Finding Common Motifs in DNA Sequences

A survey

Nuno D. Mendes

ndm@algos.inesc-id.pt

Instituto Superior Técnico

Universidade Técnica de Lisboa

Outline

- Some biology
- The problem
- An early graph-based approach: WINNOWER
- Avoiding the explicit search for cliques: PRUNER
- An approach based on suffix-trees: SMILE
- Merging WINNOWER and SMILE: MITRA
- Conclusions

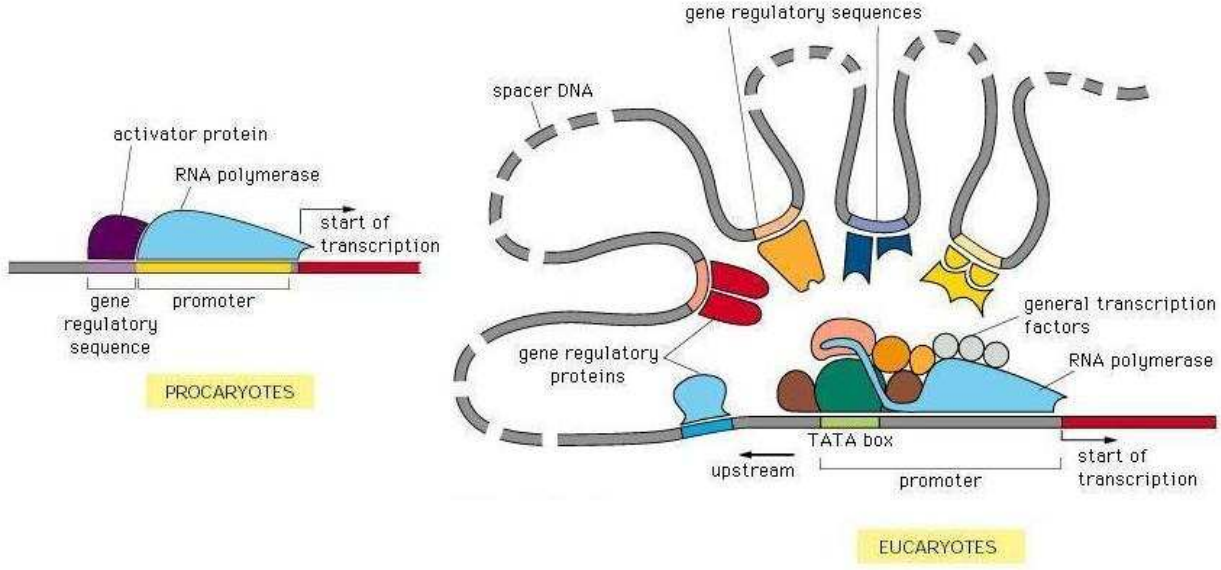
Some Biology

Some Biology

- Transcriptional regulation is one the most important mechanisms regulating gene expression

Some Biology

- Transcriptional regulation is one the most important mechanisms regulating gene expression
  - Directed by proteins which specifically bind to motifs present in regulatory regions



Some Biology

- Co-regulated genes are expected to share motifs in their regulatory regions

Some Biology

- Co-regulated genes are expected to share motifs in their regulatory regions
- Sometimes with errors

Problem

- Given a set of sequences $\mathcal{S} = \{S_1, \ldots, S_t\}$

Problem

- Given a set of sequences $\mathcal{S} = \{S_1, \ldots, S_t\}$
- Find all motifs of length $l \in (l_{\min}, \ldots, l_{\max})$ (l-mers) which occur in $q \leq t$ sequences with at most e errors

Problem

- Given a set of sequences $\mathcal{S} = \{S_1, \ldots, S_t\}$
- Find all motifs of length $l \in (l_{\min}, \ldots, l_{\max})$ (l-mers) which occur in $q \leq t$ sequences with at most e errors

$l = 5 \qquad e = 1 \qquad q = t = 3$

S_1 : GATTGCATCATAGATCCGATT
S_2 : GACCGTACGCCATACGATTCA
S_3 : ACTCATAAGCCTACTTAGCTA

should find CATAA | CATAT | CATAG | CATAC

Problem

- Given a set of sequences $\mathcal{S} = \{S_1, \ldots, S_t\}$
- Find all motifs of length $l \in (l_{\min}, \ldots, l_{\max})$ (l-mers) which occur in $q \leq t$ sequences with at most e errors

$l = 5 \qquad e = 1 \qquad q = t = 3$

S_1 : GATTGCATCATAGATCCGATT
S_2 : GACCGTACGCCATACGATTCA
S_3 : ACTCATAAGCCTACTTAGCTA

should find $\boxed{\text{CATAA}}$ $\boxed{\text{CATAT}}$ $\boxed{\text{CATAG}}$ $\boxed{\text{CATAC}}$

Note that CATAT never occurs exactly

Problem

- So far we have thought of a motif as a contiguous string: *simple motif*

Problem

- So far we have thought of a motif as a contiguous string: *simple motif*
- Sometimes it is useful to search for *composite motifs*

Problem

- So far we have thought of a motif as a contiguous string: *simple motif*
- Sometimes it is useful to search for *composite motifs*
  - combined simple motifs co-occurring in non-overlapping positions in various input sequences at a certain distance from one another

Problem

- So far we have thought of a motif as a contiguous string: *simple motif*
- Sometimes it is useful to search for *composite motifs*
  - combined simple motifs co-occurring in non-overlapping positions in various input sequences at a certain distance from one another
    - Easier to find, especially if one of the components is poorly conserved across the input sequences
    - The added distance restriction avoids the extraction of too many motifs

Notation

- All sequences are defined over $\Sigma = \{A, T, G, C\}$

Notation

- All sequences are defined over $\Sigma = \{A, T, G, C\}$
- $S_j[i]$ denotes the ith character on the jth sequence

Notation

- All sequences are defined over $\Sigma = \{A, T, G, C\}$
- $S_j[i]$ denotes the ith character on the jth sequence
- $m_{ij} \in \Sigma^l$ denotes the l-mer starting in position i of S_j

Notation

- All sequences are defined over $\Sigma = \{A, T, G, C\}$
- $S_j[i]$ denotes the ith character on the jth sequence
- $m_{ij} \in \Sigma^l$ denotes the l-mer starting in position i of S_j
- n_j denotes the length of sequence S_j

Notation

- All sequences are defined over $\Sigma = \{A, T, G, C\}$
- $S_j[i]$ denotes the ith character on the jth sequence
- $m_{ij} \in \Sigma^l$ denotes the l-mer starting in position i of S_j
- n_j denotes the length of sequence S_j
- If $m, m' \in \Sigma^l$ then $\delta(m, m')$ denotes the Hamming distance between both

Notation

- All sequences are defined over $\Sigma = \{A, T, G, C\}$
- $S_j[i]$ denotes the ith character on the jth sequence
- $m_{ij} \in \Sigma^l$ denotes the l-mer starting in position i of S_j
- n_j denotes the length of sequence S_j
- If $m, m' \in \Sigma^l$ then $\delta(m, m')$ denotes the Hamming distance between both
- Finally,

$$N = \sum_{i=1}^{t} n_i$$

WINNOWER

- Builds a graph $G = (V, E)$

WINNOWER

- Builds a graph $G = (V, E)$
- Each l-mer in the input sequences is a vertex

$$V = \{m_{ij} : 1 \leq i \leq n_j - l + 1 \wedge 1 \leq j \leq t\}$$

WINNOWER

- Builds a graph $G = (V, E)$
- Each l-mer in the input sequences is a vertex

$$V = \{m_{ij} : 1 \leq i \leq n_j - l + 1 \wedge 1 \leq j \leq t\}$$

- There is an edge between a pair of vertices coming from different input sequences provided that they do not mismatch in more than $2e$ positions

$$E = \{(m_{ij}, m_{rs}) : m_{ij}, m_{rs} \in V \wedge \delta(m_{ij}, m_{rs}) \leq 2e \wedge j \neq s\}$$

WINNOWER

- Builds a graph $G = (V, E)$
- Each l-mer in the input sequences is a vertex

$$V = \{m_{ij} : 1 \leq i \leq n_j - l + 1 \wedge 1 \leq j \leq t\}$$

- There is an edge between a pair of vertices coming from different input sequences provided that they do not mismatch in more than $2e$ positions

$$E = \{(m_{ij}, m_{rs}) : m_{ij}, m_{rs} \in V \wedge \delta(m_{ij}, m_{rs}) \leq 2e \wedge j \neq s\}$$

This way, a motif occurring on q different sequences with at most e errors corresponds to a q-clique in G

WINNOWER

- Builds a graph $G = (V, E)$
- Each l-mer in the input sequences is a vertex

$$V = \{m_{ij} : 1 \leq i \leq n_j - l + 1 \wedge 1 \leq j \leq t\}$$

- There is an edge between a pair of vertices coming from different input sequences provided that they do not mismatch in more than $2e$ positions

$$E = \{(m_{ij}, m_{rs}) : m_{ij}, m_{rs} \in V \wedge \delta(m_{ij}, m_{rs}) \leq 2e \wedge j \neq s\}$$

This way, a motif occurring on q different sequences with at most e errors corresponds to a q-clique in G

For $q > 2$, the converse is not necessarily true

WINNOWER

- Builds a graph $G = (V, E)$
- Each l-mer in the input sequences is a vertex

$$V = \{m_{ij} : 1 \leq i \leq n_j - l + 1 \wedge 1 \leq j \leq t\}$$

- There is an edge between a pair of vertices coming from different input sequences provided that they do not mismatch in more than $2e$ positions

$$E = \{(m_{ij}, m_{rs}) : m_{ij}, m_{rs} \in V \wedge \delta(m_{ij}, m_{rs}) \leq 2e \wedge j \neq s\}$$

This way, a motif occurring on q different sequences with at most e errors corresponds to a q-clique in G

For $q > 2$, the converse is not necessarily true

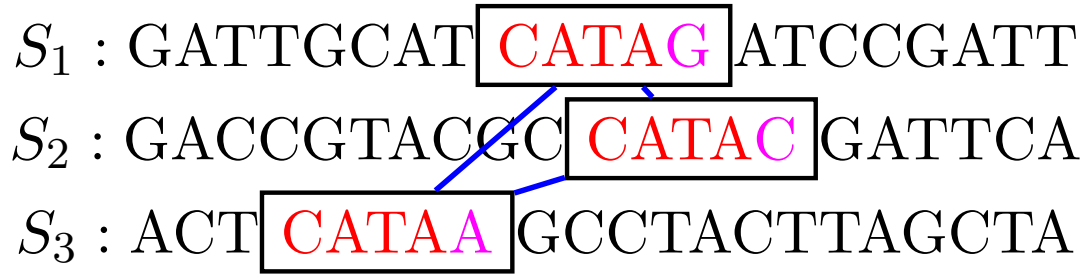
- Example:

  $\mathrm{ATAT} \qquad \mathrm{GTGT} \qquad \mathrm{TTTT}$

  are at a distance no larger than 2 from one another, but there is no pattern at a distance of 1 from all of them

WINNOWER

- Example of a 3-clique where all edges of G participate on the clique



WINNOWER

- Our problem is reduced to finding q-cliques in the t-partite graph

WINNOWER

- Our problem is reduced to finding q-cliques in the t-partite graph
- Finding a maximal k-clique in a graph for $k > 2$ is NP-Complete

WINNOWER

- Our problem is reduced to finding q-cliques in the t-partite graph
- Finding a maximal k-clique in a graph for $k > 2$ is NP-Complete
- WINNOWER adopts a winnowing strategy to try to remove all edges which cannot be part of a q-clique in hope that after the iterative process only q-cliques remain

WINNOWER

- A vertex u is a *neighbour* of a clique $C = \{v_1, \ldots, v_k\}$ if $\{v_1, \ldots, v_k, u\}$ is also a clique in the graph

WINNOWER

- A vertex u is a *neighbour* of a clique $C = \{v_1, \ldots, v_k\}$ if $\{v_1, \ldots, v_k, u\}$ is also a clique in the graph
- A clique is *extendable* if it has at least one neighbour in each partition of the graph

WINNOWER

- A vertex u is a *neighbour* of a clique $C = \{v_1, \ldots, v_k\}$ if $\{v_1, \ldots, v_k, u\}$ is also a clique in the graph
- A clique is *extendable* if it has at least one neighbour in each partition of the graph
- An edge is said to be *spurious* if it does not belong to any extendable cliques of some size k

WINNOWER

- A vertex u is a *neighbour* of a clique $C = \{v_1, \ldots, v_k\}$ if $\{v_1, \ldots, v_k, u\}$ is also a clique in the graph
- A clique is *extendable* if it has at least one neighbour in each partition of the graph
- An edge is said to be *spurious* if it does not belong to any extendable cliques of some size k

$\rightarrow$ The algorithm will remove all spurious edges for increasing values of k

WINNOWER

- For $k = 1$
  - A vertex u is a neighbour of vertex v if $(u, v) \in E$
  - We need to remove every vertex which does not have at least $q - 1$ neighbours in different partitions

WINNOWER

- For $k = 1$
  - A vertex u is a neighbour of vertex v if $(u, v) \in E$
  - We need to remove every vertex which does not have at least $q - 1$ neighbours in different partitions
- For $k = 2$
  - A vertex u is a neighbour of an edge (v, w) if $\{v, w, u\}$ is a triangle in G
  - We need to remove every edge which does not have at least $q - 2$ neighbours in different partitions

WINNOWER

- For $k > 2$
  - Observe that for any clique of size q there are $\binom{q}{k}$ extendable cliques with k vertices

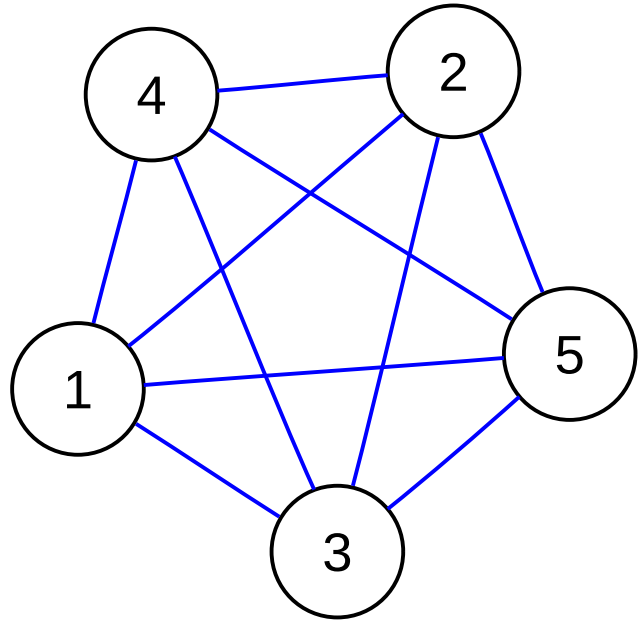
WINNOWER

- For $k > 2$
  - Observe that for any clique of size q there are $\binom{q}{k}$ extendable cliques with k vertices
  - Thus, every edge on a q-clique belongs to at least $\binom{q-2}{k-2}$ extendable clique of size k

WINNOWER

- For $k > 2$
  - Observe that for any clique of size q there are $\binom{q}{k}$ extendable cliques with k vertices
  - Thus, every edge on a q-clique belongs to at least $\binom{q-2}{k-2}$ extendable clique of size k
  - WINNOWER works up to $k = 3$, in which case, all edges not belonging to at least $q - 2$ extendable cliques are deemed inconsistent and removed

WINNOWER



A maximal 5-clique. All edges must participate in 3 different triangles.

WINNOWER

- Time complexity for $k = 3$ in $O(t^4 N^{2.66})$

WINNOWER

- Time complexity for $k = 3$ in $O(t^4 N^{2.66})$
- It is not guaranteed to find a solution

WINNOWER

- Time complexity for $k = 3$ in $O(t^4 N^{2.66})$
- It is not guaranteed to find a solution
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l

WINNOWER

- Time complexity for $k = 3$ in $O(t^4 N^{2.66})$
- It is not guaranteed to find a solution
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l
- It does not consider composite motifs directly

WINNOWER

- Time complexity for $k = 3$ in $O(t^4 N^{2.66})$
- It is not guaranteed to find a solution
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l
- It does not consider composite motifs directly
- Assumes that a motif will not repeat in the same sequence

PRUNER

- Proposed to address some of the limitations of WINNOWER and other algorithms

PRUNER

- Proposed to address some of the limitations of WINNOWER and other algorithms
- PRUNER claims to explore only a small portion of the e-mismatch neighbourhood of an l-mer by taking into account pair-wise similarities

PRUNER

- Proposed to address some of the limitations of WINNOWER and other algorithms
- PRUNER claims to explore only a small portion of the e-mismatch neighbourhood of an l-mer by taking into account pair-wise similarities
- Builds a graph identical to the one built by WINNOWER

PRUNER

- Proposed to address some of the limitations of WINNOWER and other algorithms
- PRUNER claims to explore only a small portion of the e-mismatch neighbourhood of an l-mer by taking into account pair-wise similarities
- Builds a graph identical to the one built by WINNOWER
- Treats edges differently according to the degree of similarity between its connecting vertices

PRUNER

- Proposed to address some of the limitations of WINNOWER and other algorithms
- PRUNER claims to explore only a small portion of the e-mismatch neighbourhood of an l-mer by taking into account pair-wise similarities
- Builds a graph identical to the one built by WINNOWER
- Treats edges differently according to the degree of similarity between its connecting vertices
- The price to pay is incompleteness

PRUNER

- Uses the notion of set of *consistent patterns* of two l-mers m_1 and m_2, denoted $\rho(m_1, m_2)$

$$\rho(m_1, m_2) = \{m \in \Sigma^l : \delta(m, m_1) \leq e \wedge \delta(m, m_2) \leq e\}$$

PRUNER

- Uses the notion of set of *consistent patterns* of two l-mers m_1 and m_2, denoted $\rho(m_1, m_2)$

$$\rho(m_1, m_2) = \{m \in \Sigma^l : \delta(m, m_1) \leq e \wedge \delta(m, m_2) \leq e\}$$

- Note that, regardless of the value of e, if $\delta(m_1, m_2) > 2e$ then $\rho(m_1, m_2) = \emptyset$

PRUNER

Observe that

- For each l-mer, one needs only to explore its set of consistent patterns with respect to every other l-mer

PRUNER

Observe that

- For each l-mer, one needs only to explore its set of consistent patterns with respect to every other l-mer
- The number of l-mers which are at a distance no greater than d reduces rapidly with decreasing values of d

PRUNER

Observe that

- For each l-mer, one needs only to explore its set of consistent patterns with respect to every other l-mer
- The number of l-mers which are at a distance no greater than d reduces rapidly with decreasing values of d
- The size of the set of consistent patterns for two l-mers which mismatch in d positions decreases rapidly with increasing values of d

PRUNER

Observe that

- For each l-mer, one needs only to explore its set of consistent patterns with respect to every other l-mer
- The number of l-mers which are at a distance no greater than d reduces rapidly with decreasing values of d
- The size of the set of consistent patterns for two l-mers which mismatch in d positions decreases rapidly with increasing values of d

In particular,

- For any m_1,$m_2 \in \Sigma^l$ such that $e < \delta(m_1, m_2) \leq 2e$ we have $|\rho(m_1, m_2)| \in O(l^{\frac{e}{2}} |\Sigma|^{\frac{e}{2}})$ and its size is maximal for $\delta(m_1, m_2) = e + 1$

PRUNER

Observe that

- For each l-mer, one needs only to explore its set of consistent patterns with respect to every other l-mer
- The number of l-mers which are at a distance no greater than d reduces rapidly with decreasing values of d
- The size of the set of consistent patterns for two l-mers which mismatch in d positions decreases rapidly with increasing values of d

In particular,

- For any m_1,$m_2 \in \Sigma^l$ such that $e < \delta(m_1, m_2) \leq 2e$ we have $|\rho(m_1, m_2)| \in O(l^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ and its size is maximal for $\delta(m_1, m_2) = e + 1$
- Whereas, if $\delta(m_1, m_2) \leq e$, then $|\rho(m_1, m_2)| \in O(l^e|\Sigma|^e)$

PRUNER

The edges of the graph are then divided into two disjoint sets

PRUNER

The edges of the graph are then divided into two disjoint sets

Group 1

$$E_1 = \{(m_1, m_2) \in E : e < \delta(m_1, m_2) \leq 2e\}$$

PRUNER

The edges of the graph are then divided into two disjoint sets

Group 1

$$E_1 = \{(m_1, m_2) \in E : e < \delta(m_1, m_2) \leq 2e\}$$

Group 2

$$E_2 = \{(m_1, m_2) \in E : \delta(m_1, m_2) \leq e\}$$

PRUNER

- By only evaluating edges in E_1, PRUNER avoids the larger size of the set of consistent patterns of motifs connected by edges in E_2

PRUNER

- By only evaluating edges in E_1, PRUNER avoids the larger size of the set of consistent patterns of motifs connected by edges in E_2

- In many cases the algorithm can report or discard a pattern without looking into edges in E_2

PRUNER

- Let $\gamma : V \mapsto \mathbb{N}$ denote the partition degree of a vertex, i.e., the number of different partitions it is connected to

PRUNER

- Let $\gamma : V \mapsto \mathbb{N}$ denote the partition degree of a vertex, i.e., the number of different partitions it is connected to
- PRUNER starts by building a graph G identical to the one built by WINNOWER

PRUNER

- Let $\gamma : V \mapsto \mathbb{N}$ denote the partition degree of a vertex, i.e., the number of different partitions it is connected to
- PRUNER starts by building a graph G identical to the one built by WINNOWER
- It proceeds to delete all vertices m such that $\gamma(m) < q - 1$, just like WINNOWER for $k = 1$

PRUNER

- For each $m_i \in V$ computes $\rho(m_i, m_j)$ for every $(m_i, m_j) \in E_1$

PRUNER

- For each $m_i \in V$ computes $\rho(m_i, m_j)$ for every $(m_i, m_j) \in E_1$
- (m_i, m_j) is deleted and the computed patterns are added to a list $\eta(i)$

PRUNER

- For each $m_i \in V$ computes $\rho(m_i, m_j)$ for every $(m_i, m_j) \in E_1$
- (m_i, m_j) is deleted and the computed patterns are added to a list $\eta(i)$
- For each pattern we keep the information about the partition it came from

PRUNER

- For each $m_i \in V$ computes $\rho(m_i, m_j)$ for every $(m_i, m_j) \in E_1$
- (m_i, m_j) is deleted and the computed patterns are added to a list $\eta(i)$
- For each pattern we keep the information about the partition it came from
- After processing node m_i, $\eta(i)$ is sorted using radix sort

PRUNER

- For each $m_i \in V$ computes $\rho(m_i, m_j)$ for every $(m_i, m_j) \in E_1$
- (m_i, m_j) is deleted and the computed patterns are added to a list $\eta(i)$
- For each pattern we keep the information about the partition it came from
- After processing node m_i, $\eta(i)$ is sorted using radix sort
- $\eta(i)$ is scanned and for each pattern p a partition count $c(p)$ is computed

PRUNER

Let $R = \gamma(m_i)$ be the partition-degree of m_i after processing and removing all incident edges of E_1 and let p be a pattern under consideration.

PRUNER

Let $R = \gamma(m_i)$ be the partition-degree of m_i after processing and removing all incident edges of E_1 and let p be a pattern under consideration.

- If $c(p) + R < q - 1$ then p can be safely removed because the partition count can increase by at most R if we compare p with each m_j of the remaining edges $(m_i, m_j) \in E_2$

PRUNER

Let $R = \gamma(m_i)$ be the partition-degree of m_i after processing and removing all incident edges of E_1 and let p be a pattern under consideration.

- If $c(p) + R < q - 1$ then p can be safely removed because the partition count can increase by at most R if we compare p with each m_j of the remaining edges $(m_i, m_j) \in E_2$
- If $c(p) \geq q - 1$ then p is reported since it is clear it already occurs in $q - 1$ other partitions

PRUNER

Let $R = \gamma(m_i)$ be the partition-degree of m_i after processing and removing all incident edges of E_1 and let p be a pattern under consideration.

- If $c(p) + R < q - 1$ then p can be safely removed because the partition count can increase by at most R if we compare p with each m_j of the remaining edges $(m_i, m_j) \in E_2$
- If $c(p) \geq q - 1$ then p is reported since it is clear it already occurs in $q - 1$ other partitions
- If $q - 1 \leq c(p) + R < q - 1$ then we have to compare p with every other vertex which is still connected to m_i to check whether $\delta(p, m_j) \leq e$

PRUNER

Let $R = \gamma(m_i)$ be the partition-degree of m_i after processing and removing all incident edges of E_1 and let p be a pattern under consideration.

- If $c(p) + R < q - 1$ then p can be safely removed because the partition count can increase by at most R if we compare p with each m_j of the remaining edges $(m_i, m_j) \in E_2$
- If $c(p) \geq q - 1$ then p is reported since it is clear it already occurs in $q - 1$ other partitions
- If $q - 1 \leq c(p) + R < q - 1$ then we have to compare p with every other vertex which is still connected to m_i to check whether $\delta(p, m_j) \leq e$

  After computing the new partition count

  - If it is still less than $q - 1$, p is discarded
  - Otherwise, p is reported

PRUNER

After processing each vertex we are left with a graph containing only edges in E_2 which need to be processed. Then,

- All vertices m_i with $\gamma(m_i) < q - 1$ are removed

PRUNER

After processing each vertex we are left with a graph containing only edges in E_2 which need to be processed. Then,

- All vertices m_i with $\gamma(m_i) < q - 1$ are removed
- The remaining vertices, if any, are guaranteed to have a partition degree larger than q

PRUNER

After processing each vertex we are left with a graph containing only edges in E_2 which need to be processed. Then,

- All vertices m_i with $\gamma(m_i) < q - 1$ are removed
- The remaining vertices, if any, are guaranteed to have a partition degree larger than q
- The associated l-mers are, thus, valid patterns and are reported

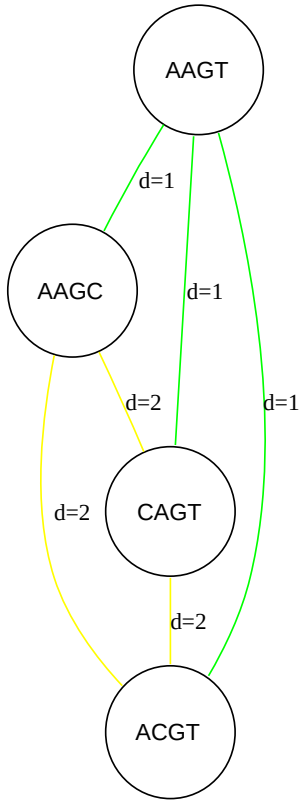
PRUNER

After processing each vertex we are left with a graph containing only edges in E_2 which need to be processed. Then,

- All vertices m_i with $\gamma(m_i) < q - 1$ are removed
- The remaining vertices, if any, are guaranteed to have a partition degree larger than q
- The associated l-mers are, thus, valid patterns and are reported
- However, consistent patterns for each pair of vertices are not computed at this stage of the algorithm which may cause some valid patterns not to be reported

PRUNER

Consider the following graph, where each vertex corresponds to a different partition
Edges in E_1 are in yellow and edges in E_2 are in green



$q = 4$

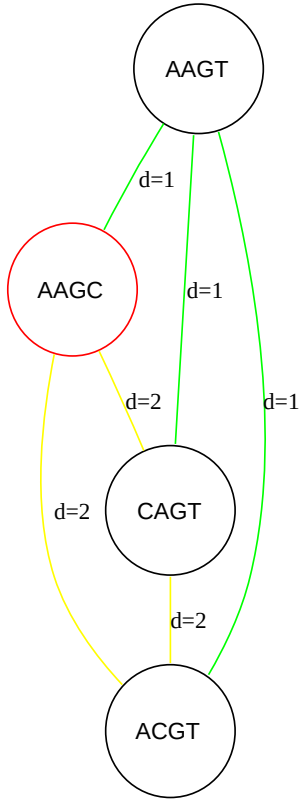
$e = 1$

For every edge $m_i \in E$

$\gamma(m_i) \geq q - 1 = 3$

No vertex is removed at this stage

PRUNER

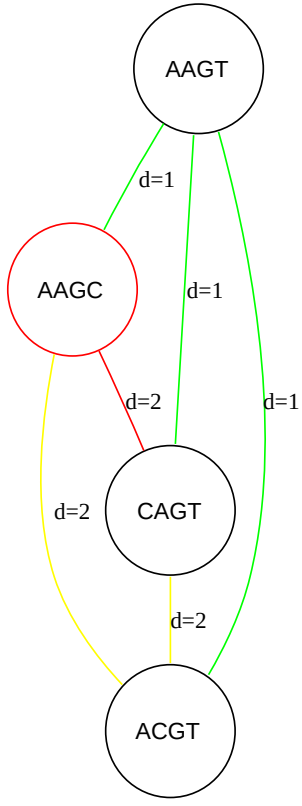


$q = 4$

$e = 1$

Consider vertex AAGC

PRUNER



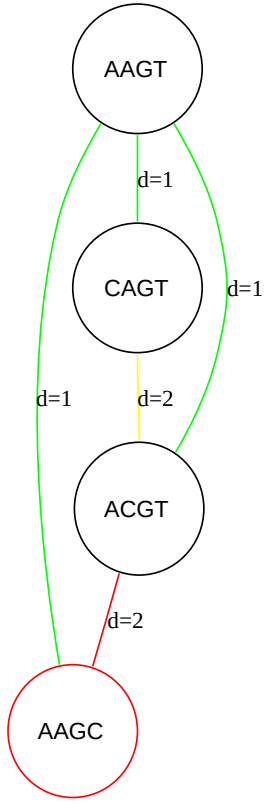
$$q = 4$$

$$e = 1$$

$$\rho(\mathrm{AAGC}, \mathrm{CAGT}) = \{\mathrm{CAGC}, \mathrm{AAGT}\}$$

$$\eta(\mathrm{AAGC}) = \{\mathrm{CAGC}, \mathrm{AAGT}\}$$

PRUNER



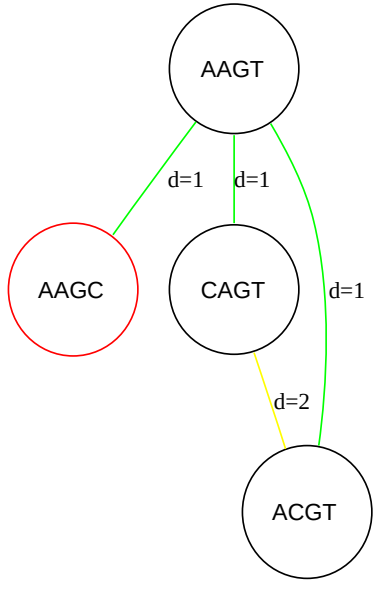
$$q = 4$$

$$e = 1$$

$$\rho(\mathrm{AAGC}, \mathrm{ACGT}) = \{\mathrm{AAGT}, \mathrm{ACGC}\}$$

$$\eta(\mathrm{AAGC}) = \{\mathrm{CAGC}, \mathrm{AAGT}, \mathrm{AAGT}, \mathrm{ACGC}\}$$

PRUNER



$q = 4$

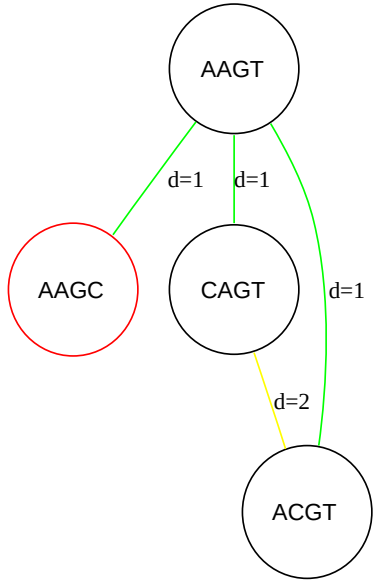
$e = 1$

Finished processing vertex AAGC

Sorting and scanning $\eta(\mathrm{AAGC})$

$\eta(\mathrm{AAGC}) = \{\mathrm{CAGC}, \mathrm{AAGT}, \mathrm{AAGT}, \mathrm{ACGC}\}$

PRUNER



$q = 4$

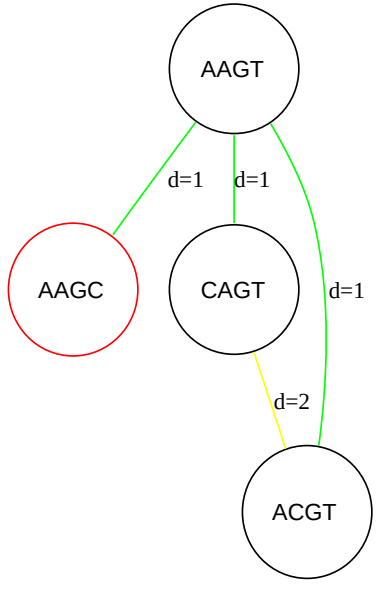
$e = 1$

Finished processing vertex AAGC

Sorting and scanning $\eta(\mathrm{AAGC})$

$\eta(\mathrm{AAGC}) = \{\mathrm{ACGC}, \mathrm{AAGT}, \mathrm{AAGT}, \mathrm{CAGC}\}$

PRUNER



$q = 4$

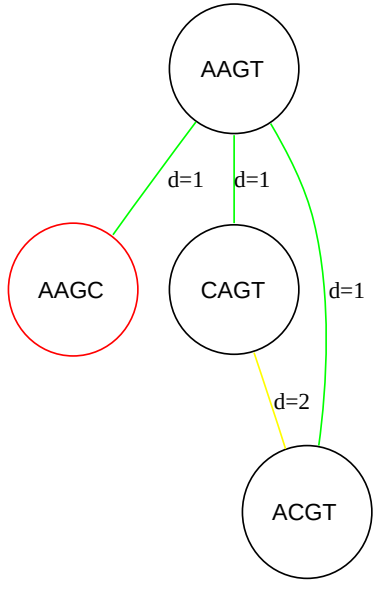
$e = 1$

Finished processing vertex AAGC

Sorting and scanning $\eta(\mathrm{AAGC})$

$\eta(\mathrm{AAGC}) = \{\overline{\mathrm{ACGC}}^{\sharp 1}, \overline{\mathrm{AAGT}, \mathrm{AAGT}}^{\sharp 2}, \overline{\mathrm{CAGC}}^{\sharp 1}\}$

PRUNER



$q = 4$

$e = 1$

$R = \gamma(\mathrm{AAGC}) = 1$

$\eta(\mathrm{AAGC}) = \{\overline{\mathrm{ACGC}}^{\sharp 1}, \overline{\mathrm{AAGT}, \mathrm{AAGT}}^{\sharp 2}, \overline{\mathrm{CAGC}}^{\sharp 1}\}$

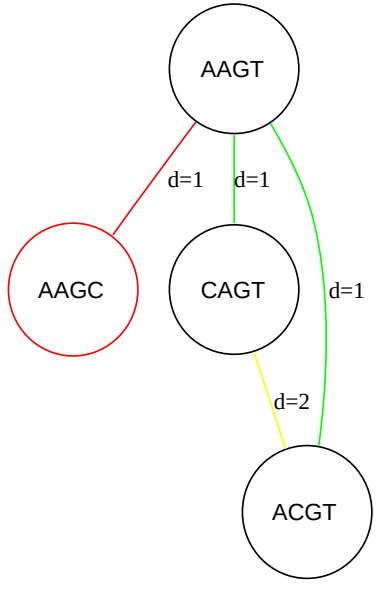
$c(\mathrm{ACGC}) + R < q - 1 \Leftrightarrow 1 + 1 < 3$ DISCARD

$c(\mathrm{CAGC}) + R < q - 1 \Leftrightarrow 1 + 1 < 3$ DISCARD

$q - 1 \leq c(\mathrm{AAGT}) + R < q - 1 + R \Leftrightarrow$

$\Leftrightarrow 3 \leq 3 < 4$ CHECK E_2 edges

PRUNER



$q = 4$

$e = 1$

$\delta(\mathrm{AAGT}, \mathrm{AAGT}) = 0 < e$

The new partition count is increased

$c(\mathrm{AAGT}) = 3 \geq q - 1$ REPORT

PRUNER would now process the remaining vertices . . .

PRUNER

- PRUNER takes $O(N^2 t^2 l^{\frac{e}{2}})$ time and $O(Ntl^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ space to operate

PRUNER

- PRUNER takes $O(N^2t^2l^{\frac{e}{2}})$ time and $O(Ntl^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ space to operate
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l

PRUNER

- PRUNER takes $O(N^2t^2l^{\frac{e}{2}})$ time and $O(Ntl^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ space to operate
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l
- It does not consider composite motifs directly

PRUNER

- PRUNER takes $O(N^2t^2l^{\frac{e}{2}})$ time and $O(Ntl^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ space to operate
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l
- It does not consider composite motifs directly
- Assumes that a motif will not repeat in the same sequence

PRUNER

- PRUNER takes $O(N^2t^2l^{\frac{e}{2}})$ time and $O(Ntl^{\frac{e}{2}}|\Sigma|^{\frac{e}{2}})$ space to operate
- To consider a range of lengths for motifs we have to re-run the algorithm for each value of l
- It does not consider composite motifs directly
- Assumes that a motif will not repeat in the same sequence
- It is incomplete

SMILE

- SMILE is a combinatorial algorithm which relies on a generalized suffix-tree

SMILE

- SMILE is a combinatorial algorithm which relies on a generalized suffix-tree
- A generalized suffix-tree $\mathcal{T}$ is a representation of all the suffixes of a set of sequences $\mathcal{S}$

SMILE

- SMILE is a combinatorial algorithm which relies on a generalized suffix-tree
- A generalized suffix-tree $\mathcal{T}$ is a representation of all the suffixes of a set of sequences $\mathcal{S}$
- $\mathcal{T}$ has multiple termination symbols (one for each sequence)

SMILE

- SMILE is a combinatorial algorithm which relies on a generalized suffix-tree
- A generalized suffix-tree $\mathcal{T}$ is a representation of all the suffixes of a set of sequences $\mathcal{S}$
- $\mathcal{T}$ has multiple termination symbols (one for each sequence)
- At each internal node v we keep a bit vector of size t ($colors_v$) indicating the sequences in $\mathcal{S}$ in which the path-label of v occurs

SMILE

- SMILE is a combinatorial algorithm which relies on a generalized suffix-tree
- A generalized suffix-tree $\mathcal{T}$ is a representation of all the suffixes of a set of sequences $\mathcal{S}$
- $\mathcal{T}$ has multiple termination symbols (one for each sequence)
- At each internal node v we keep a bit vector of size t ($colors_v$) indicating the sequences in $\mathcal{S}$ in which the path-label of v occurs
- To facilitate the discussion we will think of $\mathcal{T}$ as if it were a trie

SMILE

- The algorithm operates by traversing $\mathcal{T}$

SMILE

- The algorithm operates by traversing $\mathcal{T}$
- The traversal is guided by a virtual lexicographic trie $\mathcal{M}$ which represents all motifs $m \in \Sigma^l$

SMILE

- The algorithm operates by traversing $\mathcal{T}$
- The traversal is guided by a virtual lexicographic trie $\mathcal{M}$ which represents all motifs $m \in \Sigma^l$
- As we go down on $\mathcal{M}$ we follow all valid paths in $\mathcal{T}$, i.e. all paths for which there is still hope of finding a path-label of length l with at most e mismatches from the motif being spelt by the traversal of $\mathcal{M}$

SMILE

- The algorithm operates by traversing $\mathcal{T}$
- The traversal is guided by a virtual lexicographic trie $\mathcal{M}$ which represents all motifs $m \in \Sigma^l$
- As we go down on $\mathcal{M}$ we follow all valid paths in $\mathcal{T}$, i.e. all paths for which there is still hope of finding a path-label of length l with at most e mismatches from the motif being spelt by the traversal of $\mathcal{M}$
- The traversal stops if there are no more valid paths or if the paths do not represent occurrences in at least $q \leq t$ sequences, in which case, the sub-trie below the current node in $\mathcal{M}$ is pruned

SMILE

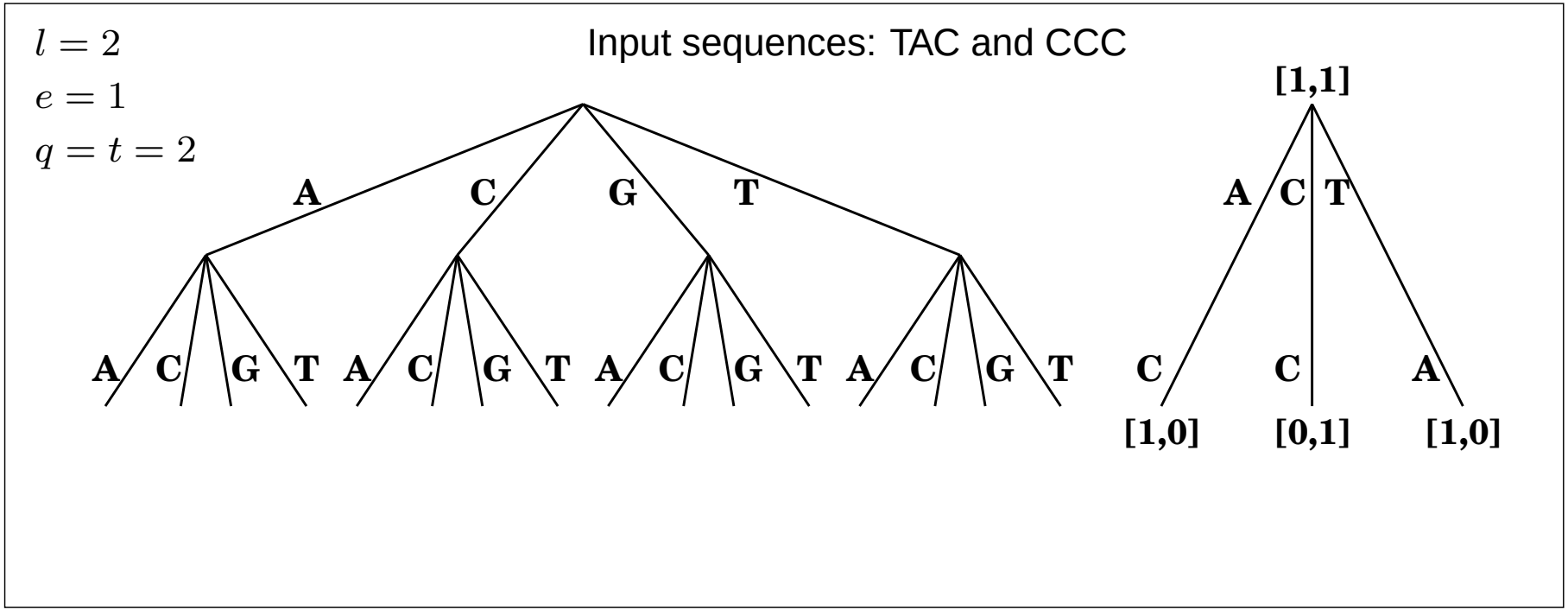
- If we are able to reach a node v in $\mathcal{M}$ at depth l and still have valid paths in $\mathcal{T}$ we report the path-label of v as a valid pattern

SMILE

- If we are able to reach a node v in $\mathcal{M}$ at depth l and still have valid paths in $\mathcal{T}$ we report the path-label of v as a valid pattern
- Note that the algorithm can easily not only extract motifs of length l but also motifs within any range of lengths $l_{\min}, \ldots, l_{\max}$
  - Continue the traversal of $\mathcal{M}$ as far as depth $l_{\max}$
  - Report any pattern with at least $l_{\min}$ characters that respects the extraction requirements

SMILE

Extraction of Simple Motifs



SMILE

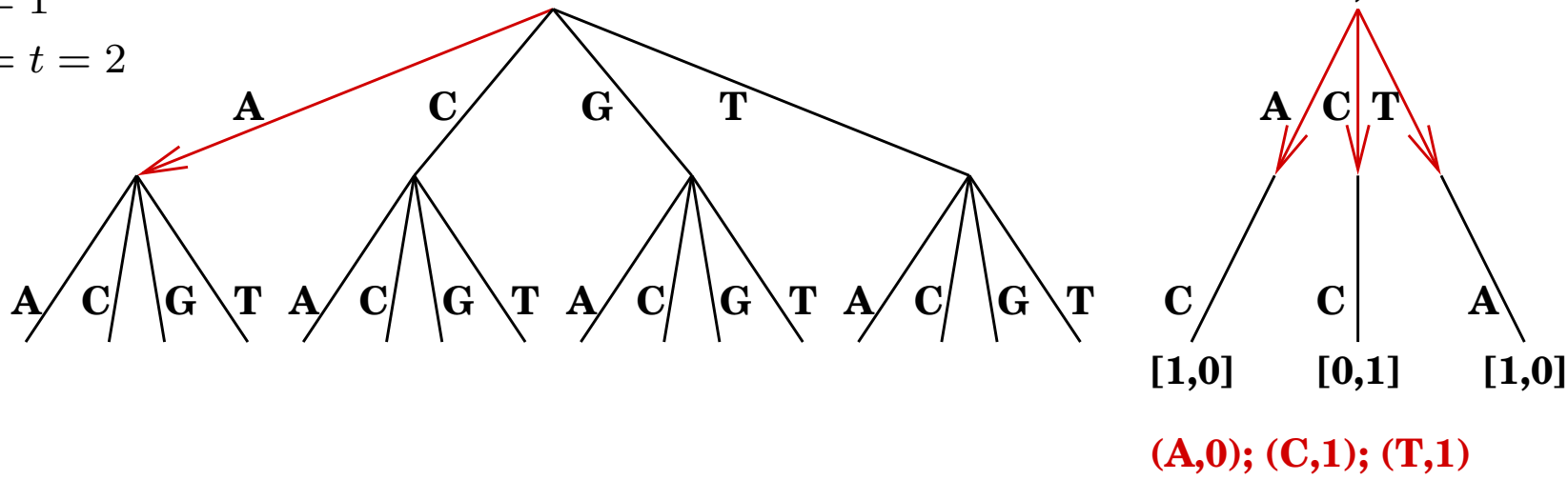
Extraction of Simple Motifs

$l = 2$

$e = 1$

$q = t = 2$

Input sequences: TAC and CCC



SMILE

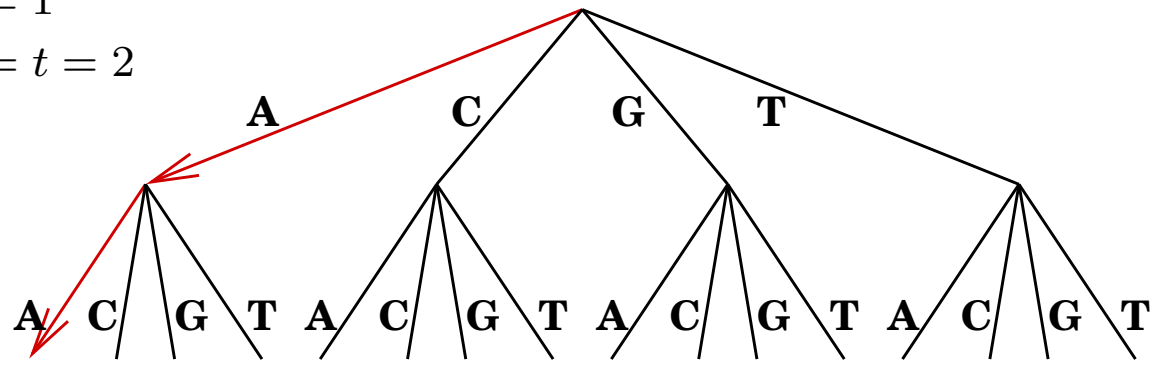
Extraction of Simple Motifs

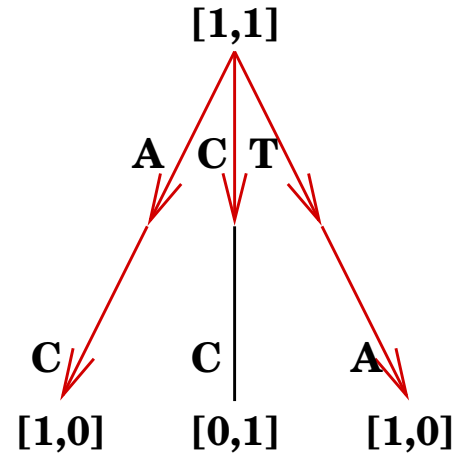
$l = 2$

$e = 1$

$q = t = 2$

Input sequences: TAC and CCC





(A,0); (C,1); (T,1)

(AC,1); (TA,1)

SMILE

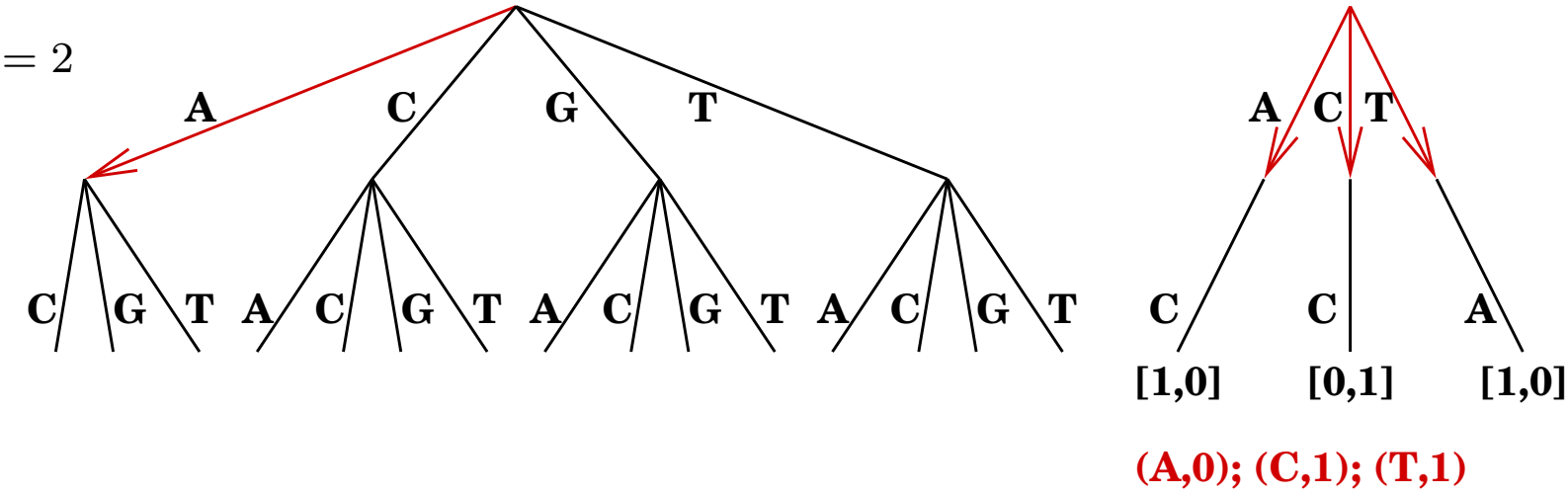
Extraction of Simple Motifs

$l = 2$

$e = 1$

$q = t = 2$

Input sequences: TAC and CCC



SMILE

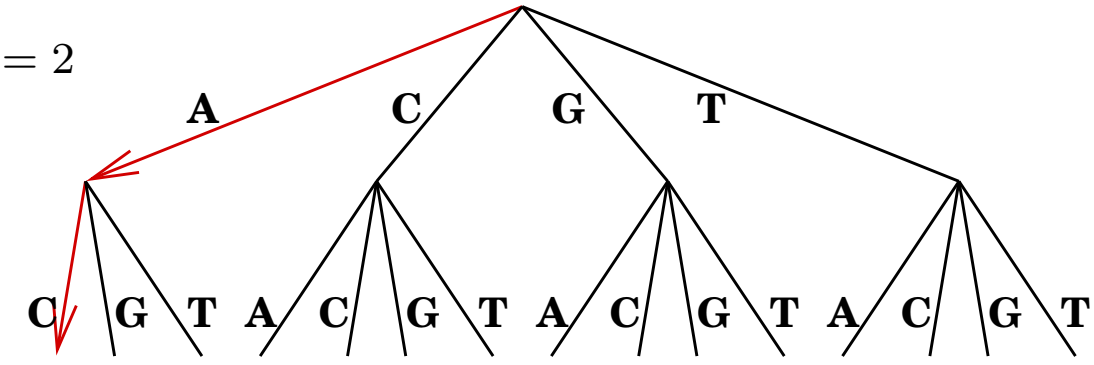
Extraction of Simple Motifs

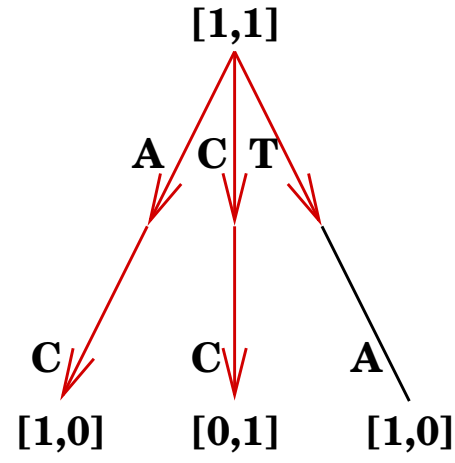
$l = 2$

$e = 1$

$q = t = 2$

Input sequences: TAC and CCC



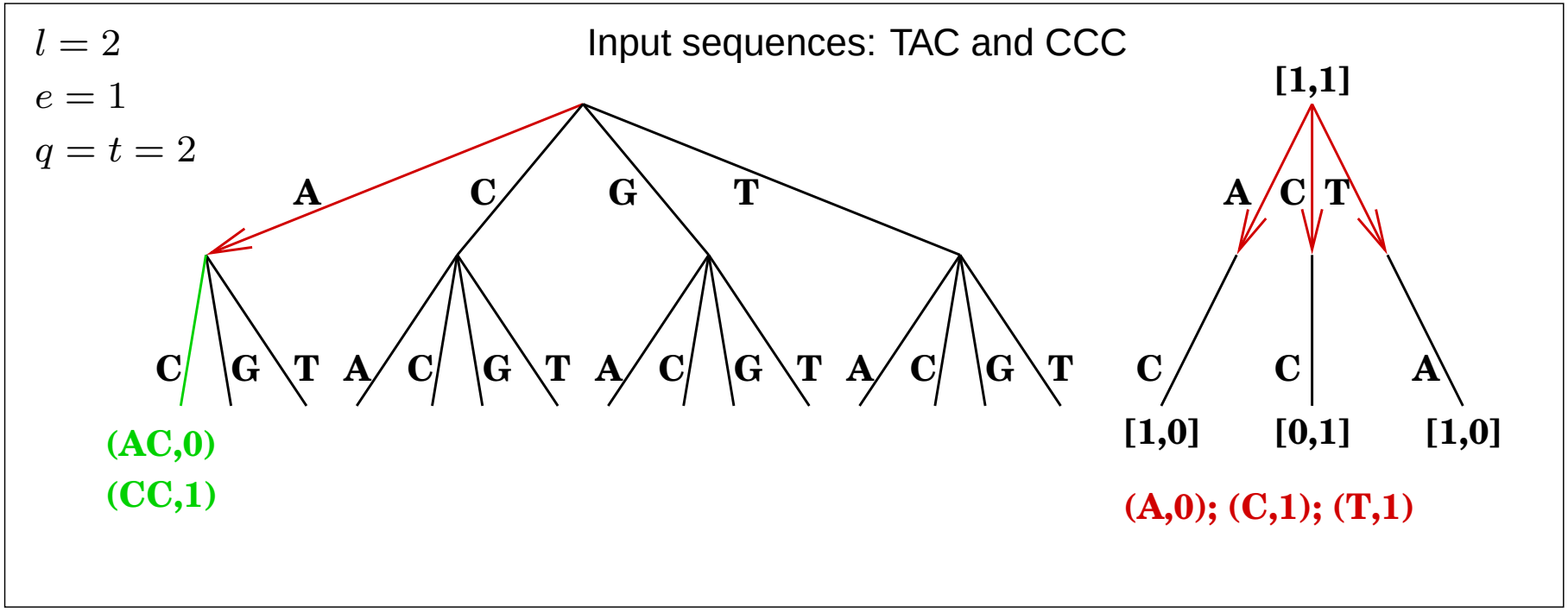


(A,0); (C,1); (T,1)

(AC,0); (CC,1)

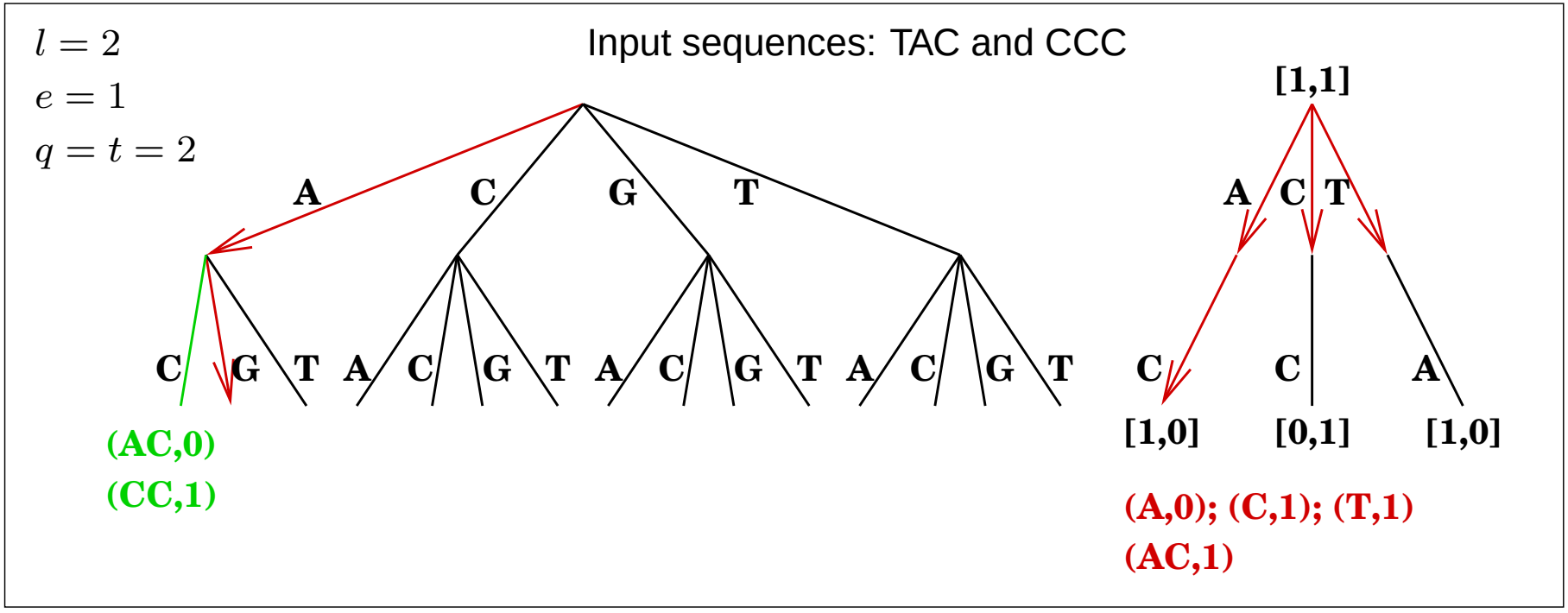
SMILE

Extraction of Simple Motifs



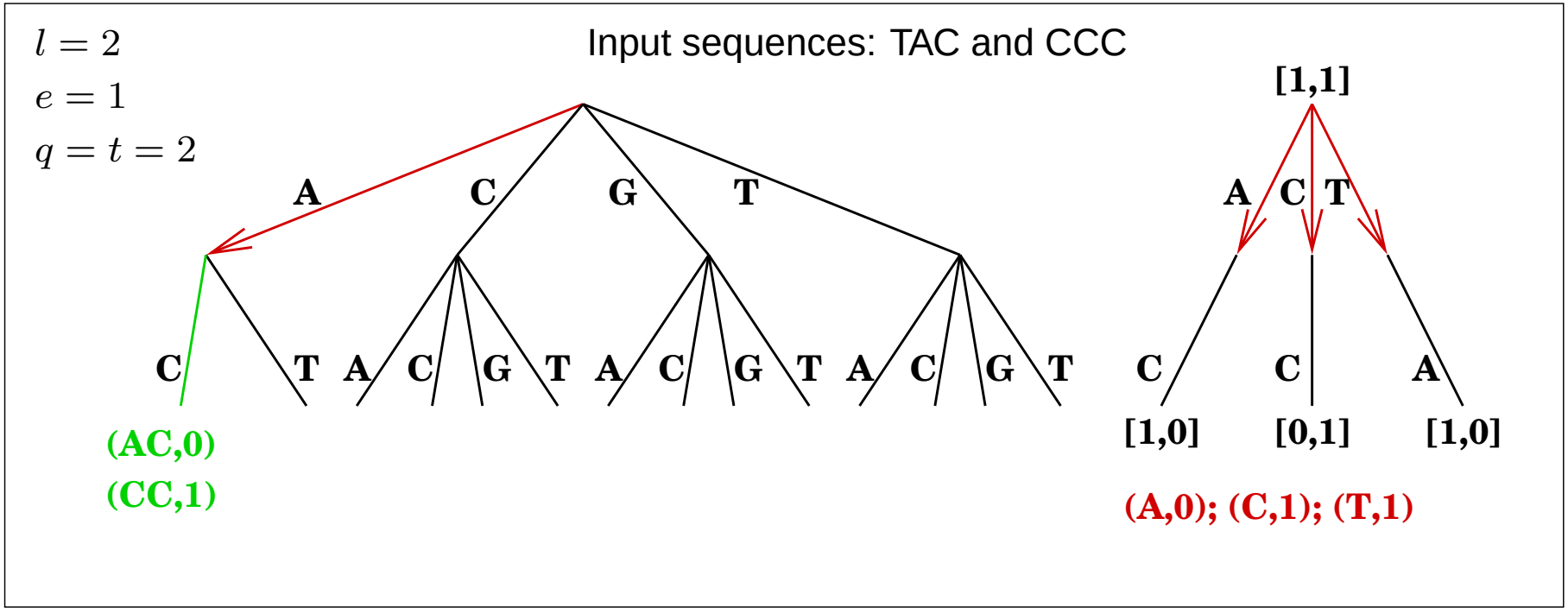
SMILE

Extraction of Simple Motifs



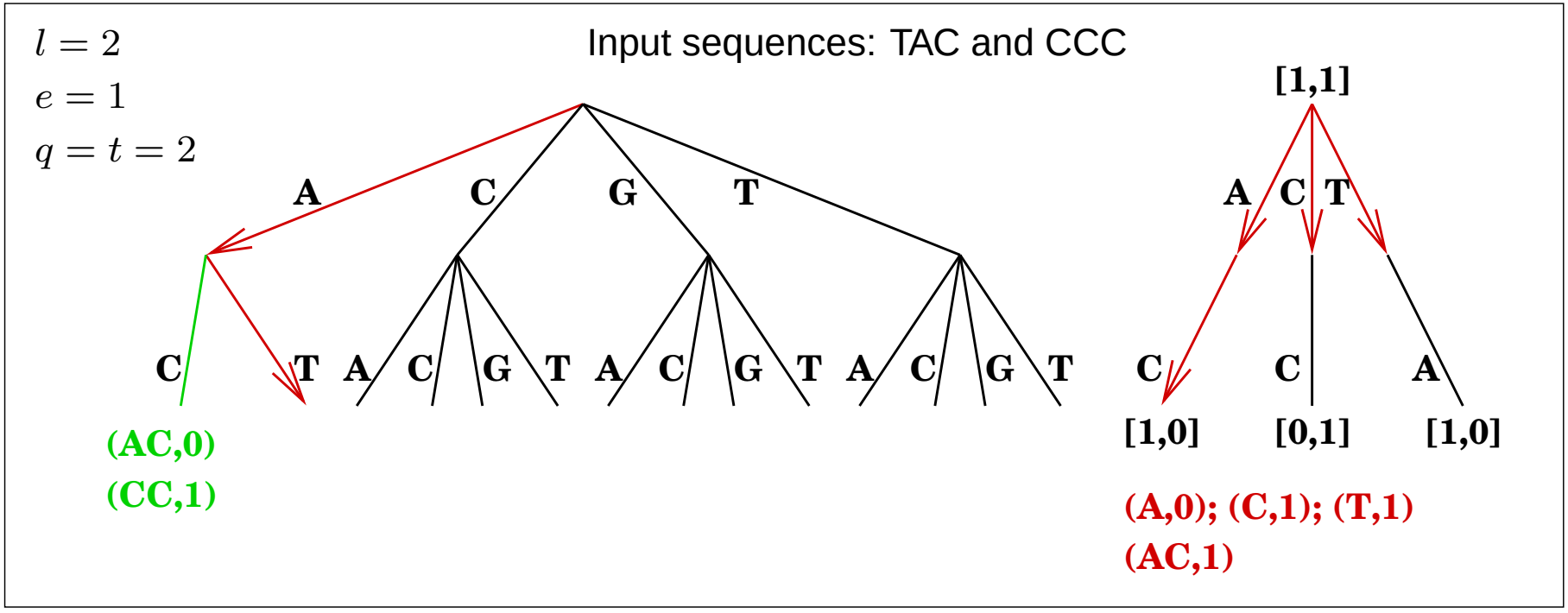
SMILE

Extraction of Simple Motifs



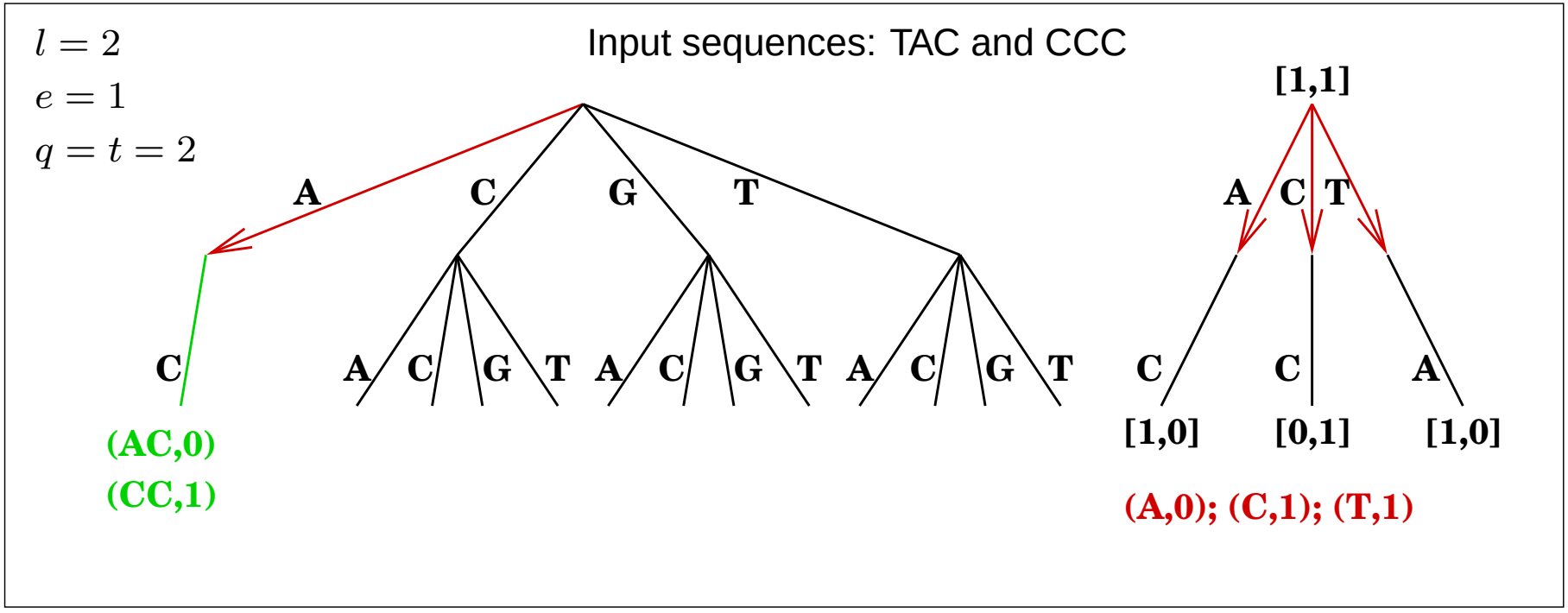
SMILE

Extraction of Simple Motifs



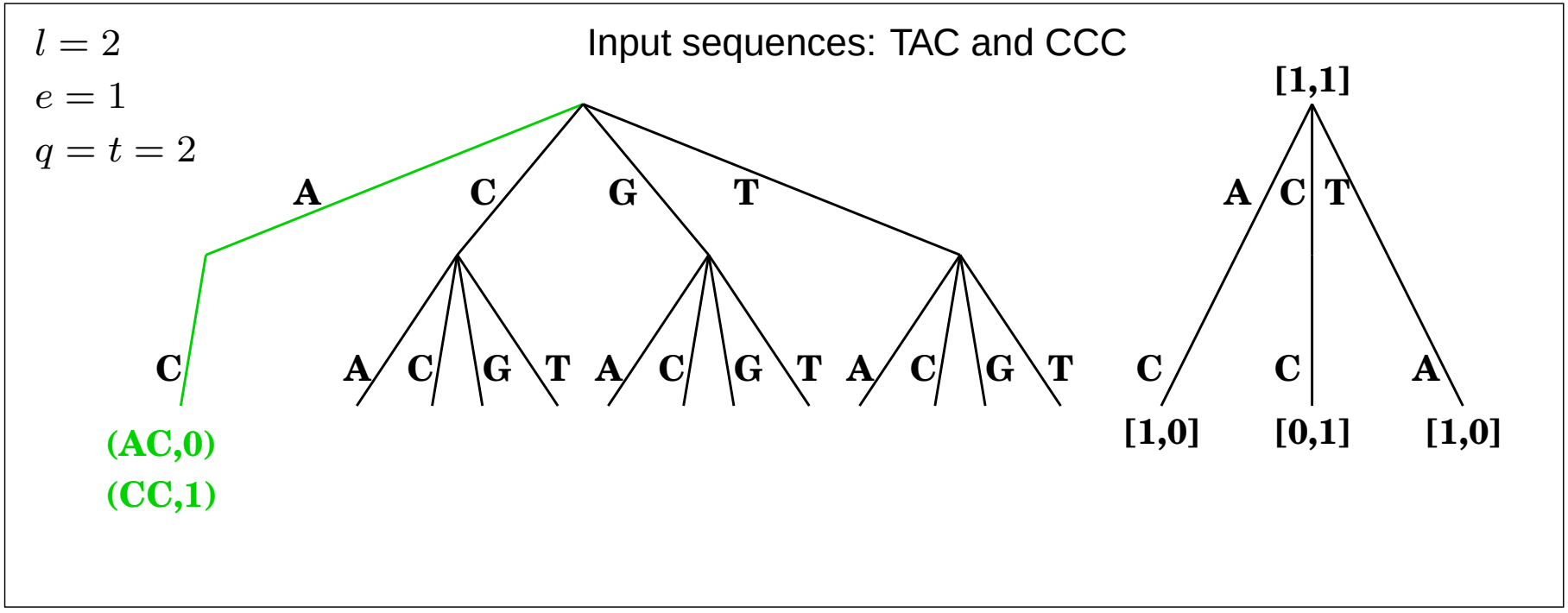
SMILE

Extraction of Simple Motifs



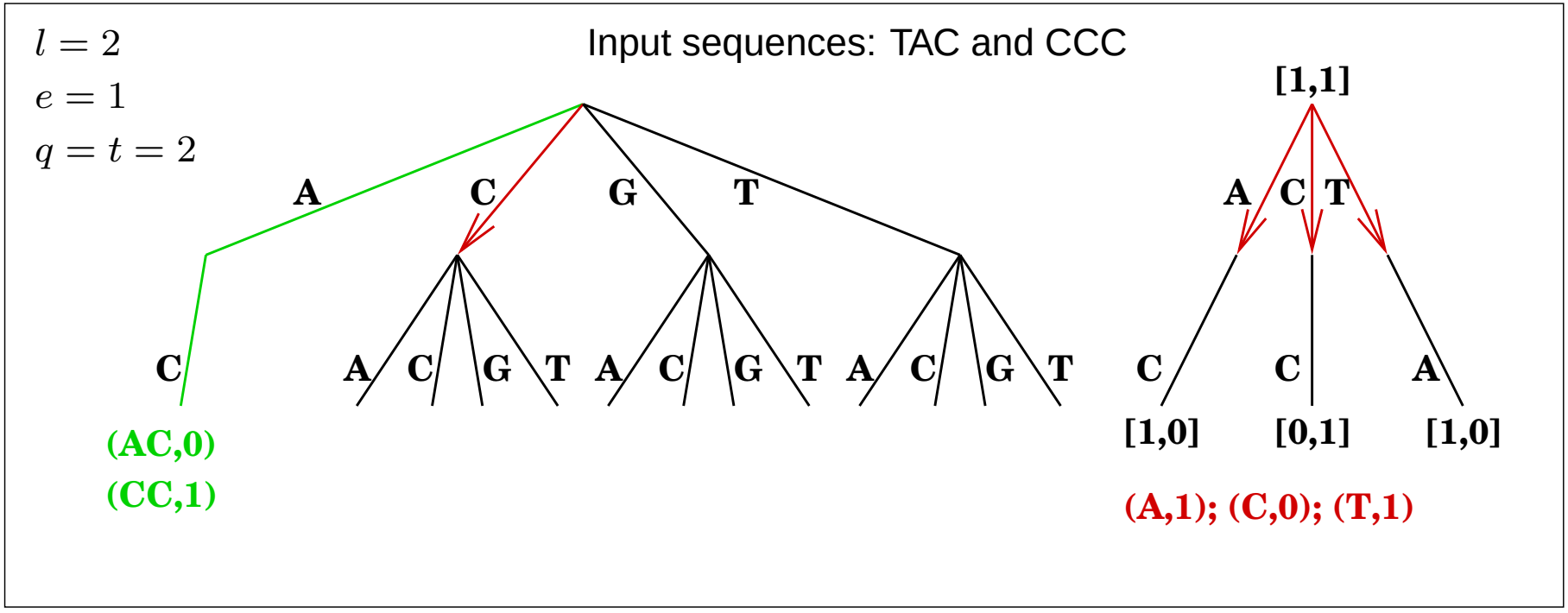
SMILE

Extraction of Simple Motifs



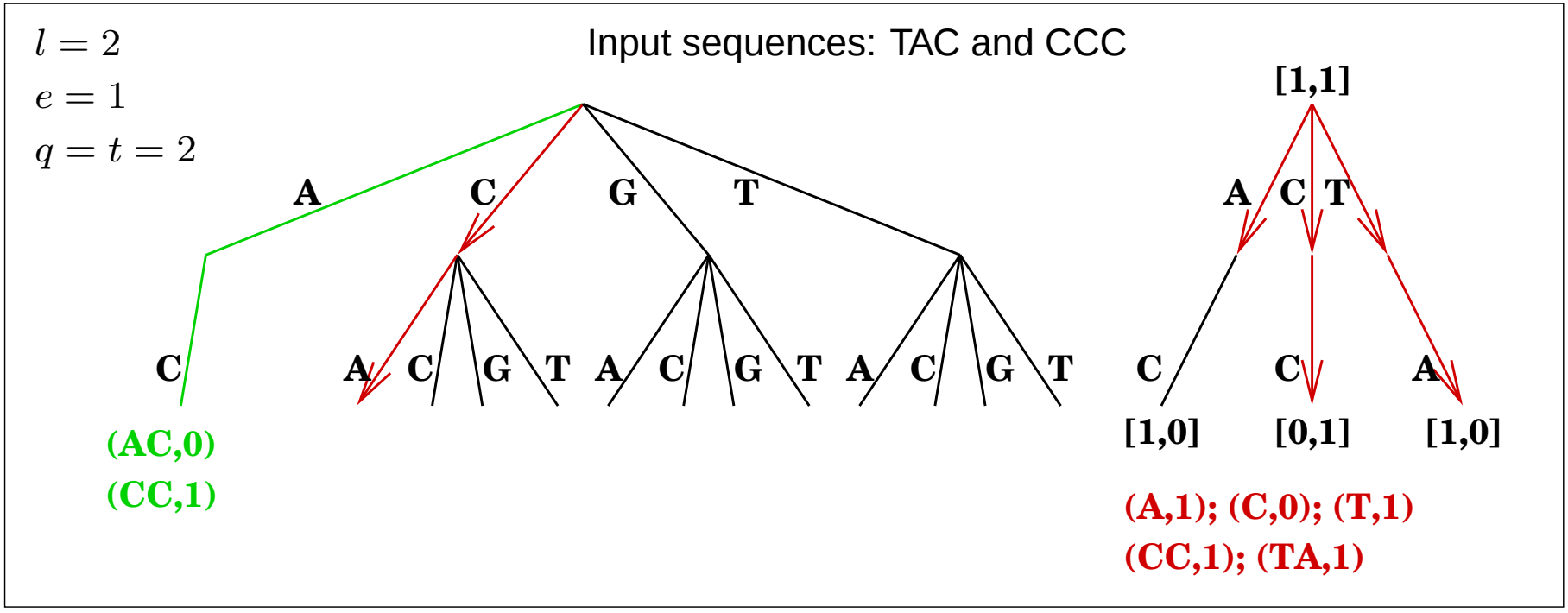
SMILE

Extraction of Simple Motifs



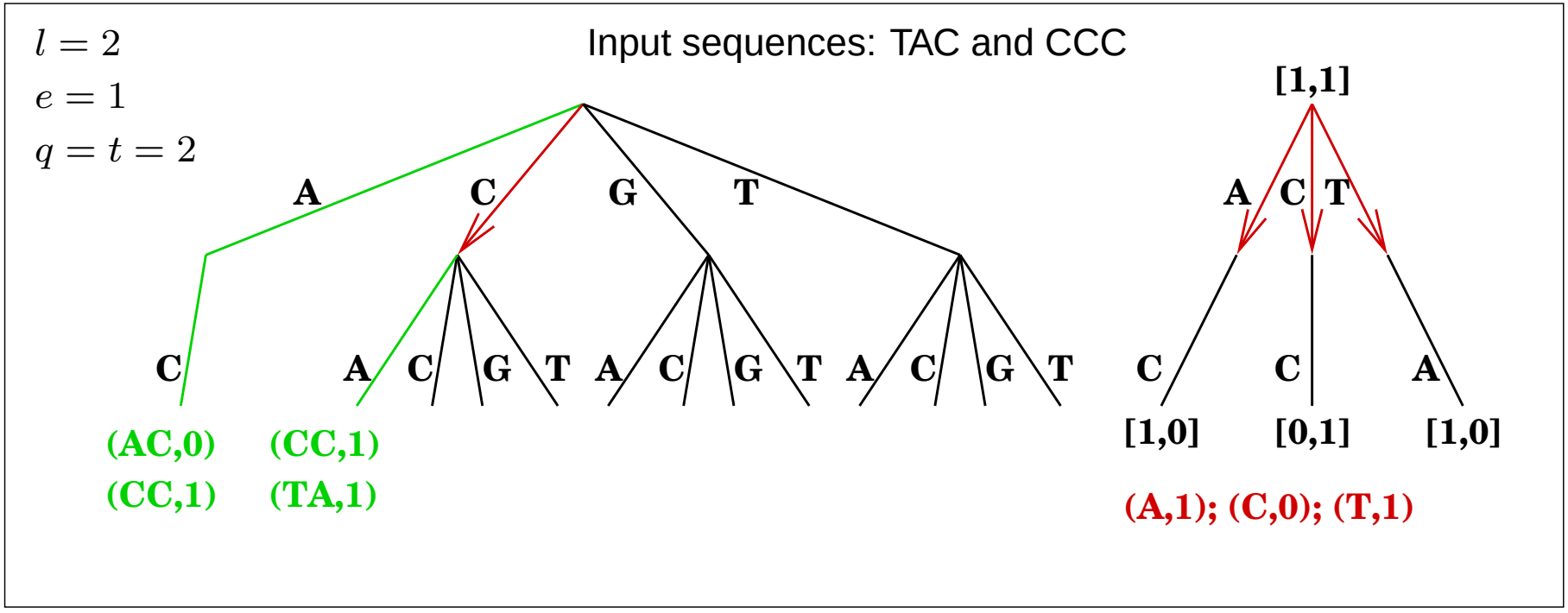
SMILE

Extraction of Simple Motifs



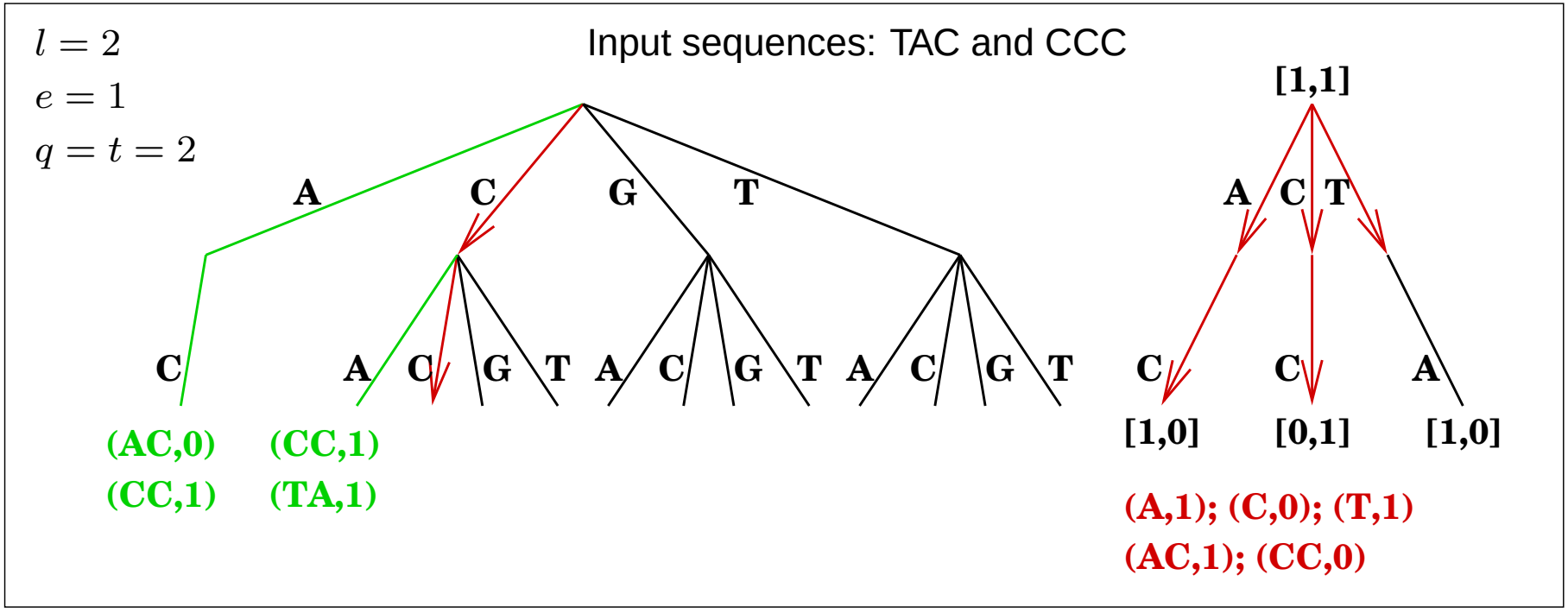
SMILE

Extraction of Simple Motifs



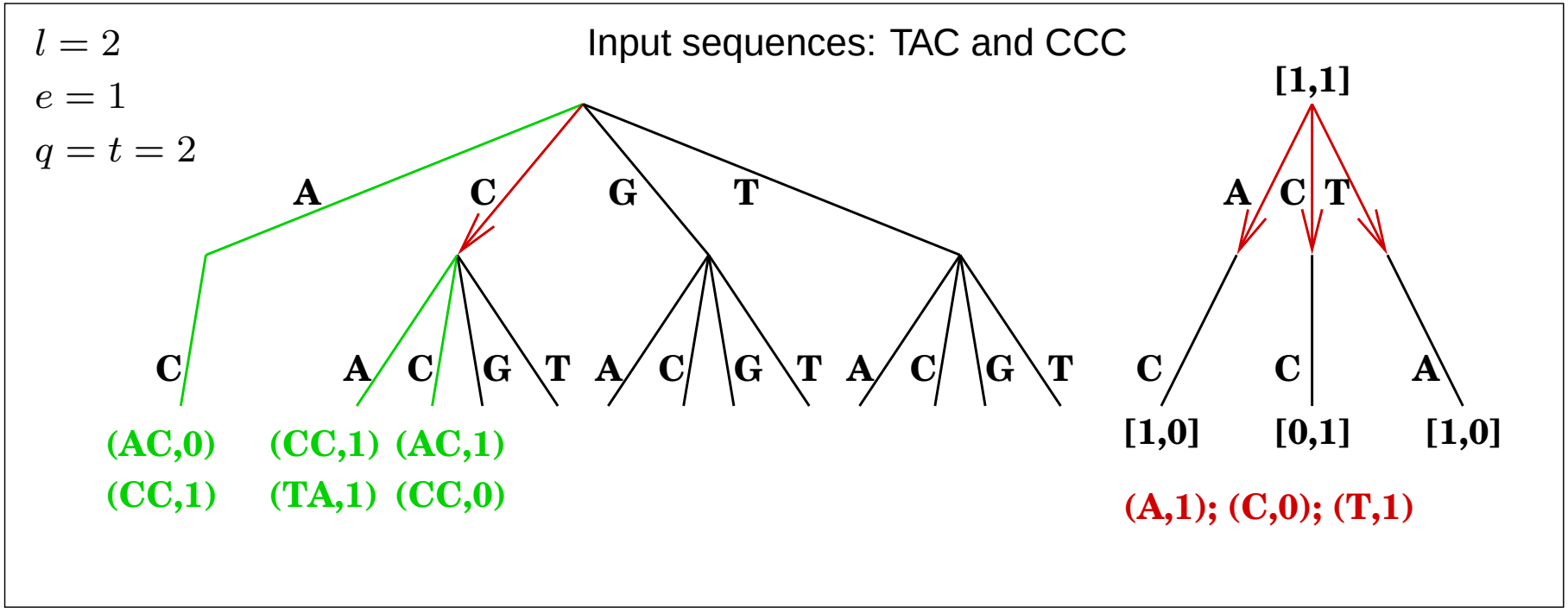
SMILE

Extraction of Simple Motifs



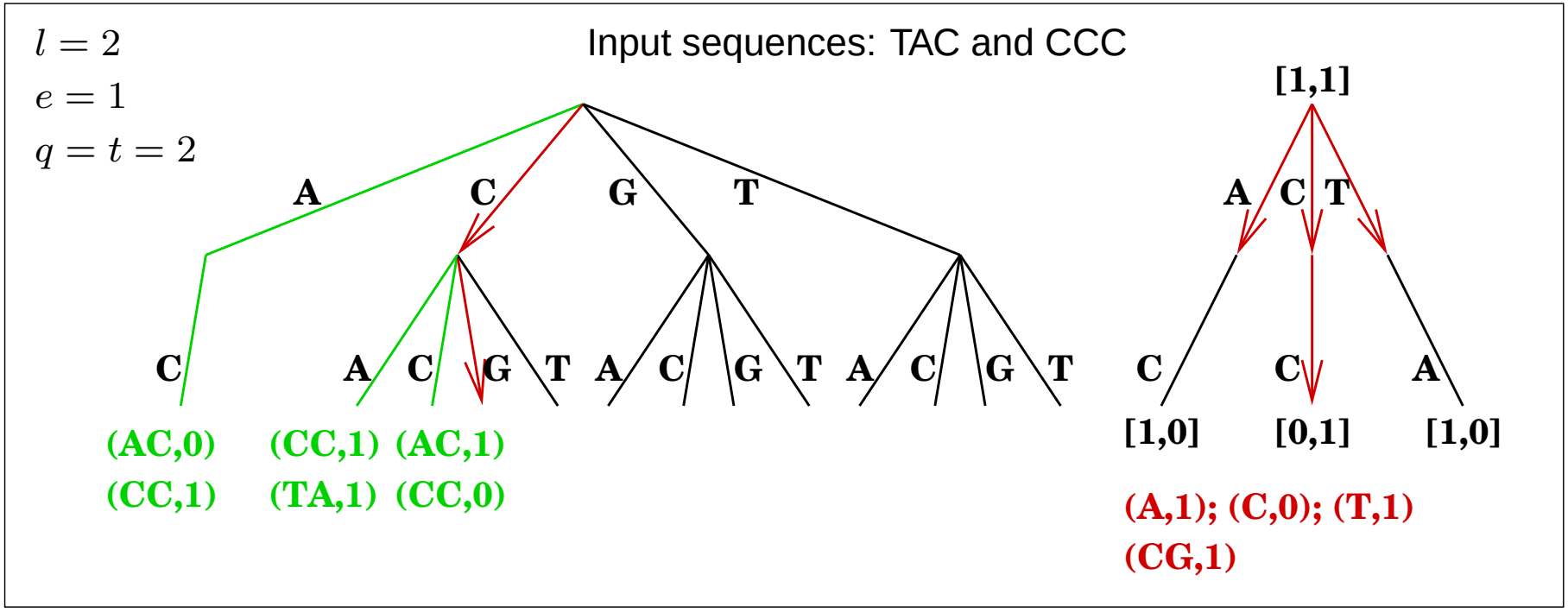
SMILE

Extraction of Simple Motifs



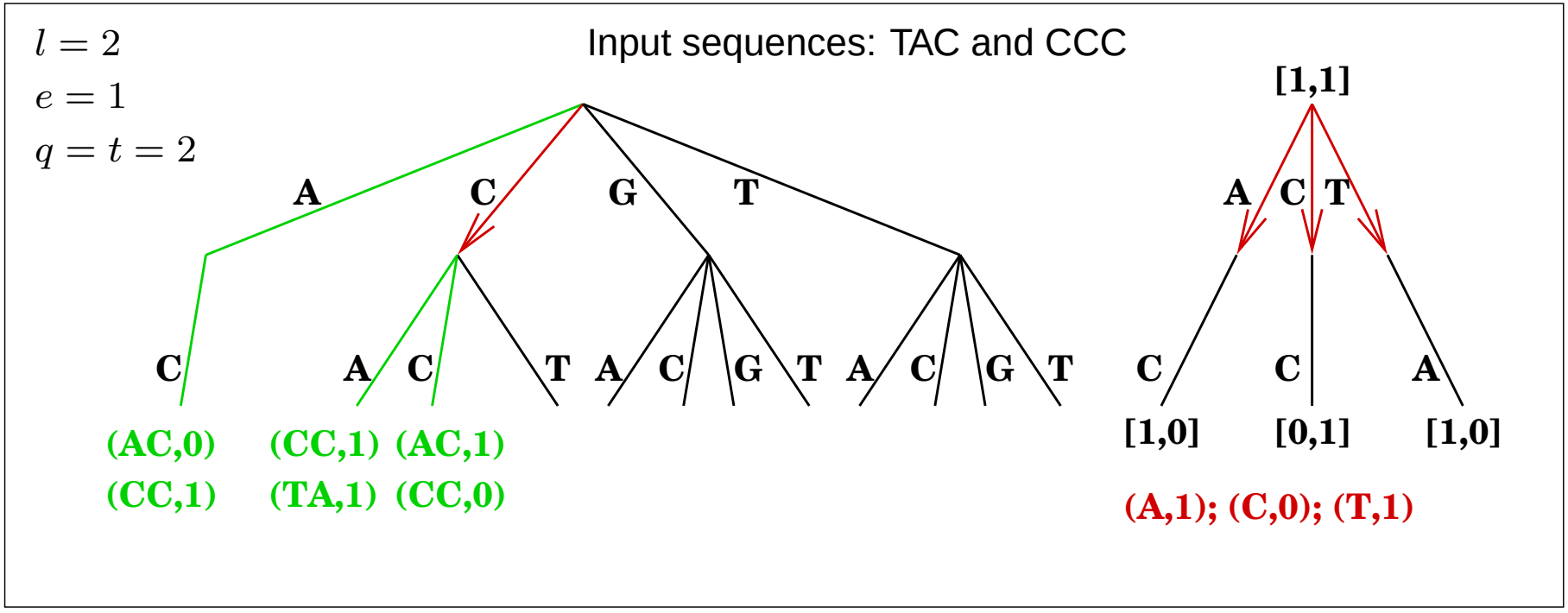
SMILE

Extraction of Simple Motifs



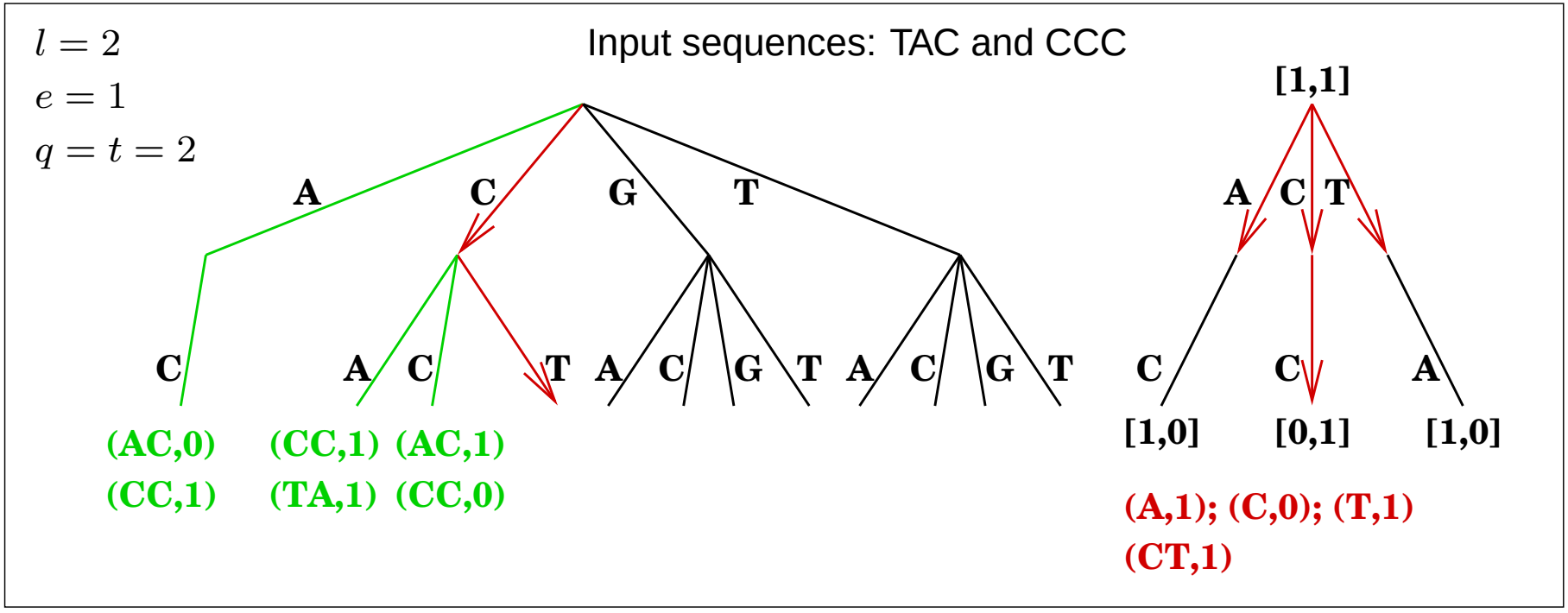
SMILE

Extraction of Simple Motifs



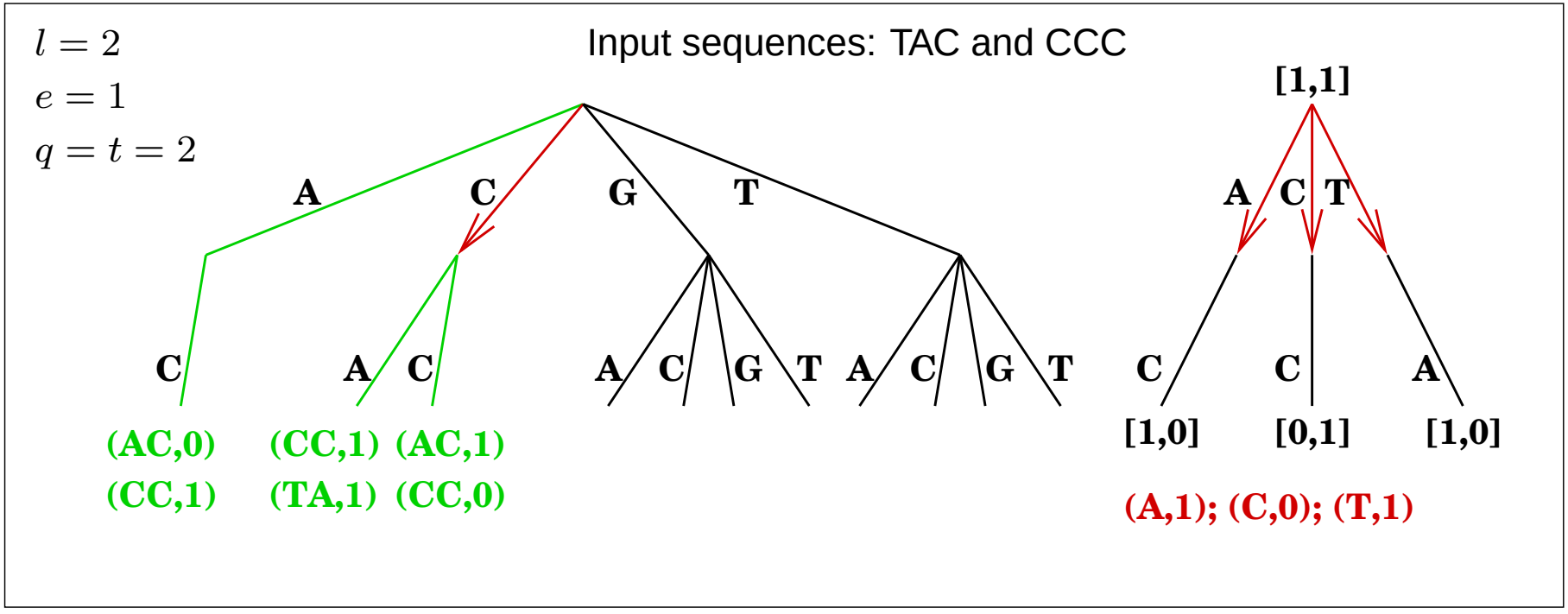
SMILE

Extraction of Simple Motifs



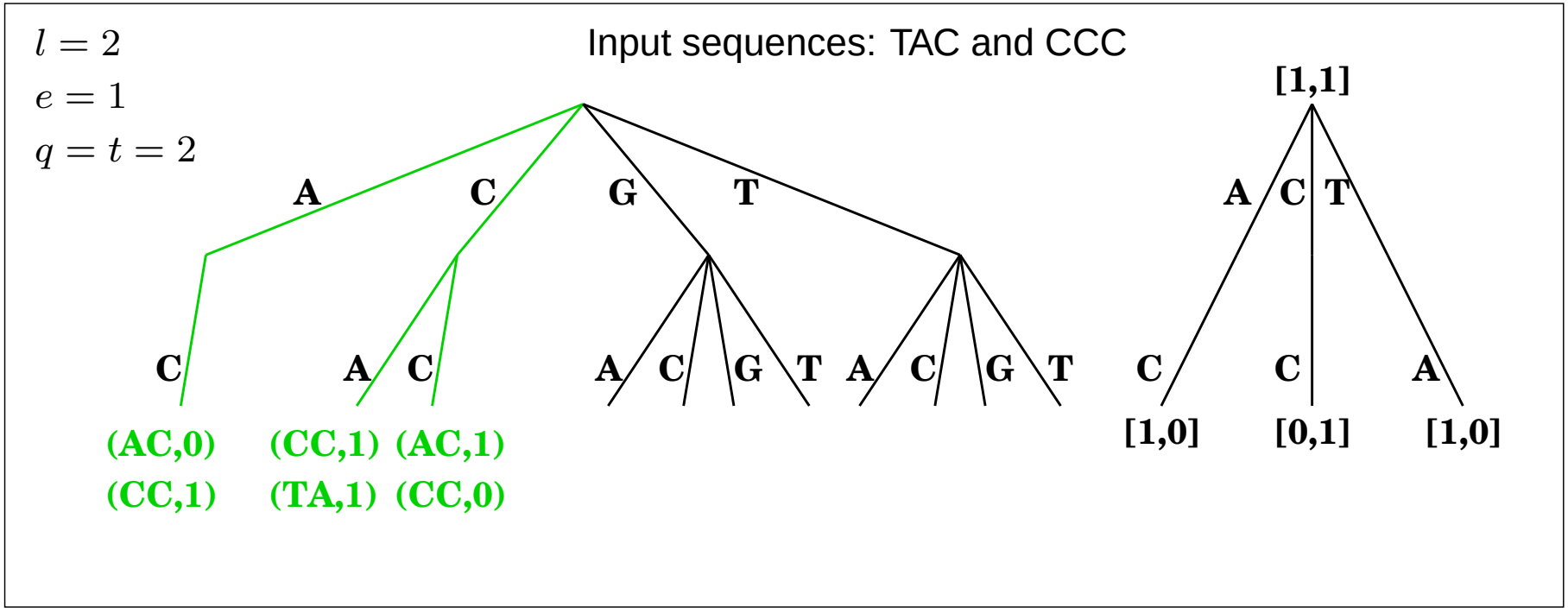
SMILE

Extraction of Simple Motifs



SMILE

Extraction of Simple Motifs



SMILE

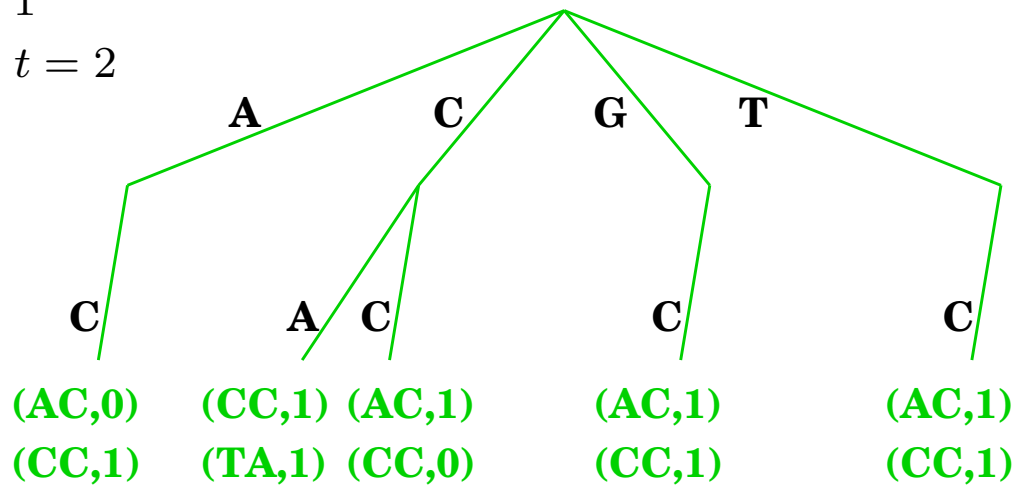
Extraction of Simple Motifs

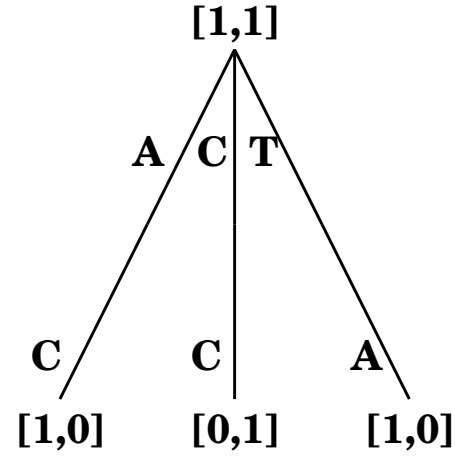
$l = 2$

$e = 1$

$q = t = 2$

Input sequences: TAC and CCC





SMILE

- Unlike the previous algorithms SMILE handles composite motifs directly

SMILE

- Unlike the previous algorithms SMILE handles composite motifs directly
- Instead of pre-processing the input sequences it will simply jump down in the suffix-tree to search for the next component

SMILE

- Unlike the previous algorithms SMILE handles composite motifs directly
- Instead of pre-processing the input sequences it will simply jump down in the suffix-tree to search for the next component
- We can consider any number of components

SMILE

- Unlike the previous algorithms SMILE handles composite motifs directly
- Instead of pre-processing the input sequences it will simply jump down in the suffix-tree to search for the next component
- We can consider any number of components
- The component motifs can be separated by $d \in \{d_{\min}, \ldots, d_{\max}\}$ characters

SMILE

- Unlike the previous algorithms SMILE handles composite motifs directly
- Instead of pre-processing the input sequences it will simply jump down in the suffix-tree to search for the next component
- We can consider any number of components
- The component motifs can be separated by $d \in \{d_{\min}, \ldots, d_{\max}\}$ characters
- We can establish a global maximum number of allowed mismatches alongside the permitted errors for each component

SMILE

Extraction of composite motifs

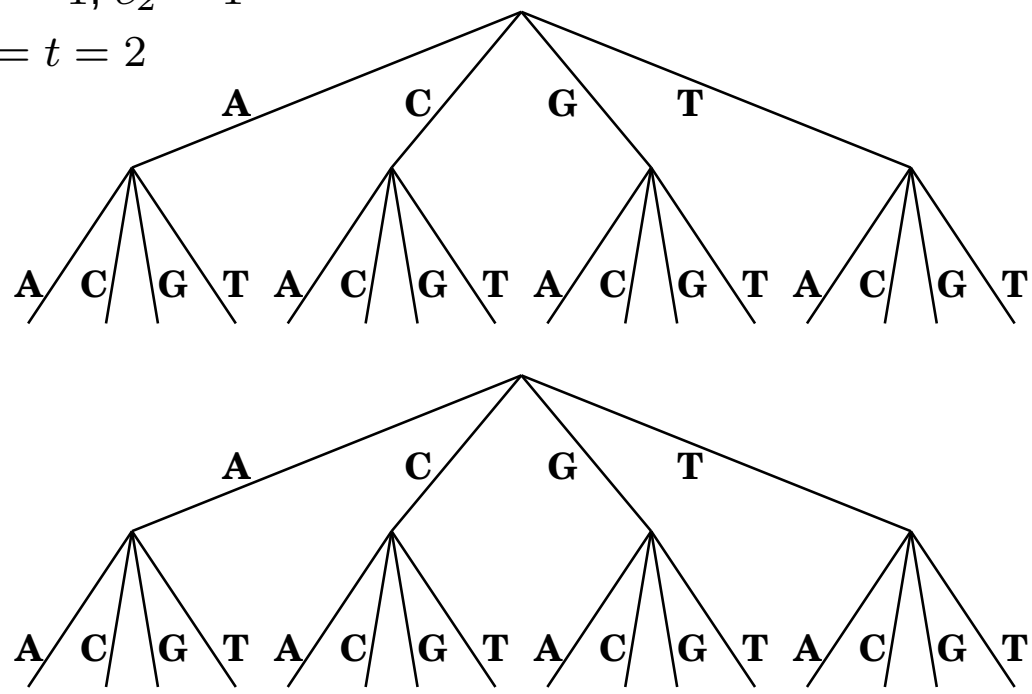
$p = 2$

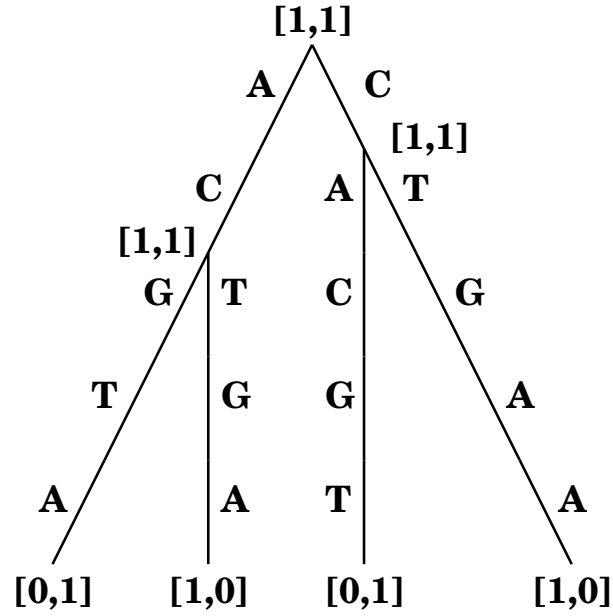
$l_1 = 2, d = 1, l_2 = 2$

$e_1 = 1, e_2 = 1$

$q = t = 2$

Input sequences: ACTGAA and CACGTA





SMILE

Extraction of composite motifs

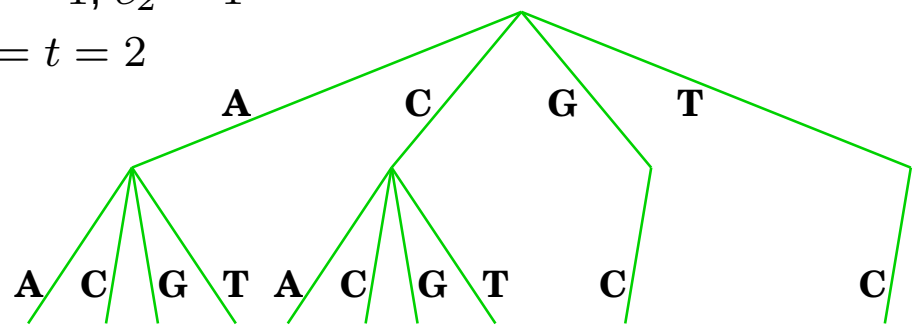
$p = 2$

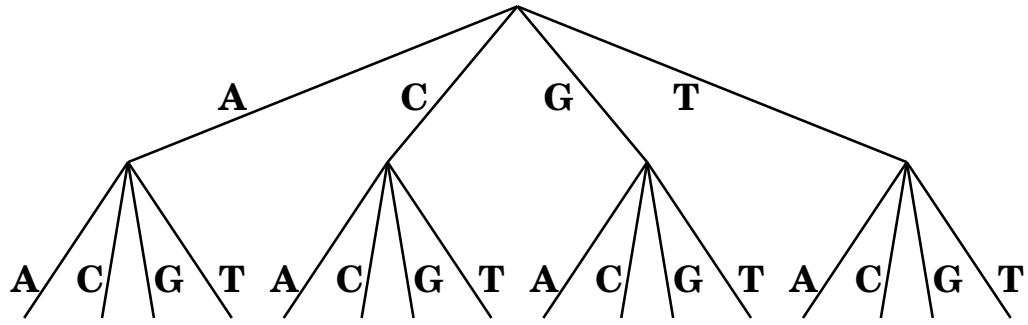
$l_1 = 2, d = 1, l_2 = 2$

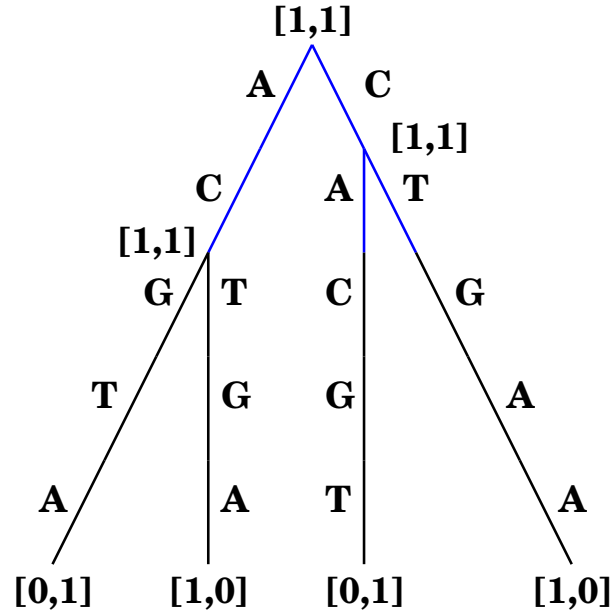
$e_1 = 1, e_2 = 1$

$q = t = 2$

Input sequences: ACTGAA and CACGTA







SMILE

Extraction of composite motifs

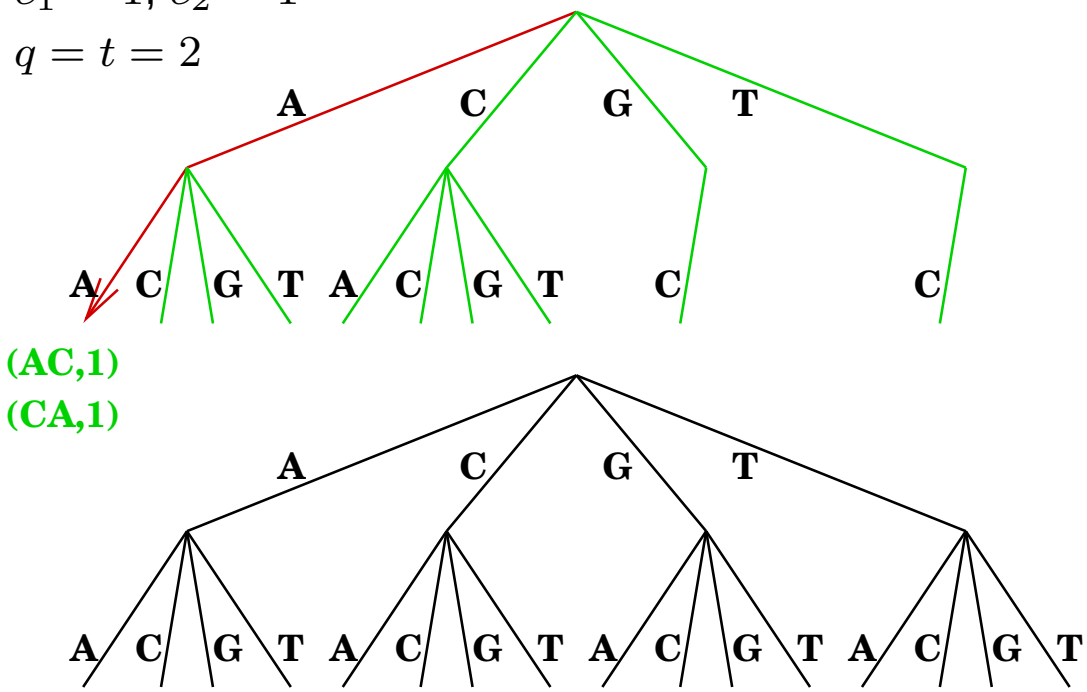
$p = 2$

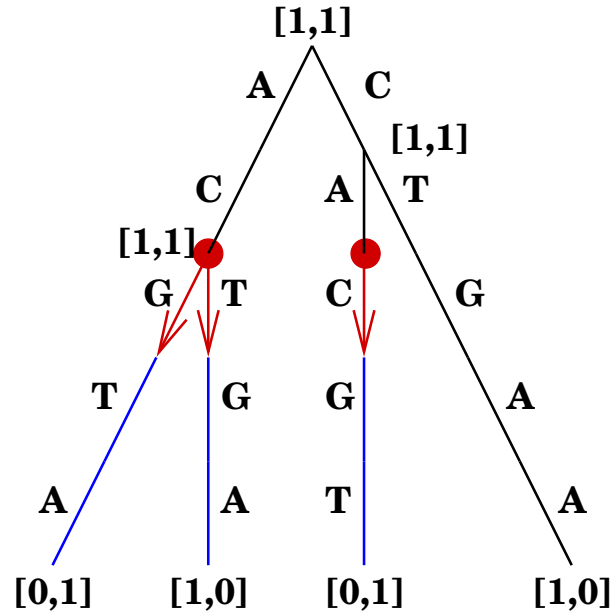
$l_1 = 2, d = 1, l_2 = 2$

$e_1 = 1, e_2 = 1$

$q = t = 2$

Input sequences: ACTGAA and CACGTA





SMILE

Extraction of composite motifs

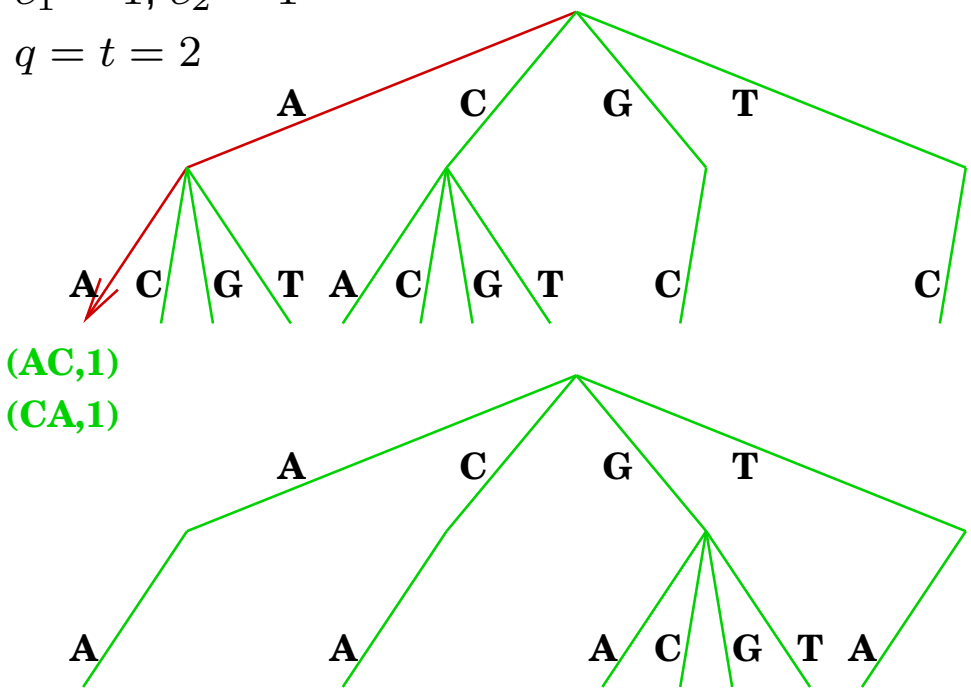
$p = 2$

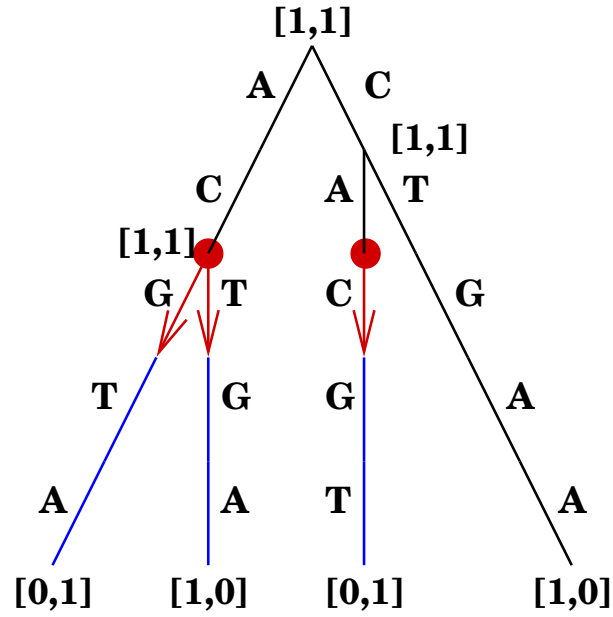
$l_1 = 2, d = 1, l_2 = 2$

$e_1 = 1, e_2 = 1$

$q = t = 2$

Input sequences: ACTGAA and CACGTA





SMILE

Extraction of composite motifs

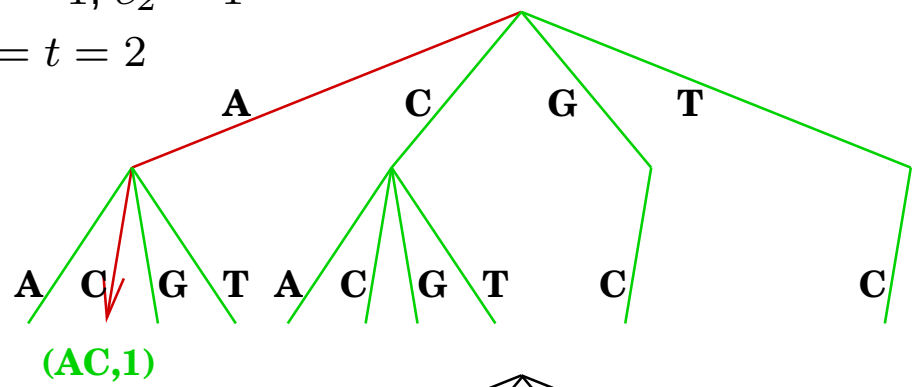
$p = 2$

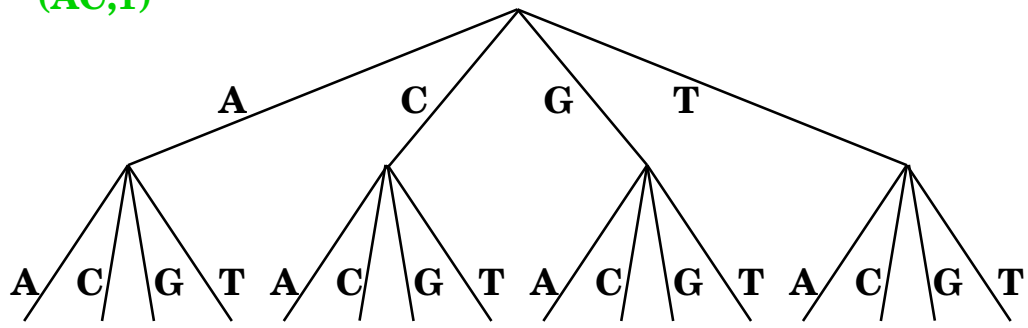
$l_1 = 2, d = 1, l_2 = 2$

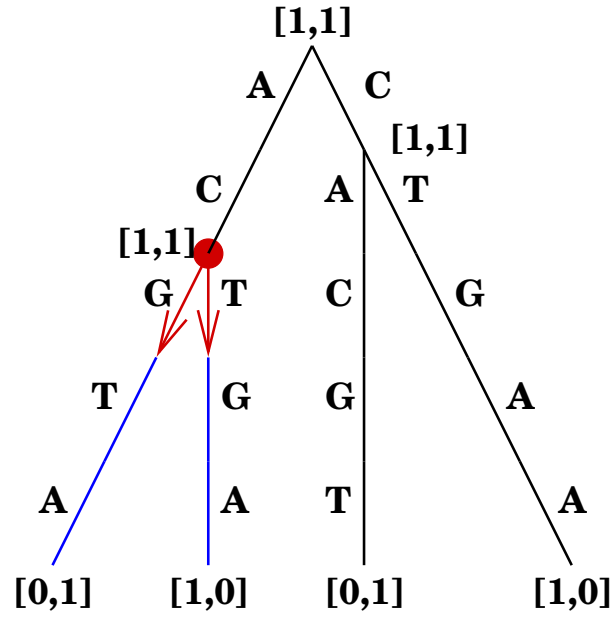
$e_1 = 1, e_2 = 1$

$q = t = 2$

Input sequences: ACTGAA and CACGTA







SMILE

Extraction of composite motifs

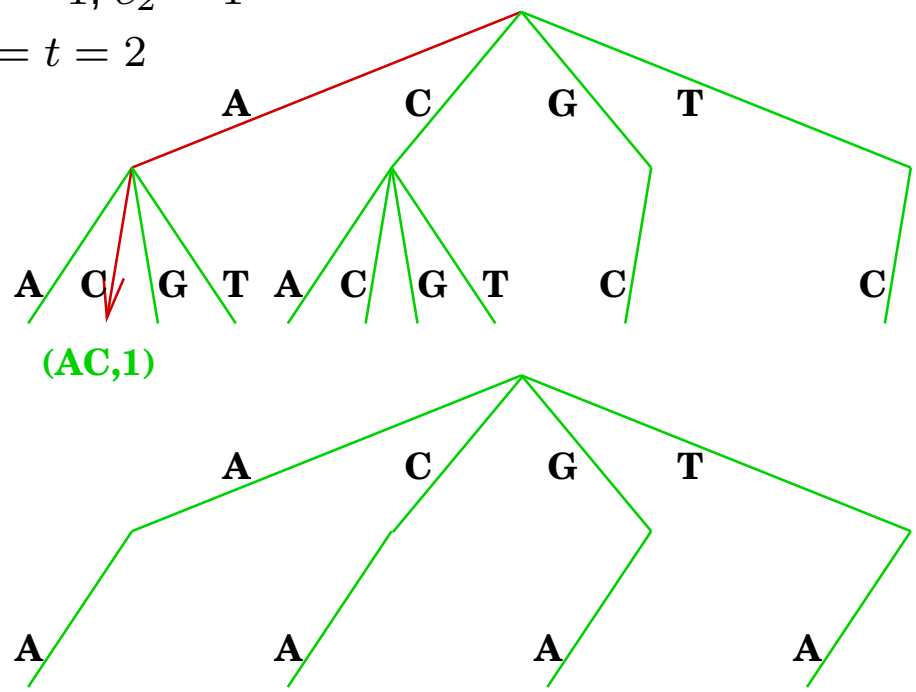
$p = 2$

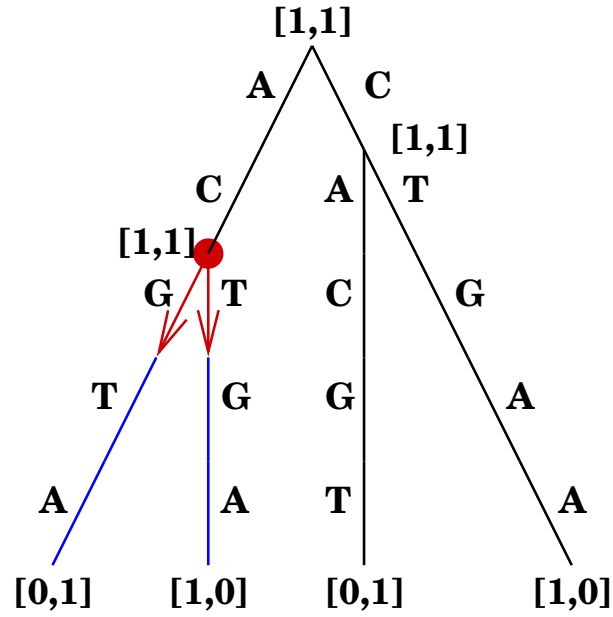
$l_1 = 2, d = 1, l_2 = 2$

$e_1 = 1, e_2 = 1$

$q = t = 2$

Input sequences: ACTGAA and CACGTA





SMILE

- The extraction of simple motifs takes $O(t^2 N \mathcal{V}(e,l))$ time and $O(t^2 N)$ space, where

$$\mathcal{V}(e,l) = \sum_{i=0}^{e} \binom{l}{i} (|\Sigma| - 1)^i \leq l^e |\Sigma|^e$$

SMILE

- The extraction of simple motifs takes $O(t^2 N \mathcal{V}(e,l))$ time and $O(t^2 N)$ space, where

$$\mathcal{V}(e,l) = \sum_{i=0}^{e} \binom{l}{i} (|\Sigma| - 1)^i \leq l^e |\Sigma|^e$$

- The simplest version for the extraction of composite motifs takes $O(p\Delta^2 \lambda_{(p-1)l+(p-1)d_{\max}} \mathcal{V}(e,l)^{p-1} + tp\Delta\lambda_{pl+(p-1)d_{\max}} \mathcal{V}(e,l)^p)$ time and $O(t^2 N)$ space where, λ_d denotes the number of nodes in $\mathcal{T}$ at depth d and $\Delta = d_{\max} - d_{\min}$

SMILE

- The extraction of simple motifs takes $O(t^2 N \mathcal{V}(e,l))$ time and $O(t^2 N)$ space, where

$$\mathcal{V}(e,l) = \sum_{i=0}^{e} \binom{l}{i} (|\Sigma| - 1)^i \leq l^e |\Sigma|^e$$

- The simplest version for the extraction of composite motifs takes $O(p\Delta^2 \lambda_{(p-1)l+(p-1)d_{\max}} \mathcal{V}(e,l)^{p-1} + tp\Delta\lambda_{pl+(p-1)d_{\max}} \mathcal{V}(e,l)^p)$ time and $O(t^2 N)$ space where, λ_d denotes the number of nodes in $\mathcal{T}$ at depth d and $\Delta = d_{\max} - d_{\min}$
- Recent versions yield even better time and space bounds

SMILE

- The extraction of simple motifs takes $O(t^2 N \mathcal{V}(e,l))$ time and $O(t^2 N)$ space, where

$$\mathcal{V}(e,l) = \sum_{i=0}^{e} \binom{l}{i} (|\Sigma| - 1)^i \leq l^e |\Sigma|^e$$

- The simplest version for the extraction of composite motifs takes $O(p\Delta^2 \lambda_{(p-1)l+(p-1)d_{\max}} \mathcal{V}(e,l)^{p-1} + tp\Delta\lambda_{pl+(p-1)d_{\max}} \mathcal{V}(e,l)^p)$ time and $O(t^2 N)$ space where, λ_d denotes the number of nodes in $\mathcal{T}$ at depth d and $\Delta = d_{\max} - d_{\min}$
- Recent versions yield even better time and space bounds
- Guarantees both convergence and completeness

SMILE

- The extraction of simple motifs takes $O(t^2 N \mathcal{V}(e,l))$ time and $O(t^2 N)$ space, where

$$\mathcal{V}(e,l) = \sum_{i=0}^{e} \binom{l}{i} (|\Sigma| - 1)^i \leq l^e |\Sigma|^e$$

- The simplest version for the extraction of composite motifs takes $O(p\Delta^2 \lambda_{(p-1)l+(p-1)d_{\max}} \mathcal{V}(e,l)^{p-1} + tp\Delta\lambda_{pl+(p-1)d_{\max}} \mathcal{V}(e,l)^p)$ time and $O(t^2 N)$ space where, λ_d denotes the number of nodes in $\mathcal{T}$ at depth d and $\Delta = d_{\max} - d_{\min}$
- Recent versions yield even better time and space bounds
- Guarantees both convergence and completeness
- Addresses several extensions to the original problem

MITRA

- MITRA is a hybrid algorithm

MITRA

- MITRA is a hybrid algorithm
- Combines the tree-like view of SMILE with information about pairwise similarities from the graph built by WINNOWER

MITRA

- MITRA is a hybrid algorithm
- Combines the tree-like view of SMILE with information about pairwise similarities from the graph built by WINNOWER
- Relies on a mismatch tree data structure $\mathcal{M}$
  - Similar to a trie
  - Splits the space of all possible motifs into disjoint spaces
  - Each subspace represents motifs with the same prefix

MITRA

- A mismatch tree $\mathcal{M}$ is a rooted tree where each internal node v has $|\Sigma|$ branches, each labelled with a different symbol of the alphabet

MITRA

- A mismatch tree $\mathcal{M}$ is a rooted tree where each internal node v has $|\Sigma|$ branches, each labelled with a different symbol of the alphabet
- The maximum depth of $\mathcal{M}$ is l

MITRA

- A mismatch tree $\mathcal{M}$ is a rooted tree where each internal node v has $|\Sigma|$ branches, each labelled with a different symbol of the alphabet
- The maximum depth of $\mathcal{M}$ is l
- Each node v corresponds to the subspace of motifs $\mathcal{P}$ with a fixed prefix defined by the path-label of v and contains a reference to all l-mers in $\mathcal{S}$ which are within e mismatches of a pattern $p \in \mathcal{P}$

MITRA

- Initially, $\mathcal{M}$ contains only the root node representing the space of all motifs

MITRA

- Initially, $\mathcal{M}$ contains only the root node representing the space of all motifs
- The nodes of $\mathcal{M}$ are expanded in a depth-first manner

MITRA

- Initially, $\mathcal{M}$ contains only the root node representing the space of all motifs
- The nodes of $\mathcal{M}$ are expanded in a depth-first manner
- While examining a node v, MITRA tries to assert whether the corresponding sub-space contains a pattern p for which there are $q \leq t$ occurrences in different input sequences with at most e mismatches

MITRA

- Initially, $\mathcal{M}$ contains only the root node representing the space of all motifs
- The nodes of $\mathcal{M}$ are expanded in a depth-first manner
- While examining a node v, MITRA tries to assert whether the corresponding sub-space contains a pattern p for which there are $q \leq t$ occurrences in different input sequences with at most e mismatches
- If a subspace does not contains a pattern in these conditions it is deemed *weak* and the mismatch tree is pruned

MITRA

- Initially, $\mathcal{M}$ contains only the root node representing the space of all motifs
- The nodes of $\mathcal{M}$ are expanded in a depth-first manner
- While examining a node v, MITRA tries to assert whether the corresponding sub-space contains a pattern p for which there are $q \leq t$ occurrences in different input sequences with at most e mismatches
- If a subspace does not contains a pattern in these conditions it is deemed *weak* and the mismatch tree is pruned
- Whenever the algorithm cannot determine whether the current node refers to a weak subspace, the node is expanded and we move down one level

MITRA

- Initially, $\mathcal{M}$ contains only the root node representing the space of all motifs
- The nodes of $\mathcal{M}$ are expanded in a depth-first manner
- While examining a node v, MITRA tries to assert whether the corresponding sub-space contains a pattern p for which there are $q \leq t$ occurrences in different input sequences with at most e mismatches
- If a subspace does not contains a pattern in these conditions it is deemed *weak* and the mismatch tree is pruned
- Whenever the algorithm cannot determine whether the current node refers to a weak subspace, the node is expanded and we move down one level
- If we reach a node v at level l the path-label of v is reported as a valid pattern

MITRA

- Recall that MITRA keeps track of all valid l-mers for each node v of $\mathcal{M}$

MITRA

- Recall that MITRA keeps track of all valid l-mers for each node v of $\mathcal{M}$
- An l-mer referred from v is valid if it matches the prefix of the path-label of v with at most e mismatches

MITRA

- Recall that MITRA keeps track of all valid l-mers for each node v of $\mathcal{M}$
- An l-mer referred from v is valid if it matches the prefix of the path-label of v with at most e mismatches
- The set of valid l-mers of a node v is a subset of the valid l-mers referred from the parent of v

MITRA

- Recall that MITRA keeps track of all valid l-mers for each node v of $\mathcal{M}$
- An l-mer referred from v is valid if it matches the prefix of the path-label of v with at most e mismatches
- The set of valid l-mers of a node v is a subset of the valid l-mers referred from the parent of v
- We can, then, efficiently generate the set of valid l-mers of a node at the expense of the computed information about its parent

MITRA

- When expanding the parent of a node v, each of its valid l-mers can fit into two situations
  - Either the position corresponding to the label of the branch to v matches the l-mer
  - or not

MITRA

- When expanding the parent of a node v, each of its valid l-mers can fit into two situations
  - Either the position corresponding to the label of the branch to v matches the l-mer
  - or not
- If we have a match, the l-mer is still valid for the child

MITRA

- When expanding the parent of a node v, each of its valid l-mers can fit into two situations
  - Either the position corresponding to the label of the branch to v matches the l-mer
  - or not
- If we have a match, the l-mer is still valid for the child
- Otherwise the mismatch count increases and
  - If the threshold e is surpassed the l-mer in not included in the list of v
  - If not, the l-mer remains valid, albeit with a greater mismatch count

MITRA

- We are left with the problem of deciding whether a subspace associated with a node v is weak

MITRA

- We are left with the problem of deciding whether a subspace associated with a node v is weak
- The authors present two alternatives
  - Counting the number of different sequences contributing to the list of valid l-mers referred by v (MITRA-COUNT), which is tantamount to the operation of SMILE when extracting simple motifs
  - Building a graph for the node v (MITRA-GRAPH)

MITRA-GRAPH

- Consider a pattern p and a set of sequences $\mathcal{S}$

MITRA-GRAPH

- Consider a pattern p and a set of sequences $\mathcal{S}$
- We construct a graph $G(p, \mathcal{S})$ where each l-mer occurring in $\mathcal{S}$ is a vertex and there is an edge connecting two l-mers if p is within e mismatches of both and they occur in different input sequences

MITRA-GRAPH

- Consider a pattern p and a set of sequences $\mathcal{S}$
- We construct a graph $G(p, \mathcal{S})$ where each l-mer occurring in $\mathcal{S}$ is a vertex and there is an edge connecting two l-mers if p is within e mismatches of both and they occur in different input sequences
- If p is under the conditions of our problem, we shall have a q-clique in G

MITRA-GRAPH

- Consider a pattern p and a set of sequences $\mathcal{S}$
- We construct a graph $G(p, \mathcal{S})$ where each l-mer occurring in $\mathcal{S}$ is a vertex and there is an edge connecting two l-mers if p is within e mismatches of both and they occur in different input sequences
- If p is under the conditions of our problem, we shall have a q-clique in G
- Now, consider a set of patterns $\mathcal{P}$

MITRA-GRAPH

- Consider a pattern p and a set of sequences $\mathcal{S}$
- We construct a graph $G(p, \mathcal{S})$ where each l-mer occurring in $\mathcal{S}$ is a vertex and there is an edge connecting two l-mers if p is within e mismatches of both and they occur in different input sequences
- If p is under the conditions of our problem, we shall have a q-clique in G
- Now, consider a set of patterns $\mathcal{P}$
- We define $G(\mathcal{P}, \mathcal{S})$ as the graph whose set of edges is the union of the edges of each $G(p, \mathcal{S})$, $p \in \mathcal{P}$

MITRA-GRAPH

- Consider a pattern p and a set of sequences $\mathcal{S}$
- We construct a graph $G(p, \mathcal{S})$ where each l-mer occurring in $\mathcal{S}$ is a vertex and there is an edge connecting two l-mers if p is within e mismatches of both and they occur in different input sequences
- If p is under the conditions of our problem, we shall have a q-clique in G
- Now, consider a set of patterns $\mathcal{P}$
- We define $G(\mathcal{P}, \mathcal{S})$ as the graph whose set of edges is the union of the edges of each $G(p, \mathcal{S})$, $p \in \mathcal{P}$
- If we can guarantee that there are no q-cliques in $G(\mathcal{P}, \mathcal{S})$ we can confidently say that $\mathcal{P}$ is weak

MITRA-GRAPH

- MITRA-GRAPH is concerned with proving that there are no q-cliques, whereas WINNOWER was trying to find some

MITRA-GRAPH

- MITRA-GRAPH is concerned with proving that there are no q-cliques, whereas WINNOWER was trying to find some
- Like WINNOWER, MITRA-GRAPH adopts a winnowing strategy, but only to remove vertices (and respective incident edges) with a partition degree less than $q - 1$

MITRA-GRAPH

- MITRA-GRAPH is concerned with proving that there are no q-cliques, whereas WINNOWER was trying to find some
- Like WINNOWER, MITRA-GRAPH adopts a winnowing strategy, but only to remove vertices (and respective incident edges) with a partition degree less than $q - 1$
- If the remaining edges are insufficient to form a q-clique we can rule out its existence, otherwise the subspace will be further divided

MITRA-GRAPH

- If MITRA-GRAPH had to build $G(\mathcal{P}, \mathcal{S})$ for each visited node it would be very innefficient

MITRA-GRAPH

- If MITRA-GRAPH had to build $G(\mathcal{P}, \mathcal{S})$ for each visited node it would be very innefficient
- Instead the graph for a node v is built at the expense of the graph of its parent u

MITRA-GRAPH

- If MITRA-GRAPH had to build $G(\mathcal{P}, \mathcal{S})$ for each visited node it would be very innefficient
- Instead the graph for a node v is built at the expense of the graph of its parent u
- Let (m_1, m_2) be an edge of the graph of u, p be the path-label of v and d be the length of p

MITRA-GRAPH

- If MITRA-GRAPH had to build $G(\mathcal{P}, \mathcal{S})$ for each visited node it would be very innefficient
- Instead the graph for a node v is built at the expense of the graph of its parent u
- Let (m_1, m_2) be an edge of the graph of u, p be the path-label of v and d be the length of p
- If $\delta(m_1, p) = e_1$, $\delta(m_2, p) = e_2$ and if the last $l - d$ characters of m_1 and m_2 mismatch in at most ε positions, then

MITRA-GRAPH

- If MITRA-GRAPH had to build $G(\mathcal{P}, \mathcal{S})$ for each visited node it would be very innefficient
- Instead the graph for a node v is built at the expense of the graph of its parent u
- Let (m_1, m_2) be an edge of the graph of u, p be the path-label of v and d be the length of p
- If $\delta(m_1, p) = e_1$, $\delta(m_2, p) = e_2$ and if the last $l - d$ characters of m_1 and m_2 mismatch in at most ε positions, then
- (m_1, m_2) is an edge of the graph of v iff $e_1 \leq e$, $e_2 \leq e$ and $e_1 + e_2 + \varepsilon \leq 2e$

MITRA-GRAPH

- If MITRA-GRAPH had to build $G(\mathcal{P}, \mathcal{S})$ for each visited node it would be very innefficient
- Instead the graph for a node v is built at the expense of the graph of its parent u
- Let (m_1, m_2) be an edge of the graph of u, p be the path-label of v and d be the length of p
- If $\delta(m_1, p) = e_1$, $\delta(m_2, p) = e_2$ and if the last $l - d$ characters of m_1 and m_2 mismatch in at most ε positions, then
- (m_1, m_2) is an edge of the graph of v iff $e_1 \leq e$, $e_2 \leq e$ and $e_1 + e_2 + \varepsilon \leq 2e$
- Note that for the root node, where $e_1 = e_2 = l - d = 0$, the condition is $\varepsilon = \delta(m_1, m_2) \leq 2e$ which is precisely the graph built by WINNOWER

Conclusions

- We presented some of the most popular combinatorial algorithms to solve the problem of finding common motifs in DNA sequences

Conclusions

- We presented some of the most popular combinatorial algorithms to solve the problem of finding common motifs in DNA sequences
- SMILE and MITRA-GRAPH are the most interesting

Conclusions

- We presented some of the most popular combinatorial algorithms to solve the problem of finding common motifs in DNA sequences
- SMILE and MITRA-GRAPH are the most interesting
- WINNOWER is not guaranteed to find a solution

Conclusions

- We presented some of the most popular combinatorial algorithms to solve the problem of finding common motifs in DNA sequences
- SMILE and MITRA-GRAPH are the most interesting
- WINNOWER is not guaranteed to find a solution
- PRUNER is incomplete

Conclusions

- We presented some of the most popular combinatorial algorithms to solve the problem of finding common motifs in DNA sequences
- SMILE and MITRA-GRAPH are the most interesting
- WINNOWER is not guaranteed to find a solution
- PRUNER is incomplete
- SMILE is unique in its approach to composite motifs

Conclusions

- We presented some of the most popular combinatorial algorithms to solve the problem of finding common motifs in DNA sequences
- SMILE and MITRA-GRAPH are the most interesting
- WINNOWER is not guaranteed to find a solution
- PRUNER is incomplete
- SMILE is unique in its approach to composite motifs
- MITRA-GRAPH has the virtue of combining different contributions of SMILE and WINNOWER to avoid unnecessary explorations of the search space

Conclusions

- SMILE can still outperform MITRA-GRAPH for smaller motifs

Conclusions

- SMILE can still outperform MITRA-GRAPH for smaller motifs
- The overhead of building the graphs is only worth for larger values of l

Conclusions

- SMILE can still outperform MITRA-GRAPH for smaller motifs
- The overhead of building the graphs is only worth for larger values of l
- Yet, SMILE is more sensitive to high degrees of degeneration since all nodes of $\mathcal{T}$ down to level e must always be visited, whereas MITRA-GRAPH can prune the mismatch tree earlier

Conclusions

- SMILE can still outperform MITRA-GRAPH for smaller motifs
- The overhead of building the graphs is only worth for larger values of l
- Yet, SMILE is more sensitive to high degrees of degeneration since all nodes of $\mathcal{T}$ down to level e must always be visited, whereas MITRA-GRAPH can prune the mismatch tree earlier
- However, it is difficult to compare the performance of different motif finders without actual tests with real or synthetic data

Conclusions

- SMILE can still outperform MITRA-GRAPH for smaller motifs
- The overhead of building the graphs is only worth for larger values of l
- Yet, SMILE is more sensitive to high degrees of degeneration since all nodes of $\mathcal{T}$ down to level e must always be visited, whereas MITRA-GRAPH can prune the mismatch tree earlier
- However, it is difficult to compare the performance of different motif finders without actual tests with real or synthetic data
- Most implementations are not publicly available

Thank you

Thank you ☺