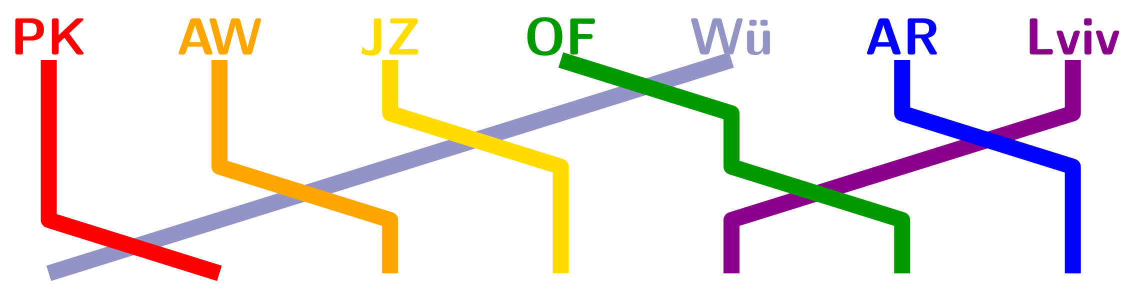
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computing Optimal Tangles Faster

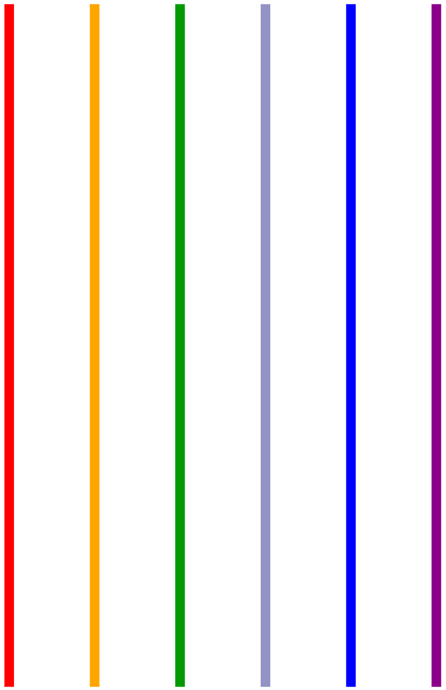


Oksana Firman Alexander Wolff
Philipp Kindermann Johannes Zink
Julius-Maximilians-Universität Würzburg, Germany

Alexander Ravsky
Pidstryhach Institute for Applied Problems of Mechanics and Mathematics,
National Academy of Sciences of Ukraine, Lviv, Ukraine

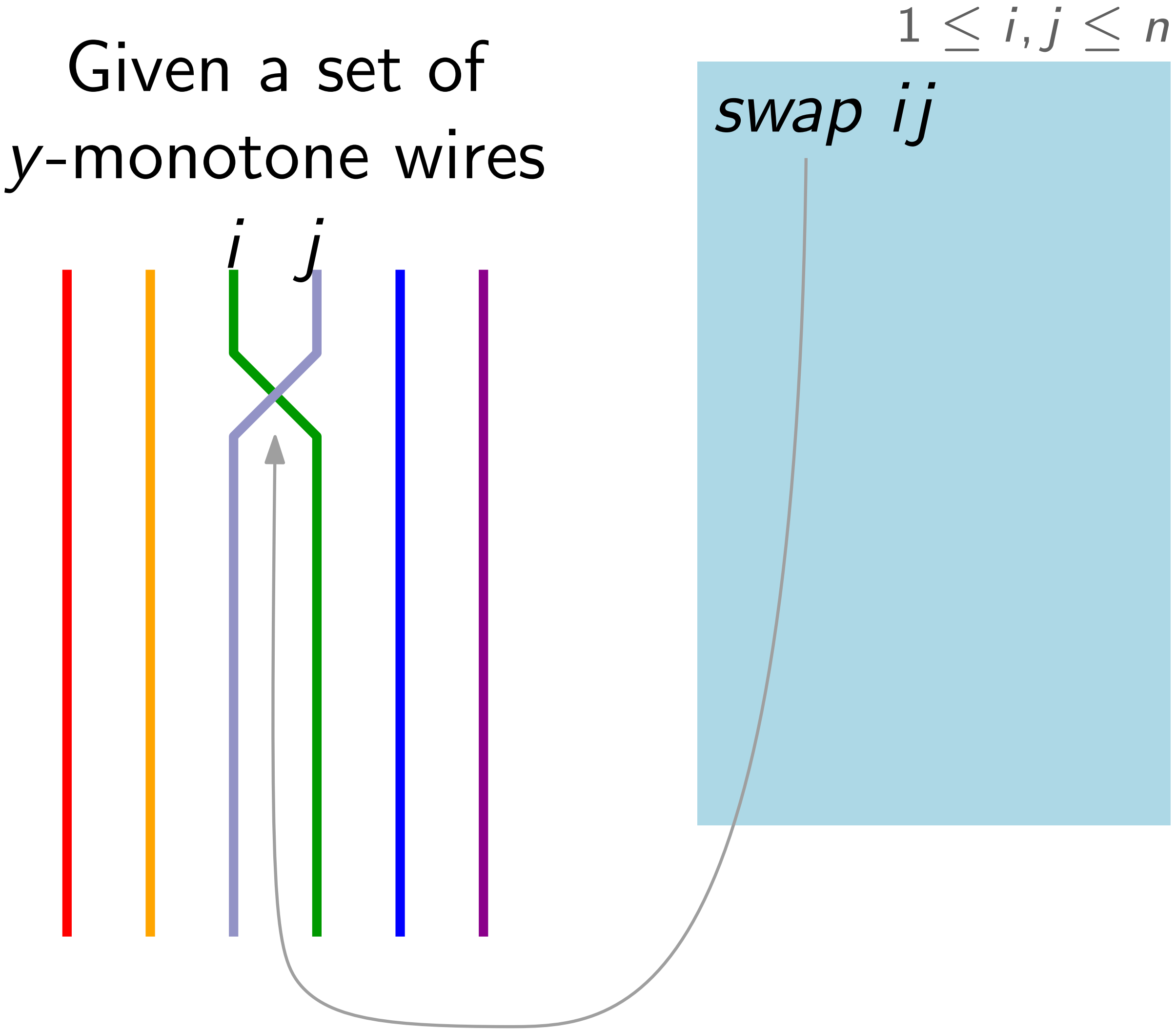
Introduction

Given a set of
y-monotone wires



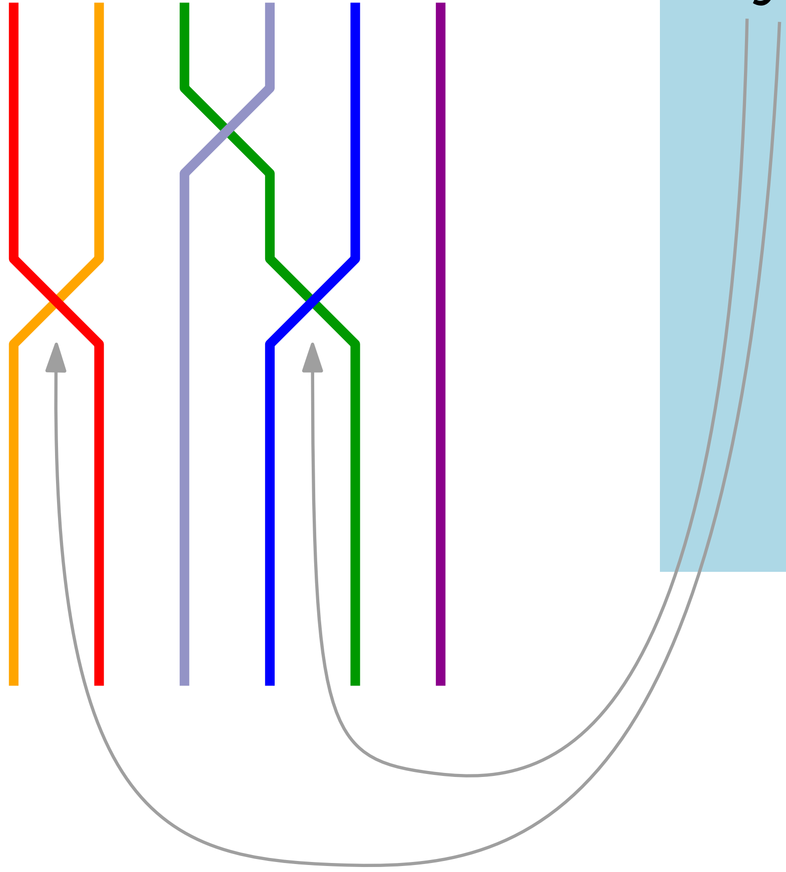
Introduction

Given a set of
y-monotone wires



Introduction

Given a set of
y-monotone wires



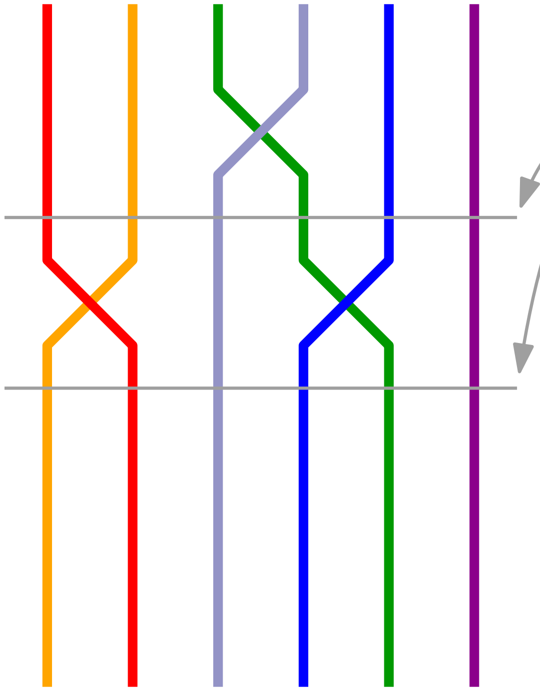
$1 \le i, j \le n$

swap ij

disjoint swaps

Introduction

Given a set of
y-monotone wires



$1 \leq i, j \leq n$

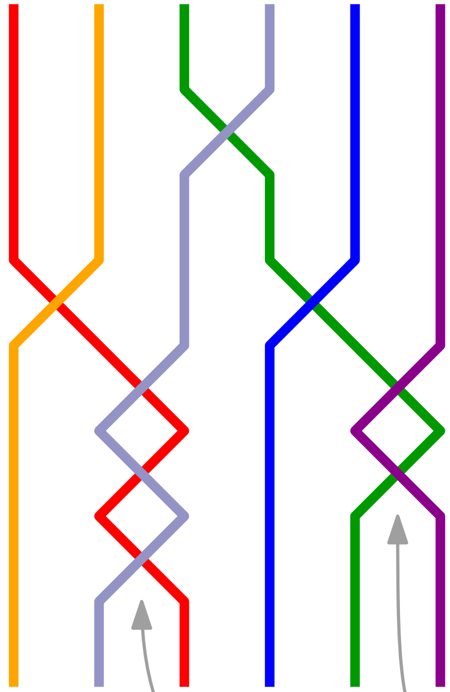
swap ij

disjoint swaps

adjacent permutations

Introduction

Given a set of
y-monotone wires



$1 \leq i, j \leq n$

swap ij

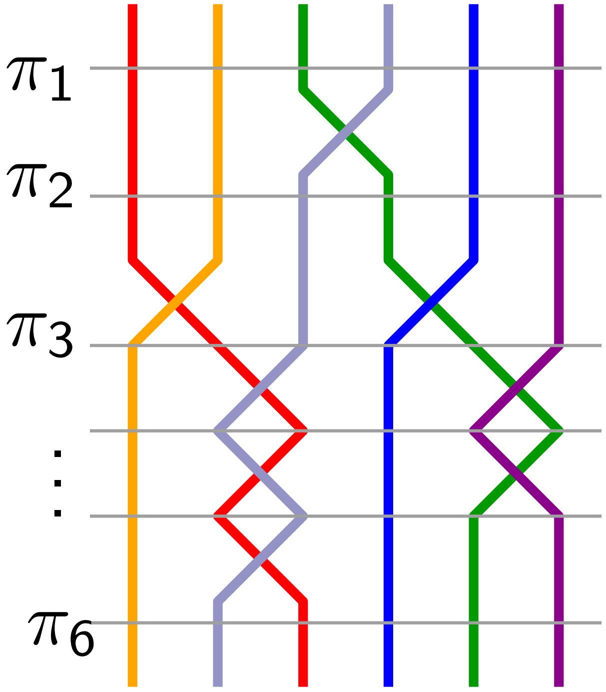
disjoint swaps

adjacent permutations

multiple swaps

Introduction

Given a set of
y-monotone wires



$1 \leq i, j \leq n$

swap ij

disjoint swaps

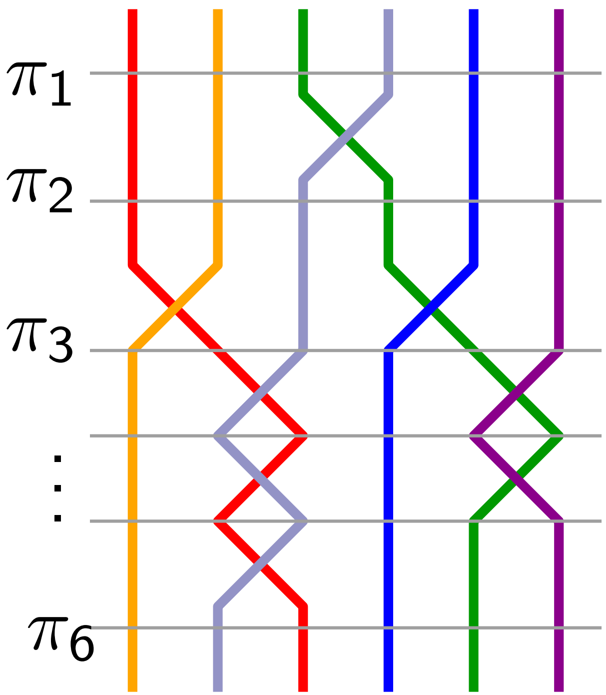
adjacent permutations

multiple swaps

tangle T of *height* $h(T)$

Introduction

Given a set of y-monotone wires



$1 \leq i, j \leq n$

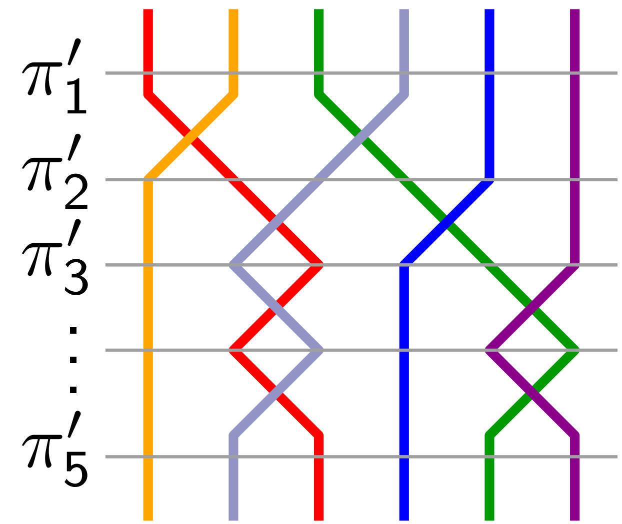
swap ij

disjoint swaps

adjacent permutations

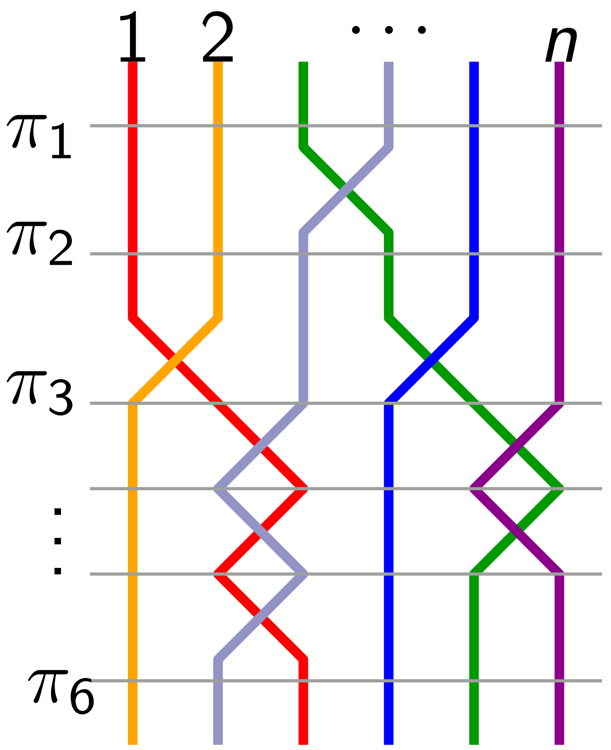
multiple swaps

tangle T of *height* $h(T)$



Introduction

Given a set of y-monotone wires



$1 \leq i, j \leq n$

swap ij

disjoint swaps

adjacent permutations

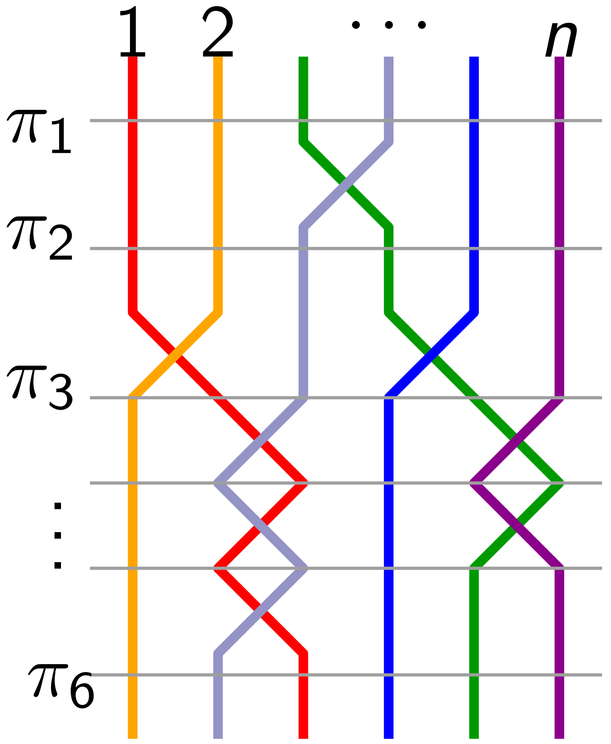
multiple swaps

tangle T of *height* $h(T)$

...and given a list of swaps L

Introduction

Given a set of
y-monotone wires



$1 \leq i, j \leq n$

swap ij

disjoint swaps

adjacent permutations

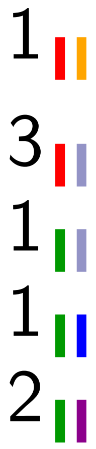
multiple swaps

tangle T of *height* $h(T)$

...and given a list of swaps L

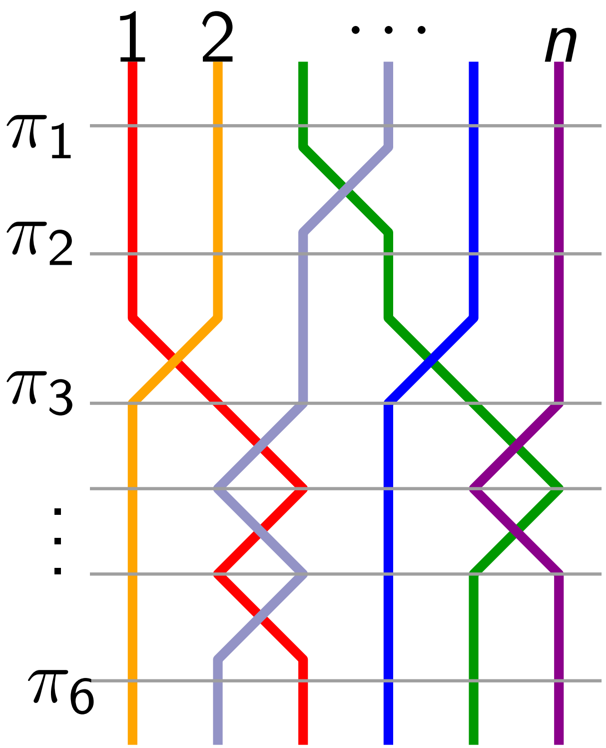
$1 \leq i < j \leq n$

- as a multiset (ℓ_{ij})



Introduction

Given a set of y-monotone wires



$1 \leq i, j \leq n$

swap ij

disjoint swaps

adjacent permutations

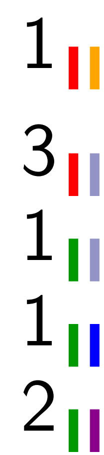
multiple swaps

tangle T of *height* $h(T)$

. . . and given a list of swaps L

$1 \leq i < j \leq n$

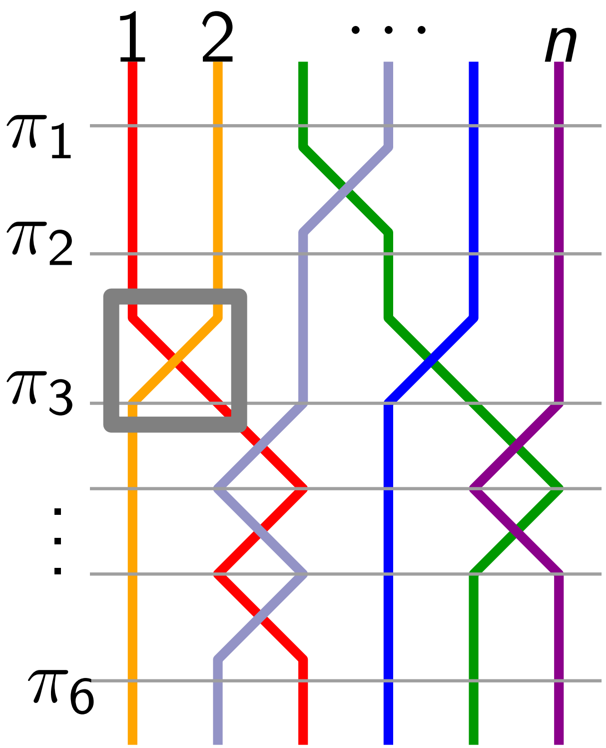
- as a multiset (ℓ_{ij})



Tangle $T(L)$ *realizes* list L

Introduction

Given a set of y-monotone wires



$1 \leq i, j \leq n$

swap ij

disjoint swaps

adjacent permutations

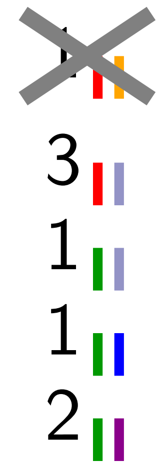
multiple swaps

tangle T of *height* $h(T)$

…and given a list of swaps L

$1 \leq i < j \leq n$

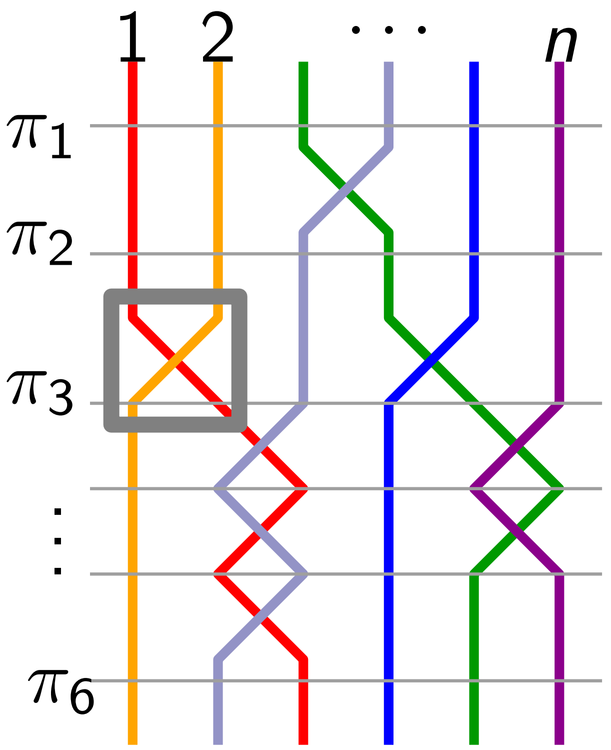
- as a multiset (ℓ_{ij})



Tangle $T(L)$ *realizes* list L

Introduction

Given a set of y-monotone wires



$1 \le i, j \le n$

swap ij

disjoint swaps

adjacent permutations

multiple swaps

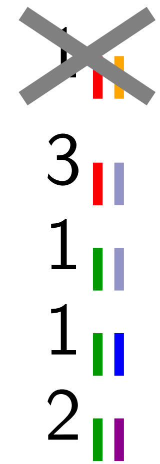
tangle T of *height* $h(T)$

Tangle $T(L)$ *realizes* list L

…and given a list of swaps L

$1 \le i < j \le n$

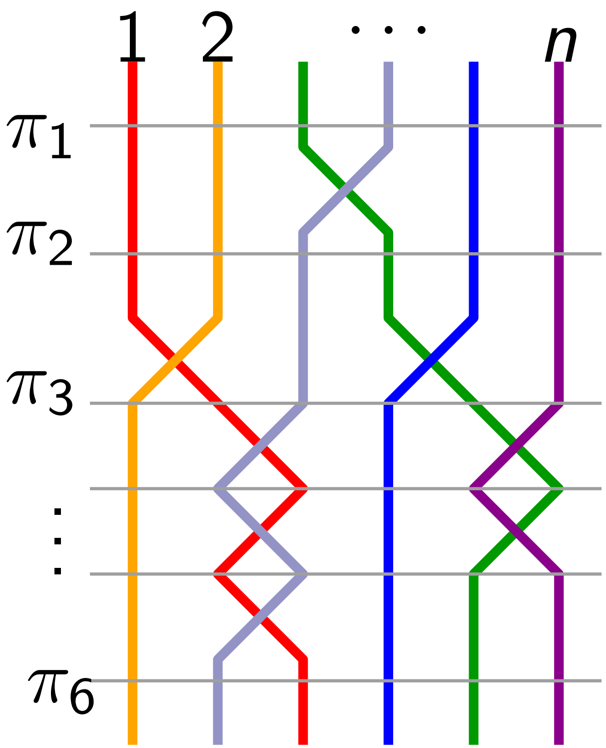
- as a multiset (ℓ_{ij})



not *feasible*

Introduction

Given a set of y-monotone wires



$1 \leq i, j \leq n$

swap ij

disjoint swaps

adjacent permutations

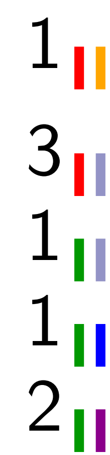
multiple swaps

tangle T of *height h(T)*

...and given a list of swaps L

$1 \leq i < j \leq n$

- as a multiset (ℓ_{ij})

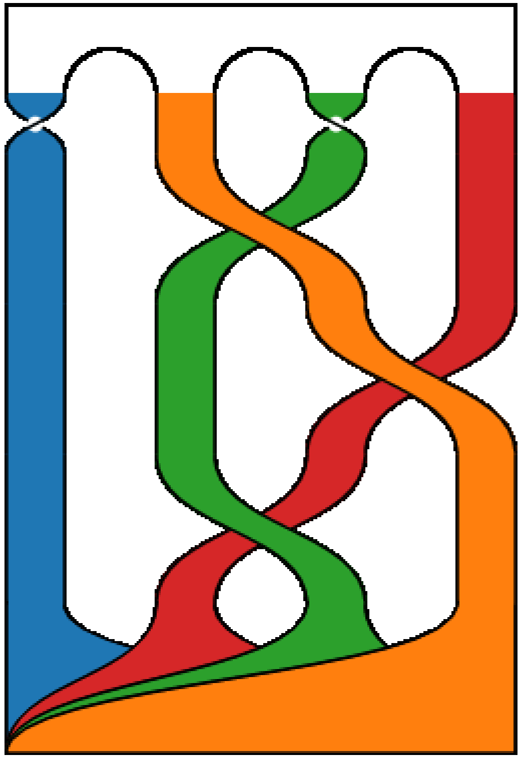


Tangle $T(L)$ *realizes* list L

A tangle $T(L)$ is *optimal* if it has the minimum height among all tangles realizing the list L.

Related work

- *Olszewski et al.* Visualizing the template of a chaotic attractor.
  GD 2018

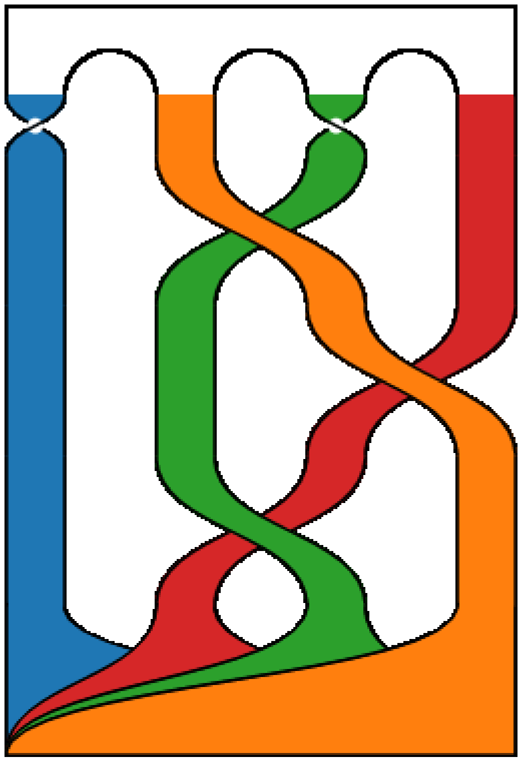


Related work

- *Olszewski et al.* Visualizing the template of a chaotic attractor.
  GD 2018

$\Updownarrow$

list



Related work

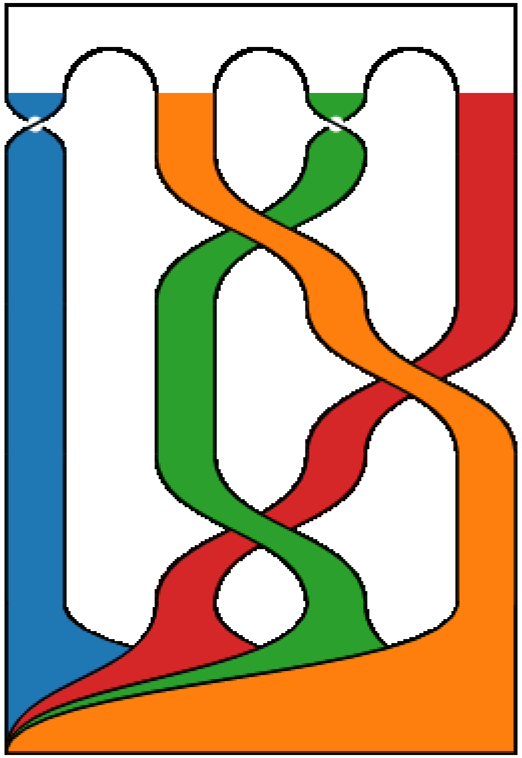
- *Olszewski et al.* Visualizing the template of a chaotic attractor. GD 2018

template

⇕

list

Algorithm to find the optimal tangle



Related work

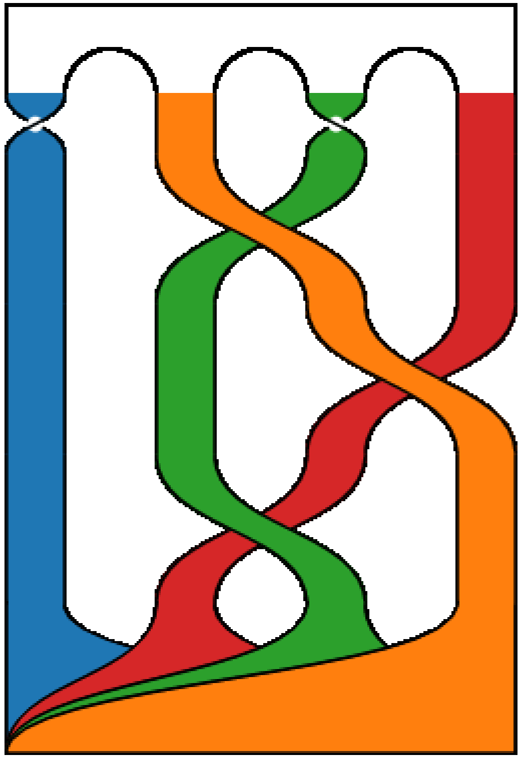
- *Olszewski et al.* Visualizing the template of a chaotic attractor. GD 2018

template

$\Updownarrow$

list

Algorithm to find the optimal tangle	*Complexity* **?**



Related work

- *Olszewski et al.* Visualizing the template of a chaotic attractor. GD 2018

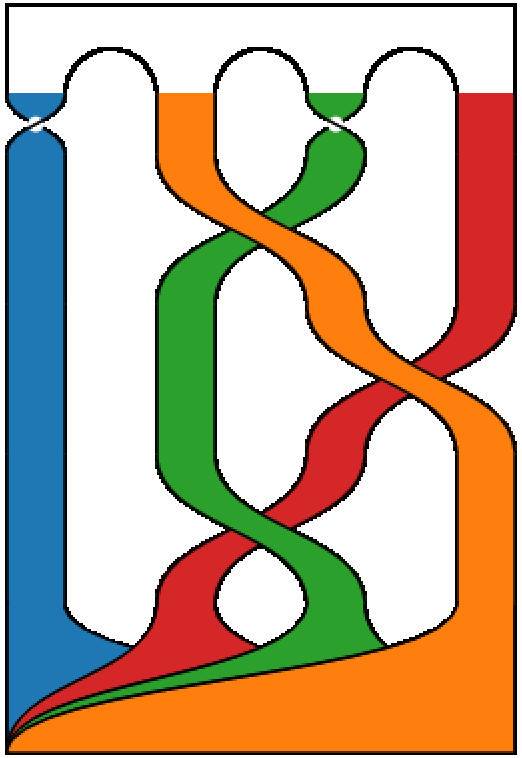
template ⇕ list

Algorithm to find the optimal tangle

Complexity ?

- *Wang.* Novel routing schemes for IC layout part I: Two-layer channel routing. DAC 1991

Given: initial and *final* permutations



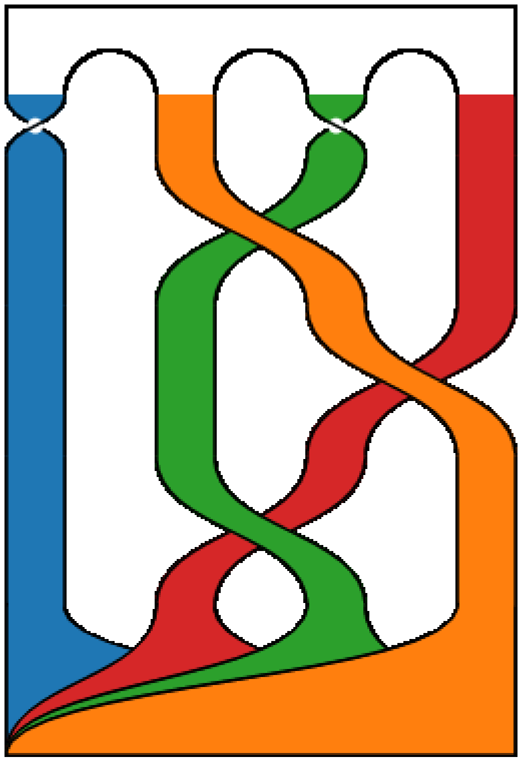
Related work

- *Olszewski et al.* Visualizing the template of a chaotic attractor. GD 2018

template
$\Updownarrow$
list

Algorithm to find the optimal tangle

Complexity ?



- *Wang.* Novel routing schemes for IC layout part I: Two-layer channel routing. DAC 1991

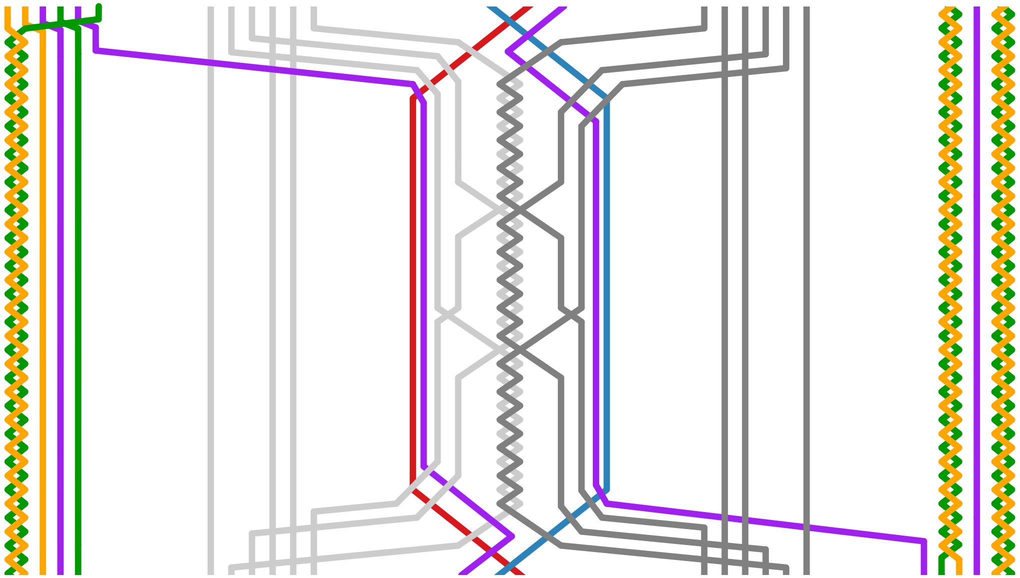
Given: initial and *final* permutations

- *Bereg et al.* Drawing Permutations with Few Corners. GD 2013

Objective: minimize the number of *bends*

Overview

- **Complexity**
  NP-hardness by reduction from 3-Partition



- **Improved the algorithm** of [Olszewski et al., GD'18]
  Using the Dynamic Program

$$O\left(\frac{\varphi^{2|L|}}{5^{|L|/n}} n\right) \longrightarrow O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}} \varphi^n n\right)$$

- Experiments

Complexity

Theorem

TANGLE-HEIGHT MINIMIZATION is NP-hard.

Complexity

Theorem

Tangle-Height Minimization is NP-hard.

Proof

Reduction from 3-Partition

Complexity

Theorem

Tangle-Height Minimization is NP-hard.

Proof

Reduction from 3-Partition

Given: a multiset A of $3m$ positive integers

$a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_{3m-2} \quad a_{3m-1} \quad a_{3m}$

Complexity

Theorem

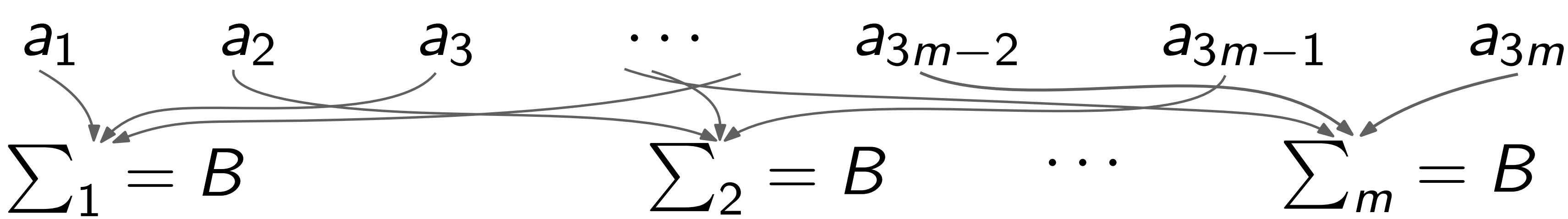
TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof

Reduction from 3-PARTITION

Given: a multiset A of $3m$ positive integers

Objective: decide whether A can be partitioned into m groups of three elements each that all sum up to the same value B



Complexity

Theorem

TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof

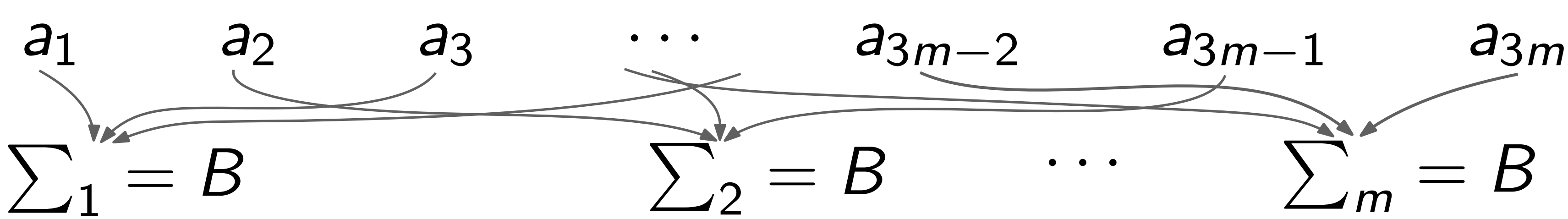
$\frac{B}{4} < a_i < \frac{B}{2}$

B is poly in m

Reduction from 3-PARTITION

Given: a multiset A of $3m$ positive integers

Objective: decide whether A can be partitioned into m groups of three elements each that all sum up to the same value B



Complexity

Theorem

TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof

$\frac{B}{4} < a_i < \frac{B}{2}$

B is poly in m

Reduction from 3-PARTITION

Given: a multiset A of $3m$ positive integers

Objective: decide whether A can be partitioned into m groups of three elements each that all sum up to the same value B

$a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_{3m-2} \quad a_{3m-1} \quad a_{3m}$

$\sum_1 = B \qquad \sum_2 = B \qquad \cdots \qquad \sum_m = B$

Given: an instance A of 3-PARTITION

Complexity

Theorem

Tangle-Height Minimization is NP-hard.

Proof

$\frac{B}{4} < a_i < \frac{B}{2}$

B is poly in m

Reduction from 3-Partition

Given: a multiset A of $3m$ positive integers

Objective: decide whether A can be partitioned into m groups of three elements each that all sum up to the same value B

$a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_{3m-2} \quad a_{3m-1} \quad a_{3m}$

$\sum_1 = B \qquad \sum_2 = B \qquad \cdots \qquad \sum_m = B$

Given: an instance A of 3-Partition

Task: construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance

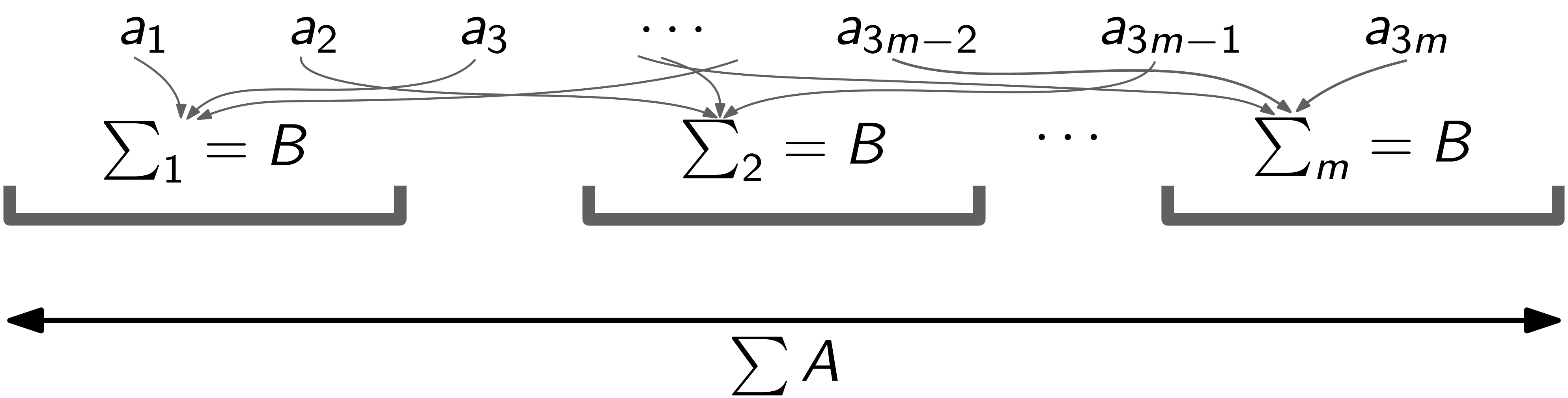
Complexity

Theorem

TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof

Reduction from 3-PARTITION



Given:	an instance A of 3-PARTITION
Task:	construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance

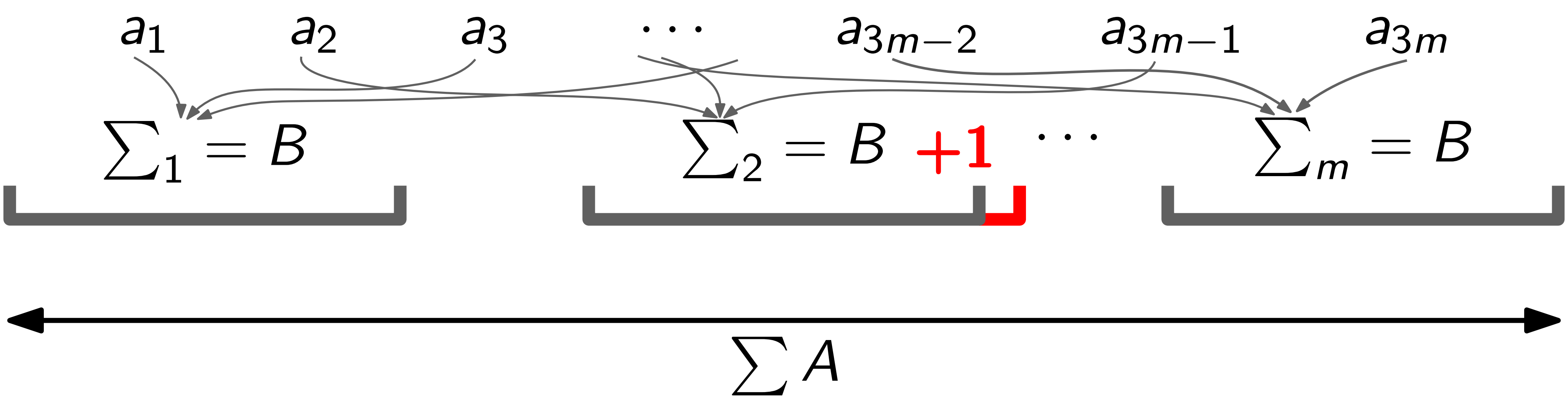
Complexity

Theorem

TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof

Reduction from 3-PARTITION



Given:	an instance A of 3-PARTITION
Task:	construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance

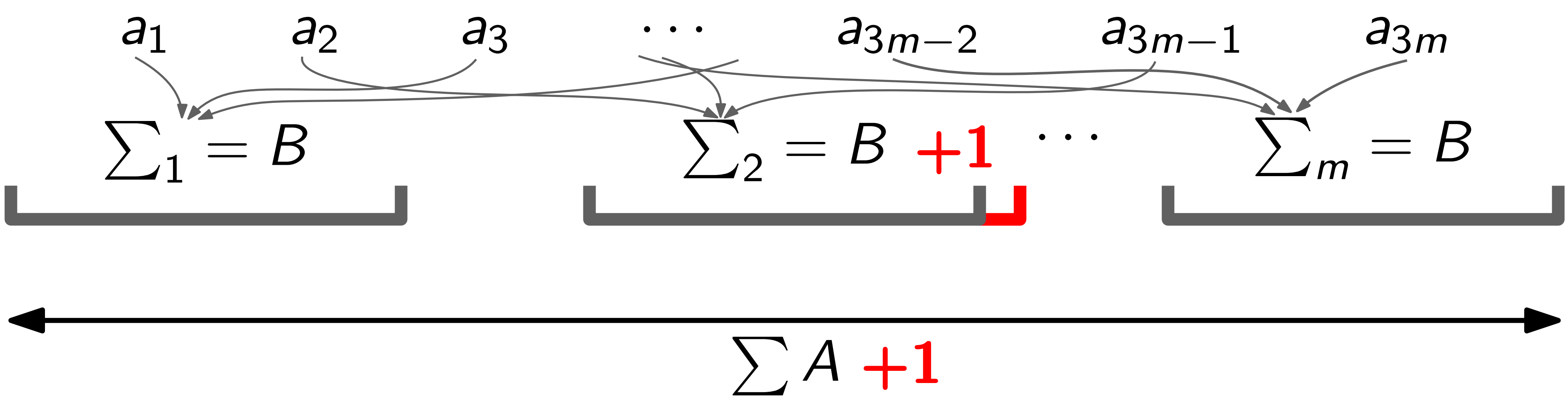
Complexity

Theorem

Tangle-Height Minimization is NP-hard.

Proof

Reduction from 3-Partition



Given: an instance A of 3-Partition

Task: construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance

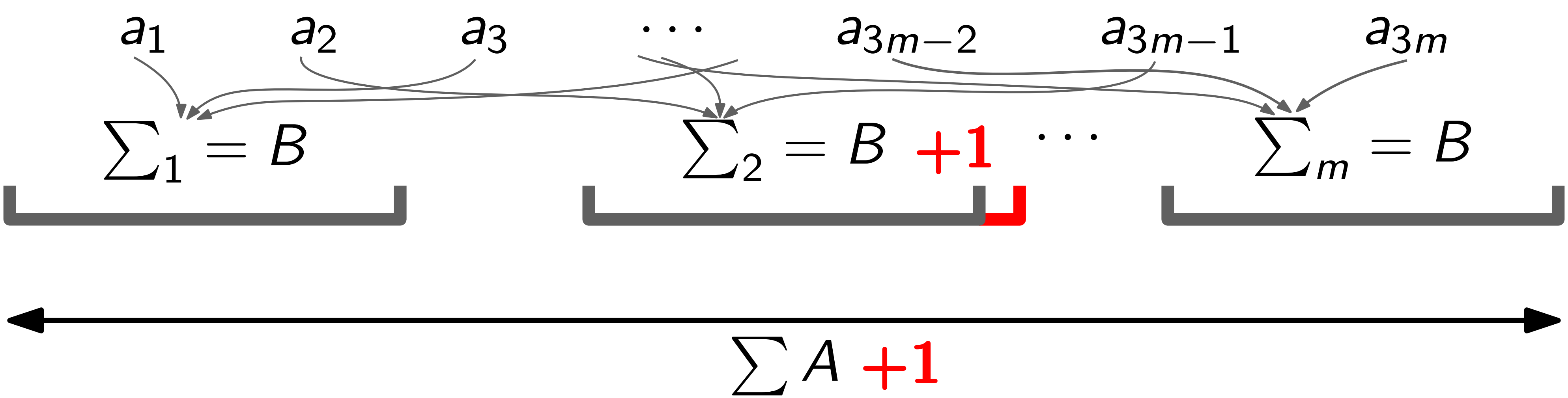
Complexity

Theorem

TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof

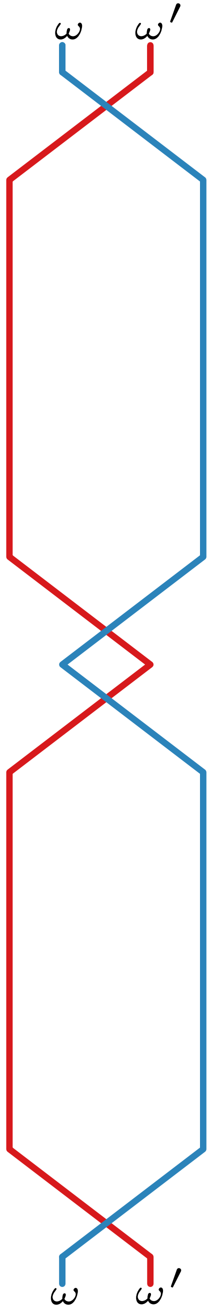
Reduction from 3-PARTITION



Given: an instance A of 3-PARTITION

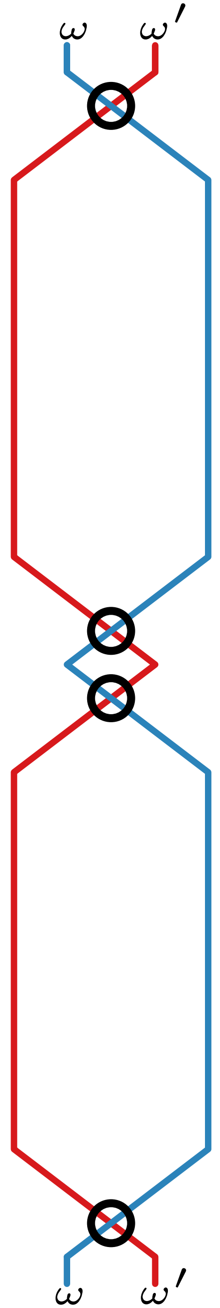
Task: construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A + 1) + 7m^2$ iff A is a yes-instance

Constructing the list L

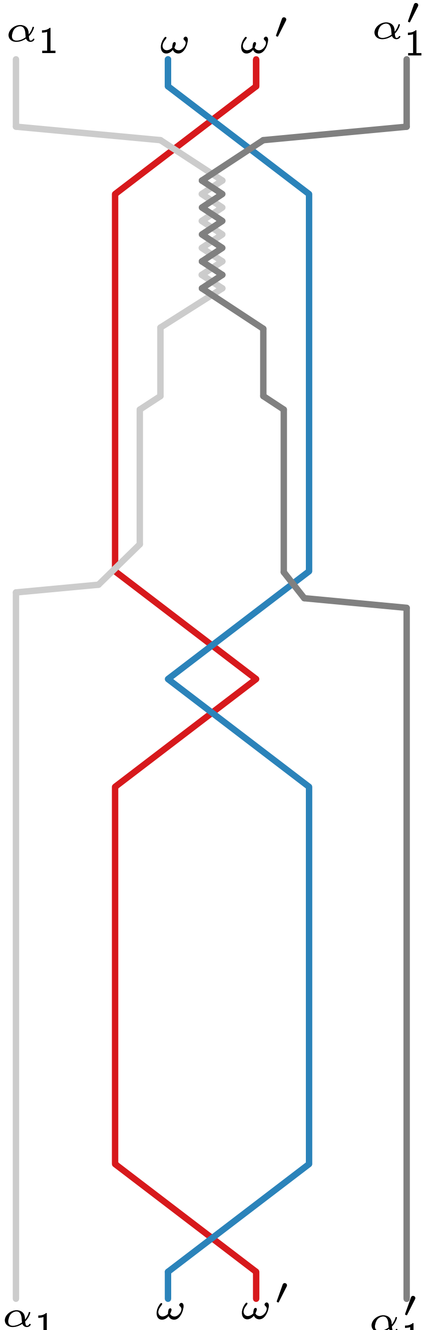


Constructing the list L

$2m$ swaps

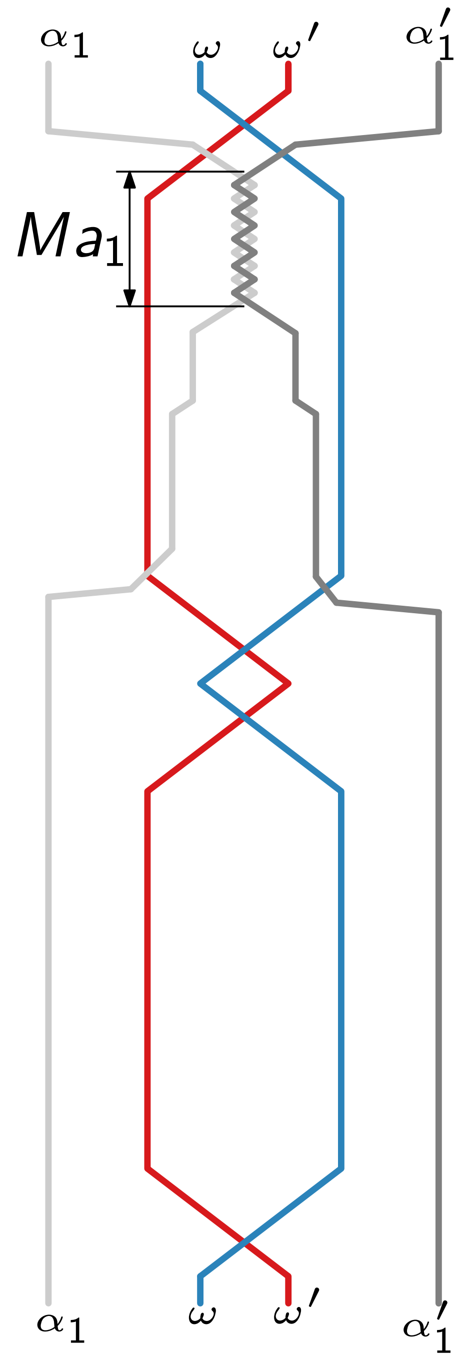


Constructing the list L



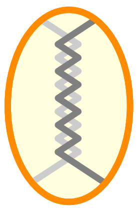
Constructing the list L

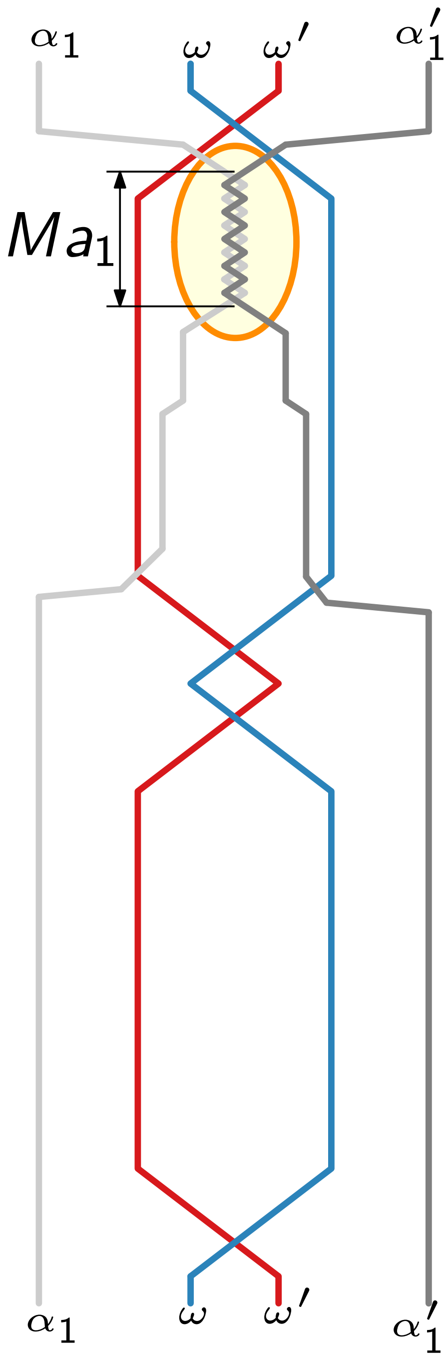
$$M = 2m^3$$



Constructing the list L

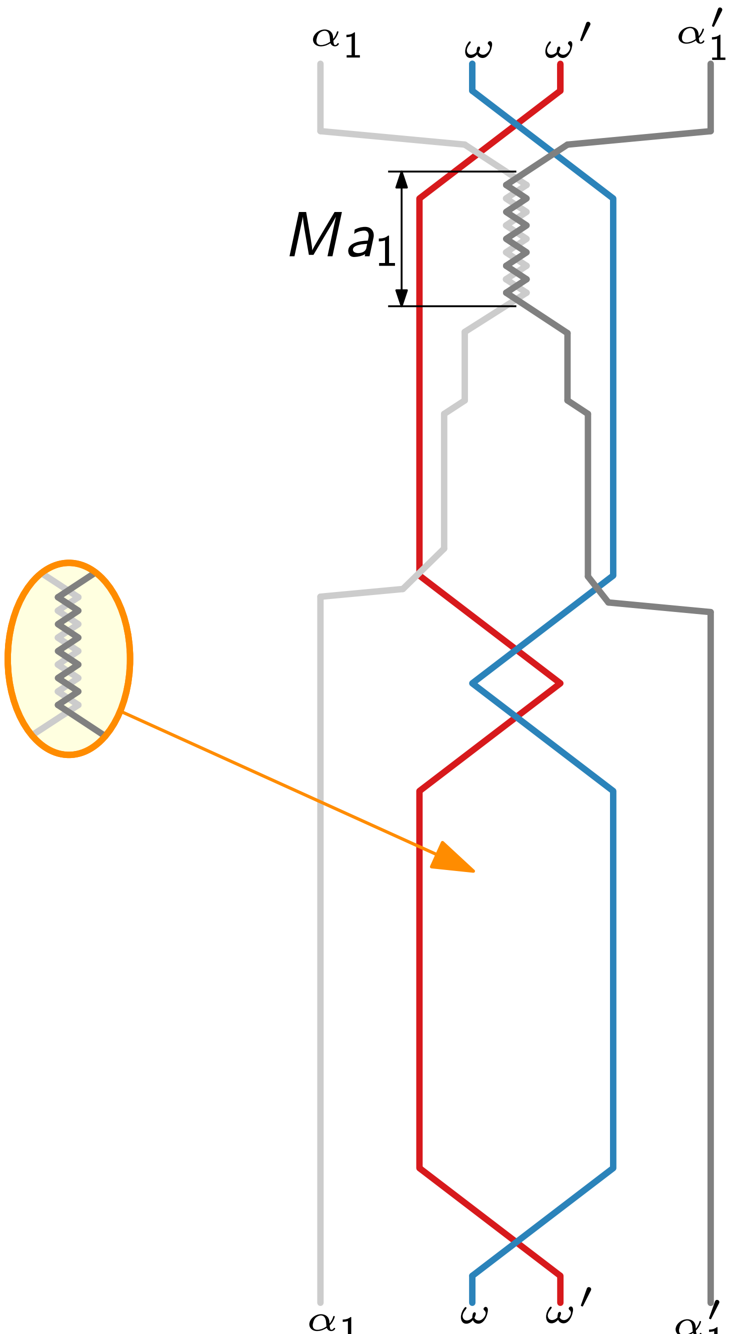
$$M = 2m^3$$





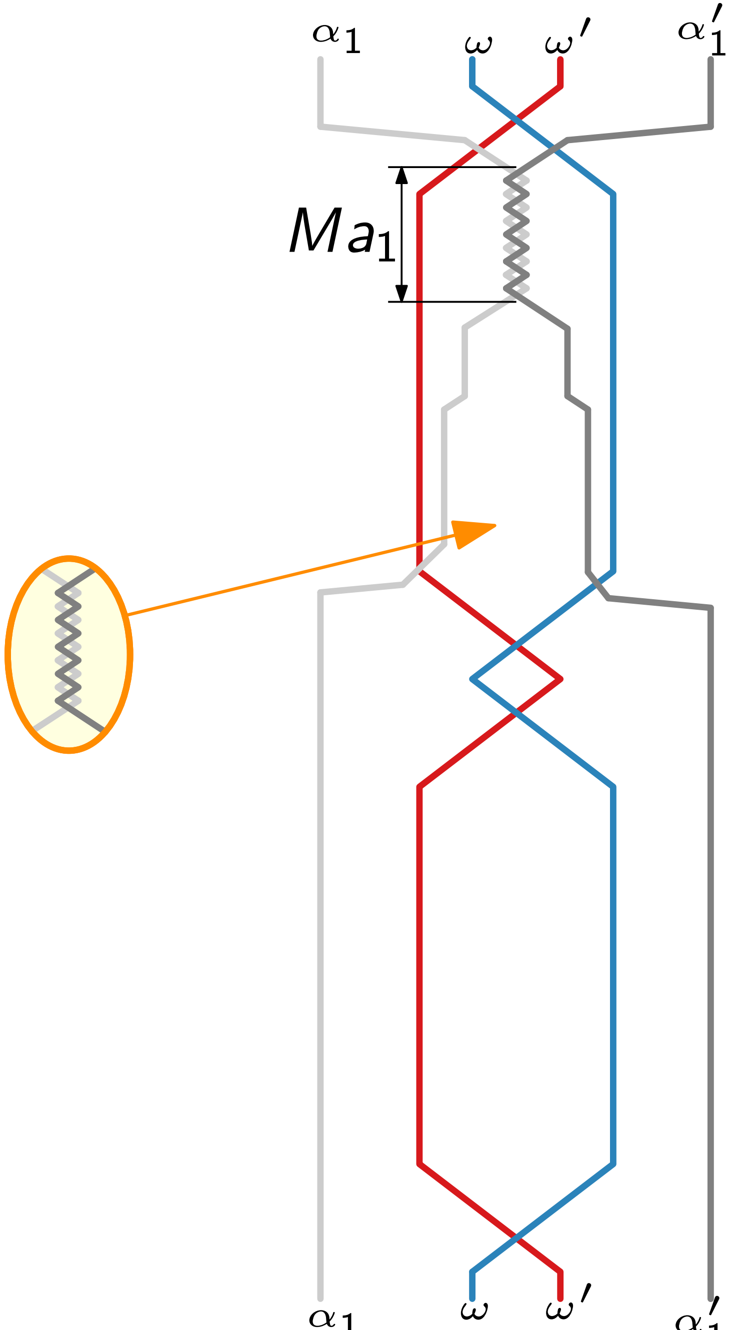
Constructing the list L

$$M = 2m^3$$



Constructing the list L

$$M = 2m^3$$

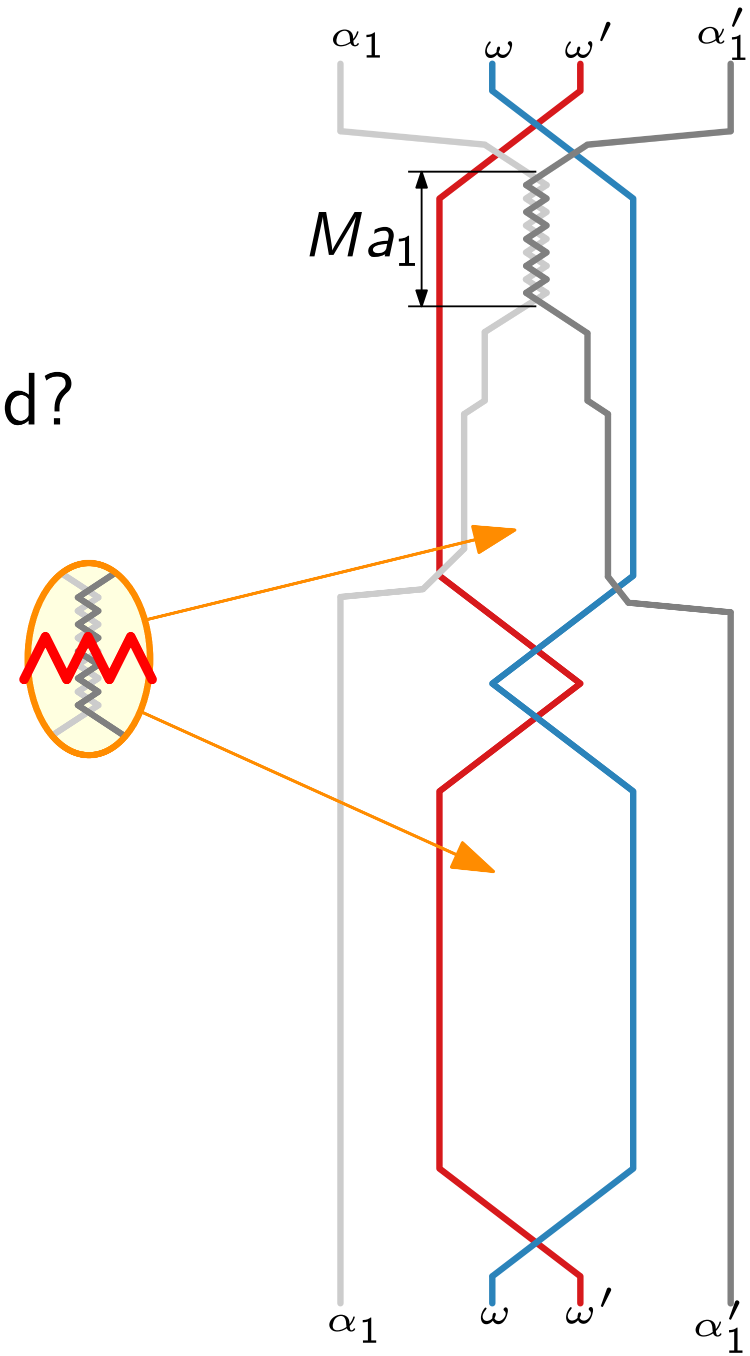


Constructing the list L

$$M = 2m^3$$

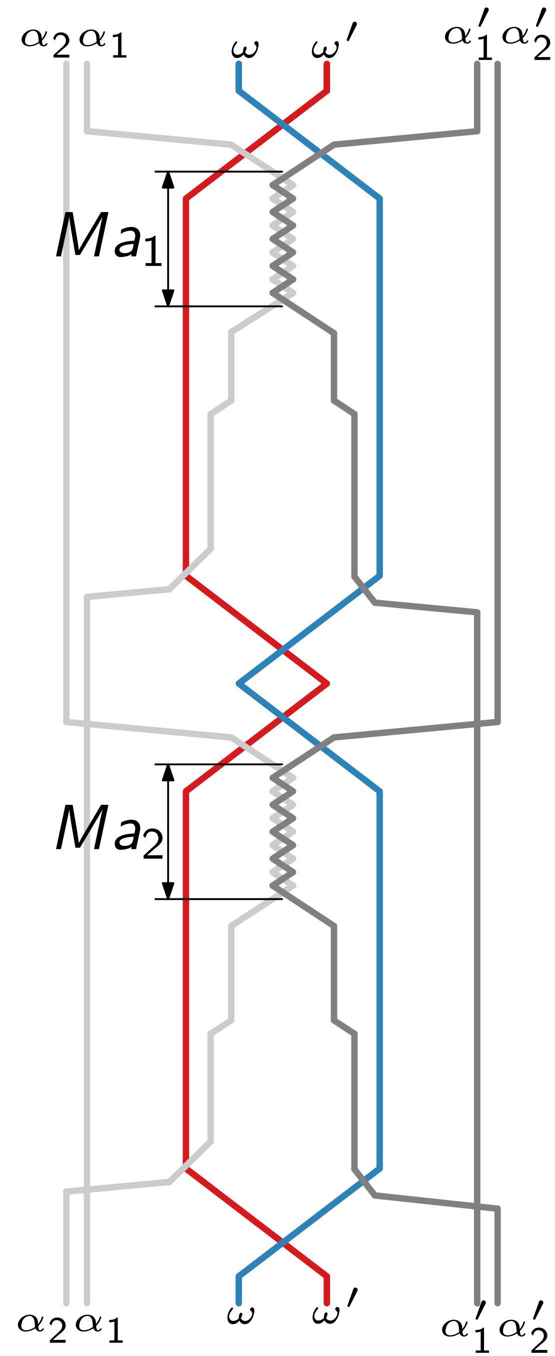
What is not allowed?

split



Constructing the list L

$$M = 2m^3$$

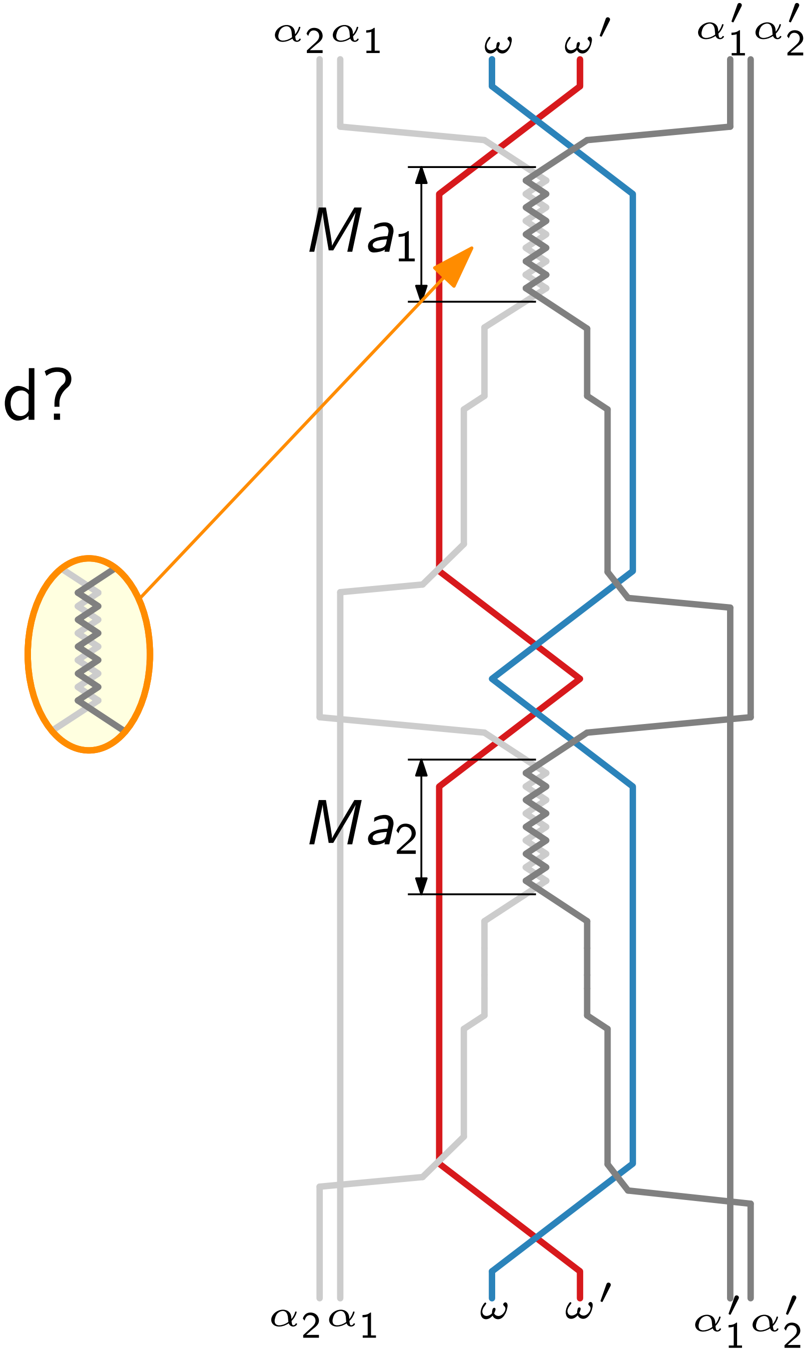


Constructing the list L

$$M = 2m^3$$

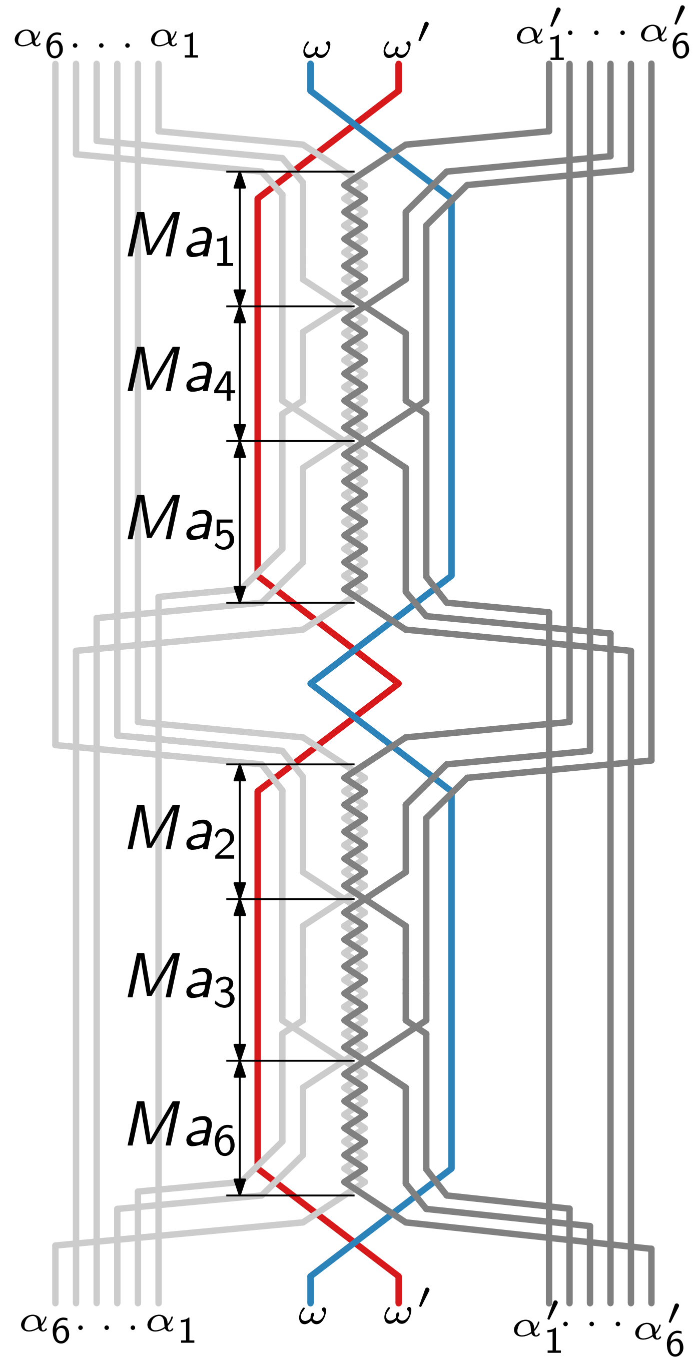
What is not allowed?

put it on the same level
with other α-α' swaps

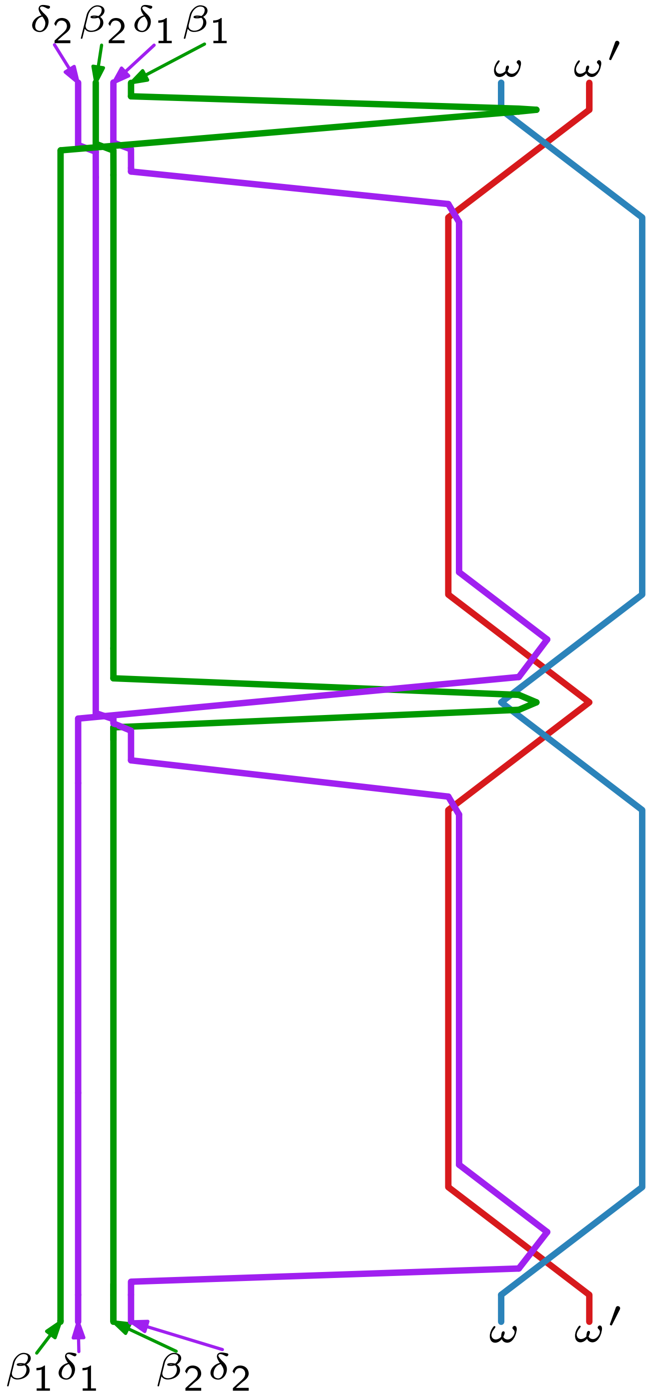


Constructing the list L

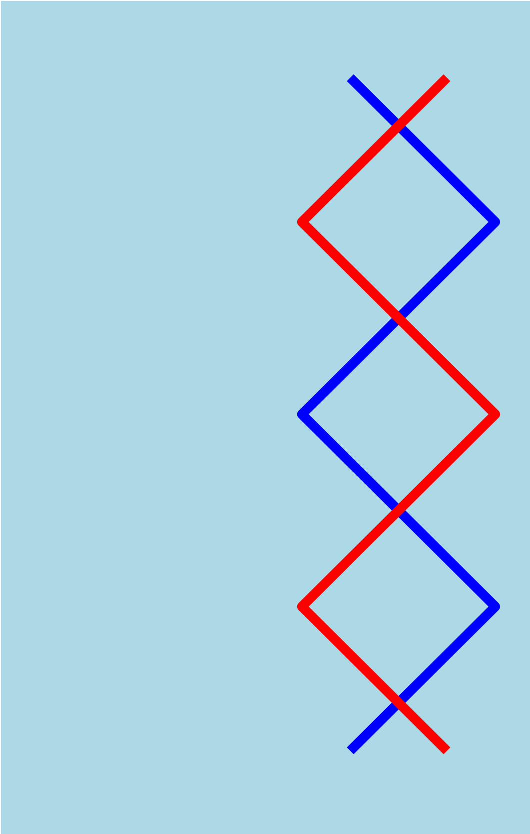
$$M = 2m^3$$

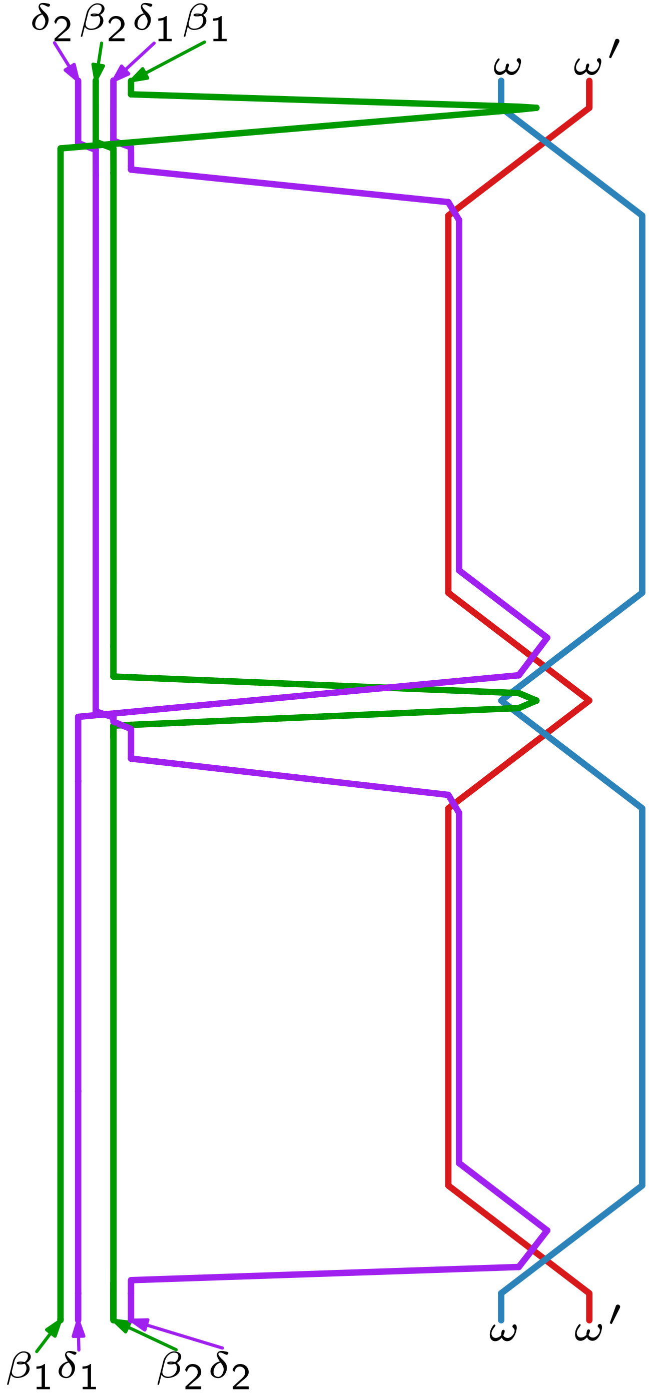


Constructing the list L

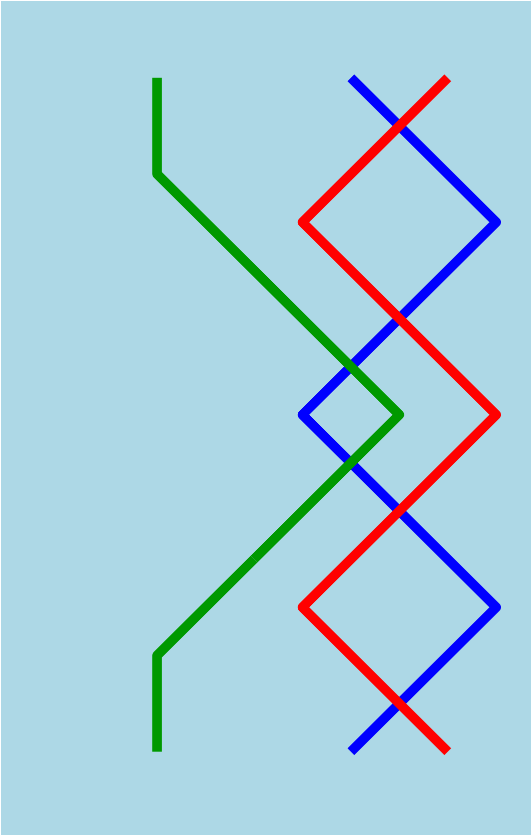


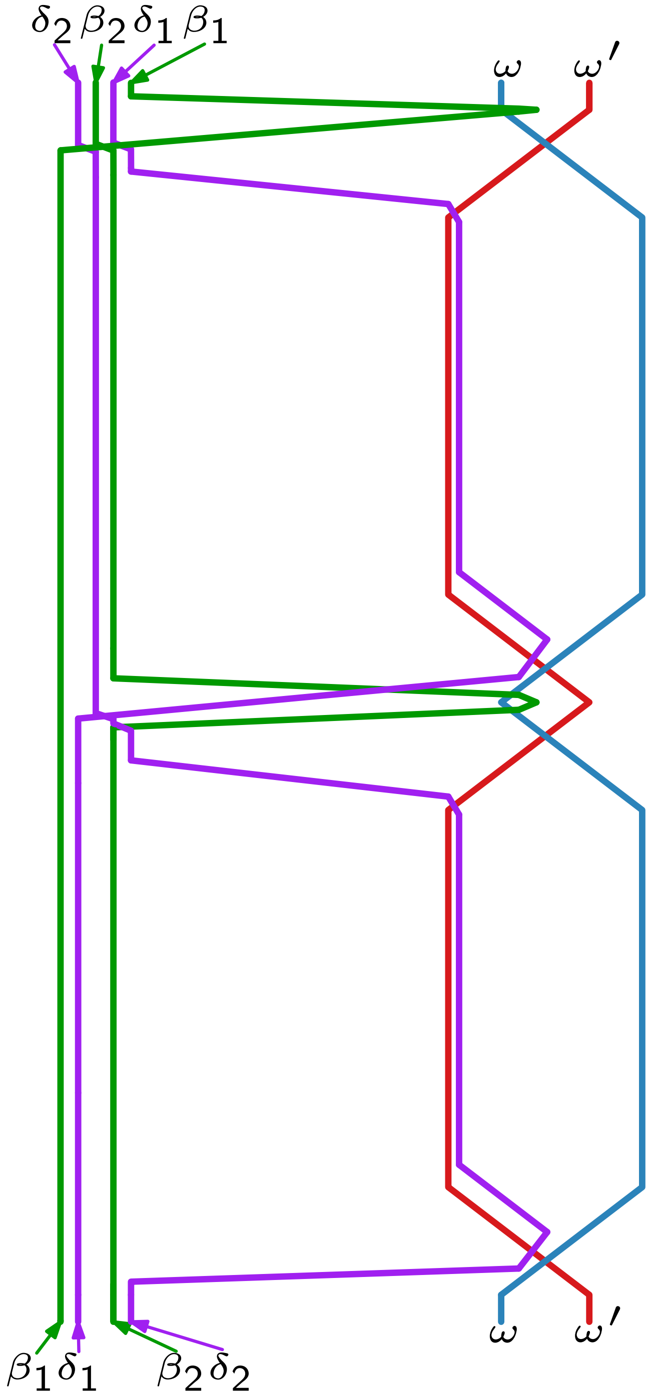
Constructing the list L



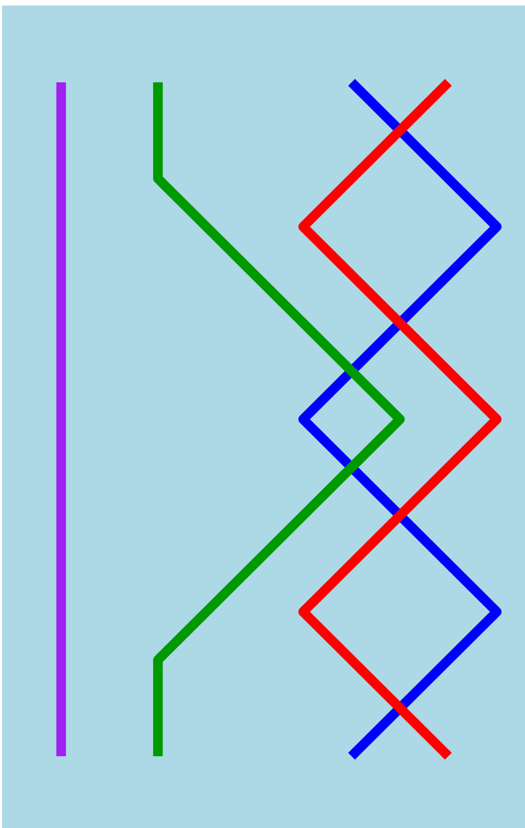


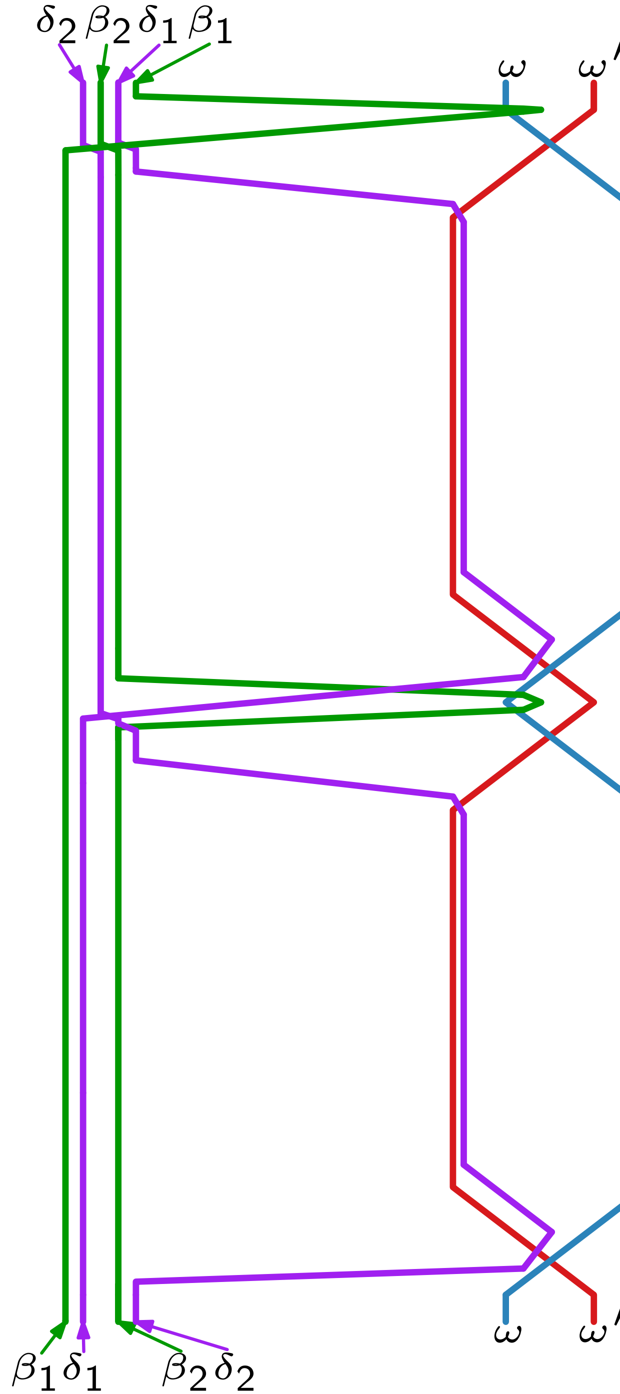
Constructing the list L



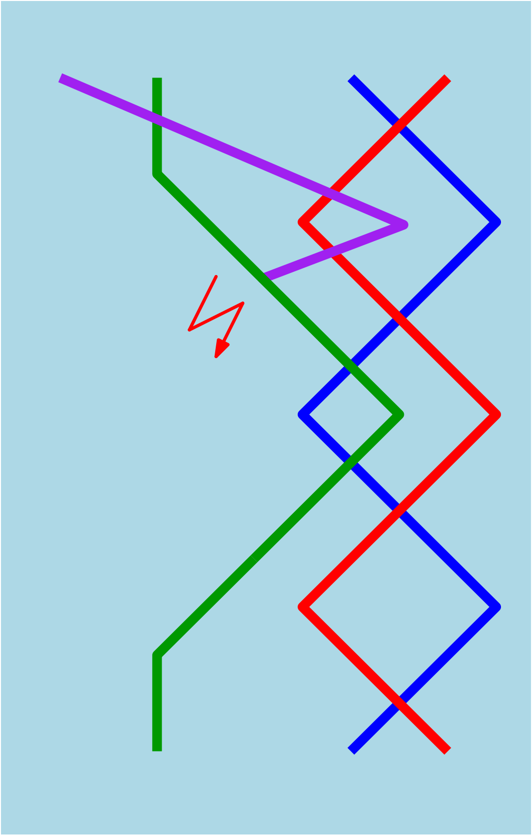


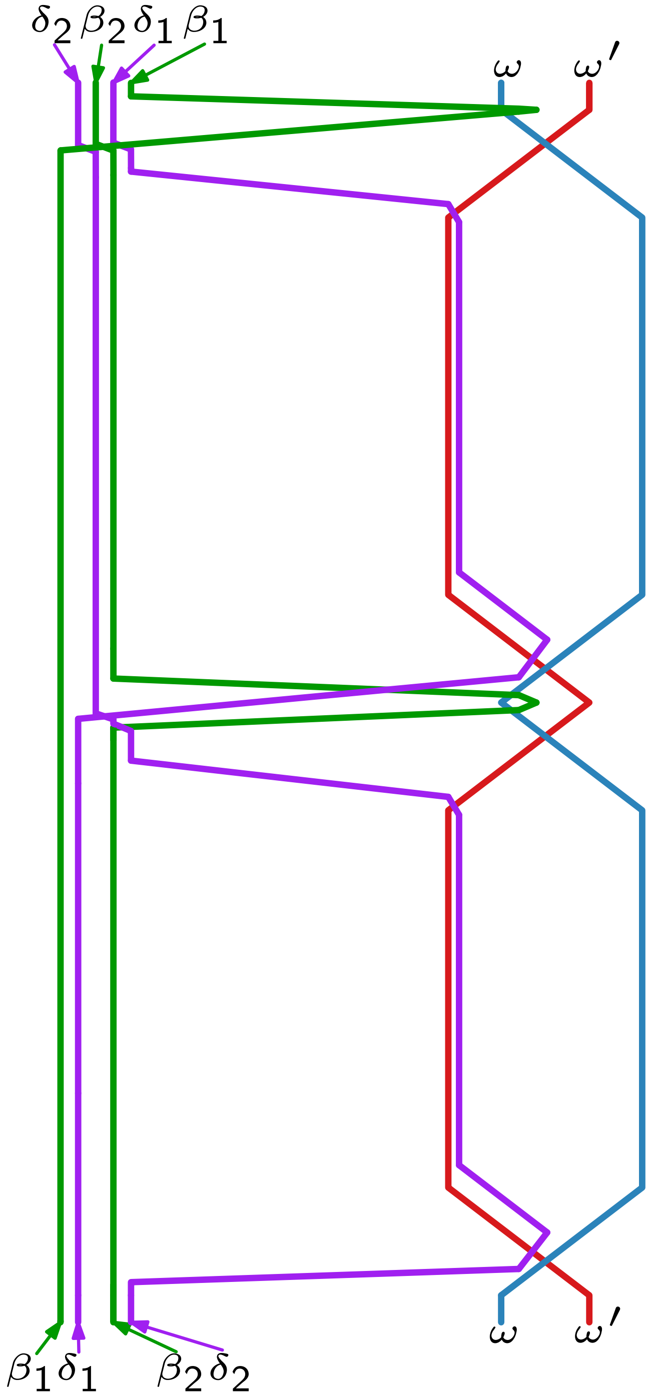
Constructing the list L



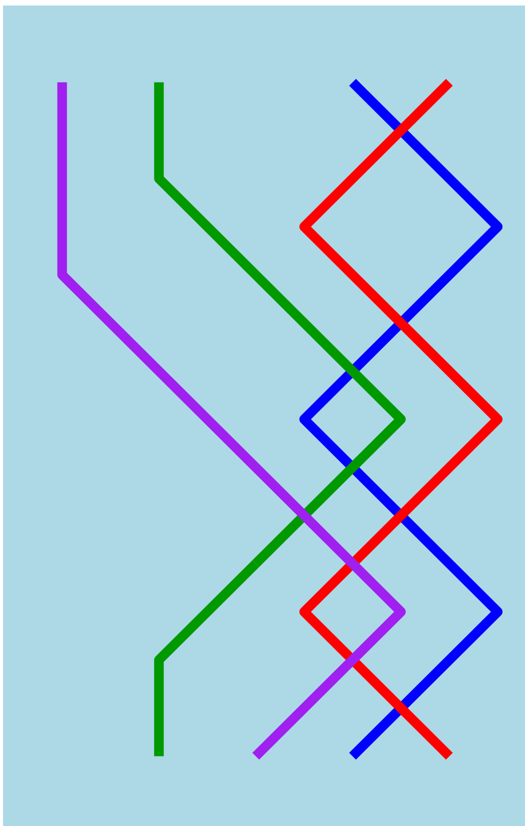


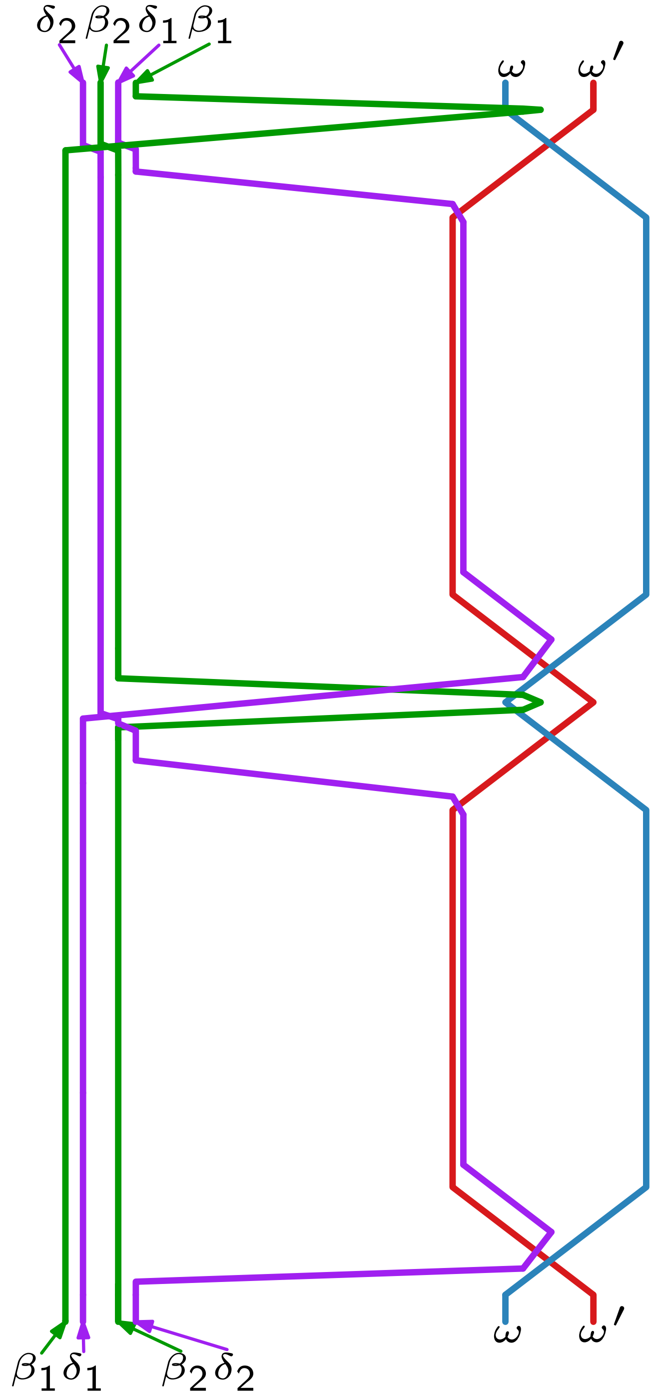
Constructing the list L



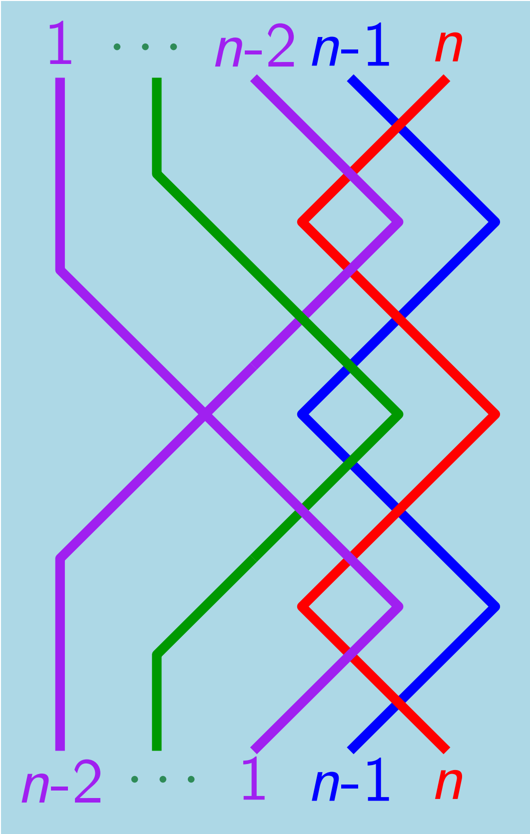


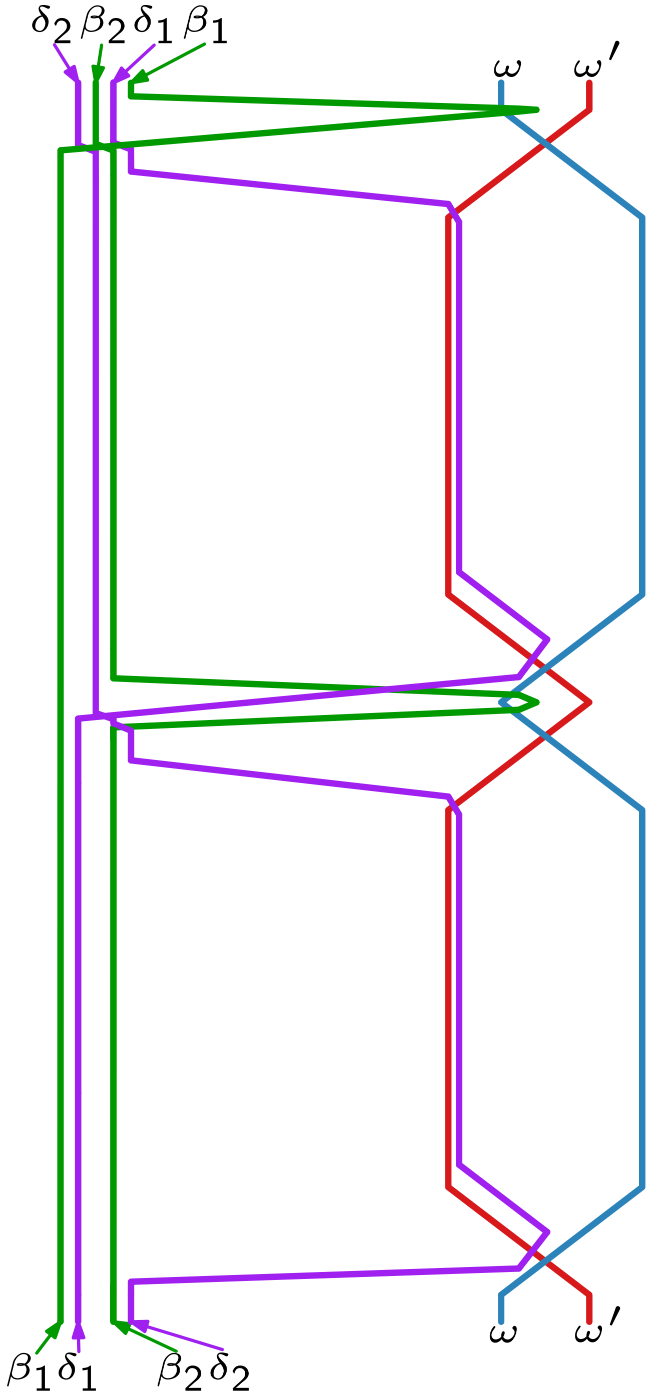
Constructing the list L





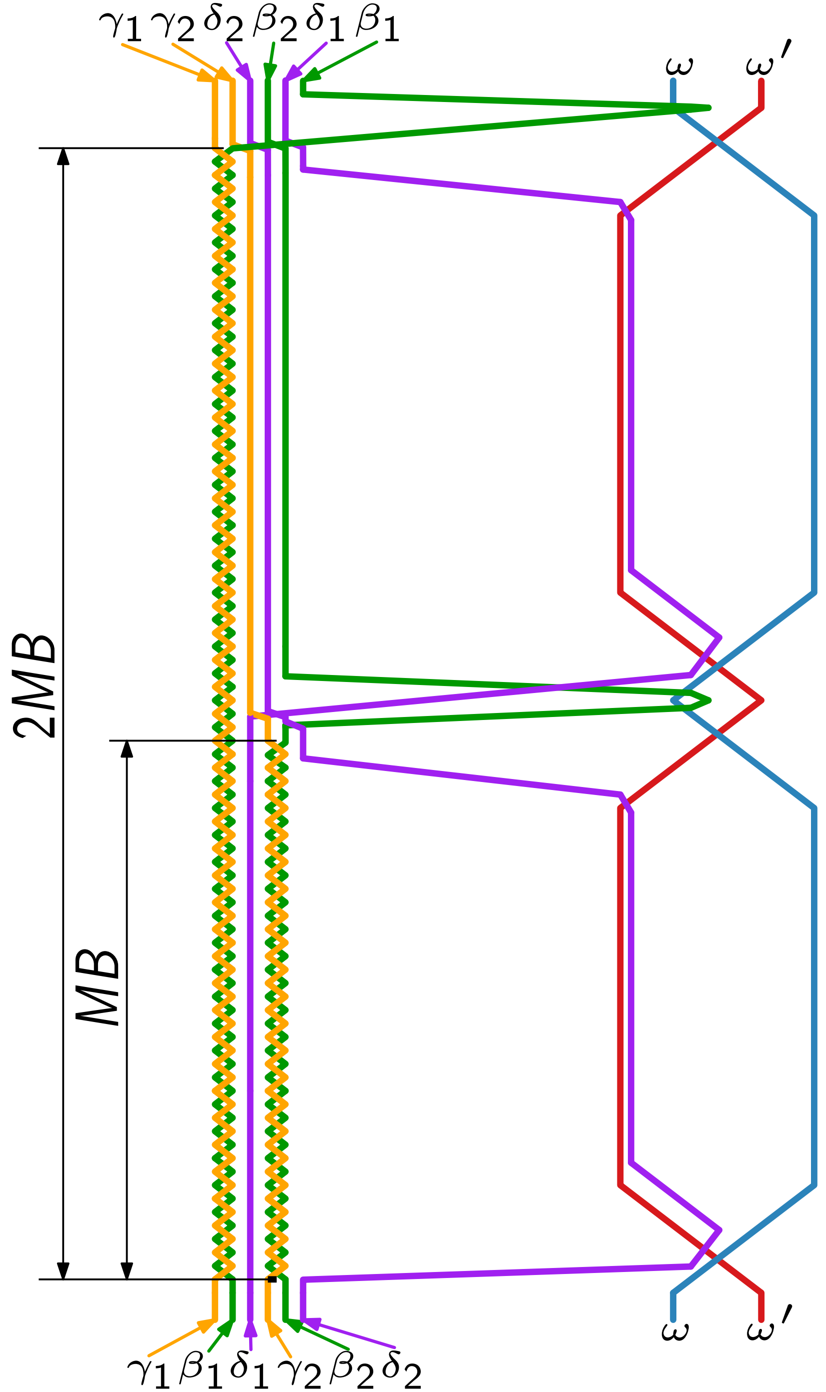
Constructing the list L





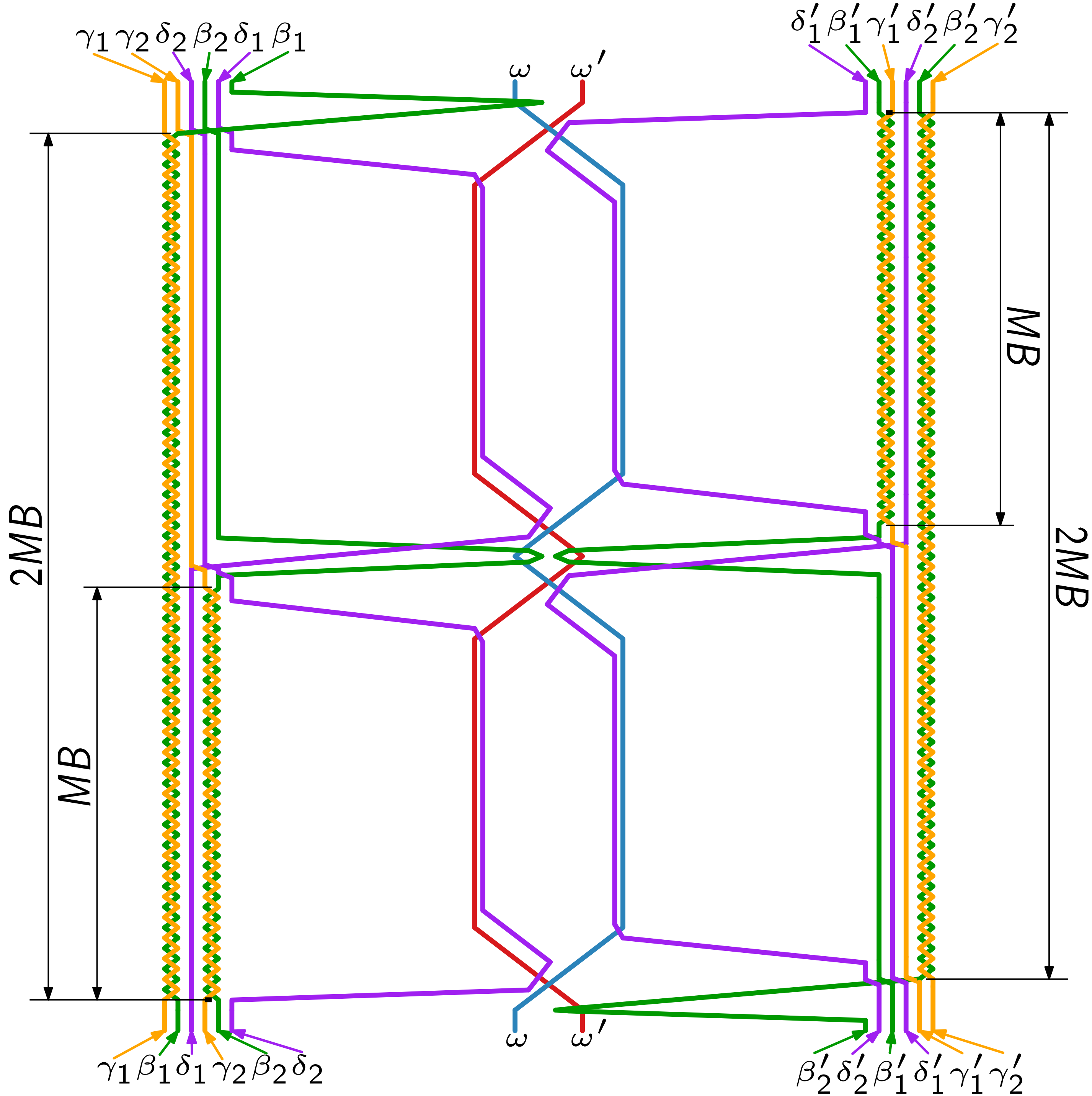
Constructing the list L

$$M = 2m^3$$



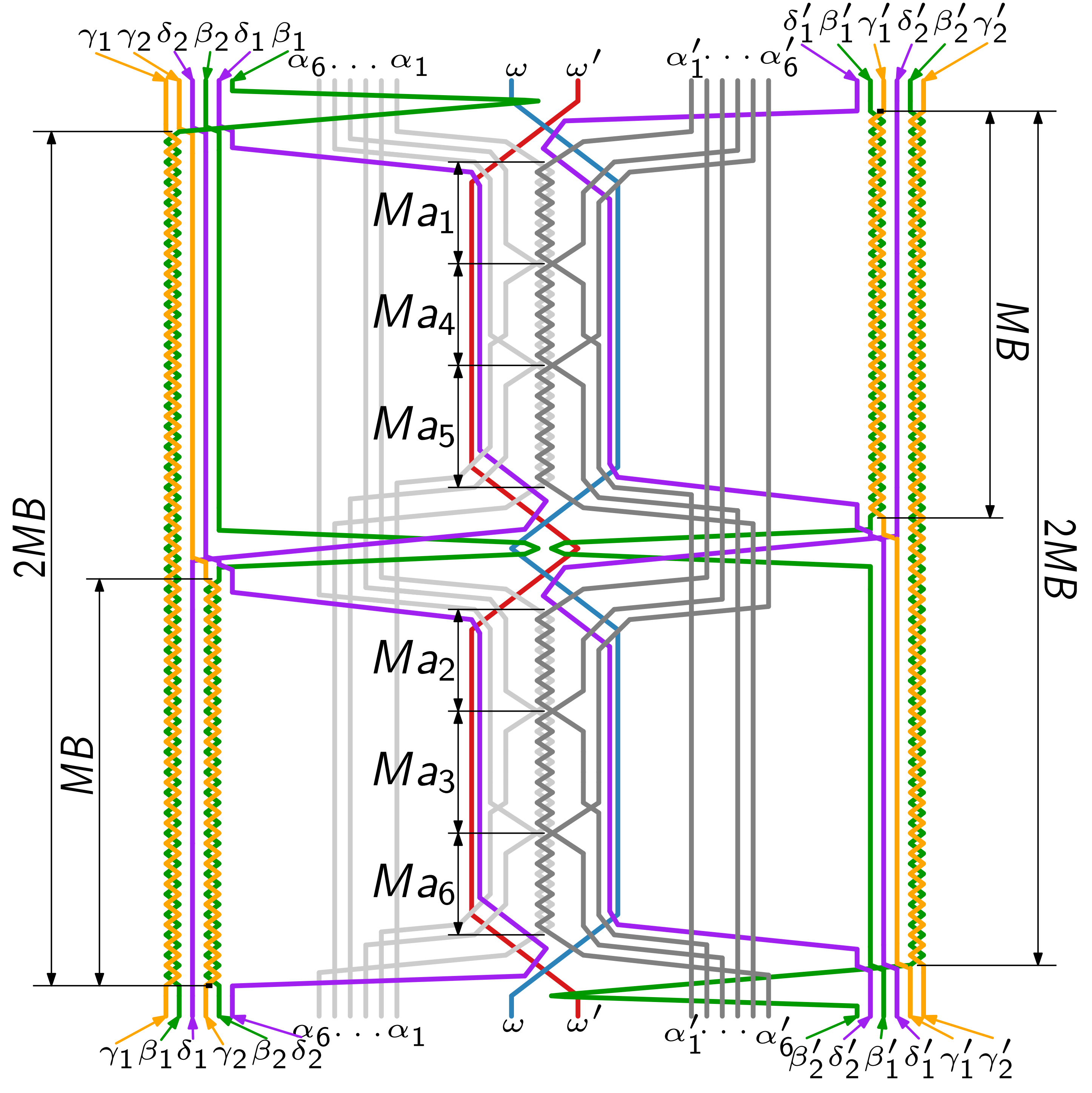
Constructing the list L

$$M = 2m^3$$



Constructing the list L

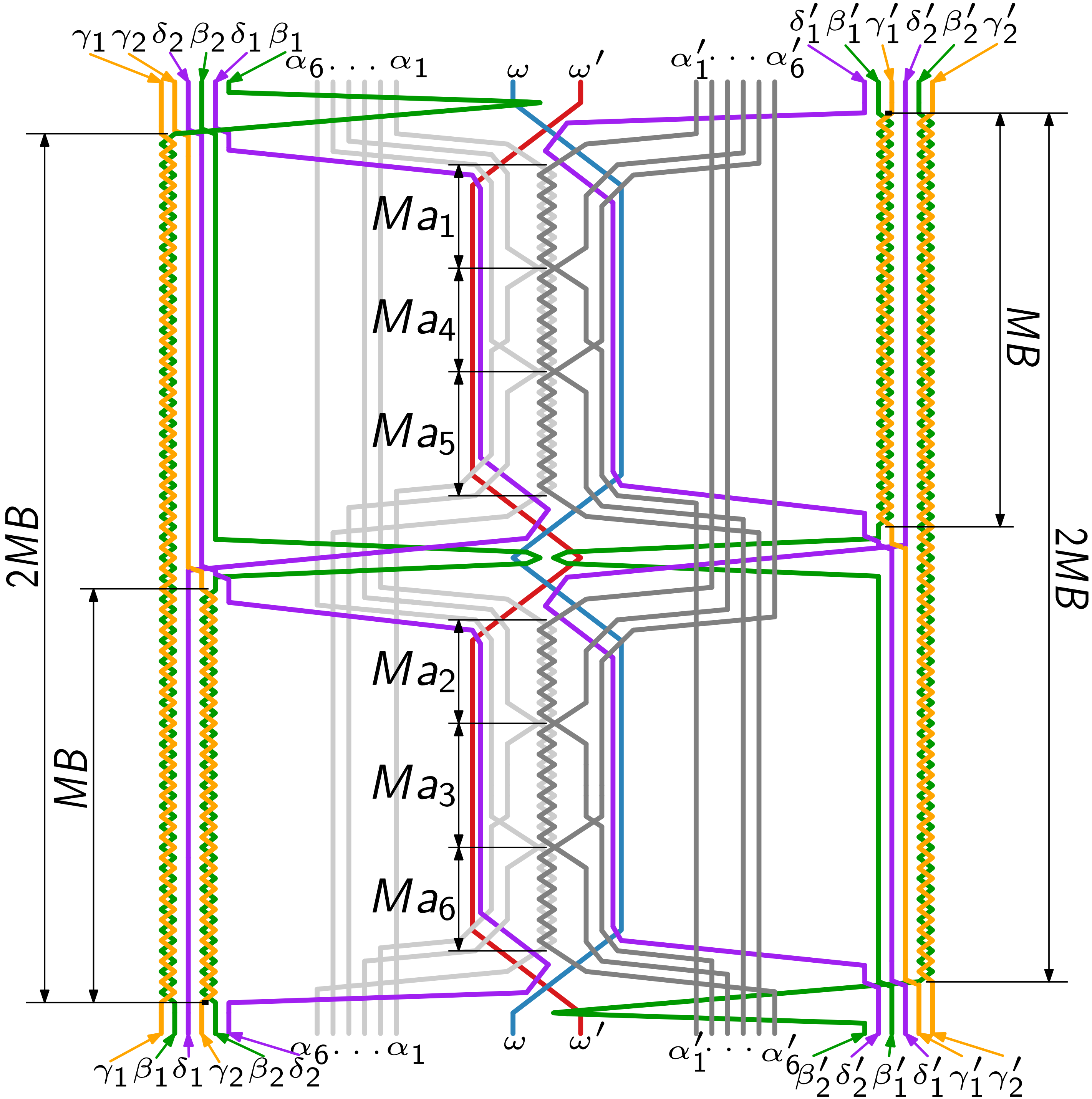
$$M = 2m^3$$



Proof of correctness

$$M = 2m^3$$

A is a yes-instance

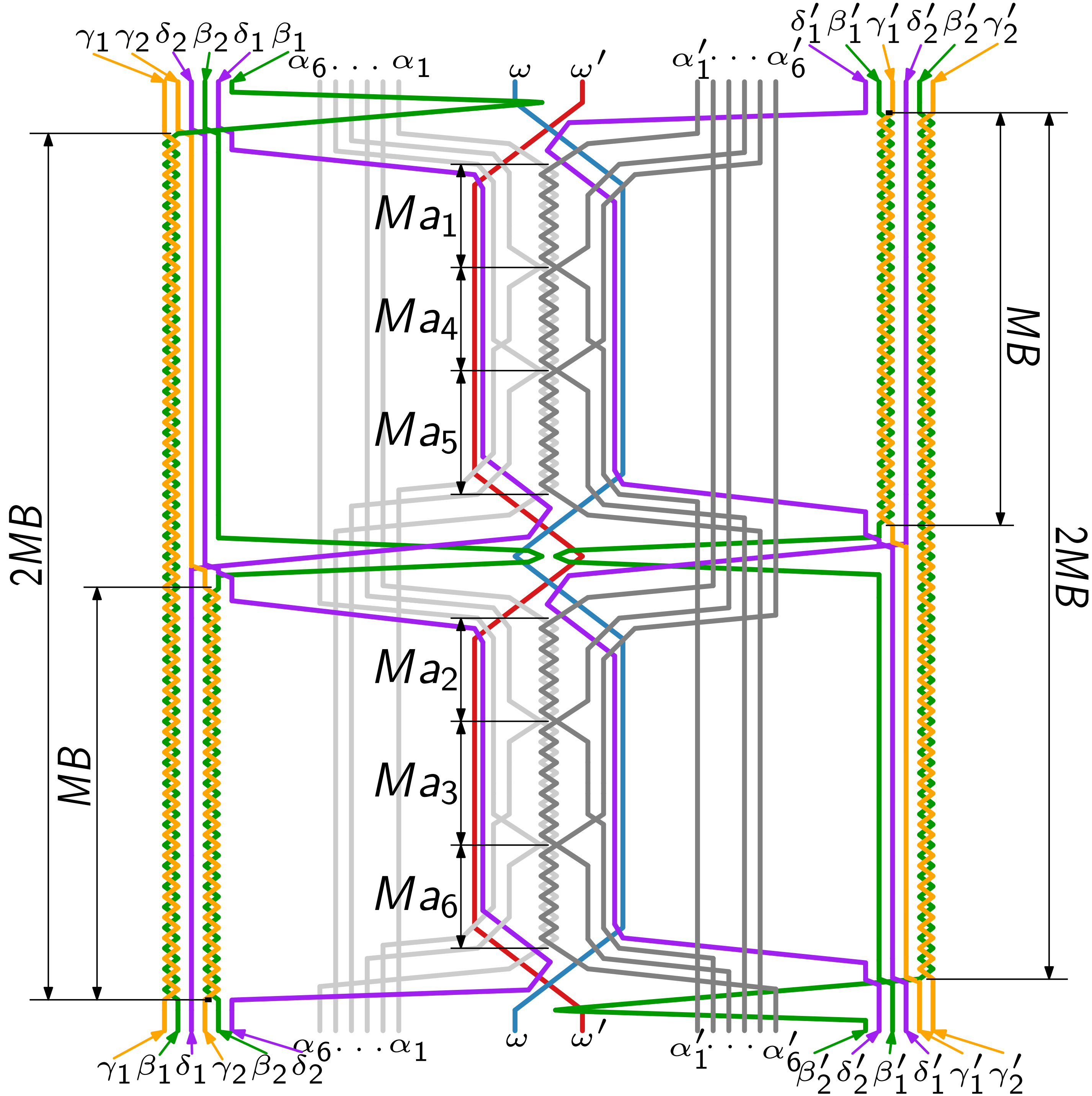


Proof of correctness

$M = 2m^3$

A is a yes-instance

by construction



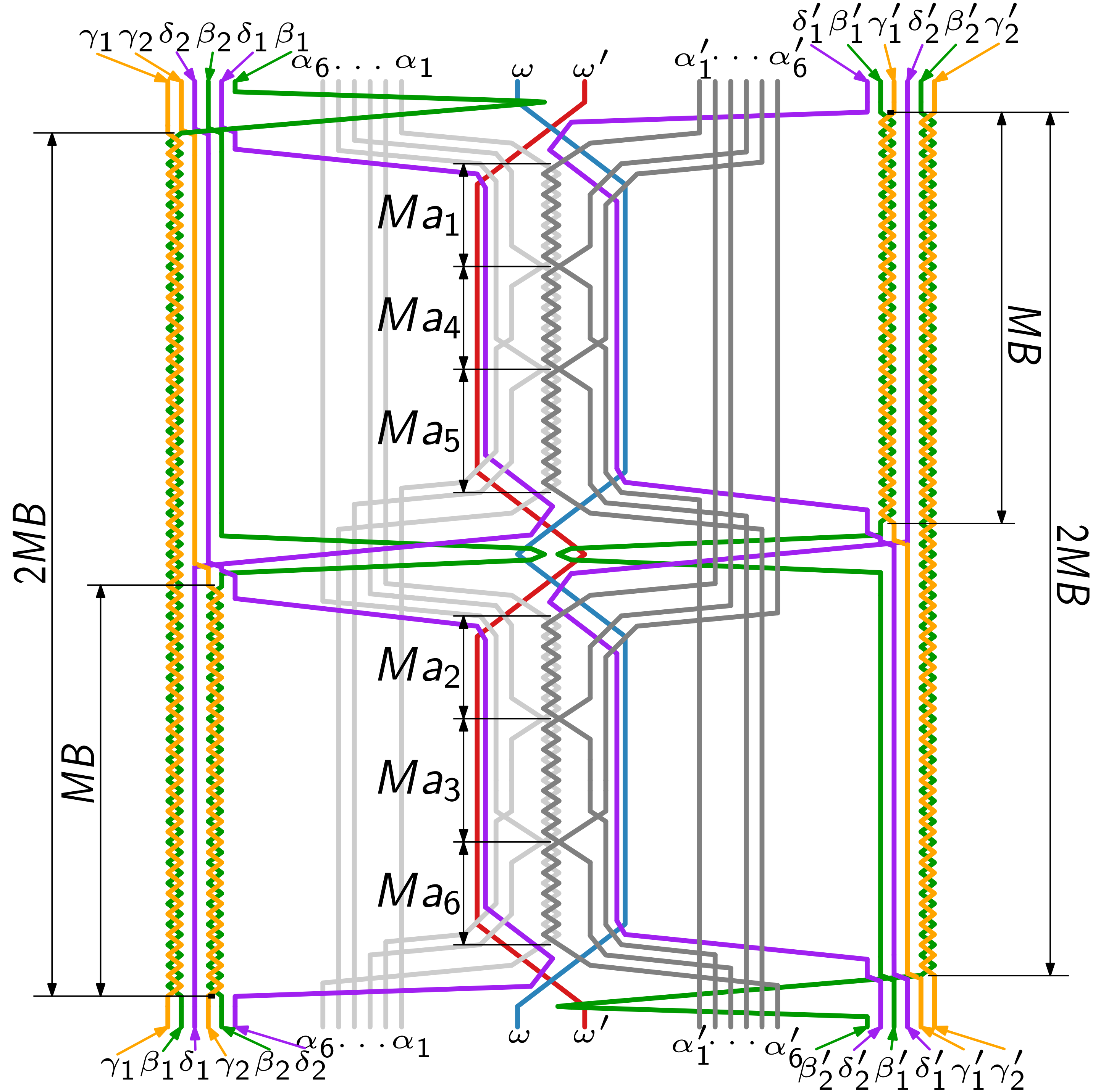
Proof of correctness

$$M = 2m^3$$

A is a yes-instance

by construction

at most H

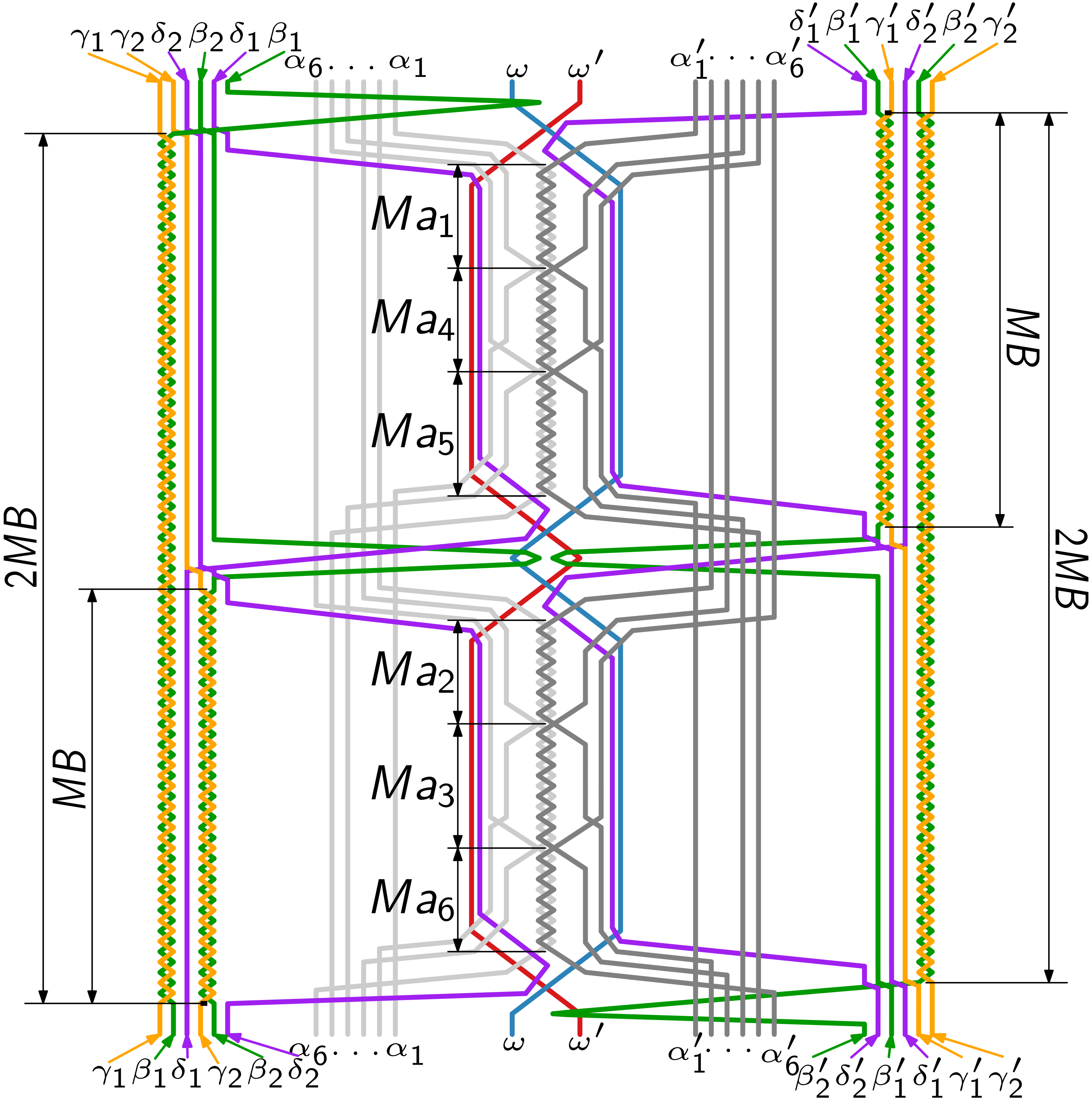


$H = 2m^3(\sum A) + 7m^2$ is the maximum allowed height for the reduction

Proof of correctness

$$M = 2m^3$$

A is a no-instance



$H = 2m^3(\sum A) + 7m^2$ is the maximum allowed height for the reduction

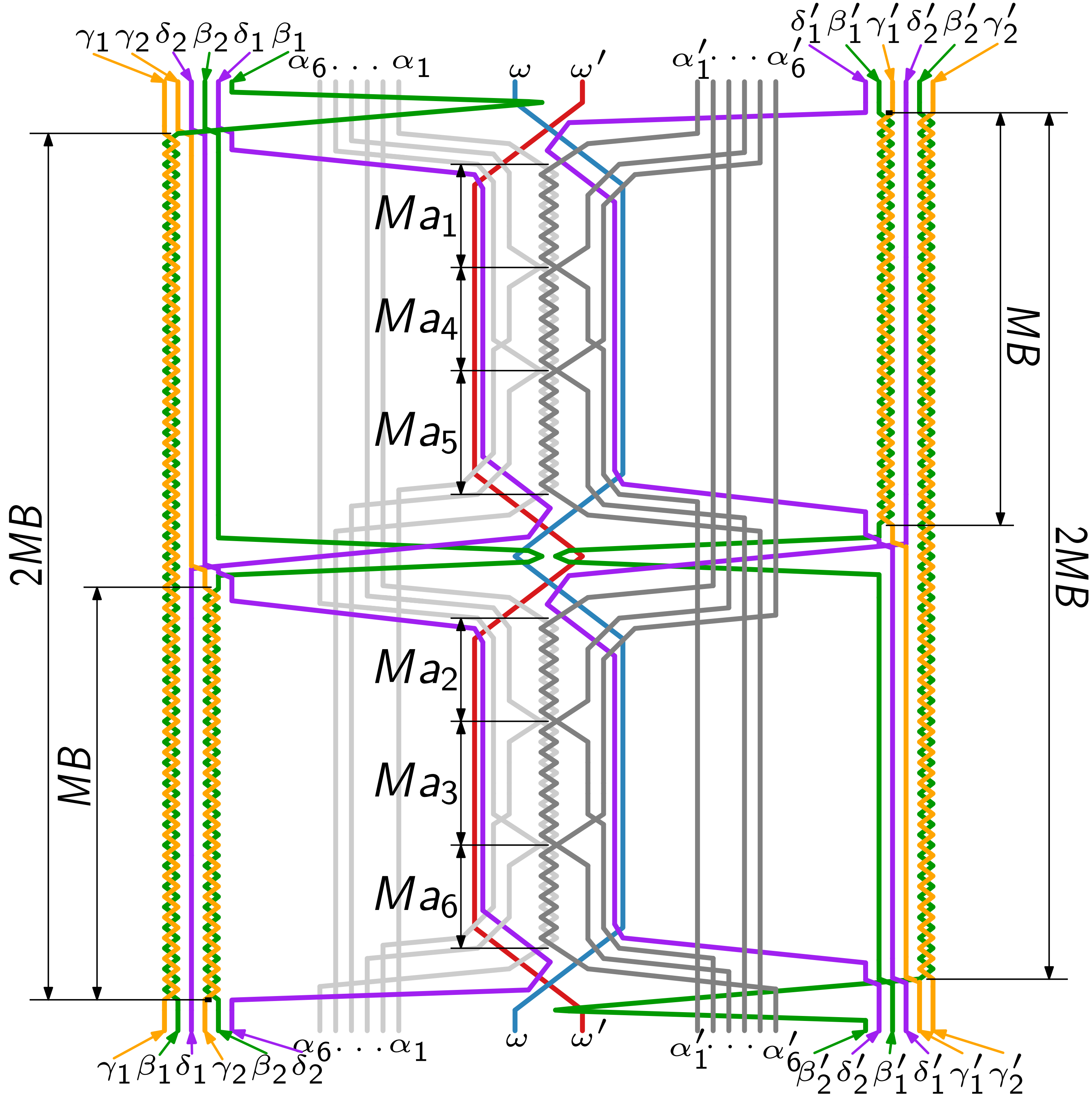
Proof of correctness

$$M = 2m^3$$

A is a no-instance

minimum height
$2m^3(\sum A + 1)$

$H = 2m^3(\sum A) + 7m^2$
is the maximum allowed height for the reduction



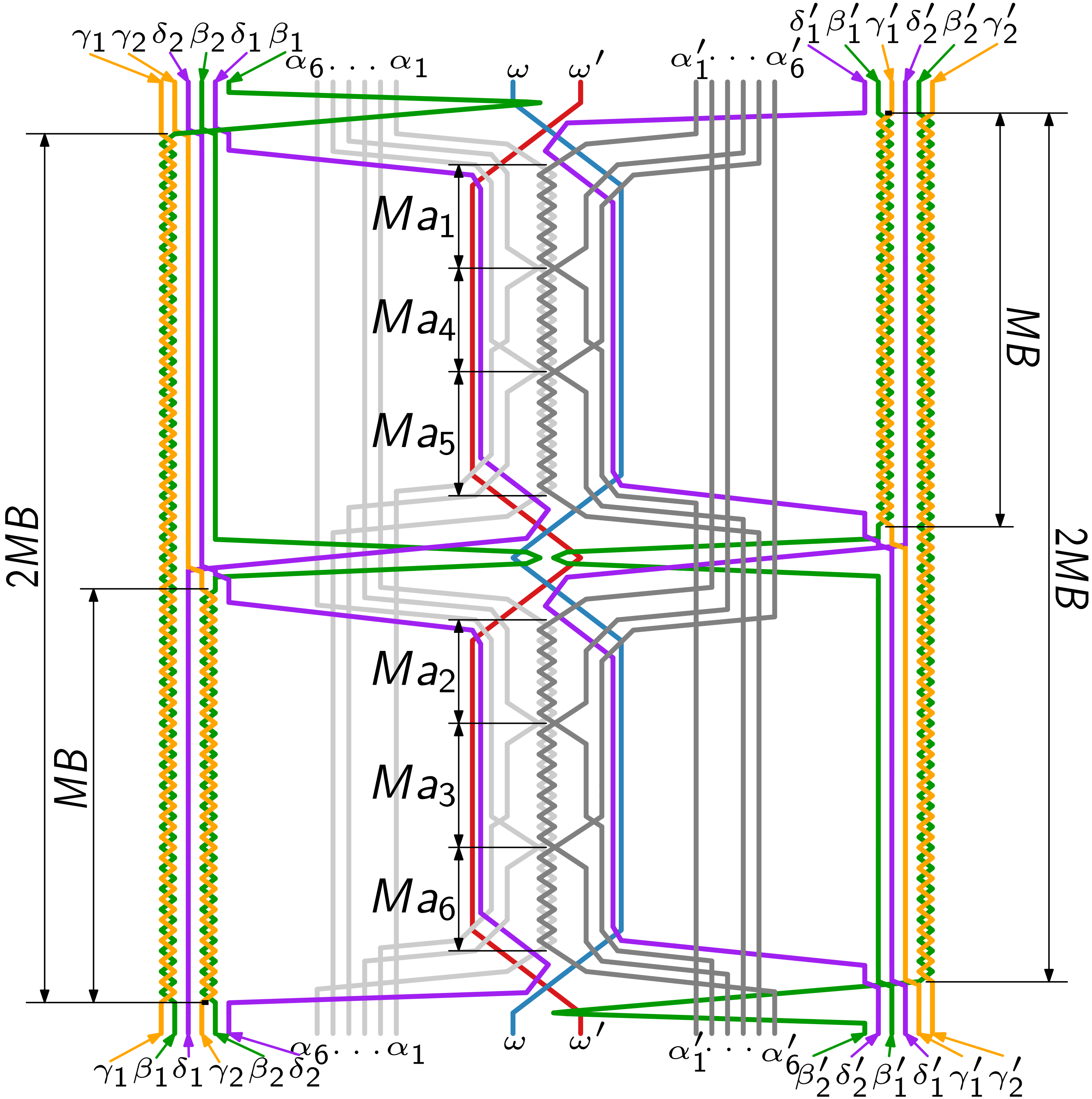
Proof of correctness

$M = 2m^3$

A is a no-instance

minimum height $2m^3(\sum A+1)$

bigger than H



$H = 2m^3(\sum A) + 7m^2$ is the maximum allowed height for the reduction

Improving of Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in . . .

Simple lists

General lists

Improving of Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in ...

n: the number of wires

Simple lists

[Olszewski et al., GD'18]

$e^{O(n^2)}$

General lists

Improving of Exact Algorithms

Tangle-Height Minimization can be solved in . . .

n: the number of wires

Simple lists

[Olszewski et al., GD'18]

$e^{O(n^2)}$

$e^{O(n \log n)}$

our result

General lists

Improving of Exact Algorithms

Tangle-Height Minimization can be solved in ...

n: the number of wires
$|L|$: the *length* of the list, i.e $\sum \ell_{ij}$
φ: the golden ratio, i.e. ≈ 1.618

Simple lists

[Olszewski et al., GD'18]

$e^{O(n^2)}$

$e^{O(n \log n)}$

our result

General lists

[Olszewski et al., GD'18]

$O\left(\frac{\varphi^{2|L|}}{5^{|L|/n}} n\right)$

Improving of Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in . . .

n:	the number of wires
$\lvert L\rvert$:	the *length* of the list, i.e $\sum \ell_{ij}$
φ:	the golden ratio, i.e. ≈ 1.618

Simple lists

[Olszewski et al., GD'18]

$e^{O(n^2)}$

$e^{O(n \log n)}$

our result

General lists

[Olszewski et al., GD'18]

$O\left(\frac{\varphi^{2|L|}}{5^{|L|/n}} n\right)$

our result

$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}} \varphi^n n\right)$

Improving of Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in ...

n: the number of wires
$|L|$: the *length* of the list, i.e $\sum \ell_{ij}$
φ: the golden ratio, i.e. ≈ 1.618

Simple lists

[Olszewski et al., GD'18]

$e^{O(n^2)}$

our result

$e^{O(n \log n)}$

General lists

[Olszewski et al., GD'18]

$O\left(\frac{\varphi^{2|L|}}{5^{|L|/n}} n\right)$

our result

$O\left(\left(\frac{2|L|}{n^2} + 1\right)^{\frac{n^2}{2}} \varphi^n n\right)$

polynomial in $|L|$
for fixed n

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n \boldsymbol{n}\right)$$

Given a list $L = (\ell_{ij})$.

Dynamic Programming Algorithm

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n \boldsymbol{n}\right)$$

L' is a *sublist* of L if $\ell'_{ij} \leq \ell_{ij}$

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n \boldsymbol{n}\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Dynamic Programming Algorithm

$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Dynamic Programming Algorithm

$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

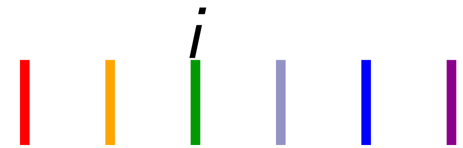
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



Dynamic Programming Algorithm

$$O\Big(\big(\tfrac{2|L|}{n^2}+1\big)^{\frac{n^2}{2}}\varphi^n n\Big)$$

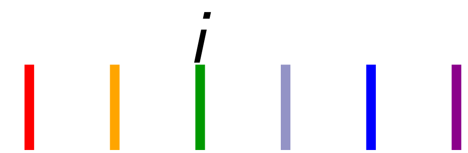
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



for each wire i:

    // find a position where it is after applying L'

Dynamic Programming Algorithm

$O\Big(\big(\frac{2|L|}{n^2}+1\big)^{\frac{n^2}{2}}\varphi^n n\Big)$

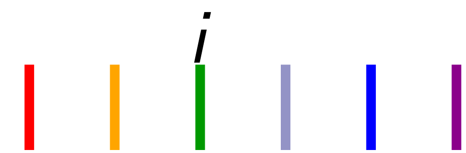
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



for each wire i:

    // find a position where it is after applying L'

    $i \mapsto i +$

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

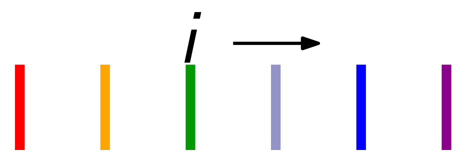
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



for each wire i:

// find a position where it is after applying L'

$i \mapsto i +$

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

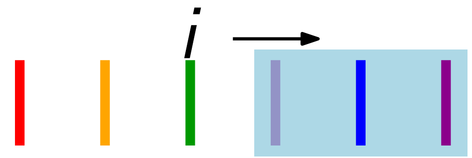
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



for each wire i:

  // find a position where it is after applying L'

  $i \mapsto i + |\{j : i < j \text{ and } l_{ij} \text{ is odd}\}|$

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

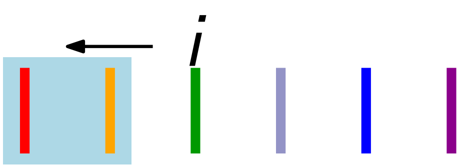
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



for each wire i:

  // find a position where it is after applying L'

  $i \mapsto i + |\{j : i < j \text{ and } l_{ij} \text{ is odd}\}| - |\{j : j < i \text{ and } l_{ij} \text{ is odd}\}|$

Dynamic Programming Algorithm

$$O\Big(\big(\tfrac{2|L|}{n^2} + 1\big)^{\frac{n^2}{2}} \varphi^n n \Big)$$

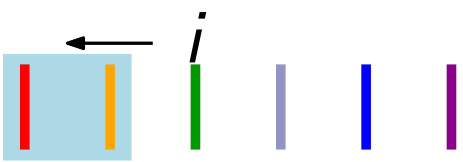
Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.



for each wire i:

    // find a position where it is after applying L'

    $i \mapsto i + |\{j : i < j \text{ and } l_{ij} \text{ is odd}\}| - |\{j : j < i \text{ and } l_{ij} \text{ is odd}\}|$

check whether the result is indeed a permutation

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2} + 1 \right)^{\frac{n^2}{2}} \varphi^n n \right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n\, L'$.

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

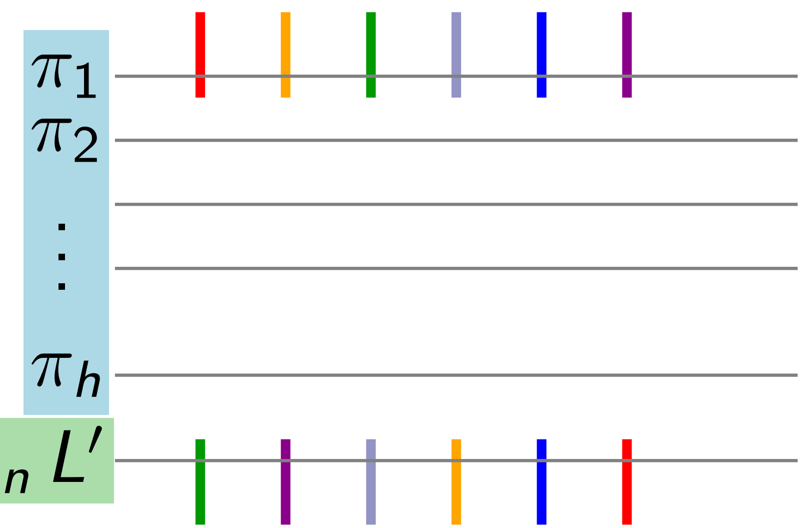
Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n L'$.

Choose the **shortest tangle** $T(L'')$



π_h and $\mathrm{id}_n L'$ are adjacent

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

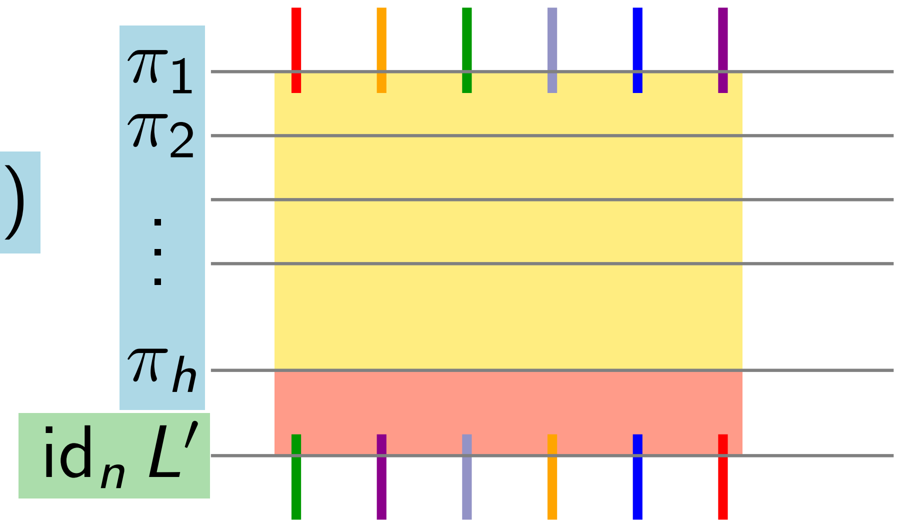
Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n\, L'$.

Choose the **shortest tangle** $T(L'')$



π_h and $\mathrm{id}_n\, L'$ are adjacent

L'' + add. swaps $= L'$

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

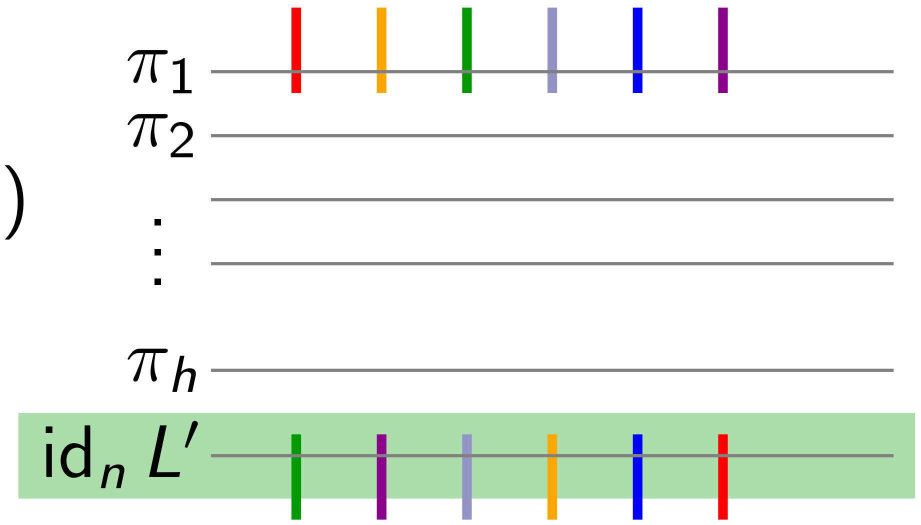
Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n L'$.

Choose the **shortest tangle** $T(L'')$

Add the final permutation to the end.



Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

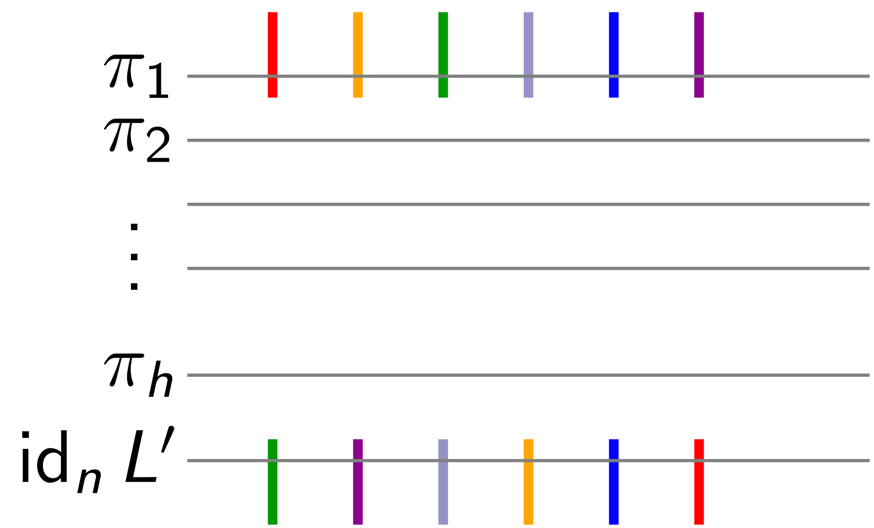
Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n\, L'$.

Choose the **shortest tangle** $T(L'')$

Add the final permutation
to the end.



Running time

$O($

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

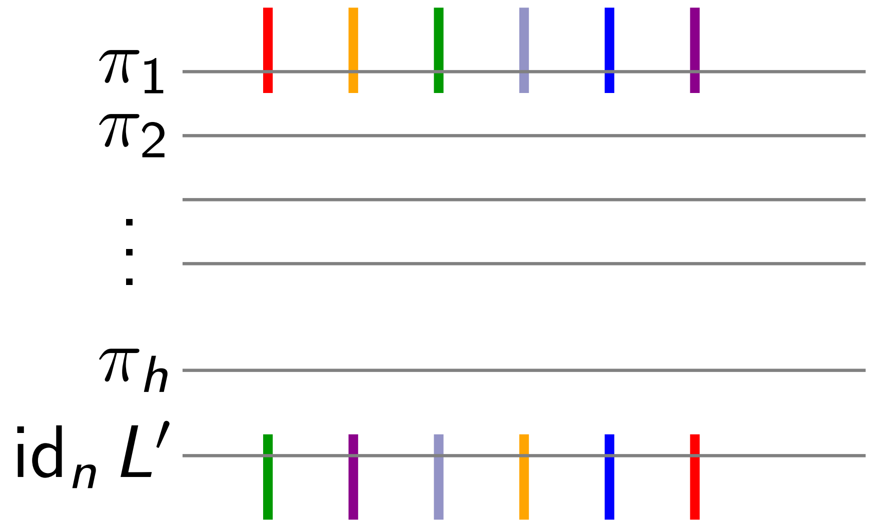
Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n L'$.

Choose the **shortest tangle** $T(L'')$

Add the final permutation to the end.



Running time

$O(\lambda$

Dynamic Programming Algorithm

$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

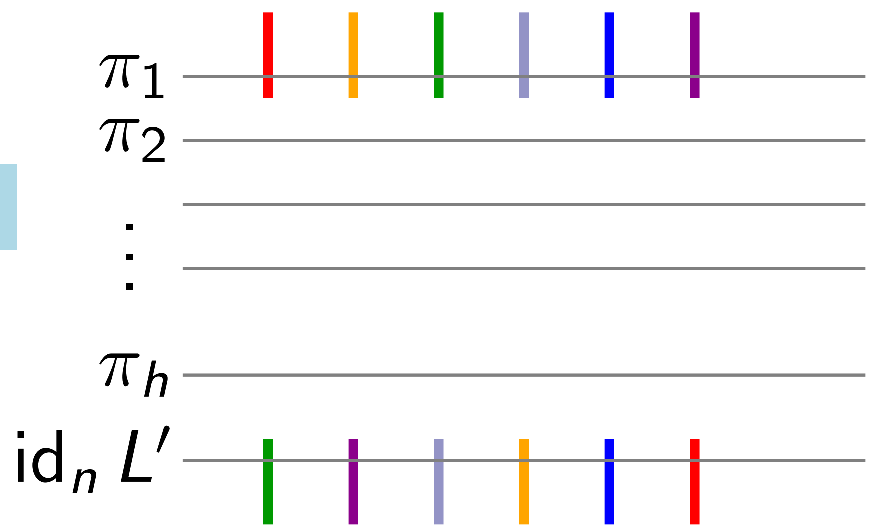
Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n\, L'$.

Choose the **shortest tangle** $T(L'')$

Add the final permutation to the end.



Running time

$O(\lambda(F_{n+1}-1)n)$

F_n is the n-th Fibonacci number

Dynamic Programming Algorithm

$$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n L'$.

Choose the **shortest tangle** $T(L'')$

Add the final permutation to the end.

Running time

$$O(\lambda(F_{n+1}-1)n) \leq \left|\begin{array}{l} \lambda = \prod_{i<j}(\ell_{ij}+1) \leq \left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}} \\ F_n \in O(\varphi^n) \end{array}\right| \leq$$

Dynamic Programming Algorithm

$$O\Big(\big(\tfrac{2|L|}{n^2}+1\big)^{\frac{n^2}{2}}\varphi^n n\Big)$$

Given a list $L = (\ell_{ij})$.

$\lambda = \#$ of **distinct sublists** of L.

Consider them in order of **increasing length**.

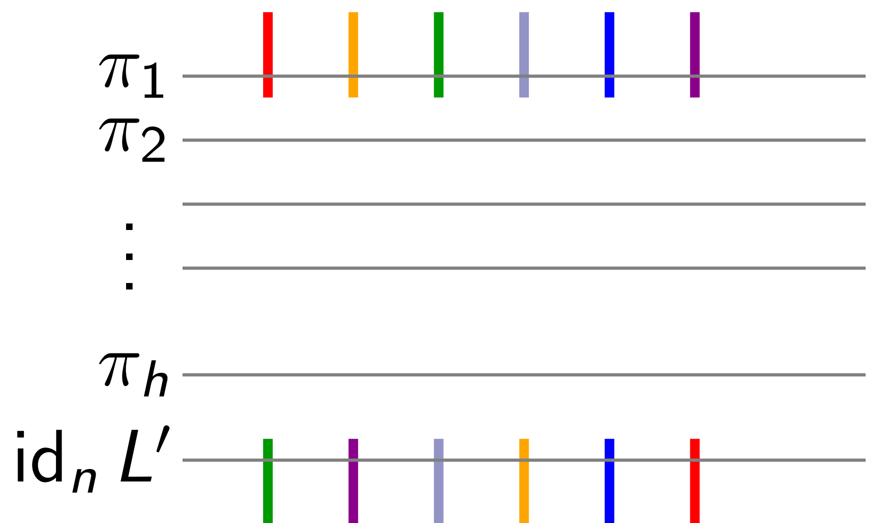
Let L' be the next list to consider.

Check its **consistency**.

Get the **final permutation** $\mathrm{id}_n L'$.

Choose the **shortest tangle** $T(L'')$

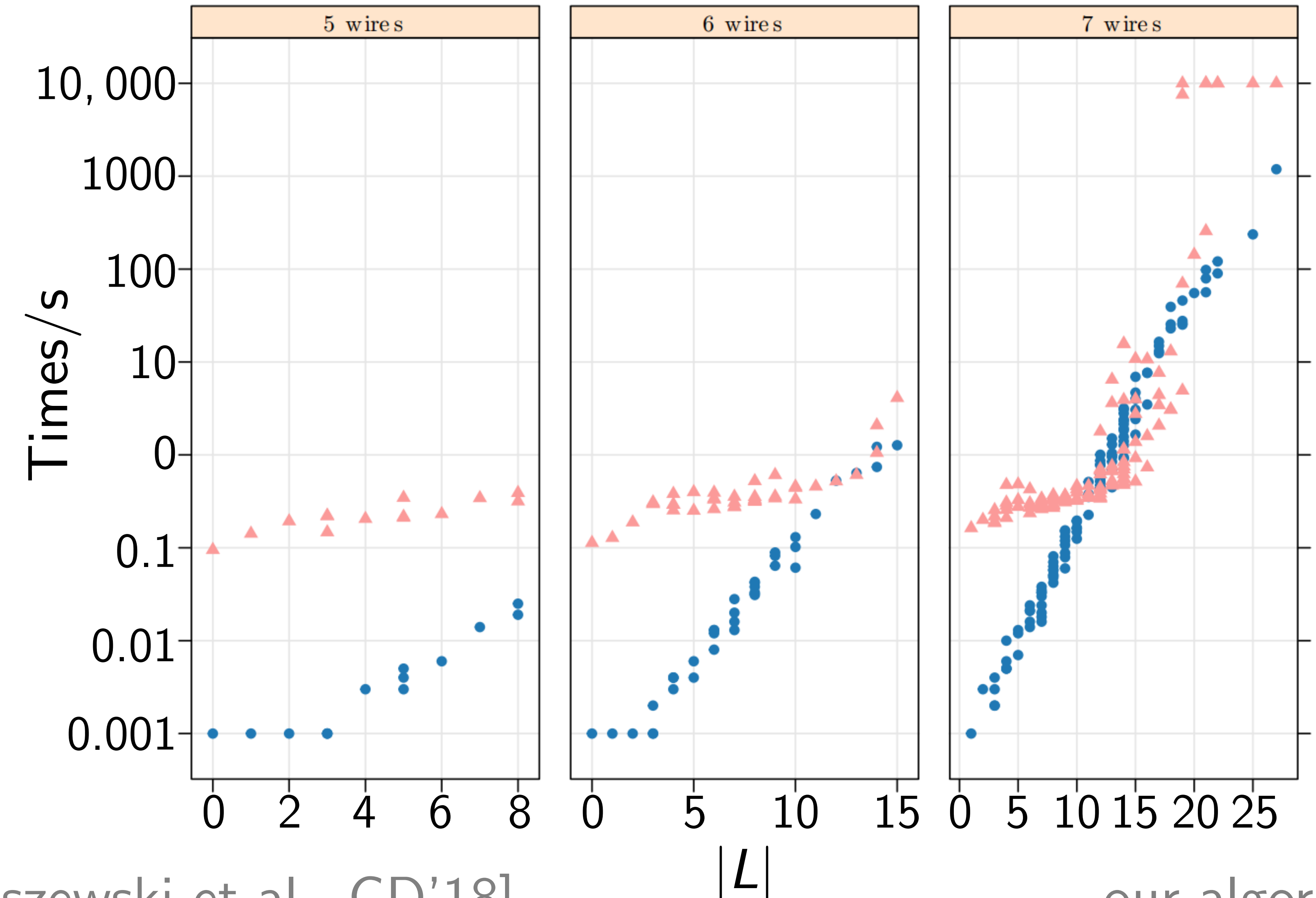
Add the final permutation
to the end.



Running time

$$O(\lambda(F_{n+1}-1)n) \leq \left|\begin{array}{l} \lambda = \prod_{i<j}(\ell_{ij}+1) \leq \left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}} \\ F_n \in O(\varphi^n) \end{array}\right| \leq$$

$|L|$: the length of the list



[Olszewski et al., GD'18]

our algorithm

$O\left(\frac{\varphi^{2|L|}}{5^{|L|/n}}n\right)$

$O\left(\left(\frac{2|L|}{n^2}+1\right)^{\frac{n^2}{2}}\varphi^n n\right)$

Open problems

Problem 1

Is it NP-hard to test the feasibility of a given (non-simple) list?

Open problems

Problem 1

Is it NP-hard to test the feasibility of a given (non-simple) list?

Problem 2

If feasibility is NP-hard, can we decide it faster than finding optimal tangles?

Open problems

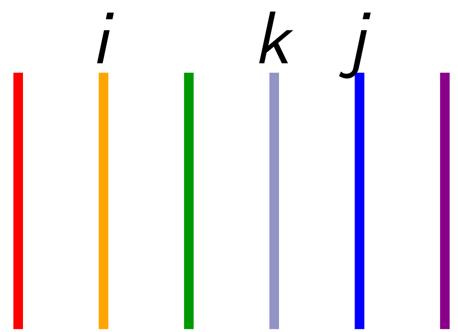
Problem 1

Is it NP-hard to test the feasibility of a given (non-simple) list?

Problem 2

If feasibility is NP-hard, can we decide it faster than finding optimal tangles?

Problem 3



A list (ℓ_{ij}) is *non-separable* if, for any $i<k<j$, $\ell_{ik} = \ell_{kj} = 0$ implies $\ell_{ij} = 0$.

Open problems

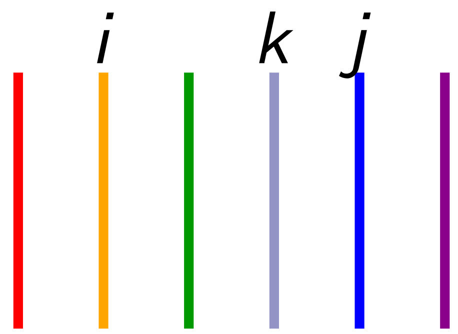
Problem 1

Is it NP-hard to test the feasibility of a given (non-simple) list?

Problem 2

If feasibility is NP-hard, can we decide it faster than finding optimal tangles?

Problem 3



A list (ℓ_{ij}) is *non-separable* if, for any $i<k<j$, $\ell_{ik} = \ell_{kj} = 0$ implies $\ell_{ij} = 0$.

necessary

Open problems

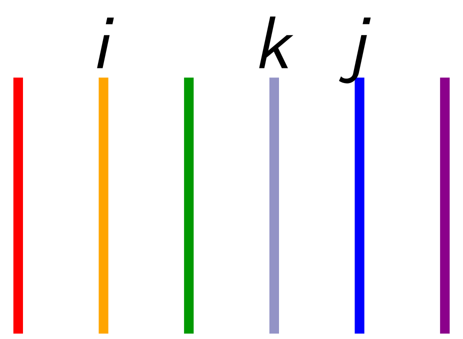
Problem 1

Is it NP-hard to test the feasibility of a given (non-simple) list?

Problem 2

If feasibility is NP-hard, can we decide it faster than finding optimal tangles?

Problem 3



A list (ℓ_{ij}) is *non-separable* if, for any $i<k<j$, $\ell_{ik} = \ell_{kj} = 0$ implies $\ell_{ij} = 0$.

necessary

For lists where all entries are even, is this sufficient?

Open problems

Thank you!

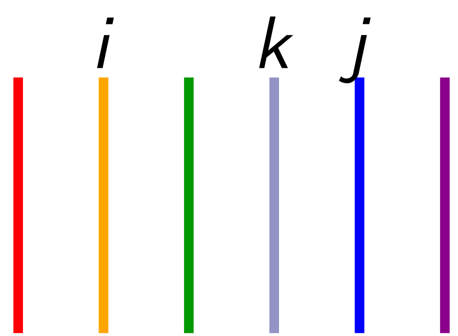
Problem 1

Is it NP-hard to test the feasibility of a given (non-simple) list?

Problem 2

If feasibility is NP-hard, can we decide it faster than finding optimal tangles?

Problem 3



A list (ℓ_{ij}) is *non-separable* if, for any $i<k<j$, $\ell_{ik} = \ell_{kj} = 0$ implies $\ell_{ij} = 0$.

necessary

For lists where all entries are even, is this sufficient?