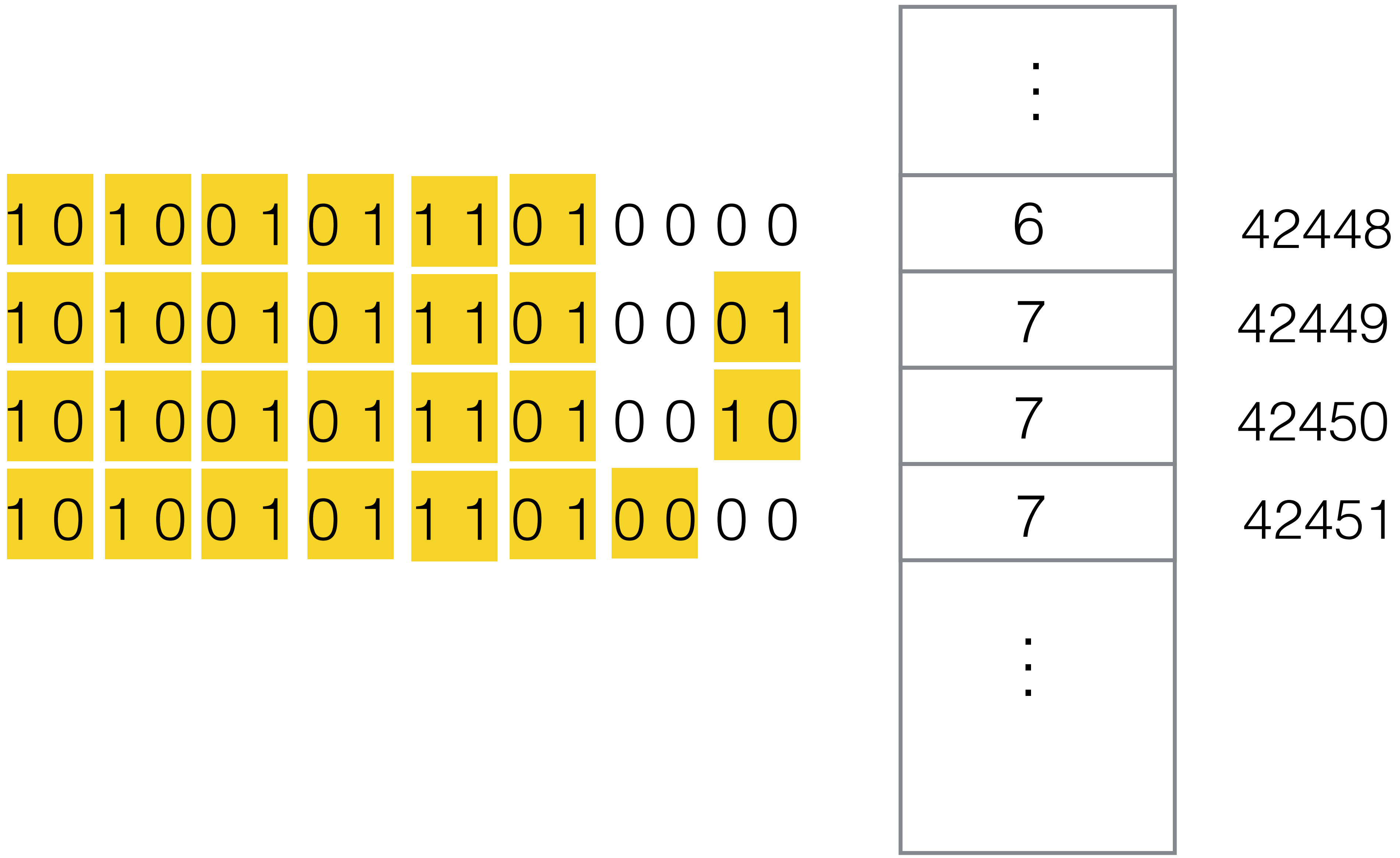
Originally, EMS-GT counts each pairs of bits in the XOR result for the Hamming distance computation.

2. Pre-computation of Mismatch values for Hamming distance Computation

We can improve this by referring to a pre-computed mismatch count values instead of counting the pair of bits every Hamming distance computation.

Pre-computation of Mismatch values

The pre-computed array of mismatch values lookup



	⋮	
1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0	6	42448
1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 1	7	42449
1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 0	7	42450
1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0	7	42451
	⋮	

Pre-computation of Mismatch values

Improved Hamming distance computation:

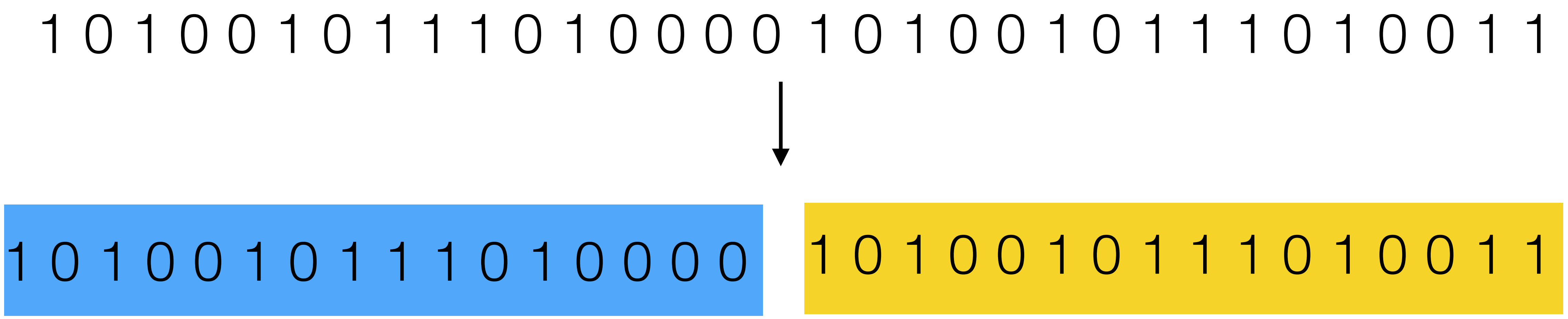
Example: Given a pre-computed values up to 16-bits and an XOR result:

1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

Pre-computation of Mismatch values

Improved Hamming distance computation:

Example: Given a pre-computed values up to 16-bits and an XOR result:



1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

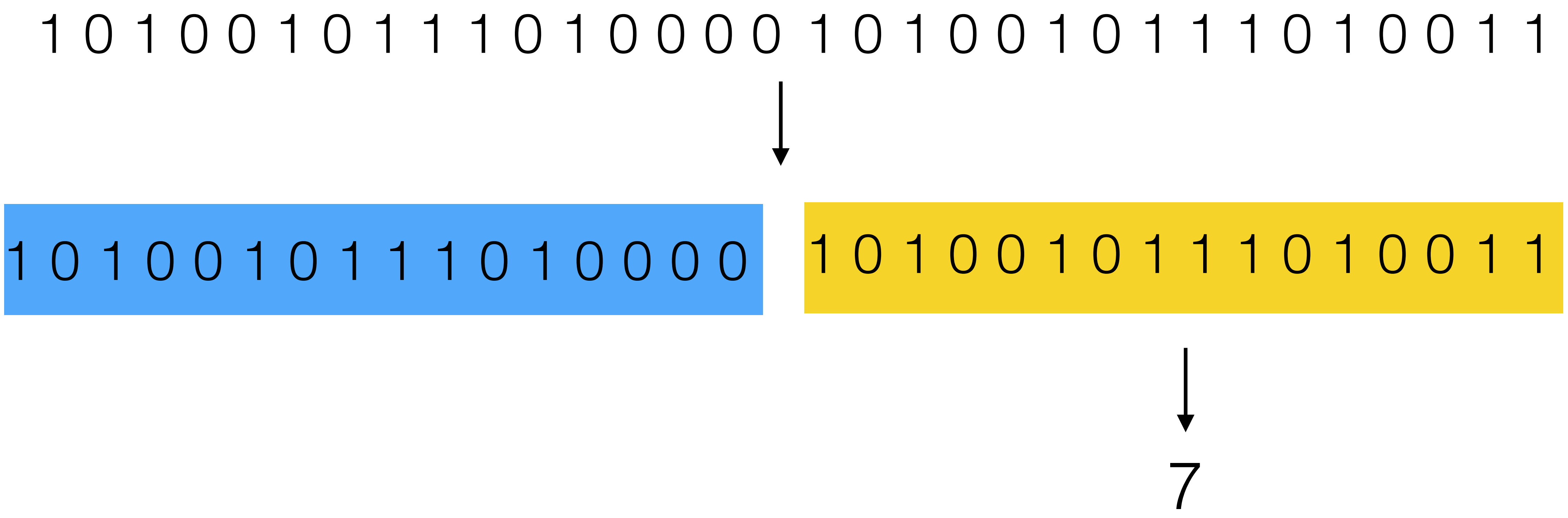
1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0

1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

Pre-computation of Mismatch values

Improved Hamming distance computation:

Example: Given a pre-computed values up to 16-bits and an XOR result:



1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0

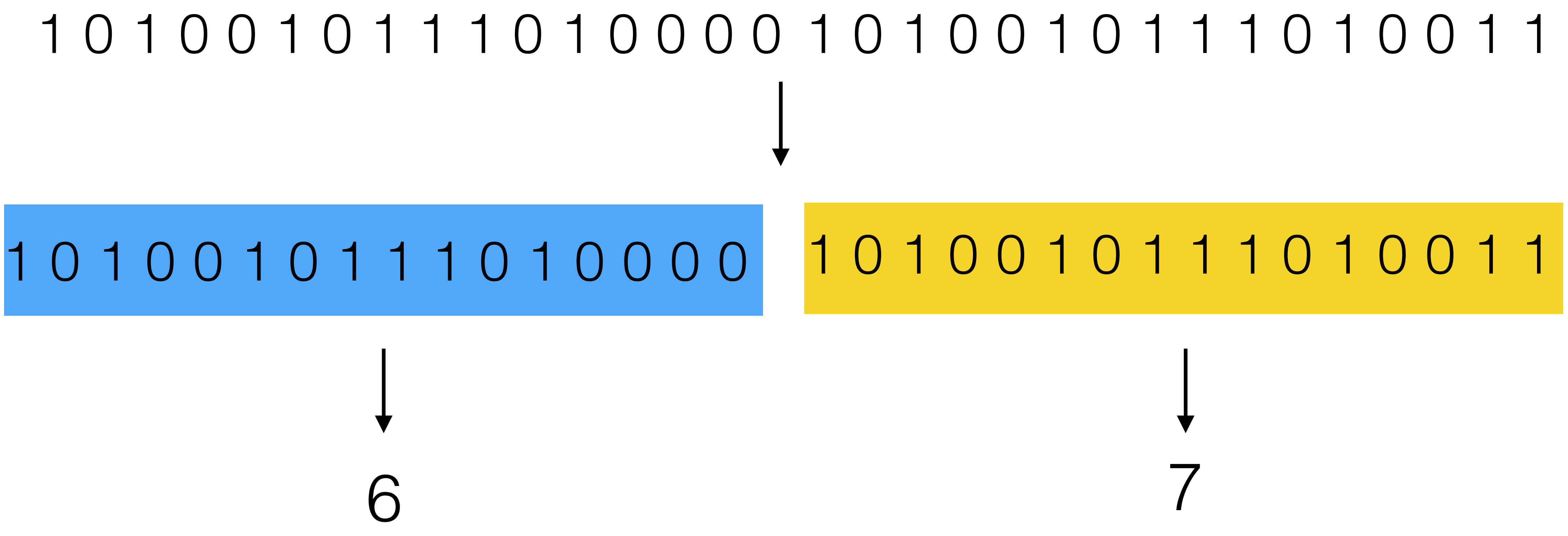
1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

7

Pre-computation of Mismatch values

Improved Hamming distance computation:

Example: Given a pre-computed values up to 16-bits and an XOR result:



1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 0

1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1

6

7

6 + 7 = 13 the hamming distance is **13**

Methods

Methods

Dataset

20 DNA sequences (n)

each sequence is 600 characters long (m)

used (l, d)-challenge instances, where l < 18.

Methods

Evaluation

Runtime evaluation of EMS-GT, qPMS9 and EMS-GT2 using ***(9, 2), (11, 3), (13, 4), (15, 5) and (17, 6)*** *challenge instances over 20 iterations each.*

Performance

Performance:

Experimentation Results:

	qPMS9	EMS-GT	EMS-GT2
(9, 2)	0.60 s	**0.04 s**	0.05 s
(11, 3)	1.26 s	**0.17 s**	0.26 s
(13, 4)	4.58 s	1.03 s	**0.82 s**
(15, 5)	25.73 s	12.39 s	**10.43 s**
(17, 6)	123.17 s	143.87 s	**111.22 s**

Conclusions:

EMS-GT2 is efficient in finding short motifs
where l < 18

EMS-GT2 proved its competitiveness by beating
qPMS9 in all (l, d)-challenge instances
where l < 18.

In practice, motifs are typically 10 base pairs.