Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

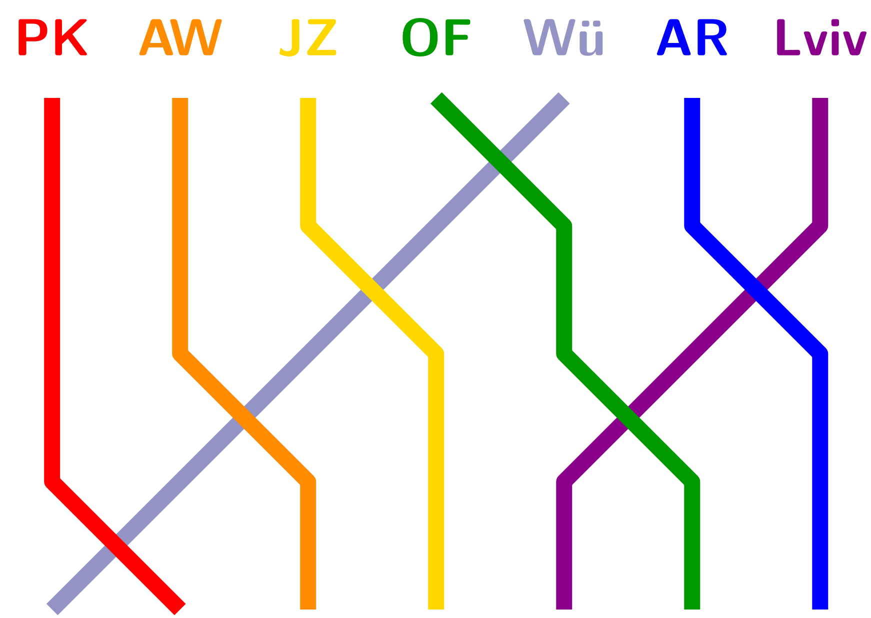
Computing Height-Optimal Tangles Faster

Oksana Firman
Philipp Kindermann
Alexander Wolff
Johannes Zink

Julius-Maximilians-Universität Würzburg, Germany

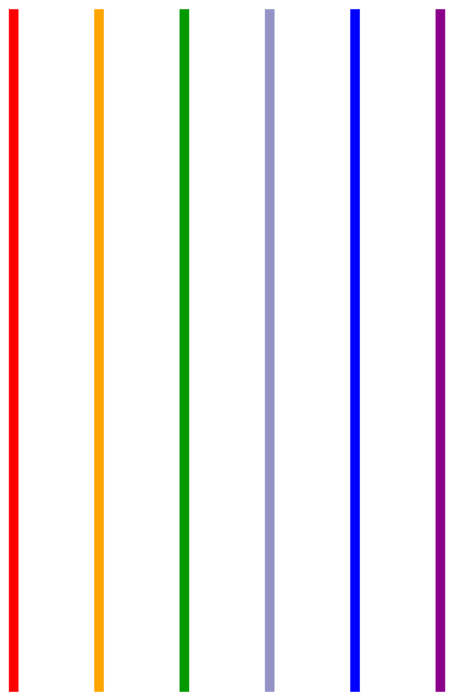
Alexander Ravsky

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



Introduction

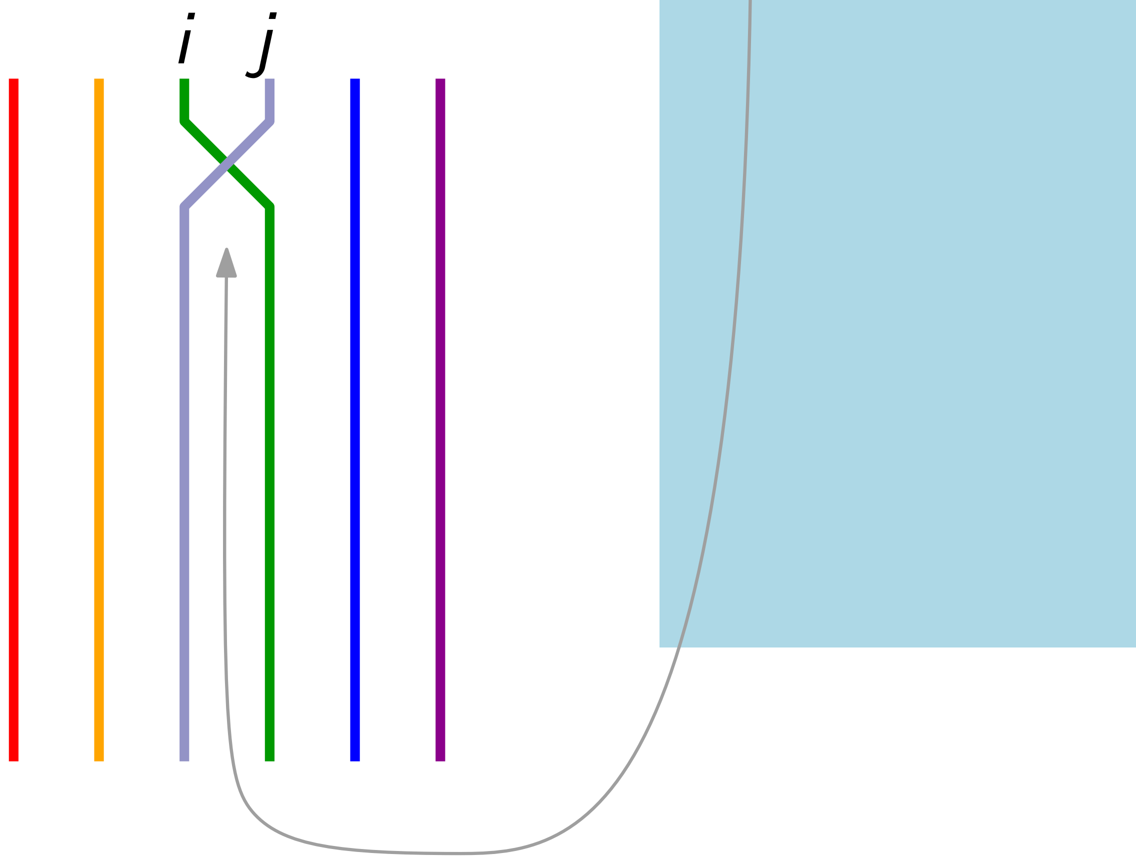
Given a set of n y-monotone wires



Introduction

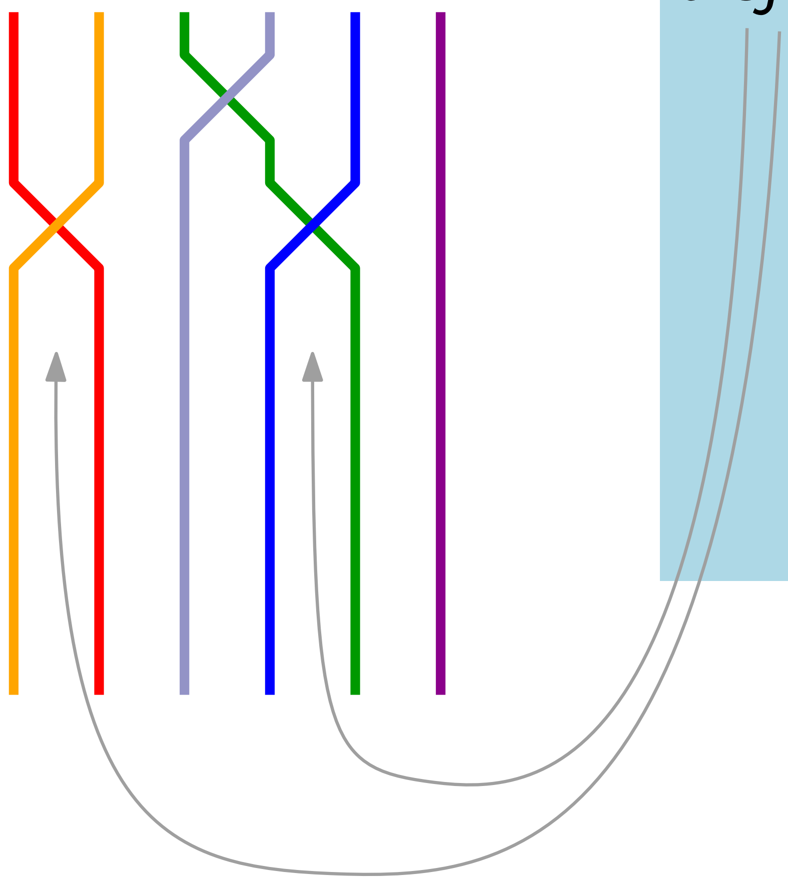
Given a set of n
y-monotone wires

$1 \leq i < j \leq n$



Introduction

Given a set of n y-monotone wires



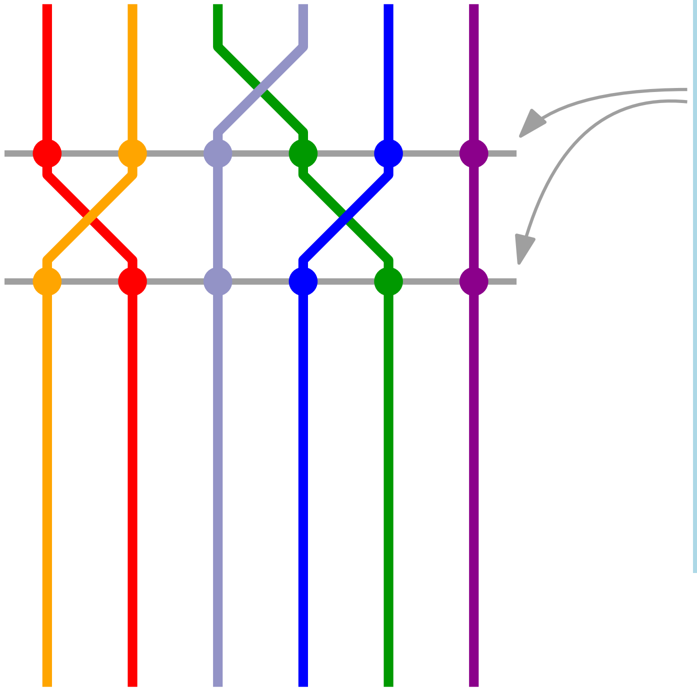
$1 \leq i < j \leq n$

swap ij

disjoint swaps

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

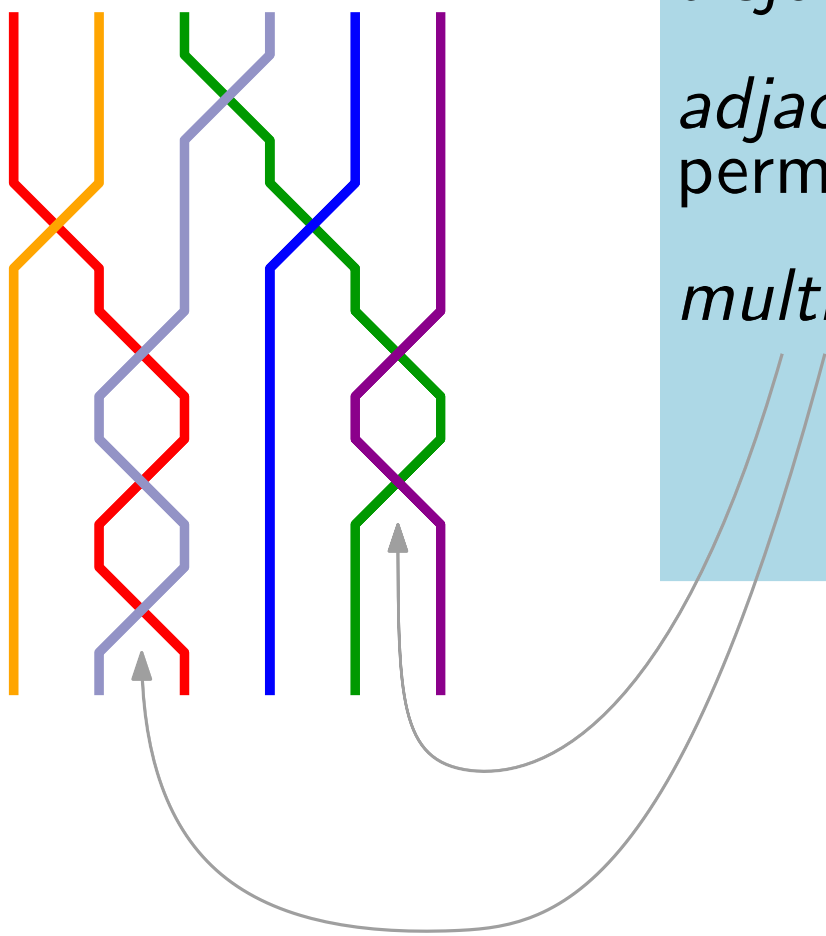
swap ij

disjoint swaps

adjacent permutations

Introduction

Given a set of n
y-monotone wires



$1 \leq i < j \leq n$

swap ij

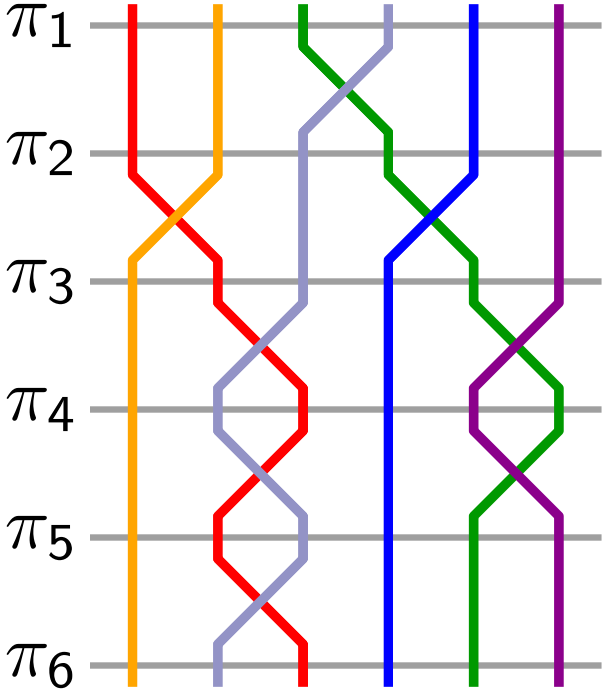
disjoint swaps

adjacent
permutations

multiple swaps

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

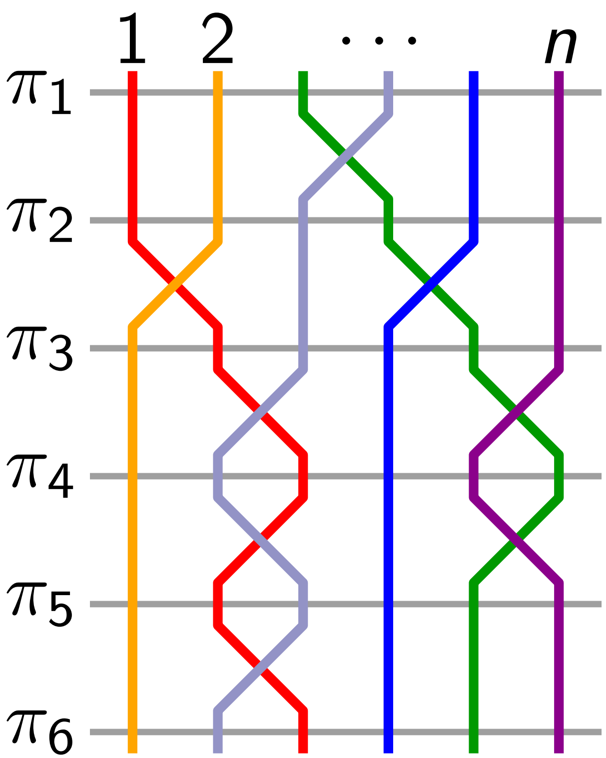
adjacent permutations

multiple swaps

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

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

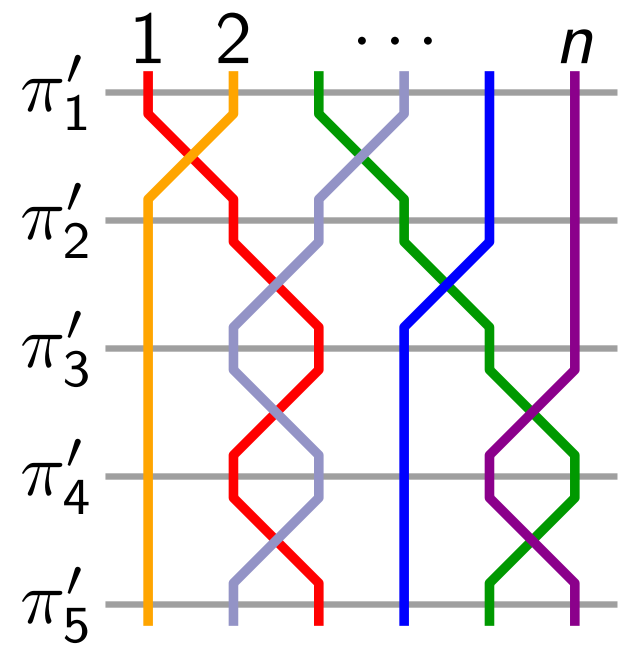
swap ij

disjoint swaps

adjacent permutations

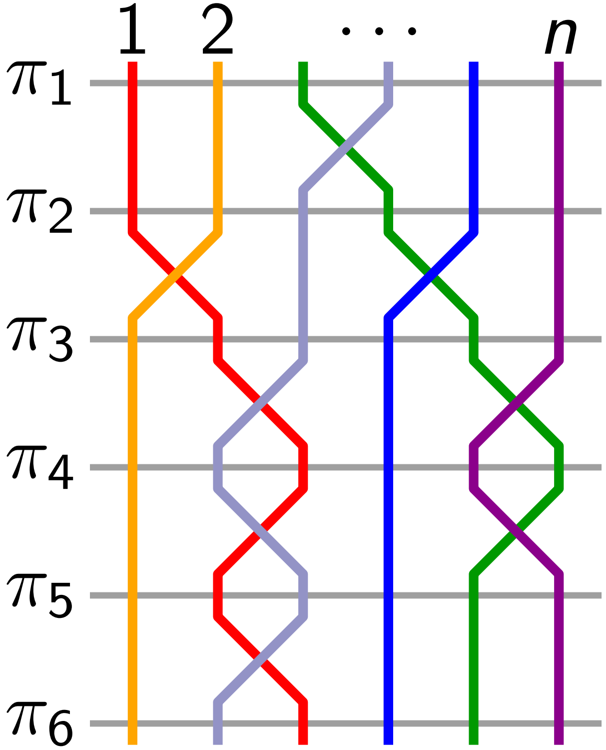
multiple swaps

tangle T of *height h(T)*



Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

adjacent permutations

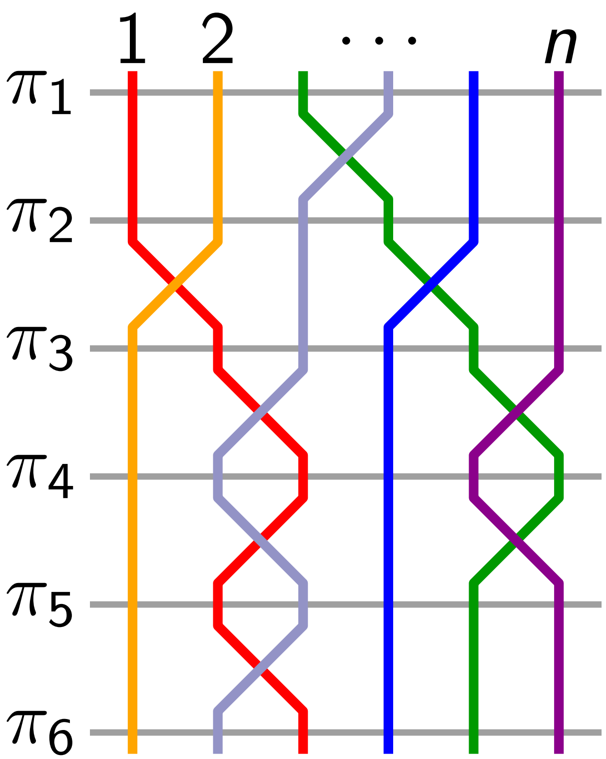
multiple swaps

tangle T of *height h(T)*

…and given a list of swaps L

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

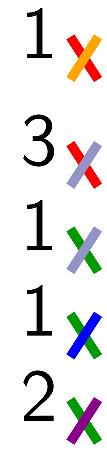
adjacent permutations

multiple swaps

tangle T of *height h*(T)

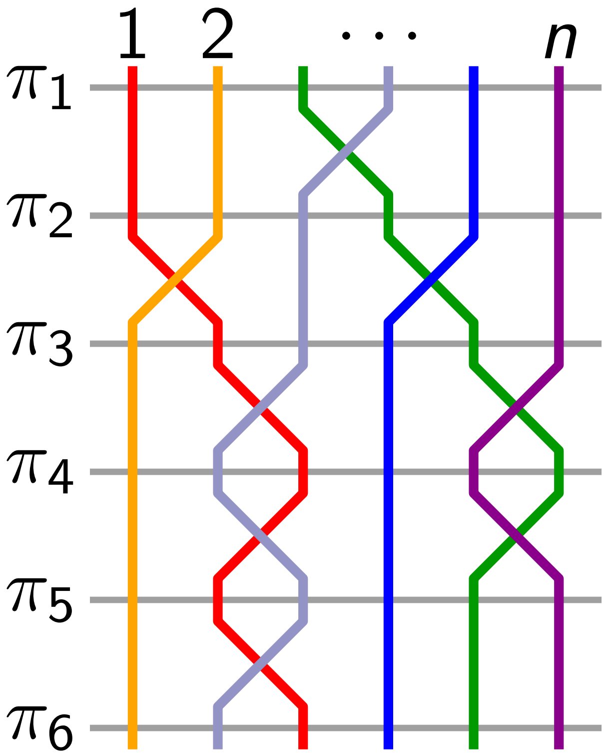
. . . and given a list of swaps L

as a multiset (ℓ_{ij})



Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

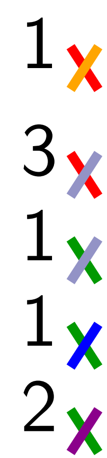
adjacent permutations

multiple swaps

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

...and given a list of swaps L

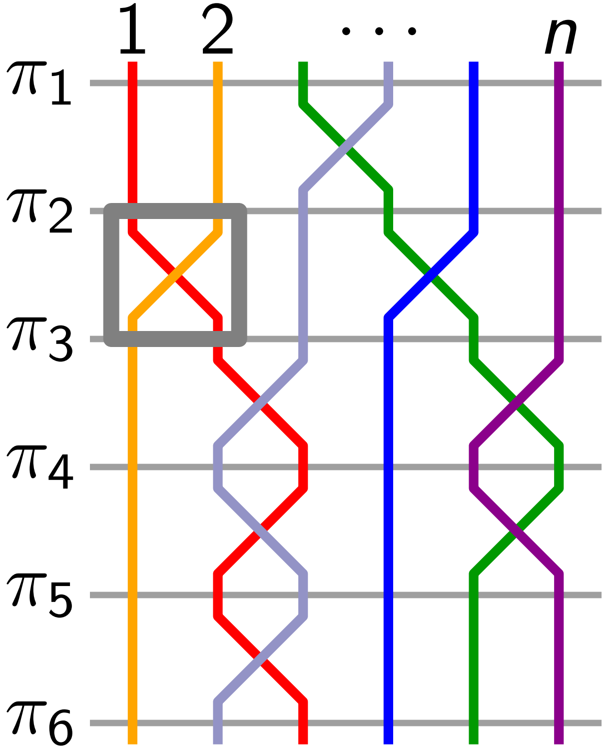
as a multiset (ℓ_{ij})



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

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

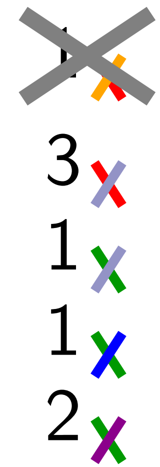
adjacent permutations

multiple swaps

tangle T of *height h(T)*

…and given a list of swaps L

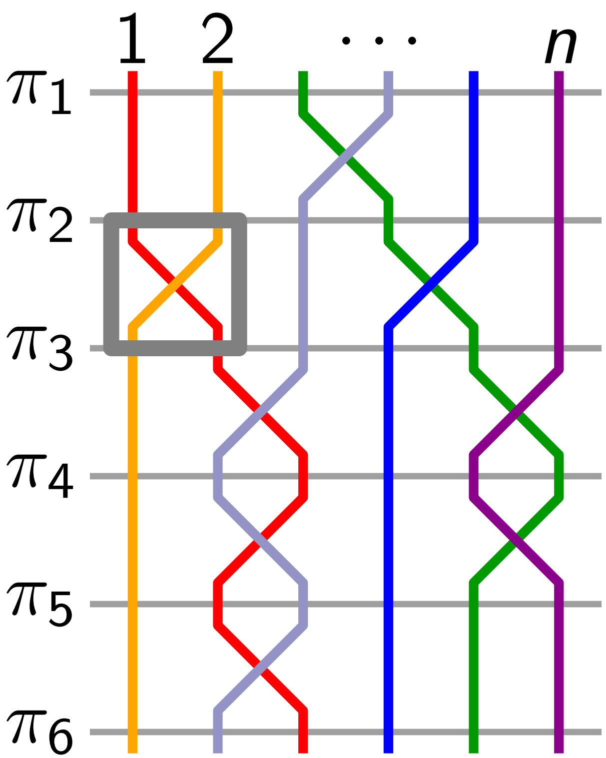
as a multiset (ℓ_{ij})



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

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

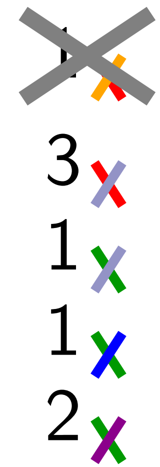
adjacent permutations

multiple swaps

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

...and given a list of swaps L

as a multiset (ℓ_{ij})

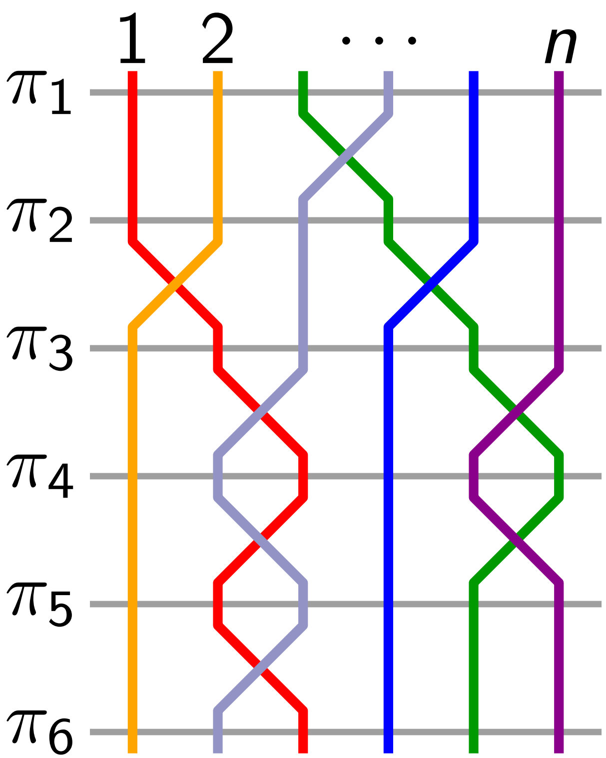


not *feasible*

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

Introduction

Given a set of n y-monotone wires



$1 \leq i < j \leq n$

swap ij

disjoint swaps

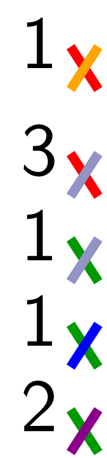
adjacent permutations

multiple swaps

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

…and given a list of swaps L

as a multiset (ℓ_{ij})

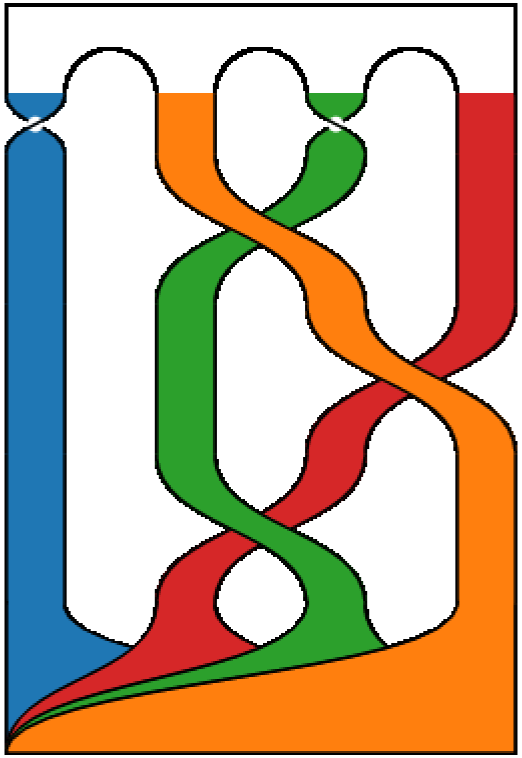


Tangle $T(L)$ *realizes* list L.

A tangle $T(L)$ is *height-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



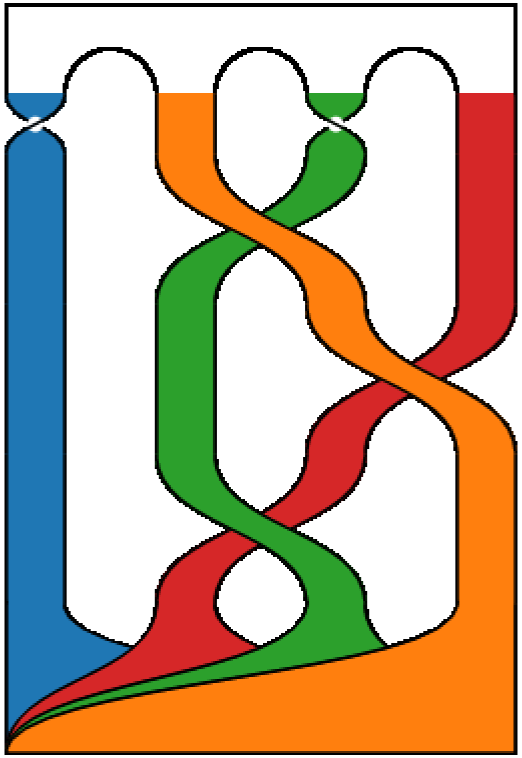
Related Work

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

template

⇕

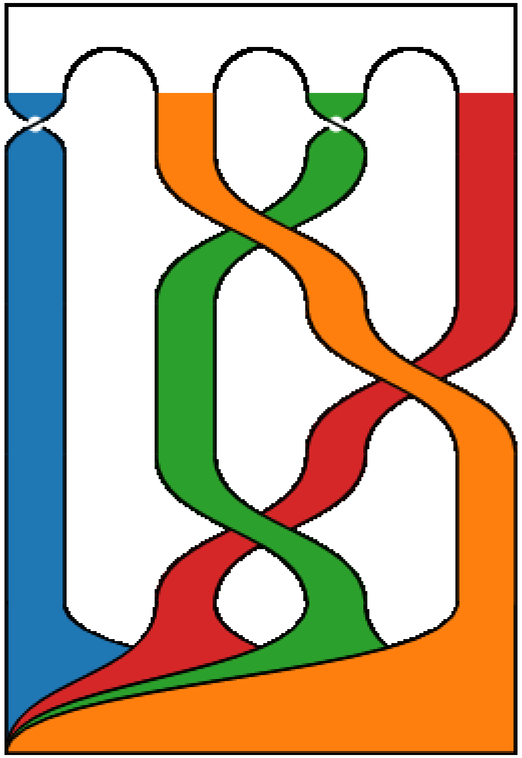
list



Related Work

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

Algorithm for finding optimal tangles



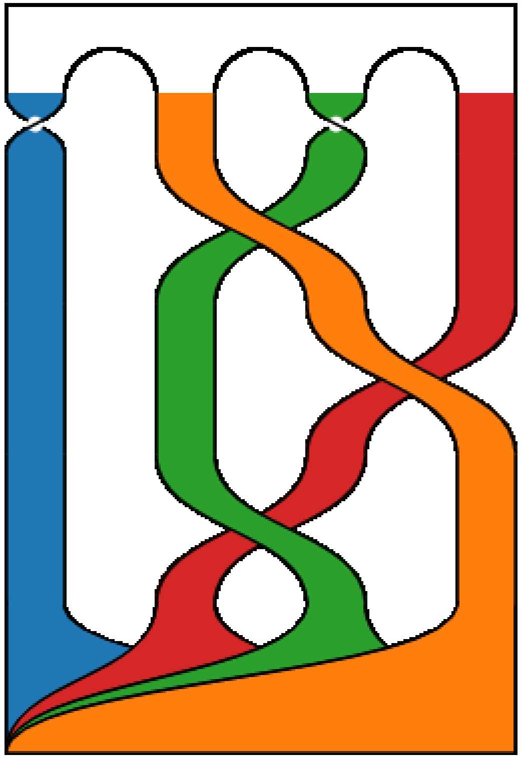
Related Work

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

template
⇕
list

Algorithm for finding optimal tangles

Complexity **?**



Related Work

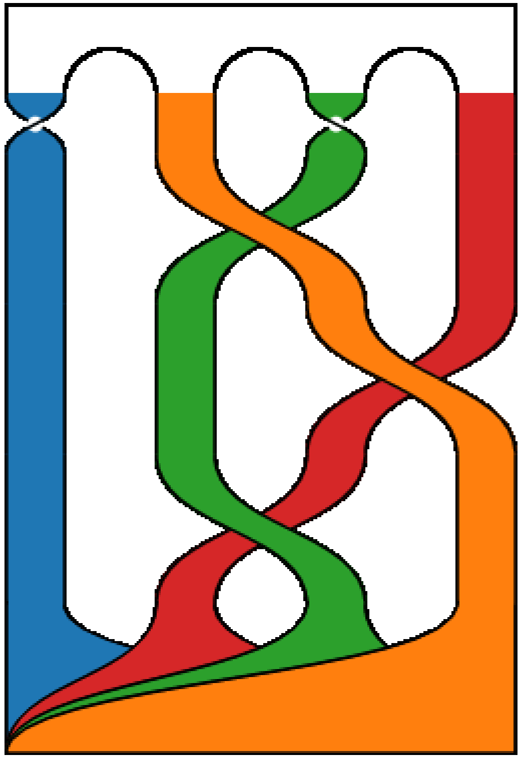
- *Olszewski et al.* Visualizing the template of a chaotic attractor. GD 2018

template ⇕ list

Algorithm for finding optimal tangles | *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 ⇕ list

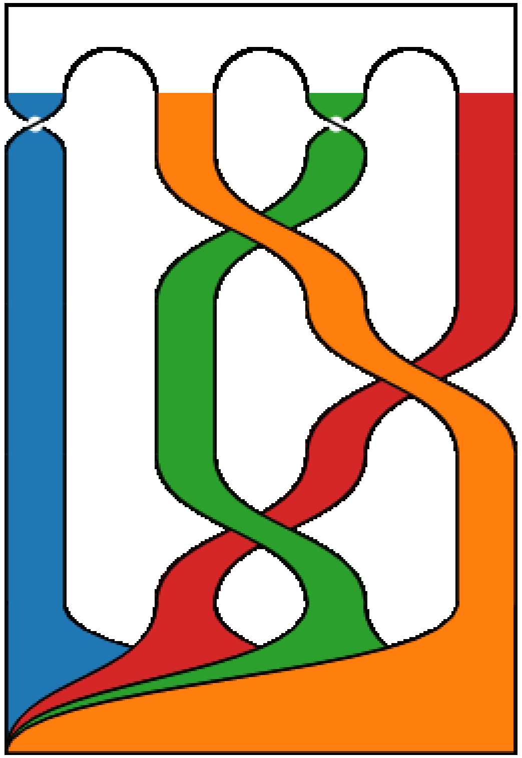
Algorithm for finding optimal tangles | *Complexity* ?

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

Given: initial and *final* permutations

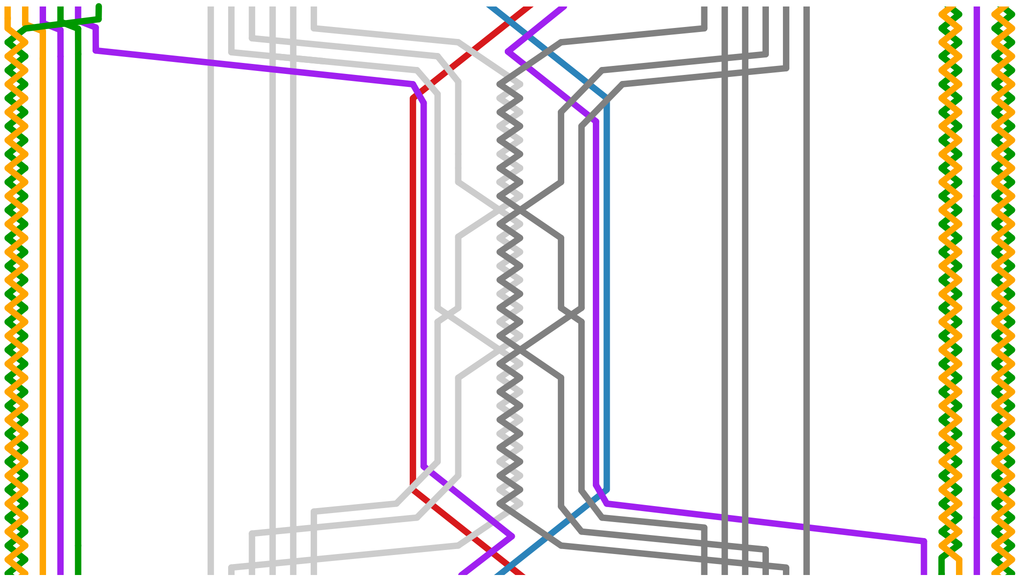
- *Bereg et al.* Drawing Permutations with Few Corners. GD 2013

Objective: minimize the number of *bends*



Overview

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



- **New algorithm:** using dynamic programming; asymptotically faster than [Olszewski et al., GD'18].

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

- **Experiments:** comparison with [Olszewski et al., GD'18]

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: Multiset A of $3m$ positive integers.

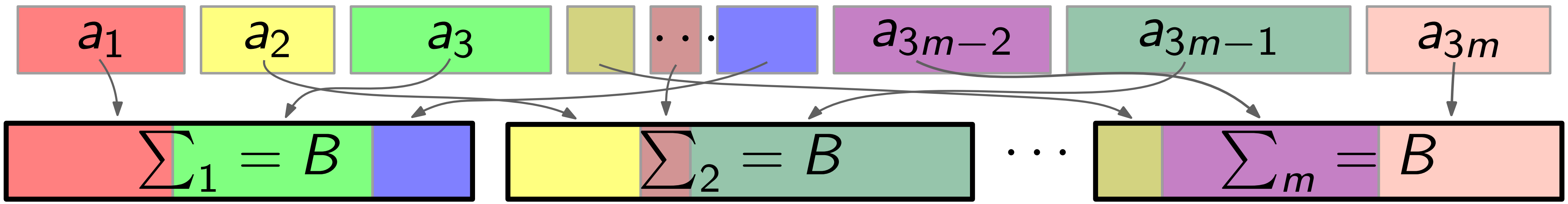
Complexity

Theorem.
Tangle-Height Minimization is NP-hard.

Proof.
Reduction from 3-Partition

Given: Multiset A of $3m$ positive integers.

Question: Can A be partitioned into m groups of three elements s.t. each group sums up to the same value B?



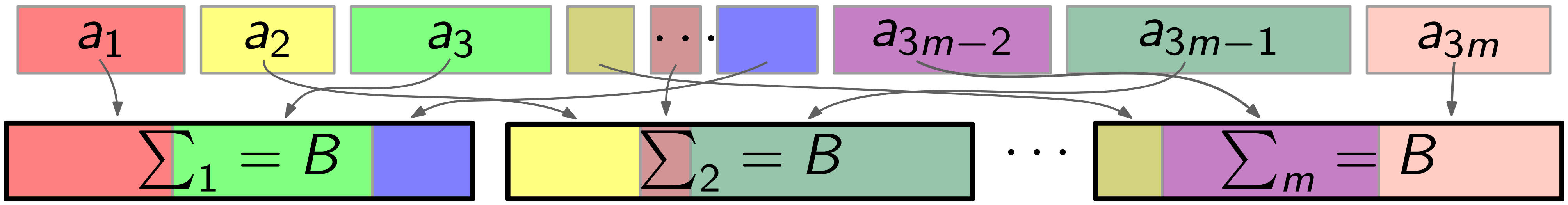
Complexity

Theorem.
TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof.
Reduction from 3-PARTITION

$\frac{B}{4} < a_i < \frac{B}{2}$
B is poly in m

Given: Multiset A of $3m$ positive integers.
Question: Can A be partitioned into m groups of three elements s.t. each group sums up to the same value B?



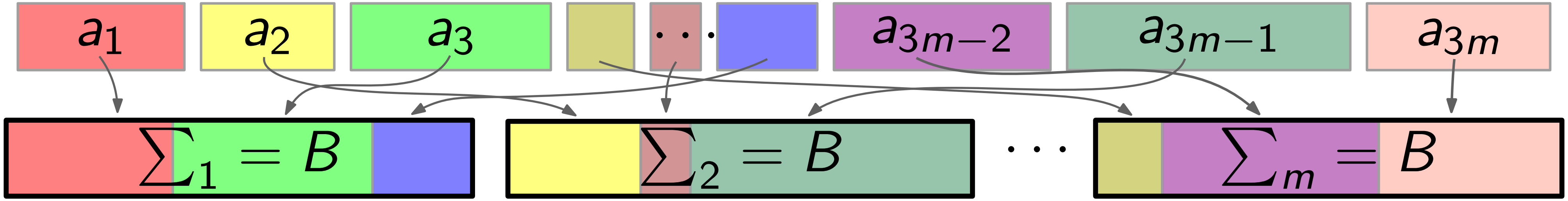
Complexity

Theorem.
TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof.
Reduction from 3-PARTITION

$\frac{B}{4} < a_i < \frac{B}{2}$
B is poly in m

Given: Multiset A of $3m$ positive integers.
Question: Can A be partitioned into m groups of three elements s.t. each group sums up to the same value B?



Given: Instance A of 3-PARTITION.

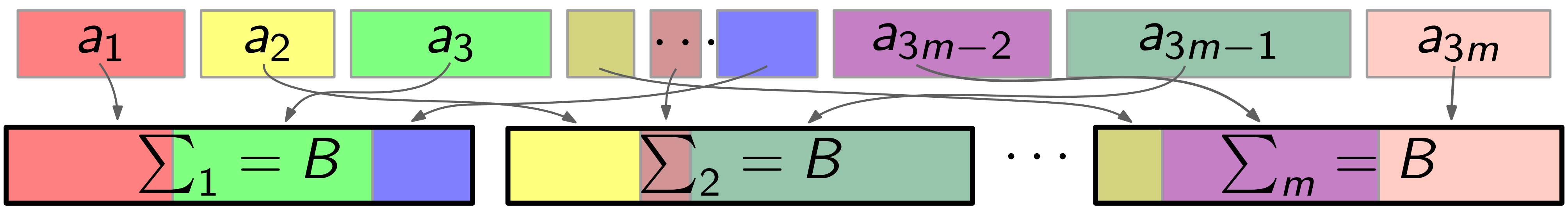
Complexity

Theorem.
Tangle-Height Minimization is NP-hard.

Proof.
Reduction from 3-Partition

$\frac{B}{4} < a_i < \frac{B}{2}$
B is poly in m

Given: Multiset A of $3m$ positive integers.
Question: Can A be partitioned into m groups of three elements s.t. each group sums up to the same value B?

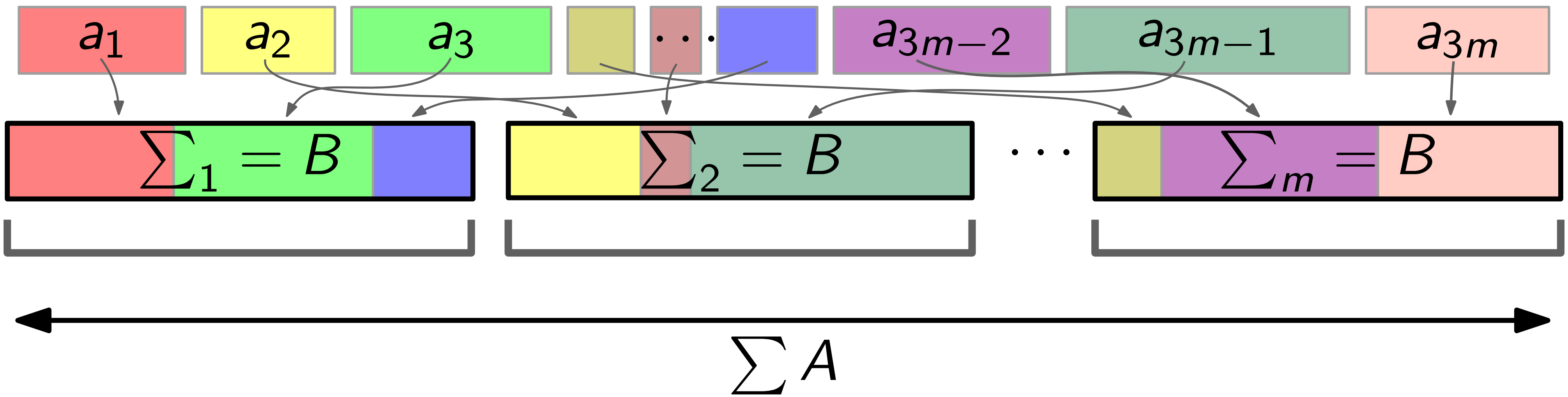


Given: Instance A of 3-Partition.
Task: Construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance.

Complexity

Theorem.
TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof.
Reduction from 3-PARTITION

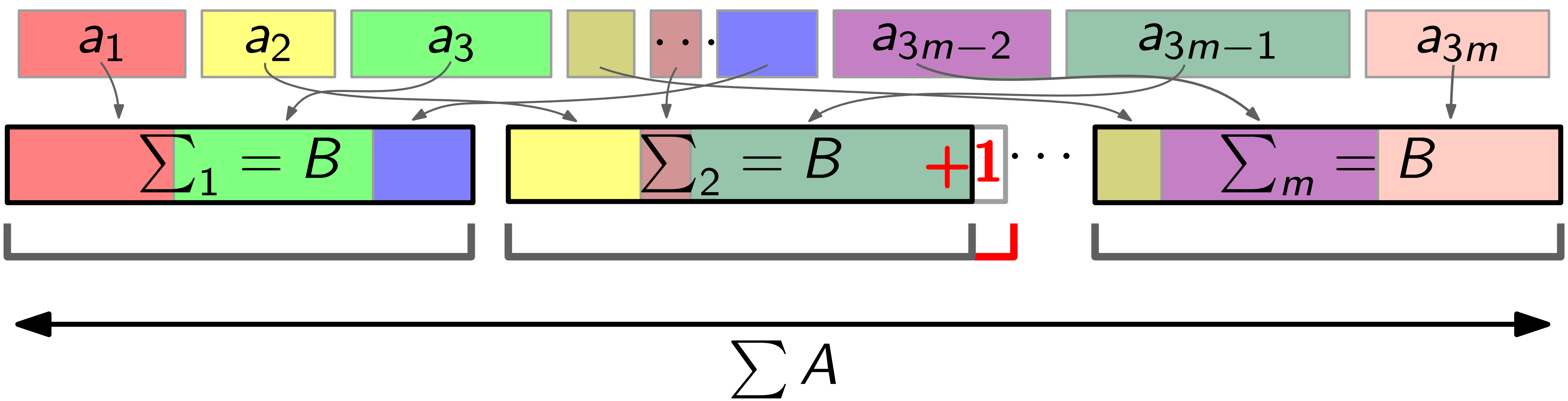


Given:	Instance A of 3-PARTITION.
Task:	Construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance.

Complexity

Theorem.
TANGLE-HEIGHT MINIMIZATION is NP-hard.

Proof.
Reduction from 3-PARTITION



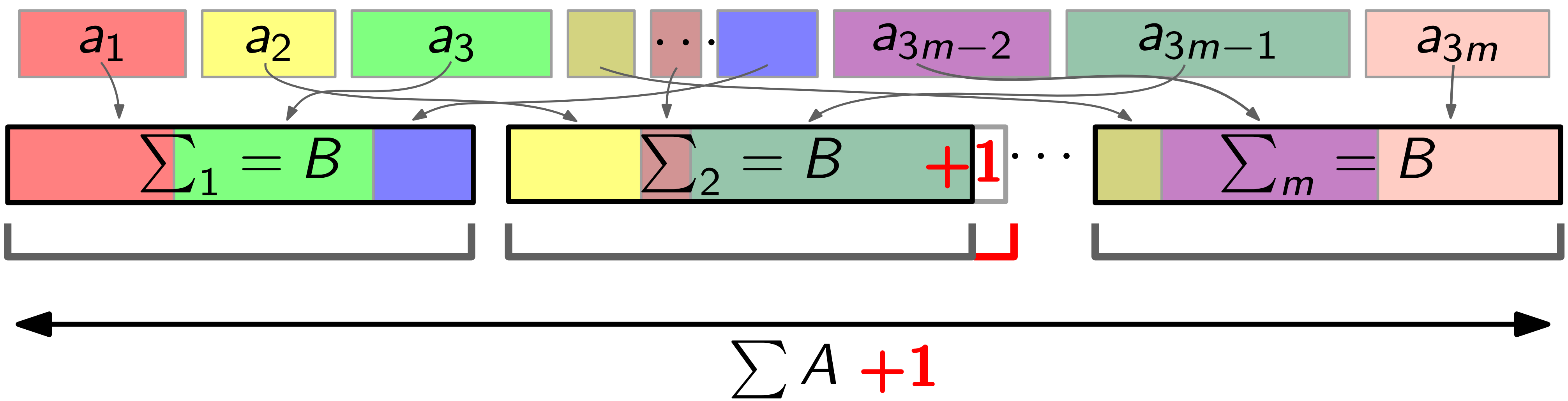
Given: Instance A of 3-PARTITION.

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

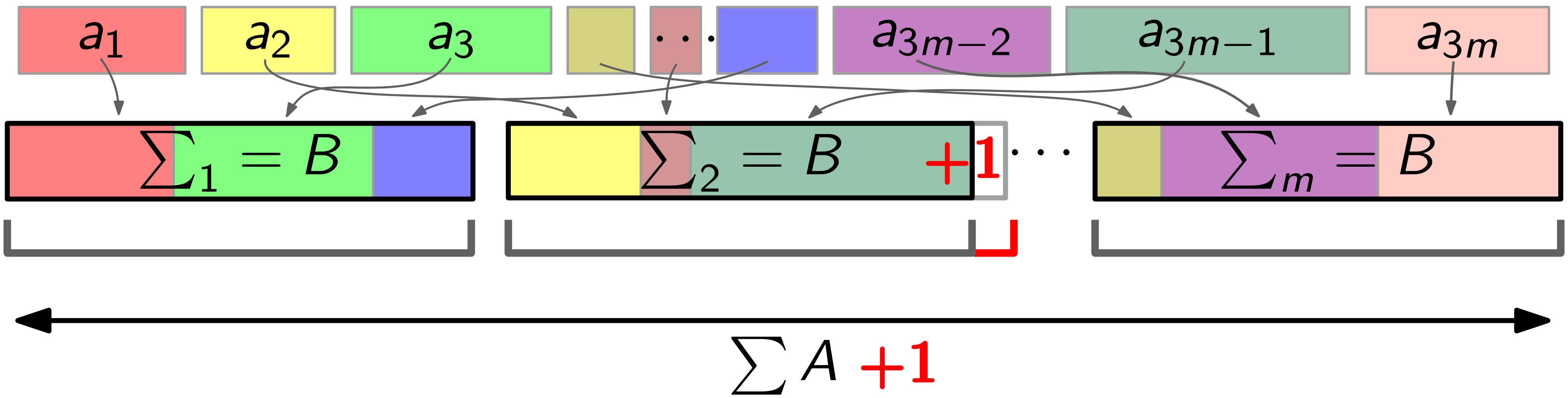


Given:	Instance A of 3-Partition.
Task:	Construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A) + 7m^2$ iff A is a yes-instance.

Complexity

Theorem.
TANGLE-HEIGHT MINIMIZATION is NP-hard.

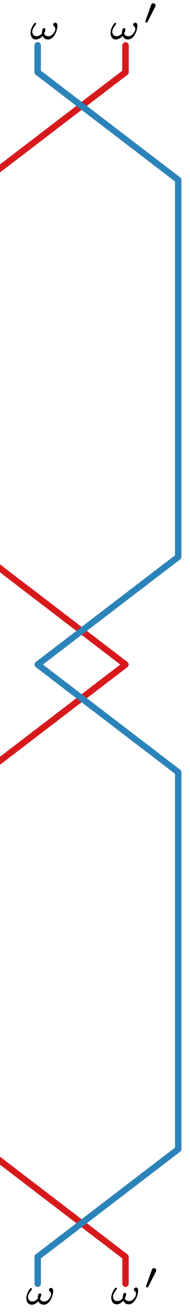
Proof.
Reduction from 3-PARTITION



Given: Instance A of 3-PARTITION.

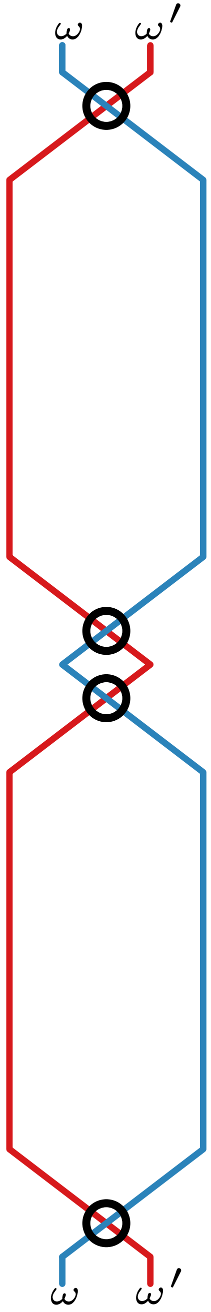
Task: construct L s.t. there is T realizing L with height at most $H = 2m^3(\sum A + 1) + 7m^2$ iff A is a yes-instance

Transforming the Instance A into a List L

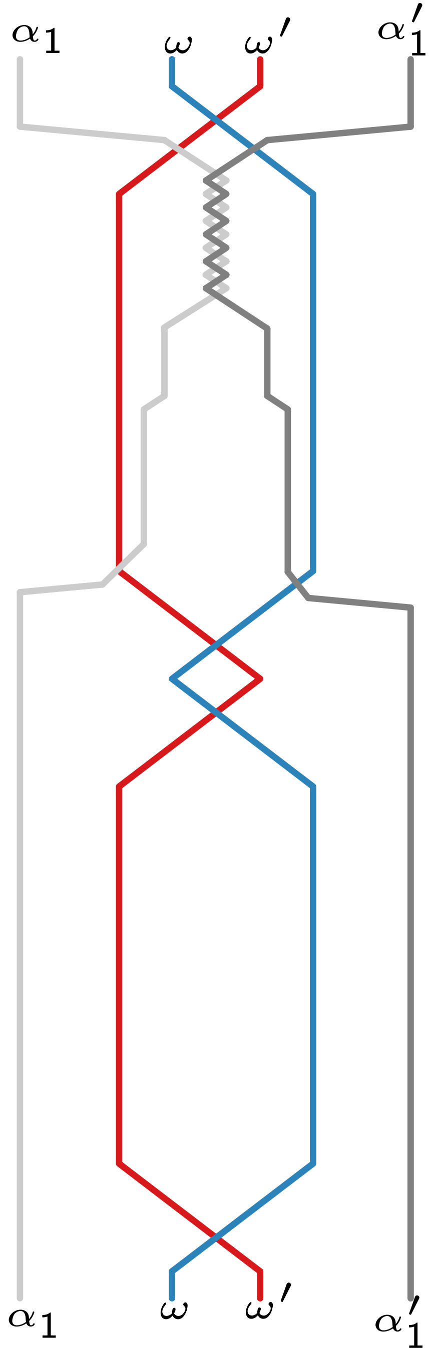


Transforming the Instance A into a List L

$2m$ swaps

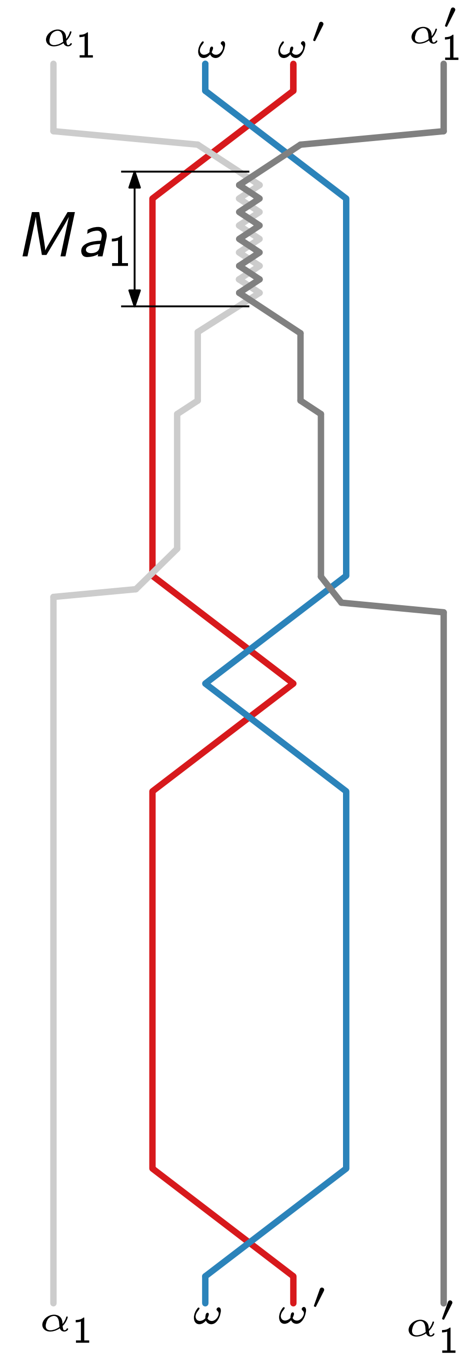


Transforming the Instance A into a List L



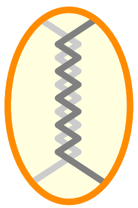
Transforming the Instance A into a List L

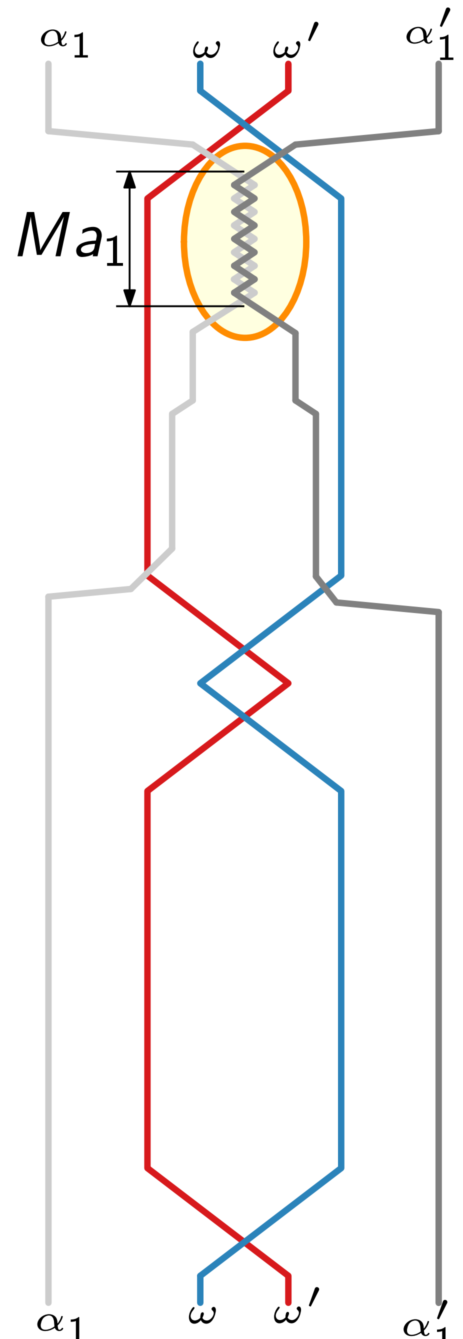
$$M = 2m^3$$



Transforming the Instance A into a List L

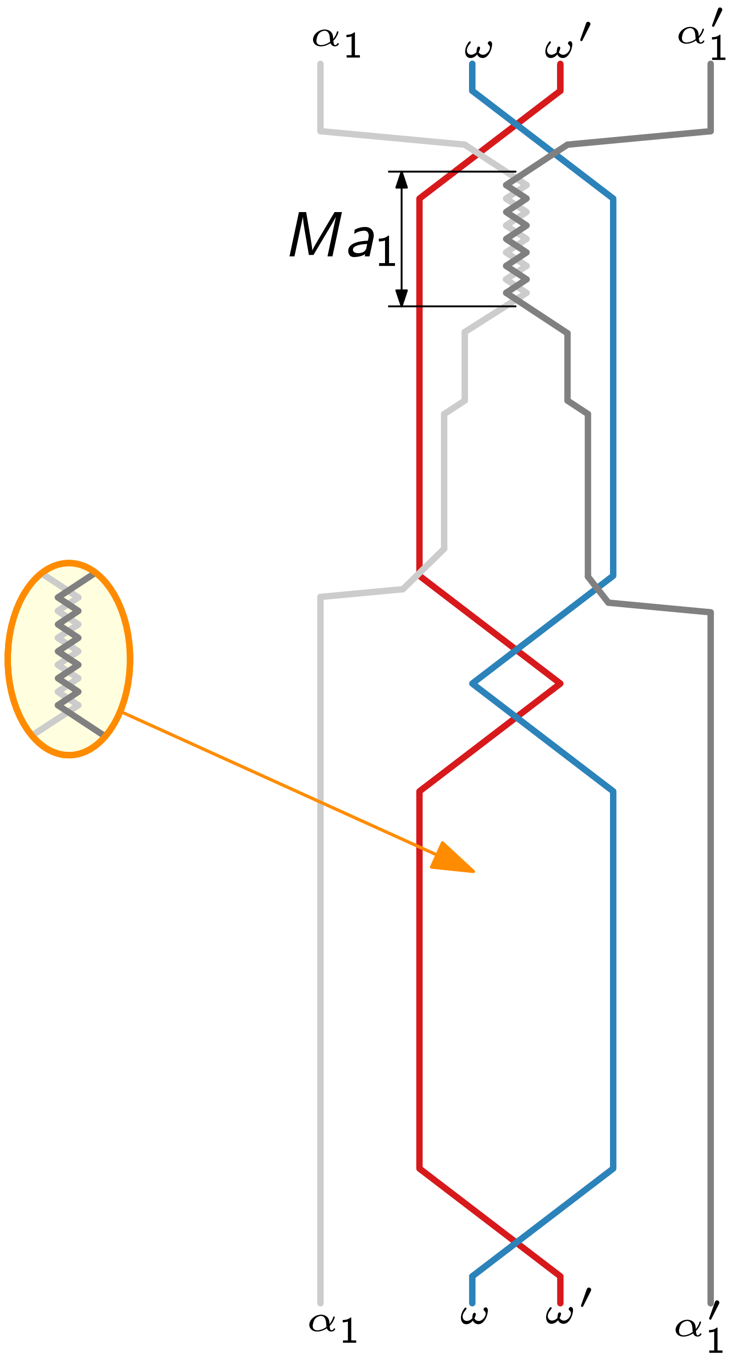
$$M = 2m^3$$





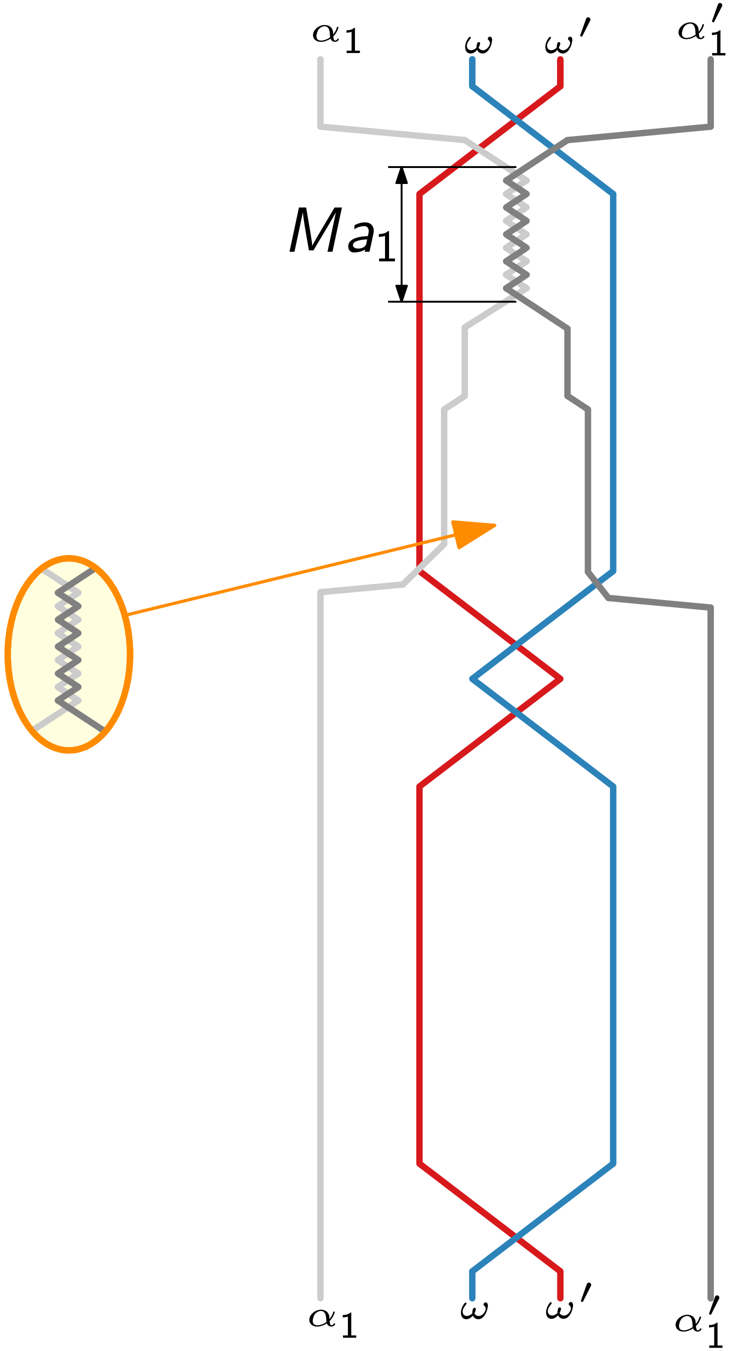
Transforming the Instance A into a List L

$$M = 2m^3$$



Transforming the Instance A into a List L

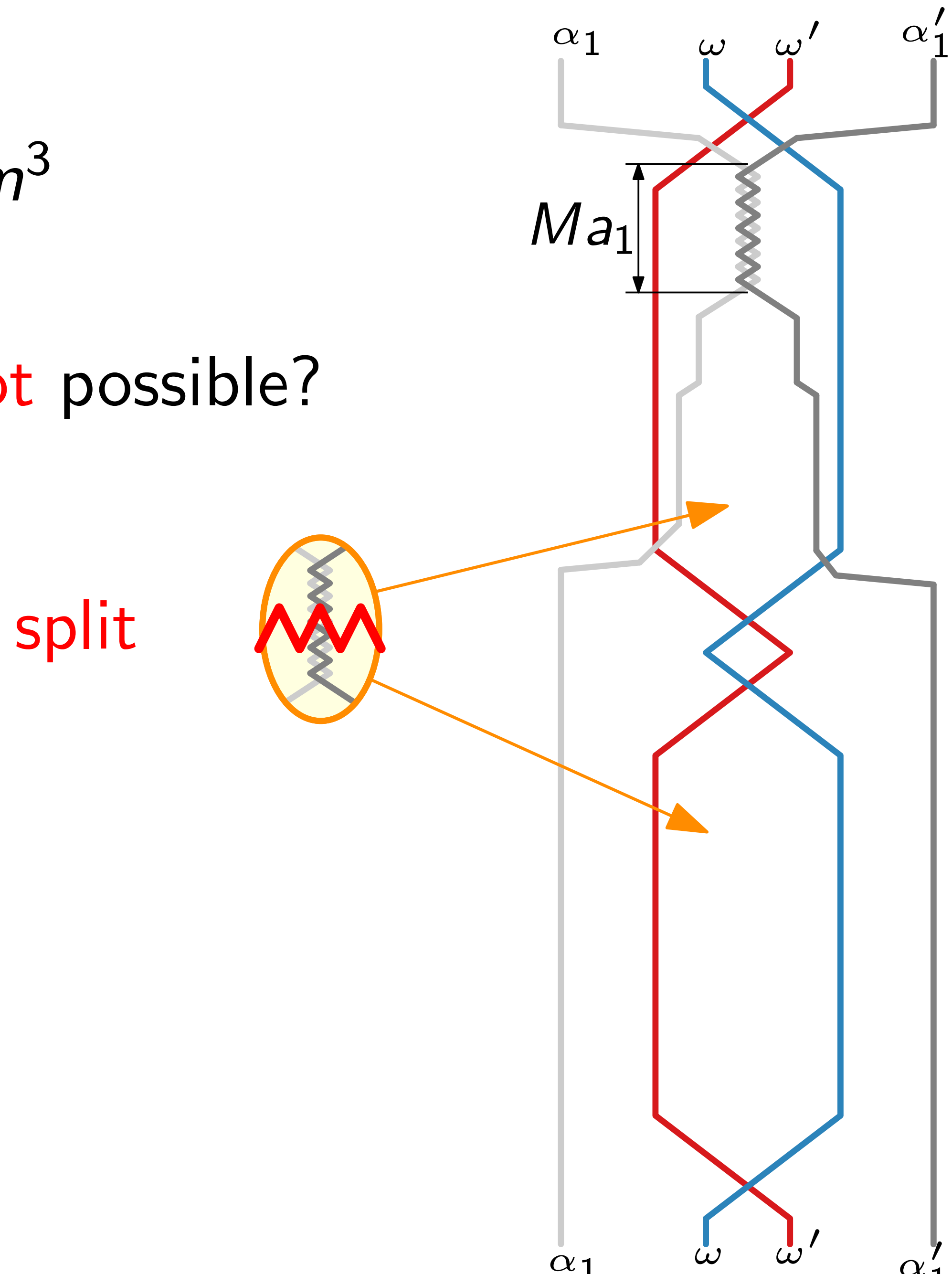
$$M = 2m^3$$



Transforming the Instance A into a List L

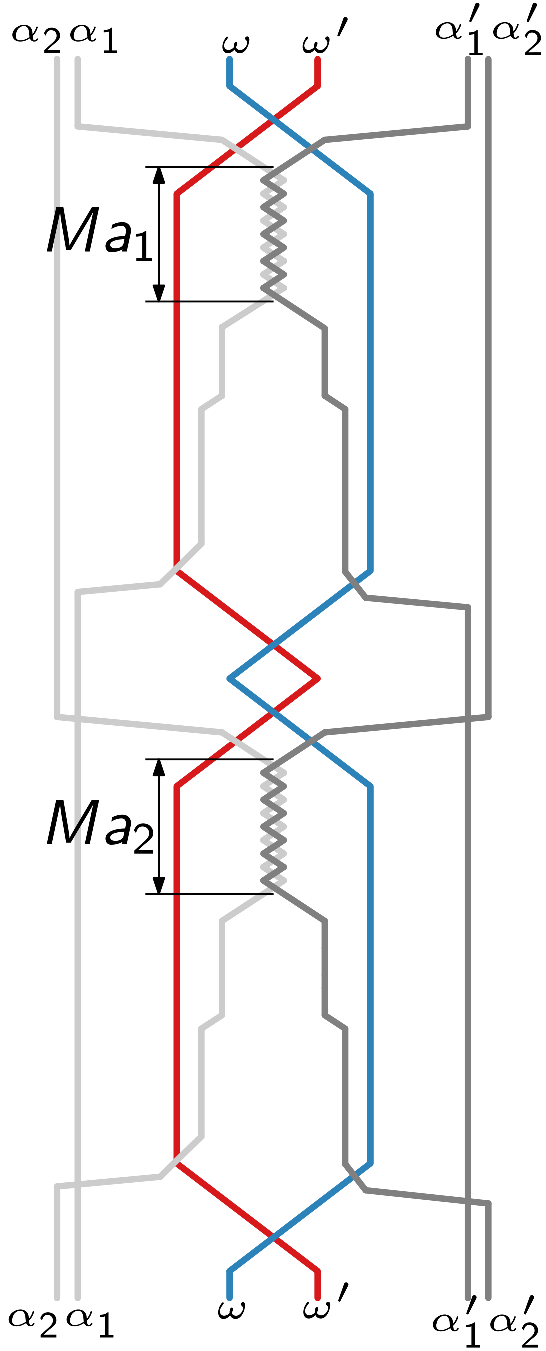
$$M = 2m^3$$

What is not possible?



Transforming the Instance A into a List L

$$M = 2m^3$$

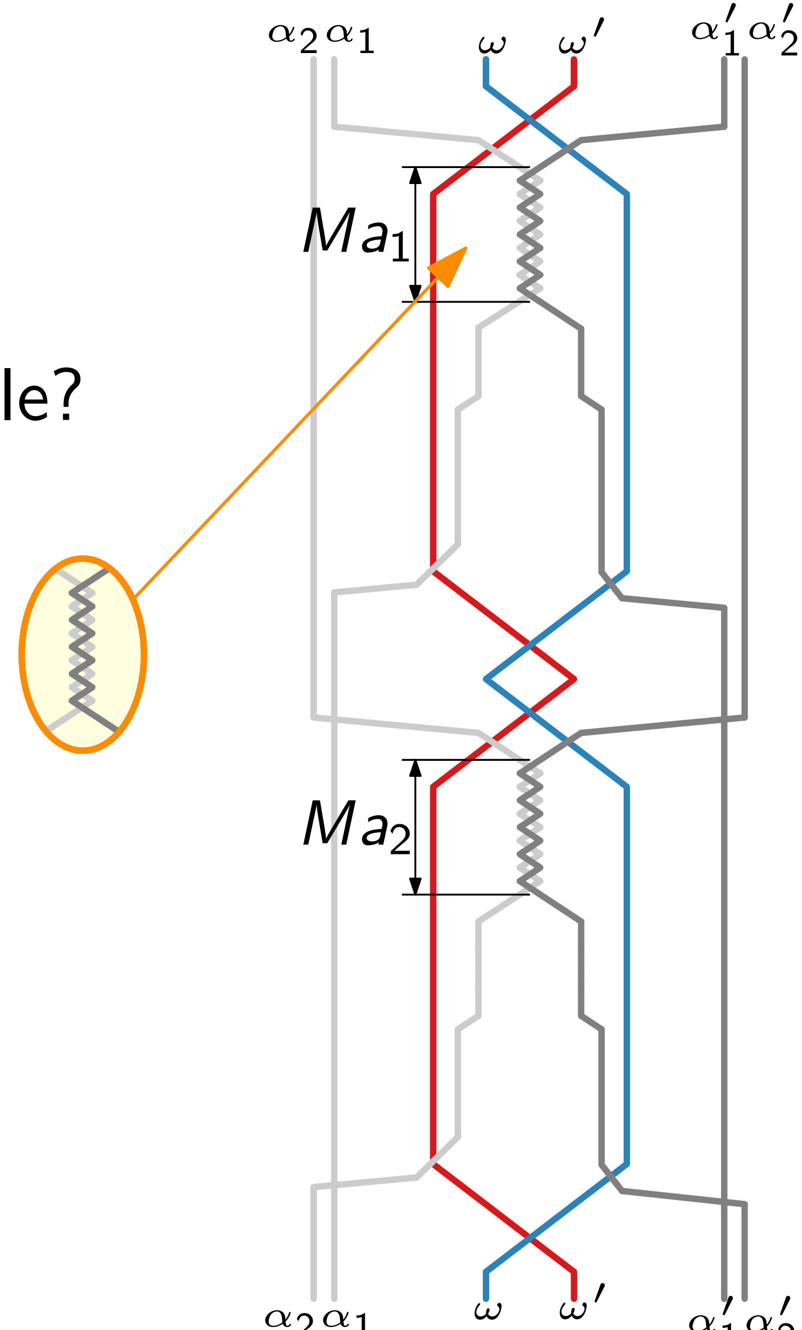


Transforming the Instance A into a List L

$$M = 2m^3$$

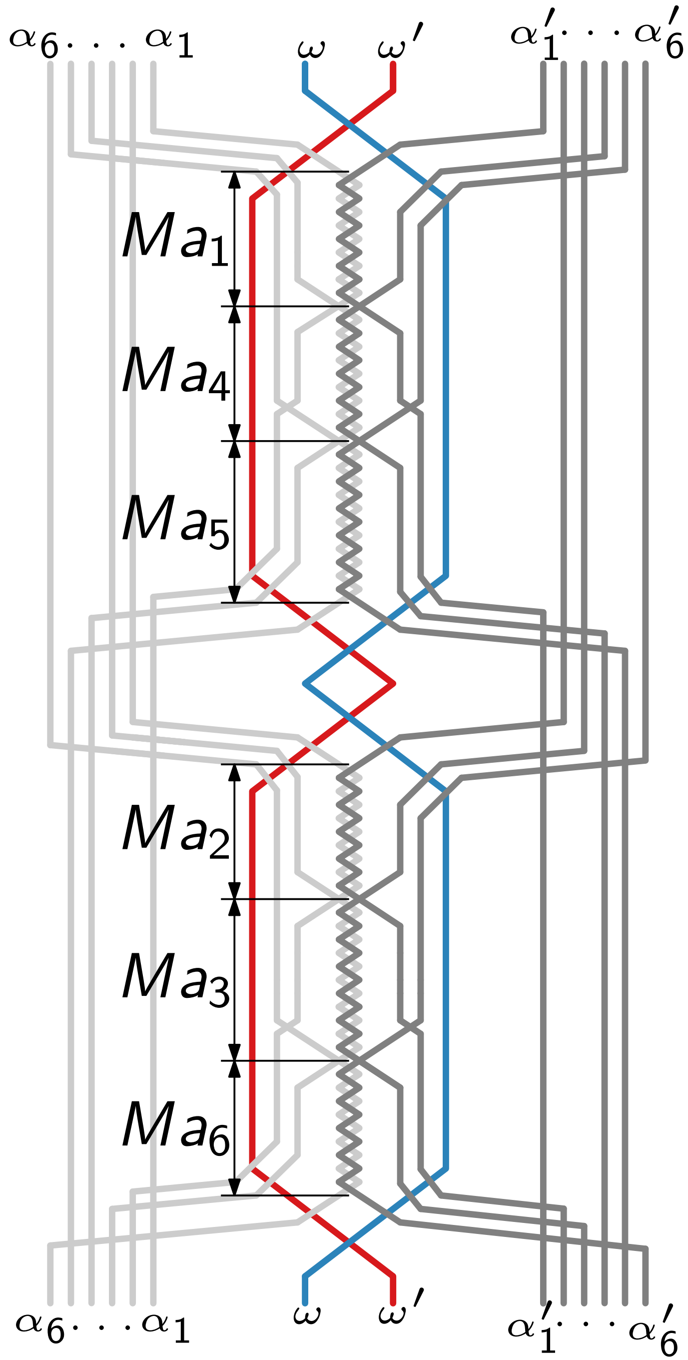
What is not possible?

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

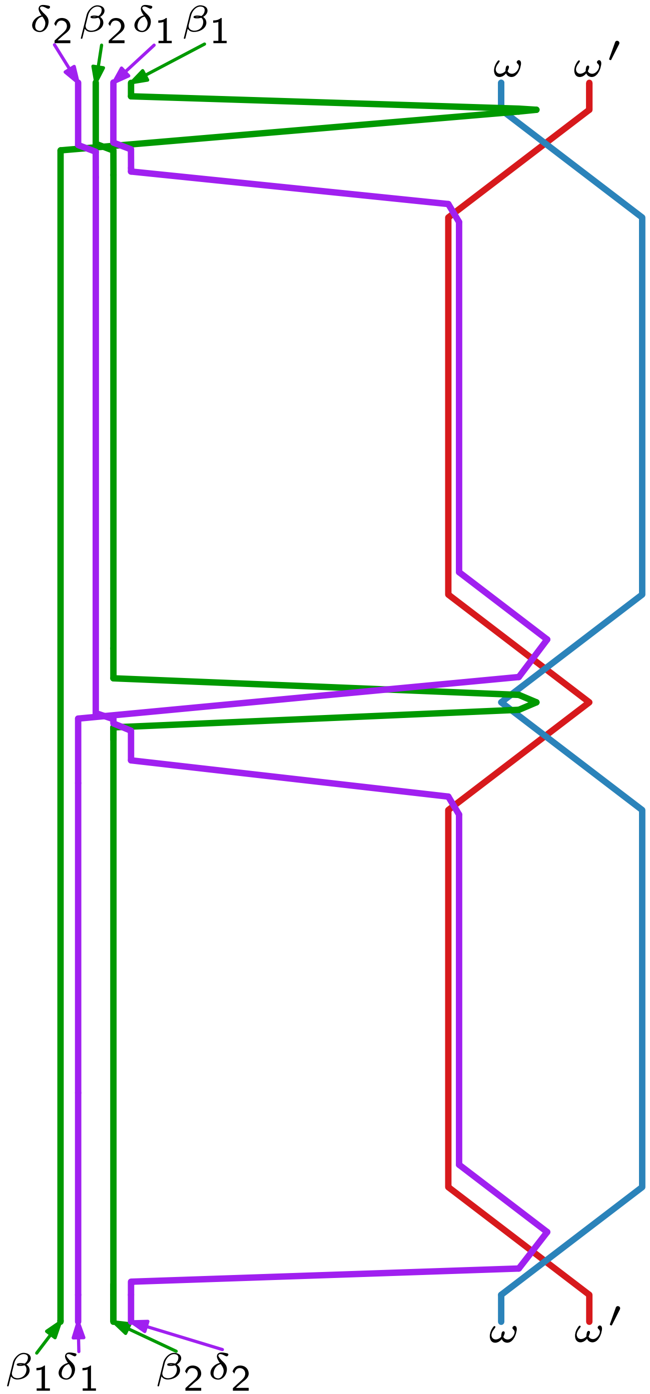


Transforming the Instance A into a List L

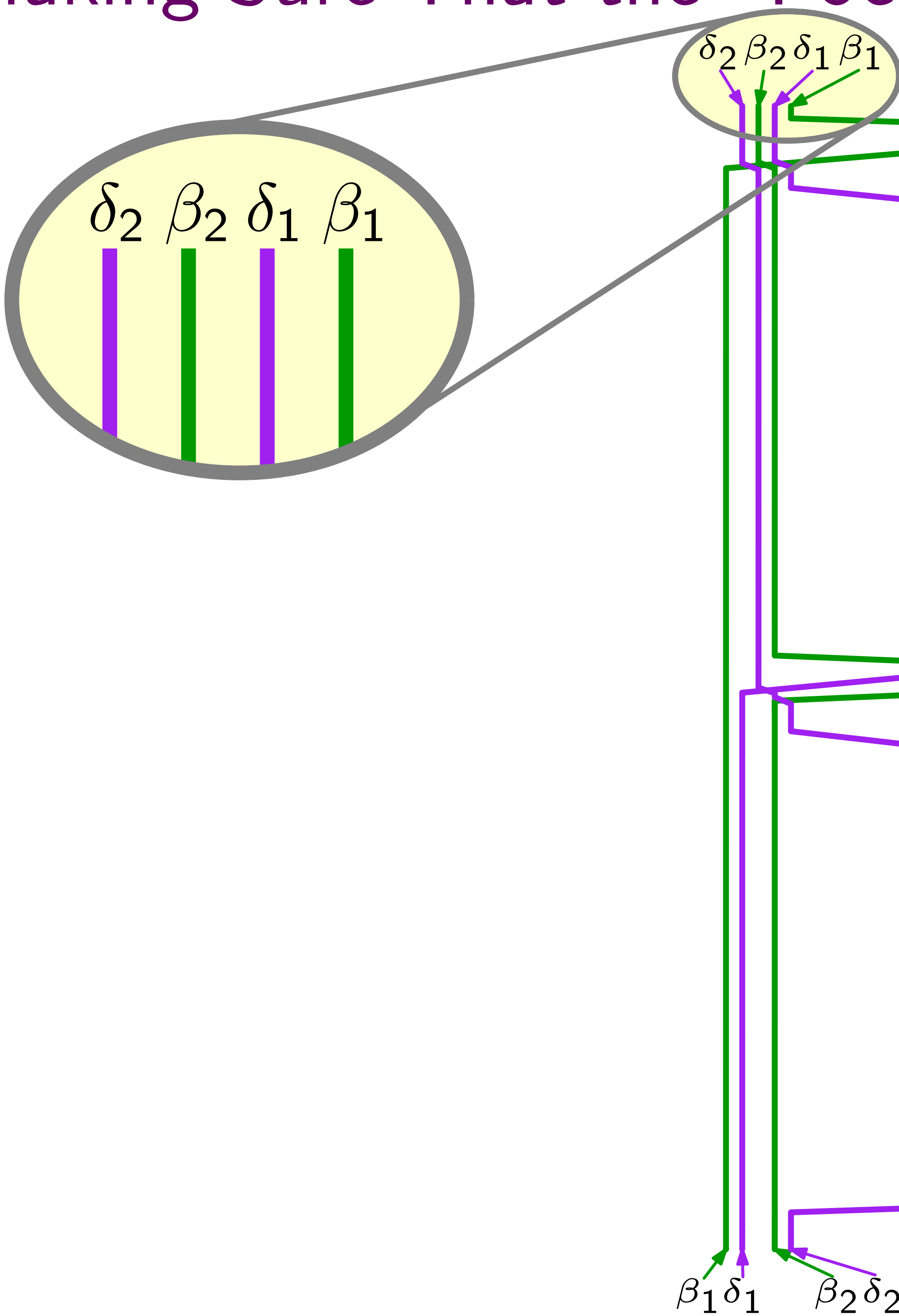
$M = 2m^3$



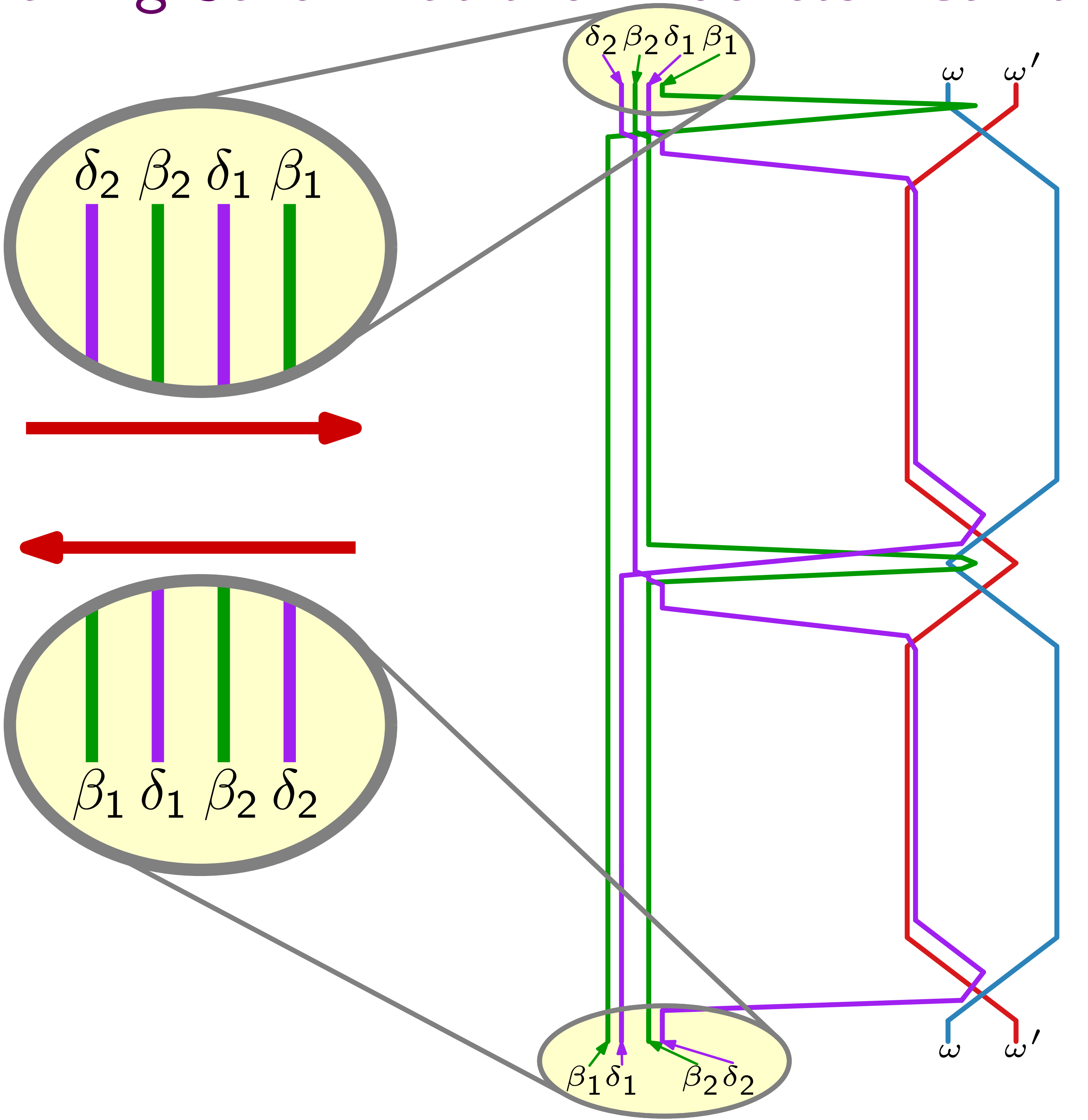
Making Sure That the "Pockets" Can't Be Squeezed



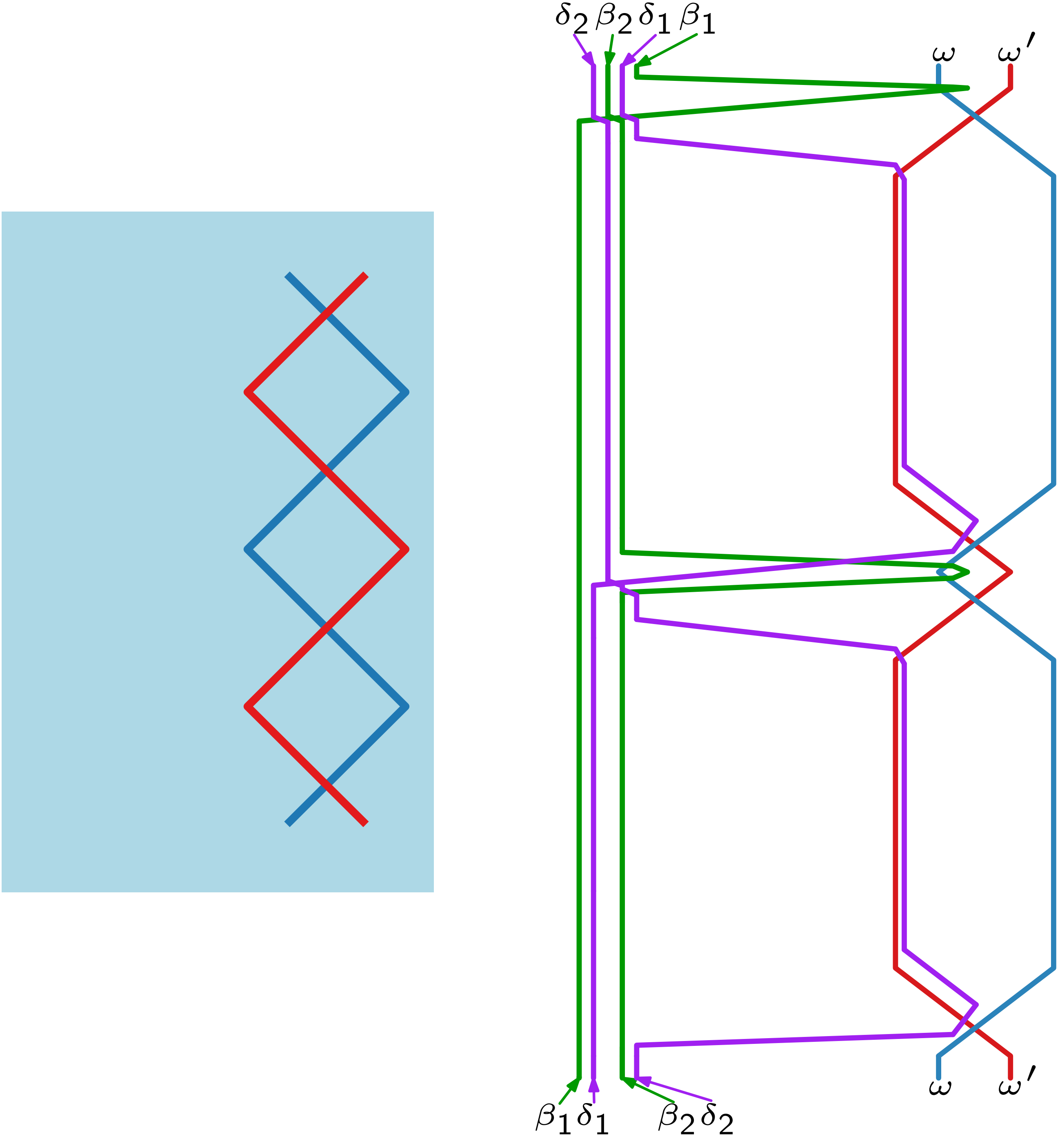
Making Sure That the "Pockets" Can't Be Squeezed



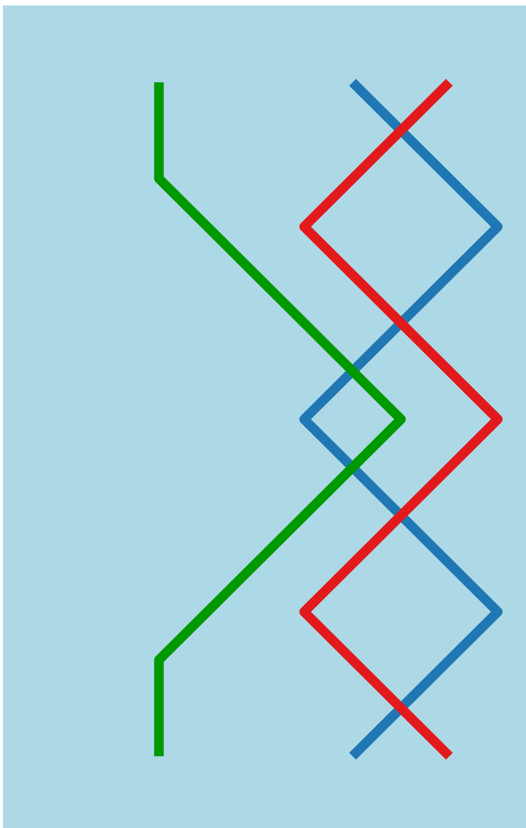
Making Sure That the “Pockets” Can’t Be Squeezed

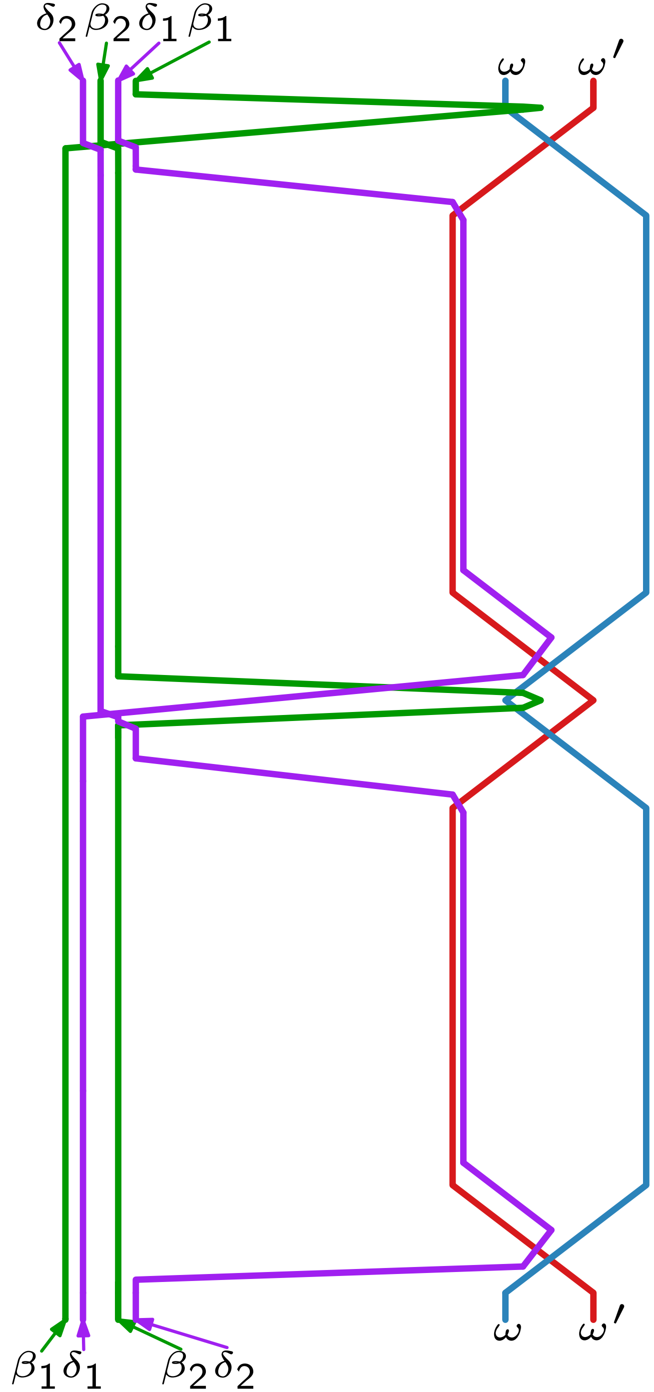


Making Sure That the “Pockets” Can’t Be Squeezed

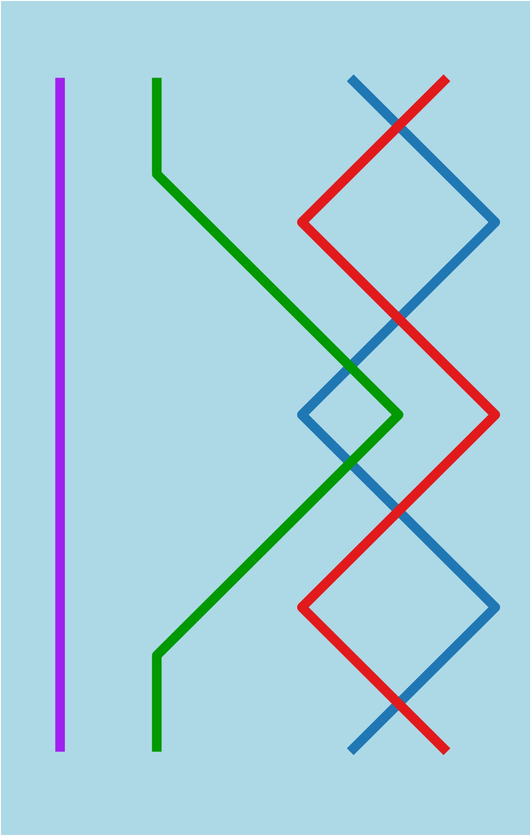


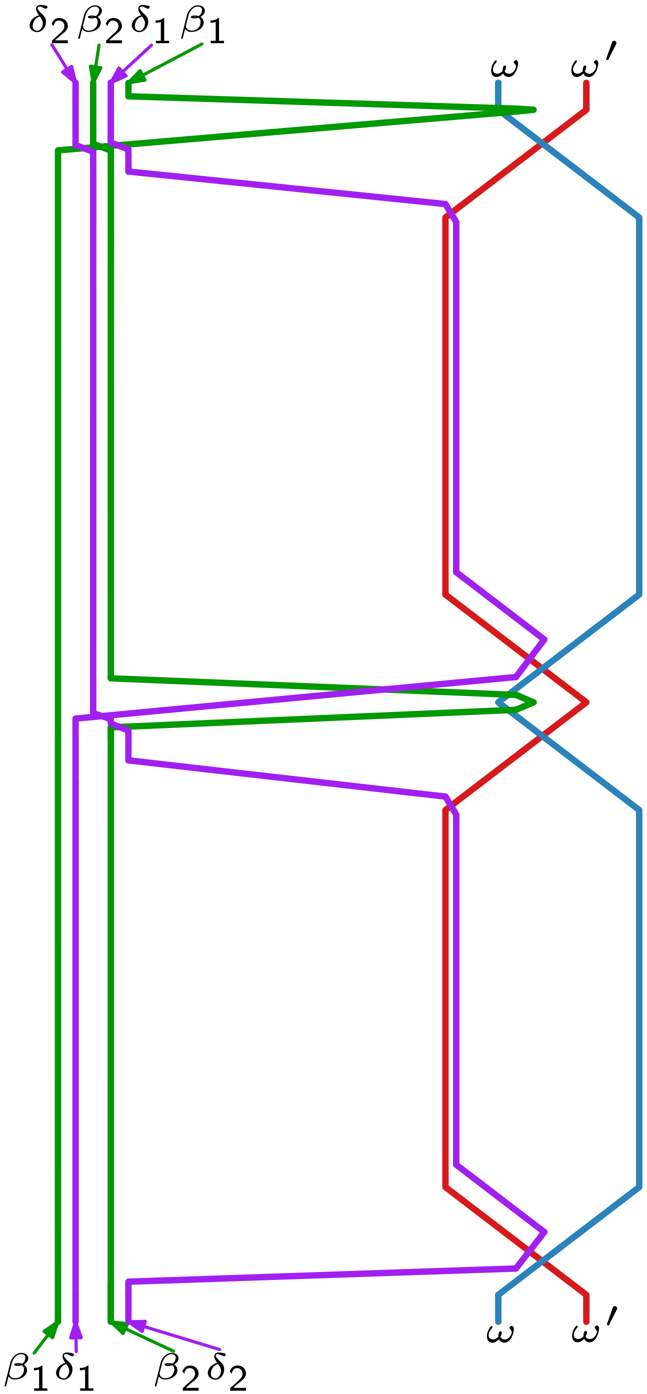
Making Sure That the "Pockets" Can't Be Squeezed



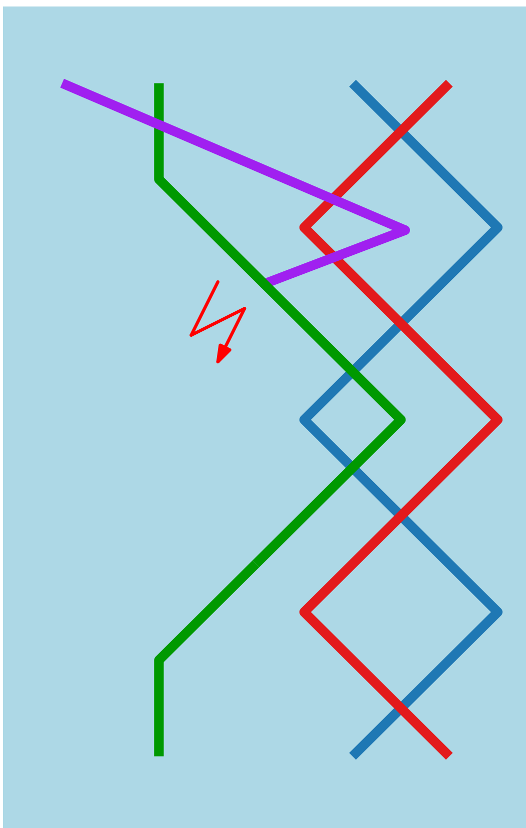


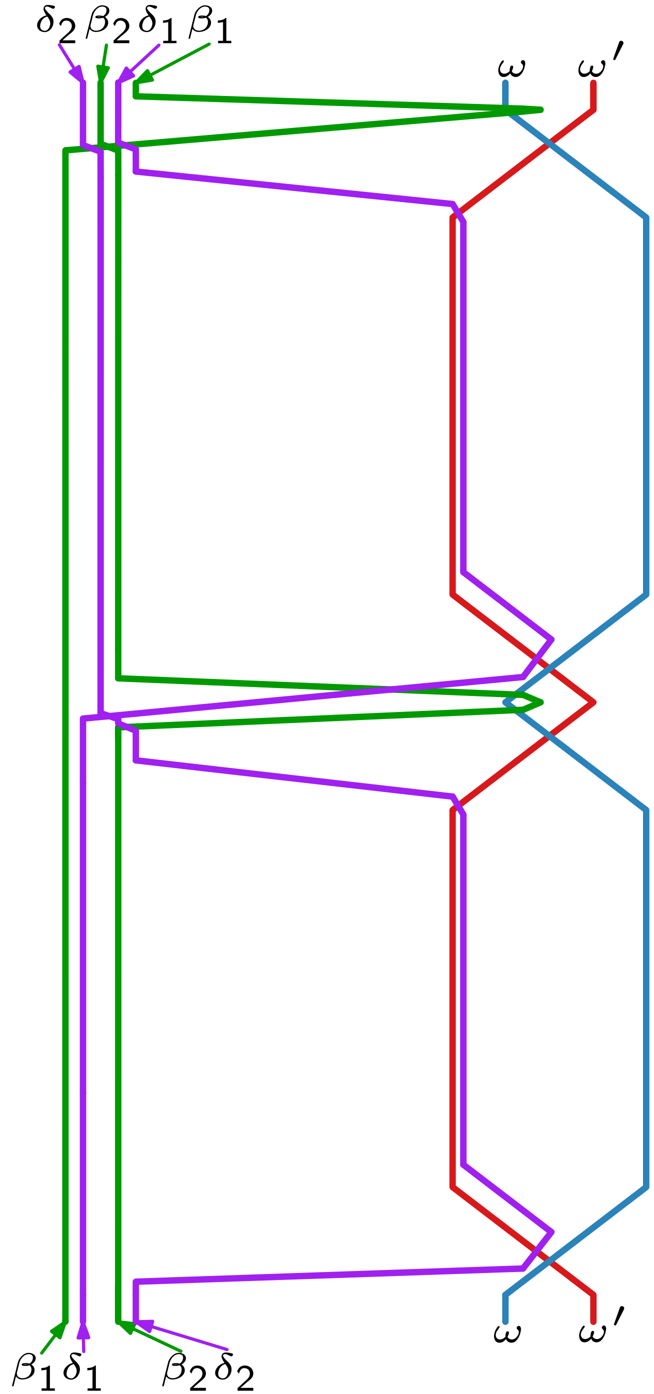
Making Sure That the "Pockets" Can't Be Squeezed



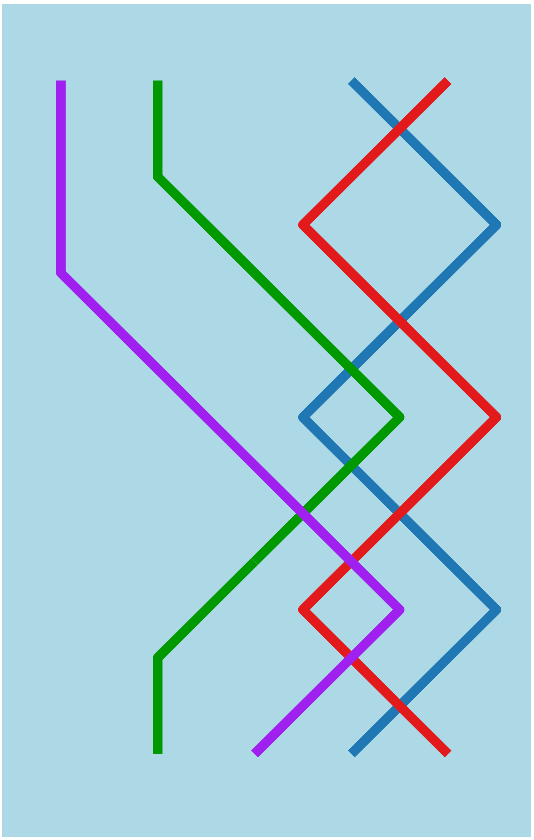


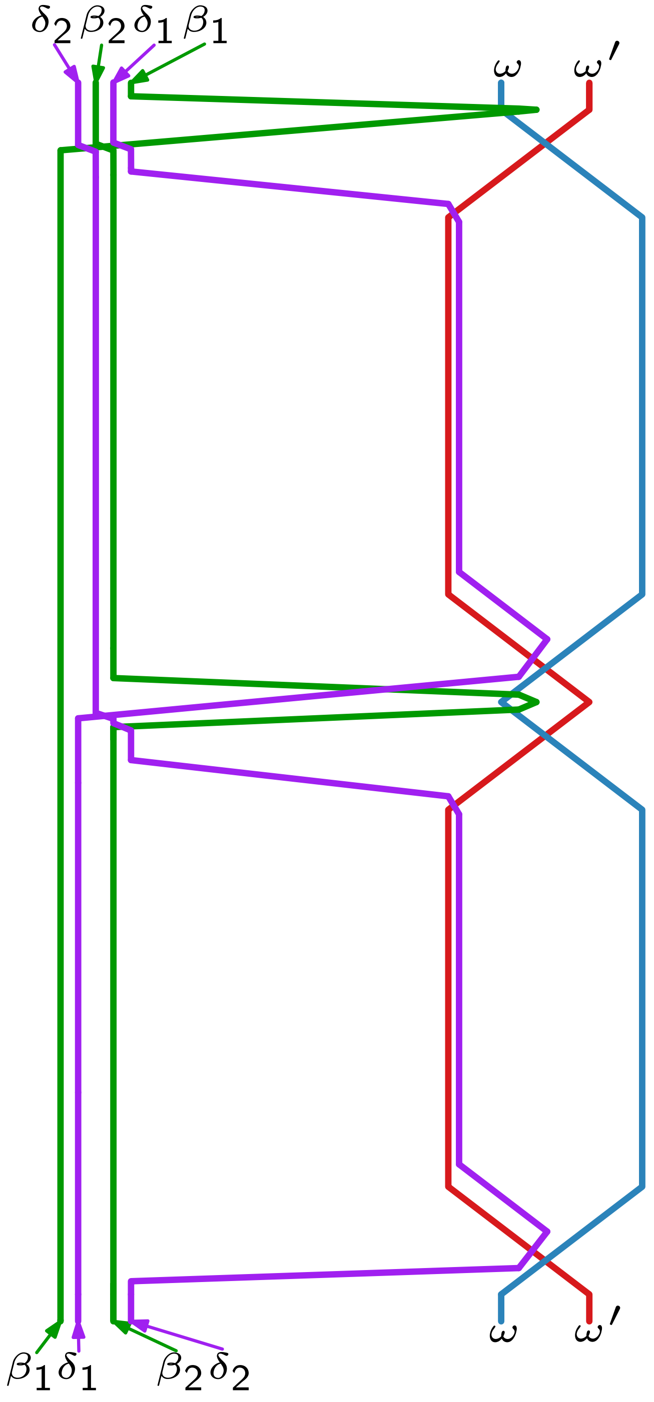
Making Sure That the “Pockets” Can’t Be Squeezed



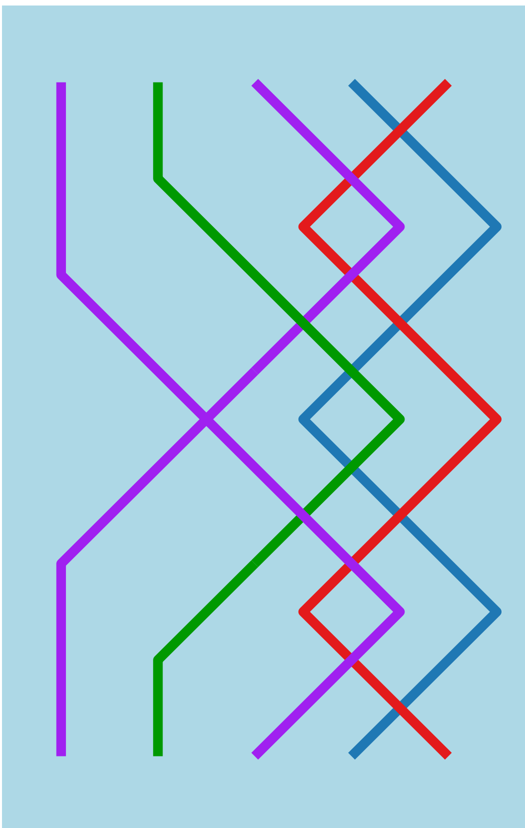


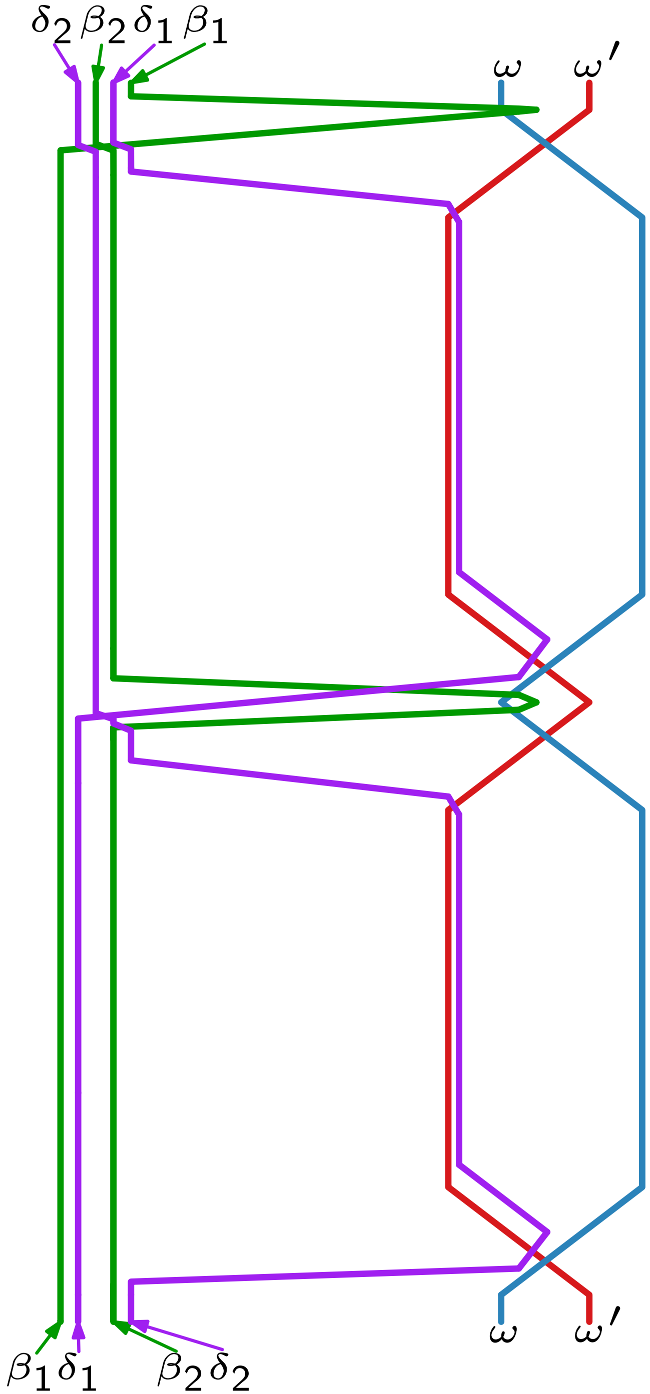
Making Sure That the “Pockets” Can’t Be Squeezed





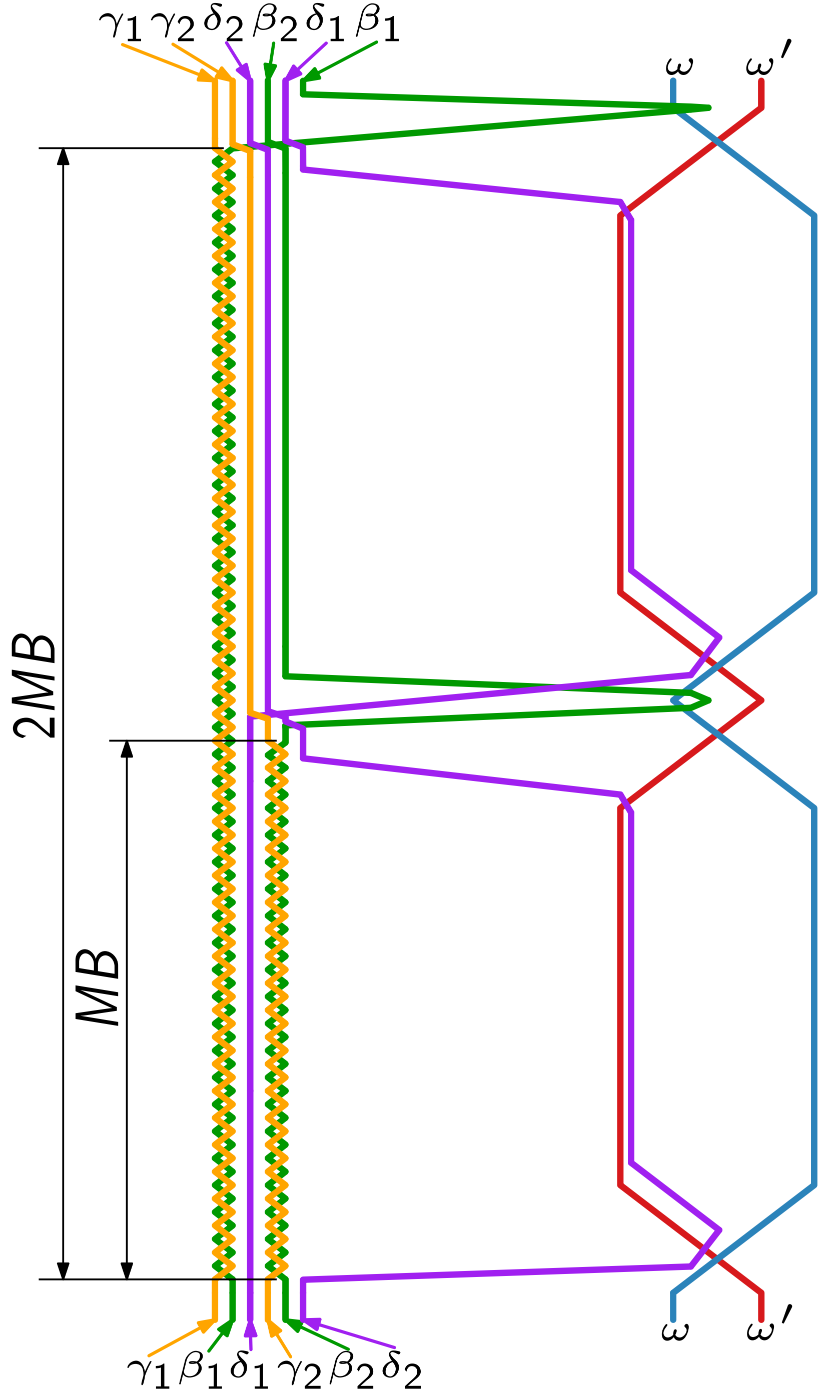
Making Sure That the "Pockets" Can't Be Squeezed





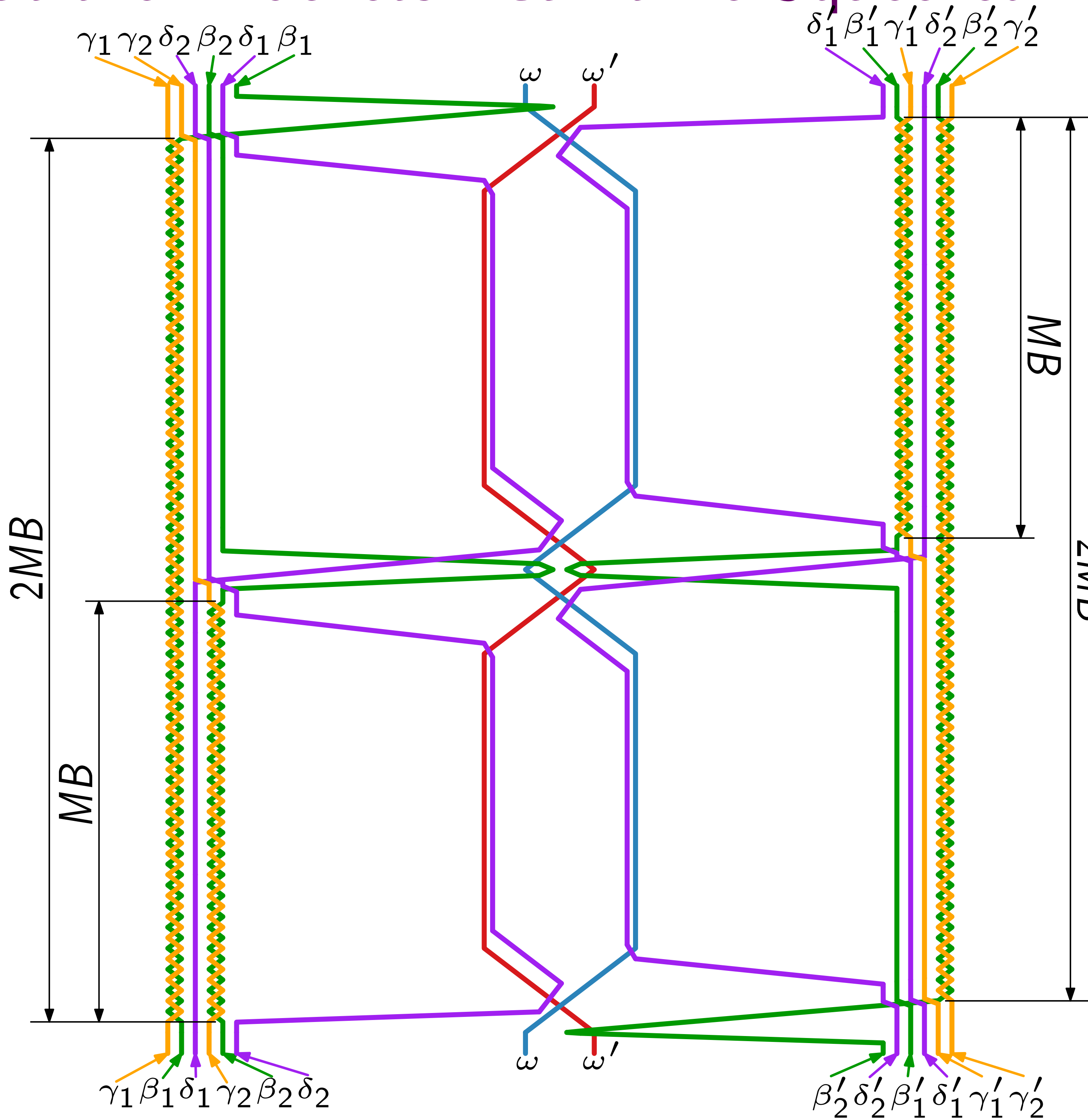
Making Sure That the "Pockets" Can't Be Squeezed

$M = 2m^3$



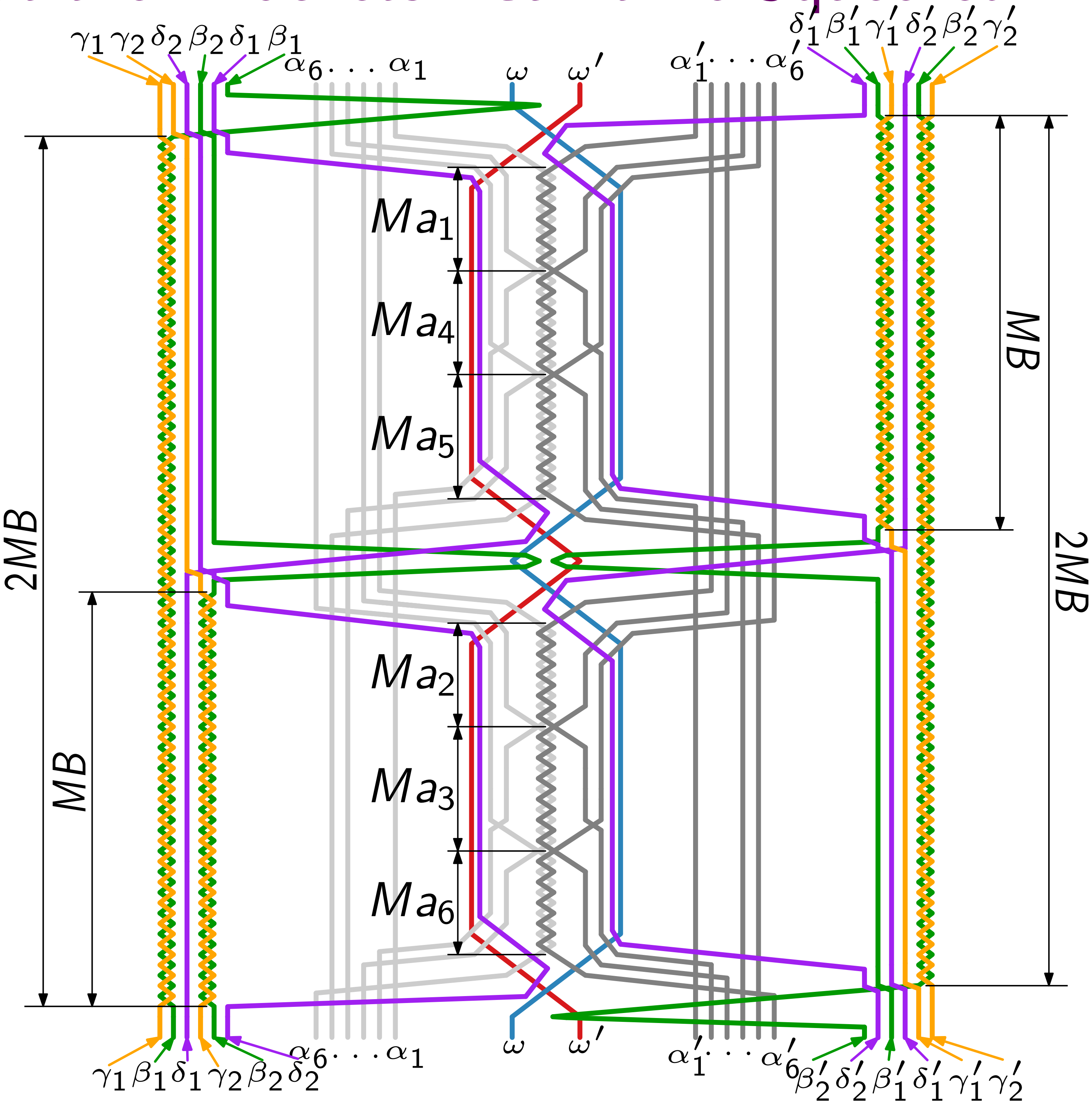
Making Sure That the “Pockets” Can’t Be Squeezed

$$M = 2m^3$$



Making Sure That the “Pockets” Can't Be Squeezed

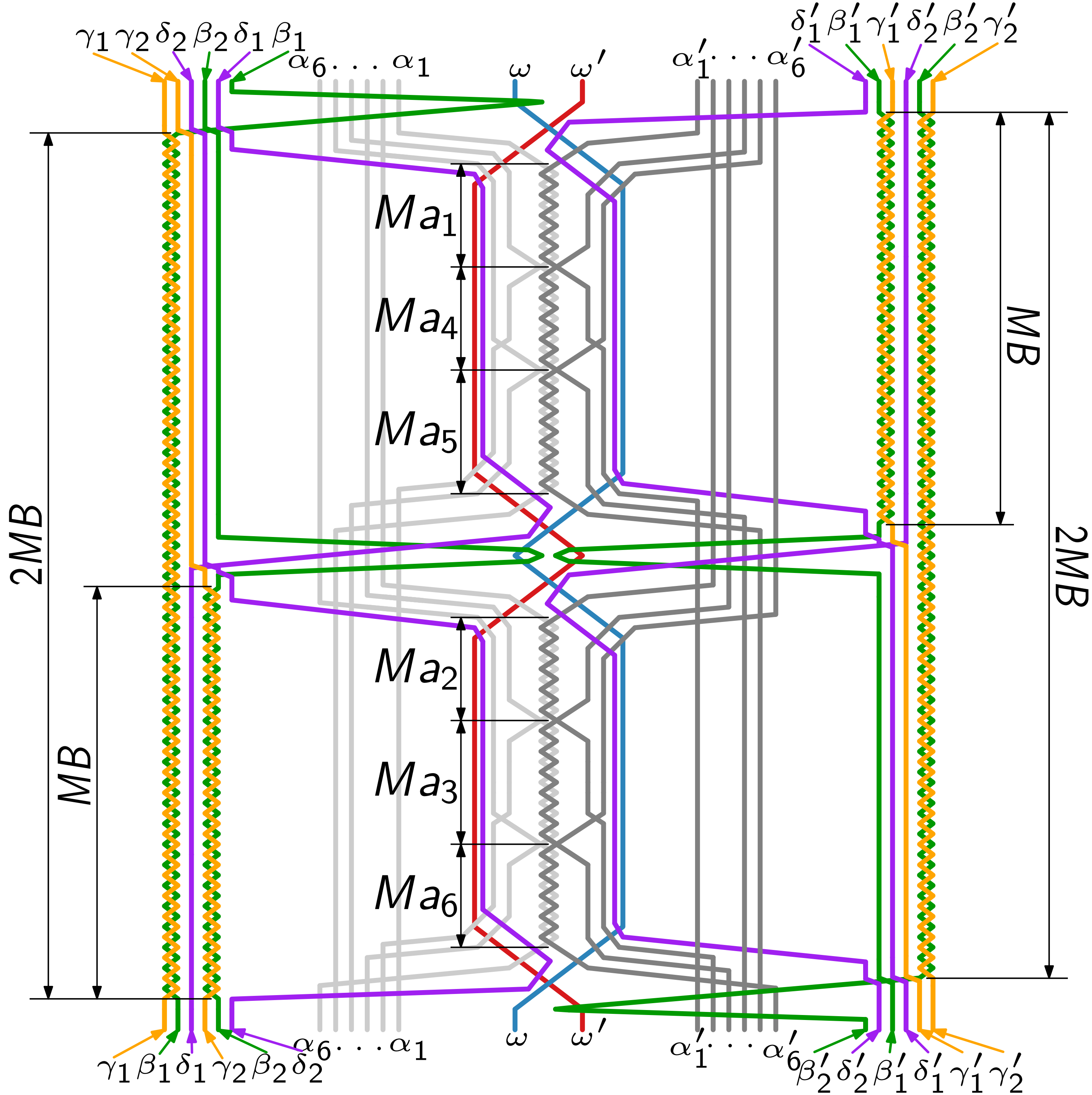
$$M = 2m^3$$



Proof of Correctness

$$M = 2m^3$$

A is a yes-instance



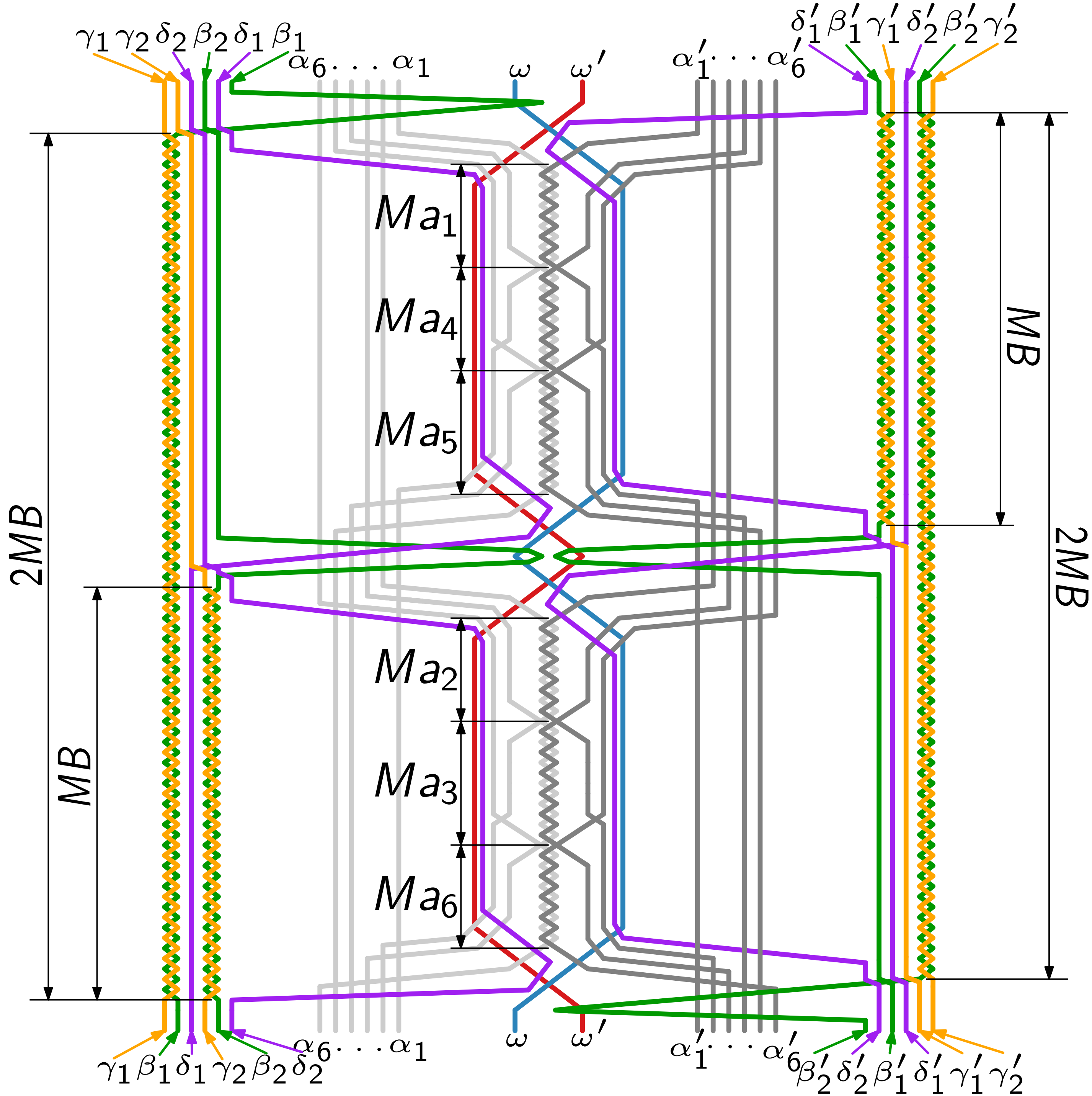
$H = 2m^3(\sum A) + 7m^2$ is the maximum allowed height for the reduction

Proof of Correctness

$M = 2m^3$

A is a yes-instance

by construction



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

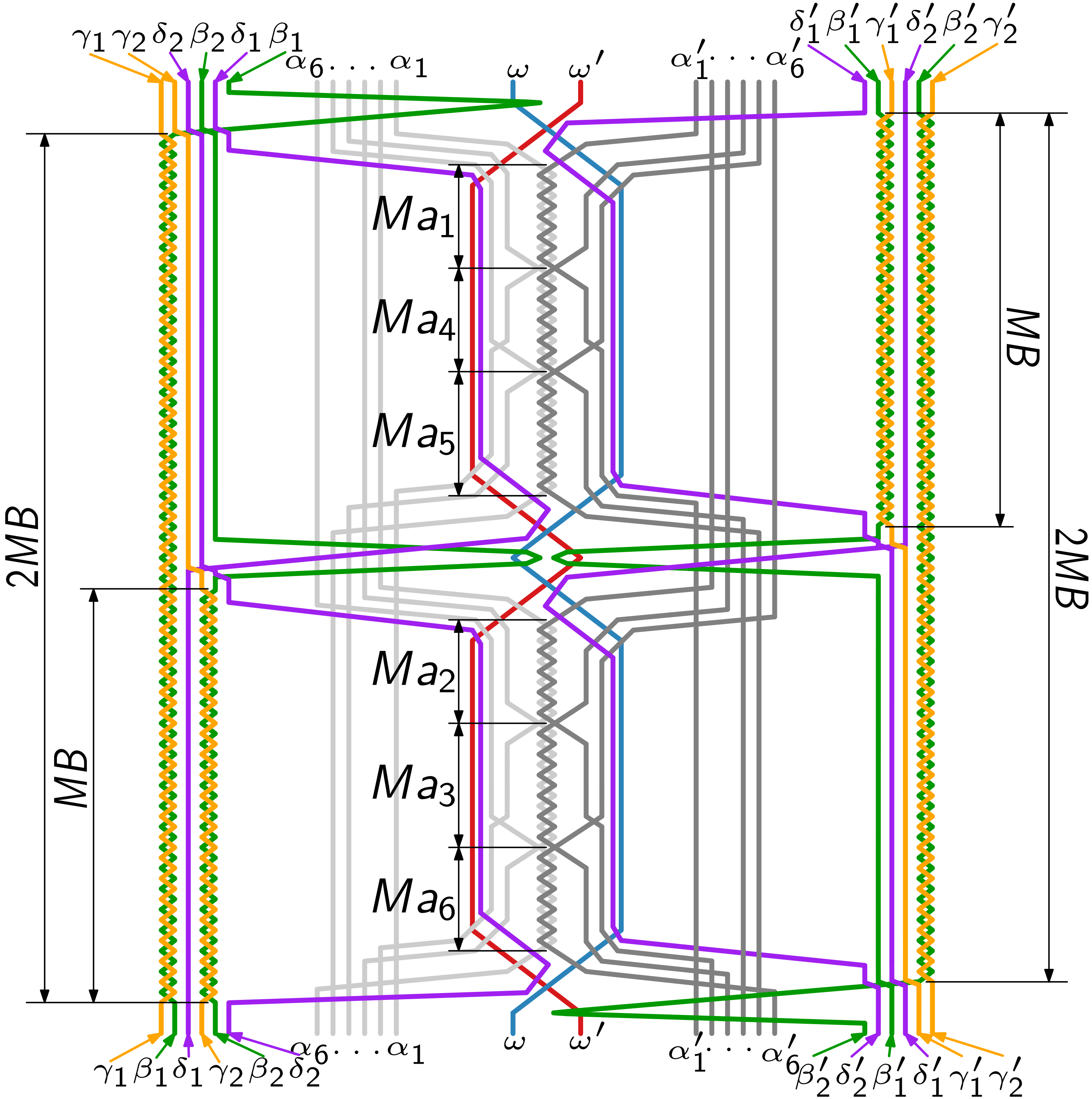
Proof of Correctness

$M = 2m^3$

A is a yes-instance

by construction

height $\leq H$

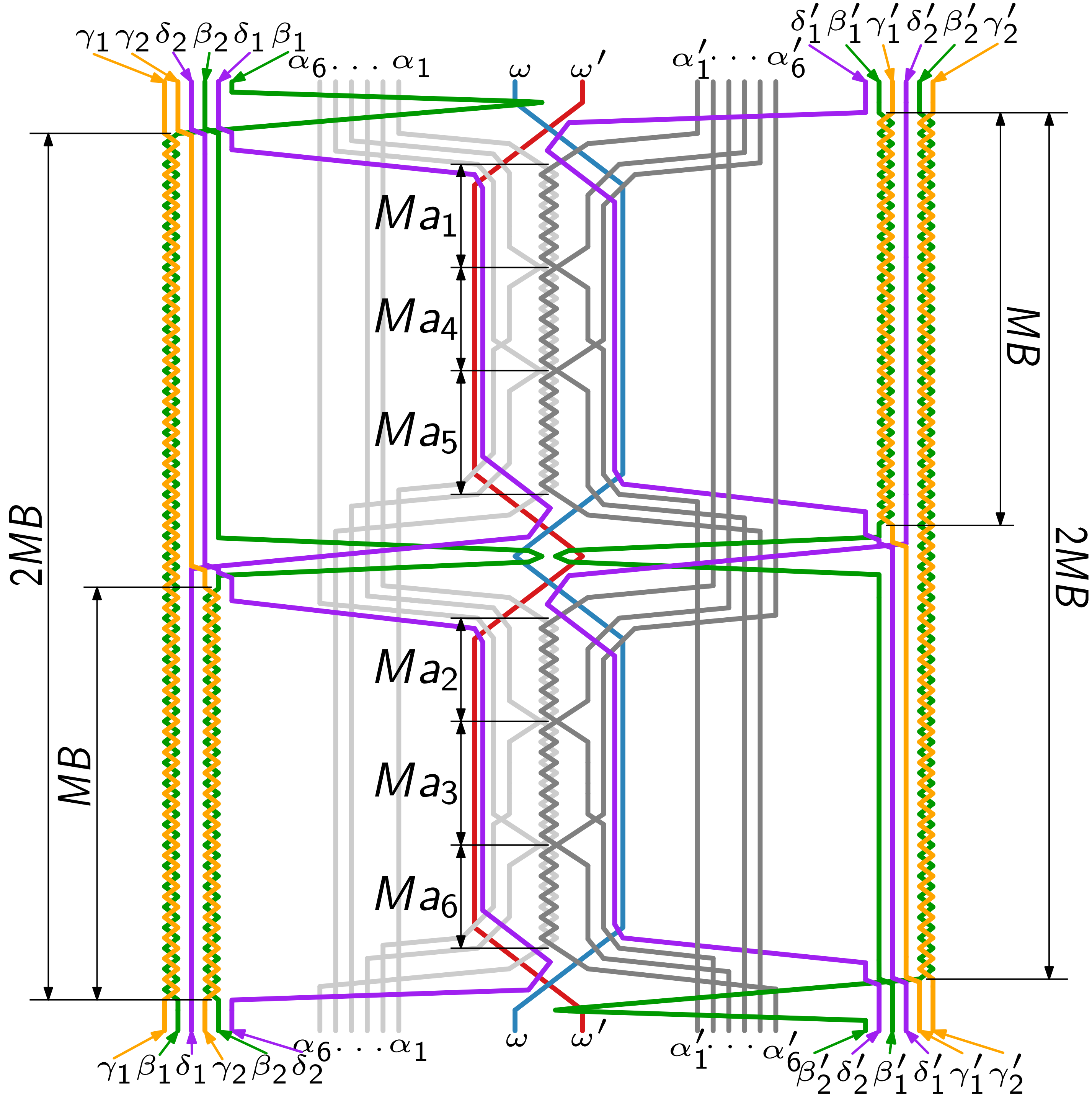


$H = 2m^3(\sum A) + 7m^2$
is the maximum allowed height for the reduction

Proof of Correctness

$$M = 2m^3$$

A is a no-instance



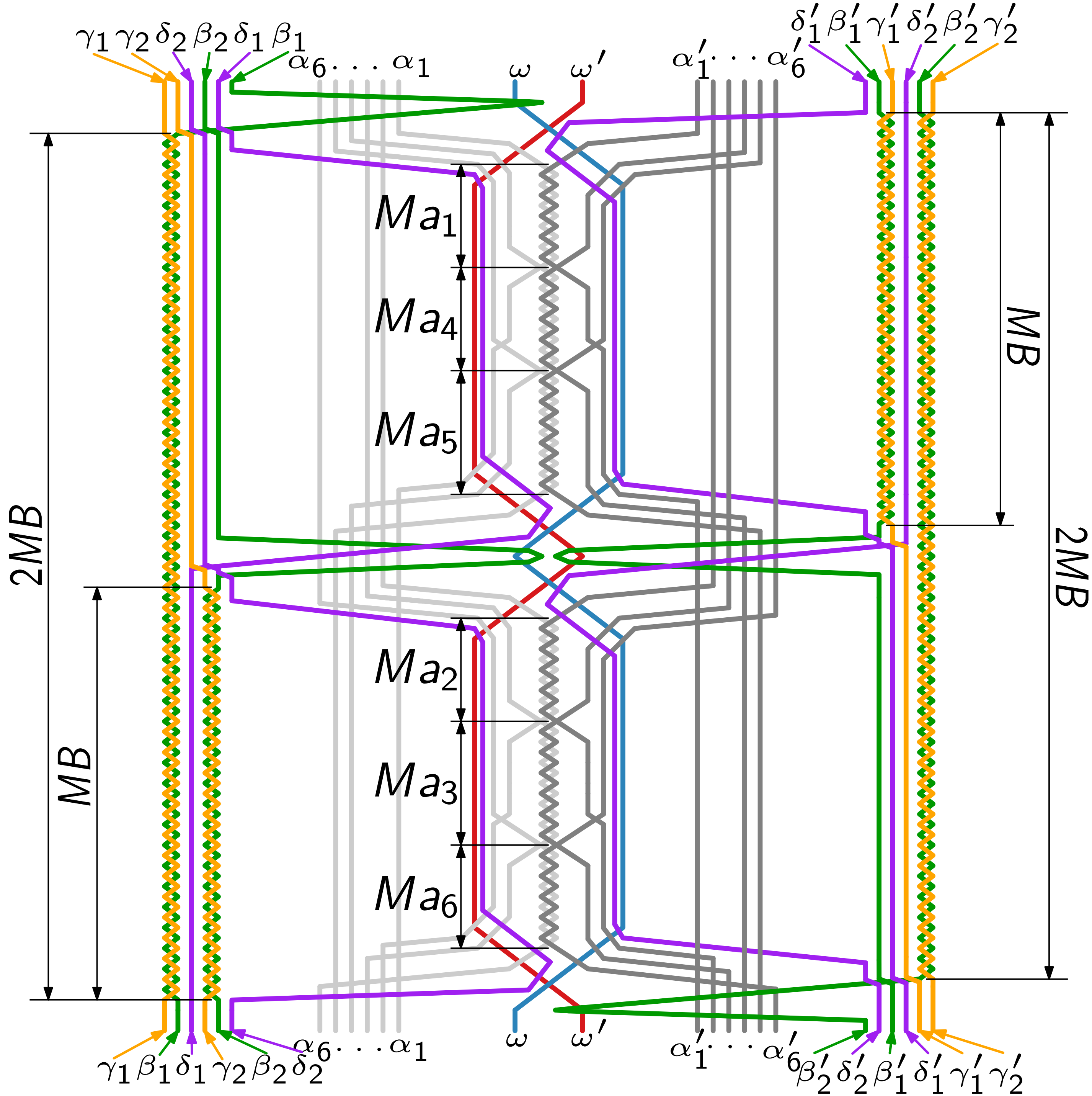
$H = 2m^3(\sum A) + 7m^2$ is the maximum allowed height for the reduction

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

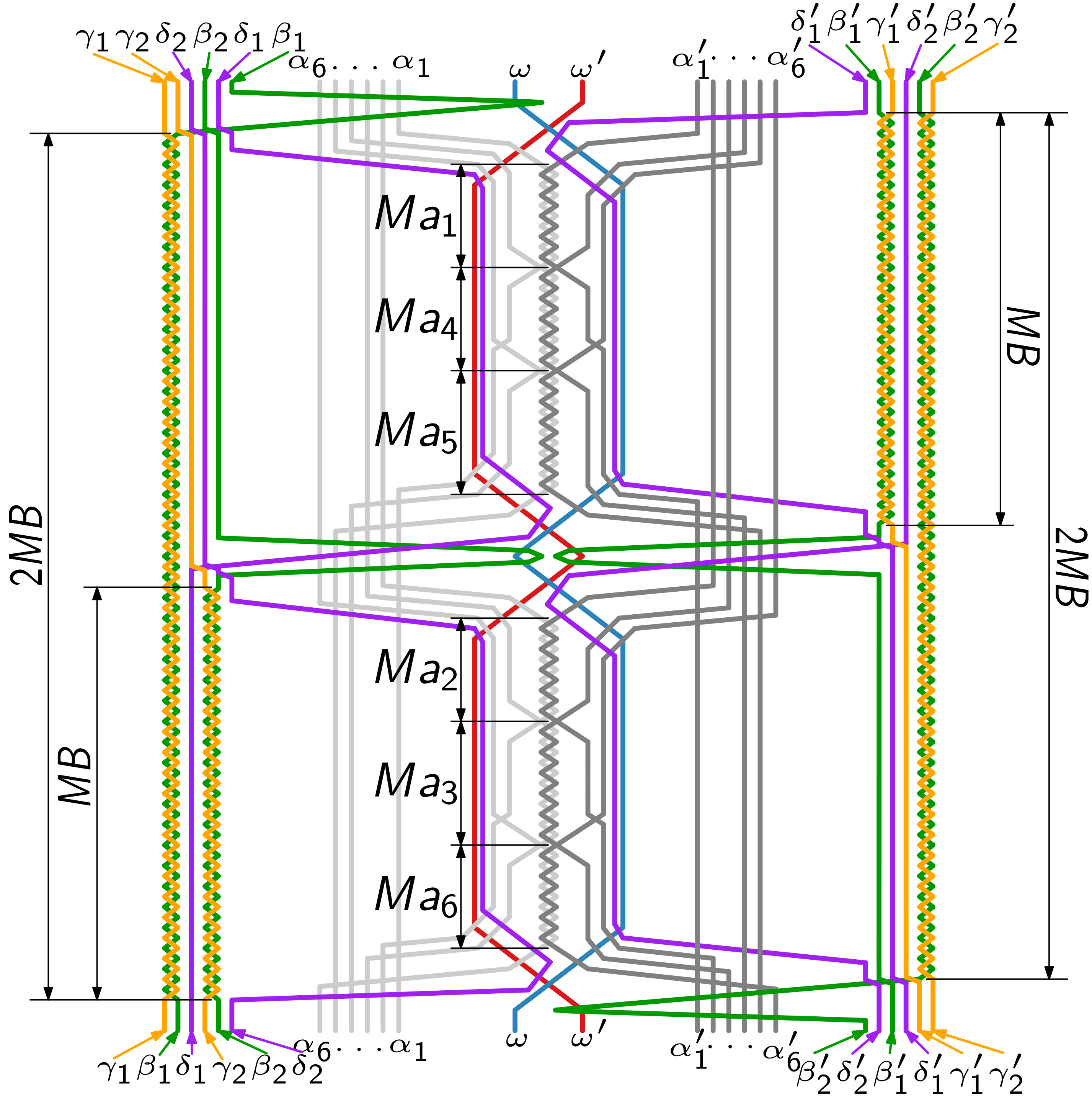
↓

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

↓

height $> H$

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



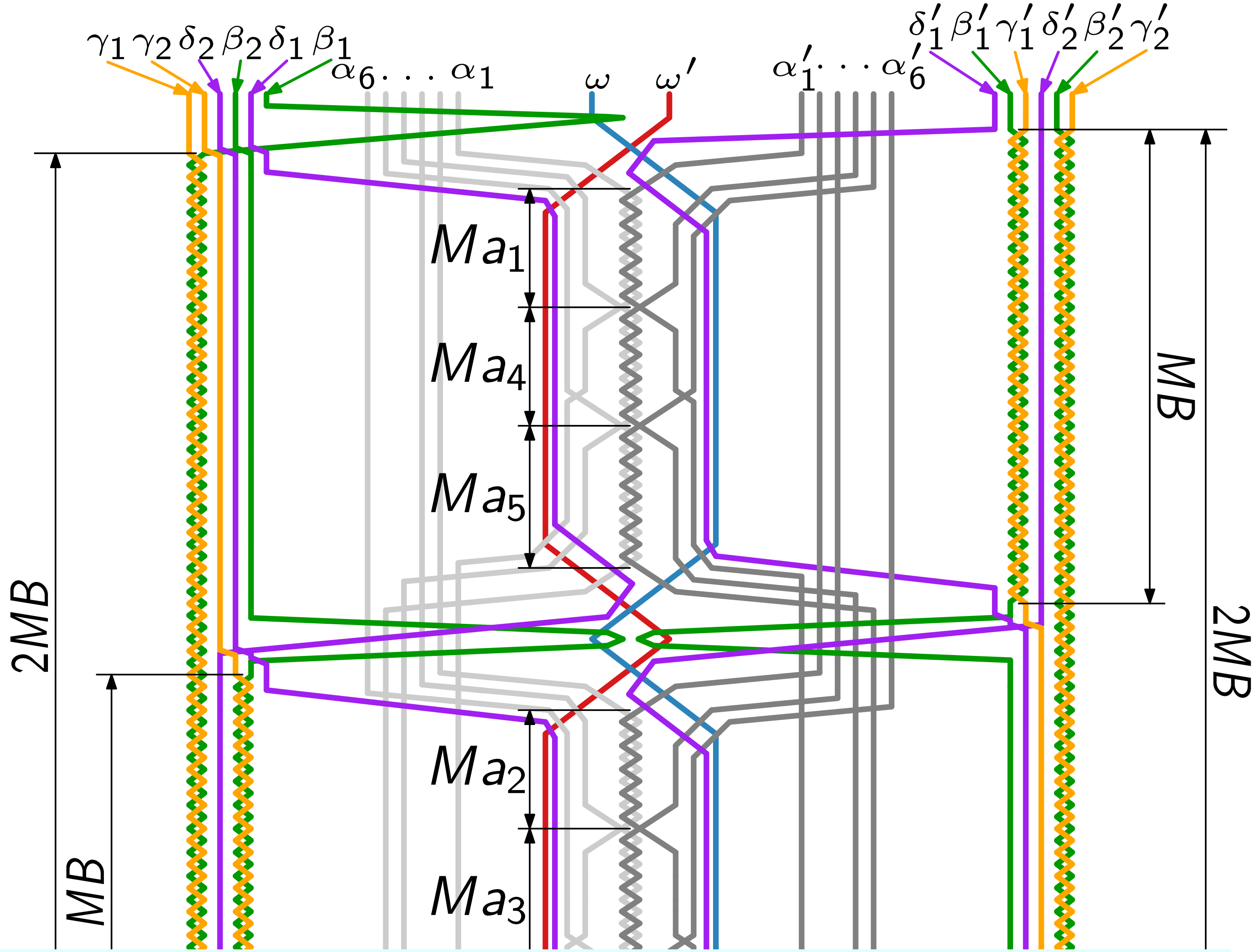
Proof of Correctness

$M = 2m^3$

A is a no-instance

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

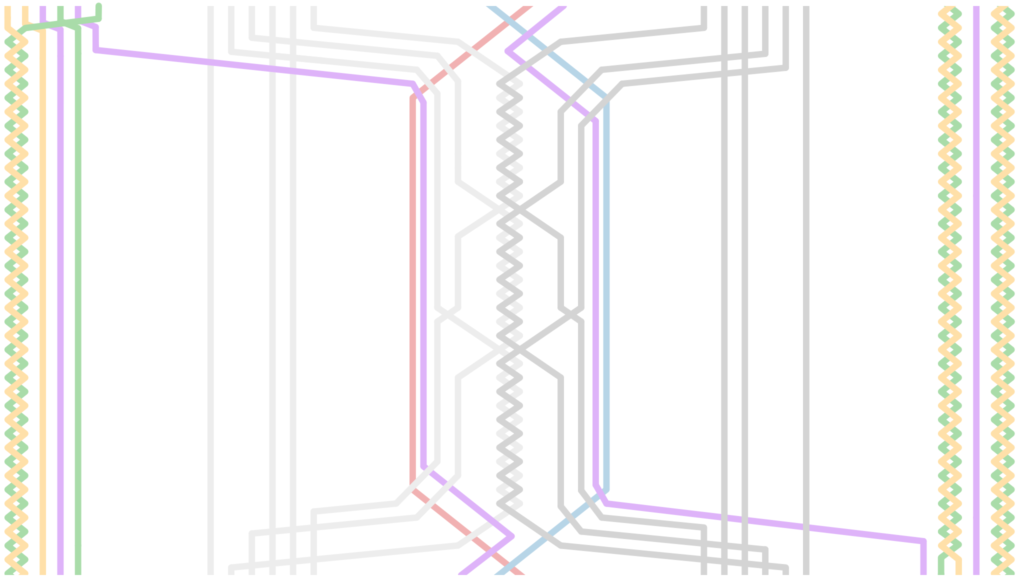
height $> H$



Theorem.
Tangle-Height Minimization is NP-hard. ✓

Overview

- **Complexity:** NP-hardness by reduction from 3-PARTITION.



- **New algorithm:** using dynamic programming; asymptotically faster than [Olszewski et al., GD'18].

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

- **Experiments:** comparison with [Olszewski et al., GD'18]

Improving Exact Algorithms

Tangle-Height Minimization can be solved in . . .

Simple lists

General lists

Improving Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in ...

n – number of wires

Simple lists

[Olszewski et al., GD'18]

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

General lists

Improving Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in . . .

n – number of wires

Simple lists

[Olszewski et al., GD'18]

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

$2^{O(n \log n)}$

our runtime

General lists

Improving Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in . . .

n – number of wires
$|L|$ – length of the list L $(= \sum \ell_{ij})$
φ – golden ratio (≈ 1.618)

Simple lists

[Olszewski et al., GD'18]

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

our runtime

$2^{O(n \log n)}$

General lists

[Olszewski et al., GD'18]

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

Improving Exact Algorithms

Tangle-Height Minimization can be solved in . . .

n – number of wires
$|L|$ – length of the list L $(= \sum \ell_{ij})$
φ – golden ratio (≈ 1.618)

Simple lists

[Olszewski et al., GD'18]

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

our runtime

$2^{O(n \log n)}$

General lists

[Olszewski et al., GD'18]

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

our runtime

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

Improving Exact Algorithms

TANGLE-HEIGHT MINIMIZATION can be solved in . . .

n – number of wires
$|L|$ – length of the list L $(= \sum \ell_{ij})$
φ – golden ratio (≈ 1.618)

Simple lists

[Olszewski et al., GD'18]

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

our runtime

$2^{O(n \log n)}$

General lists

[Olszewski et al., GD'18]

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

our runtime

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

polynomial in $|L|$
for fixed n

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

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

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

Let L' be the next list to consider.

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

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

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

Let L' be the next list to consider.

Check its **consistency**.

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

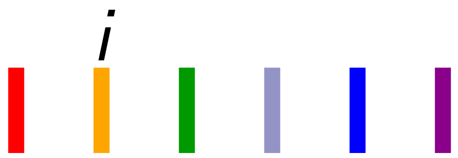
$\lambda = \#$ of **distinct sublists** of L.

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

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

Let L' be the next list to consider.

Check its **consistency**.



Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

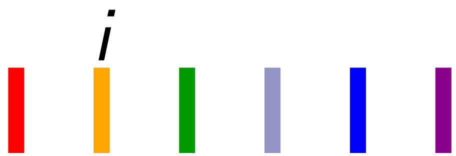
$\lambda = \#$ of **distinct sublists** of L.

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

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

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

Let $L = (\ell_{ij})$ be the given list of swaps.

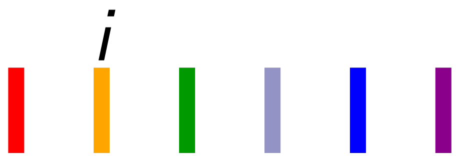
$\lambda = \#$ of **distinct sublists** of L.

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

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

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

Let $L = (\ell_{ij})$ be the given list of swaps.

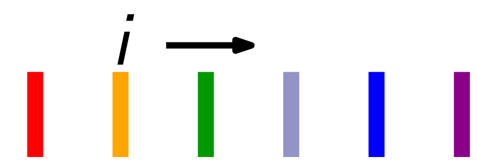
$\lambda = \#$ of **distinct sublists** of L.

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

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

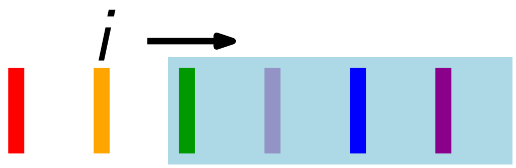
$\lambda = \#$ of **distinct sublists** of L.

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

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'

  $i \mapsto i + |\{j : j > i \text{ and } \ell'_{ij} \text{ is odd}\}|$

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

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'

$i \mapsto i + |\{j : j > i \text{ and } \ell'_{ij} \text{ is odd}\}| - |\{j : j < i \text{ and } \ell'_{ij} \text{ is odd}\}|$

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

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'

$i \mapsto i + |\{j : j > i \text{ and } \ell'_{ij} \text{ is odd}\}| - |\{j : j < i \text{ and } \ell'_{ij} \text{ is odd}\}|$

check whether the map is indeed a permutation

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

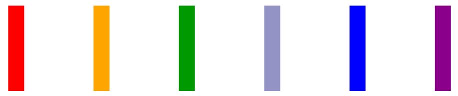
$\lambda = \#$ of **distinct sublists** of L.

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

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

Let L' be the next list to consider.

Check its **consistency**.



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

$\mathrm{id}_n\, L'$

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

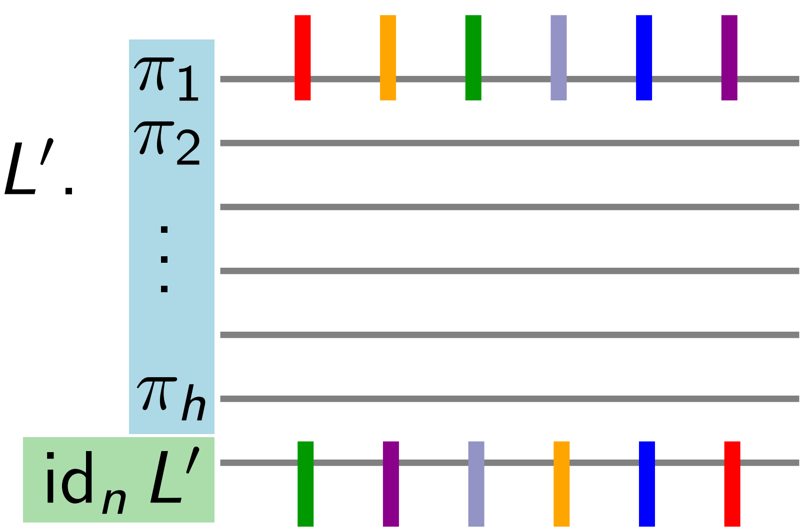
Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

Compute 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

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

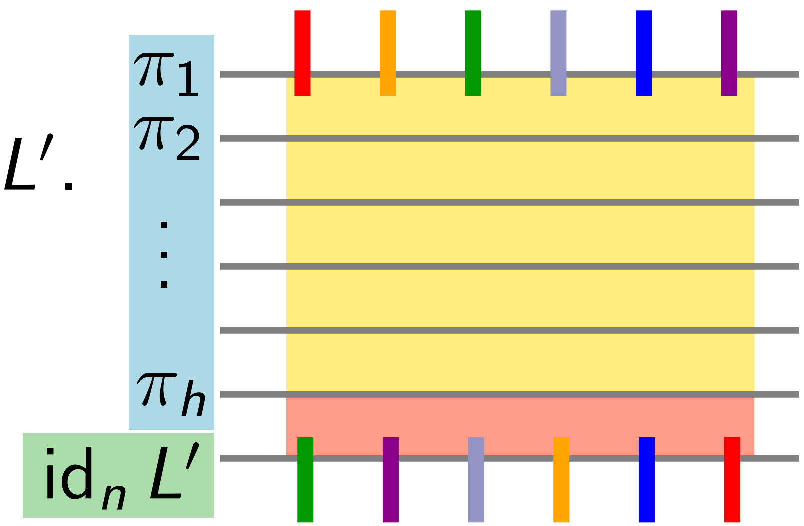
Consider them in order of **increasing length**.

Let L' be the next list to consider.

Check its **consistency**.

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

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



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

$L'' \cup$ add. swaps $= L'$

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

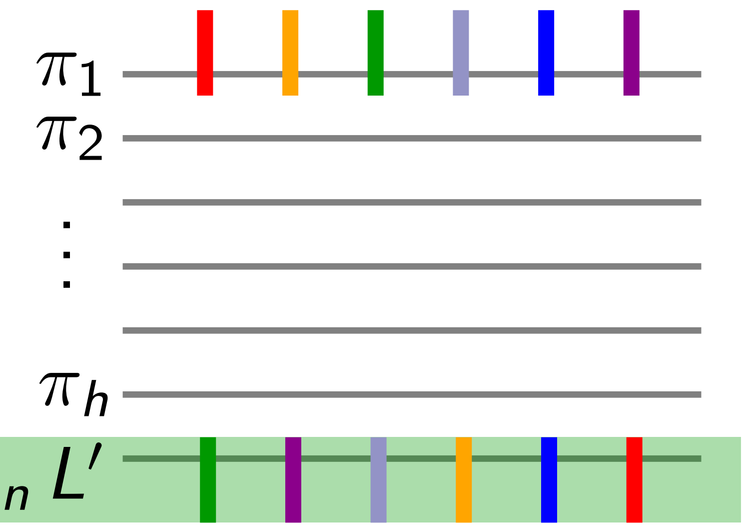
Let L' be the next list to consider.

Check its **consistency**.

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

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

Add the final permutation to its end.



Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

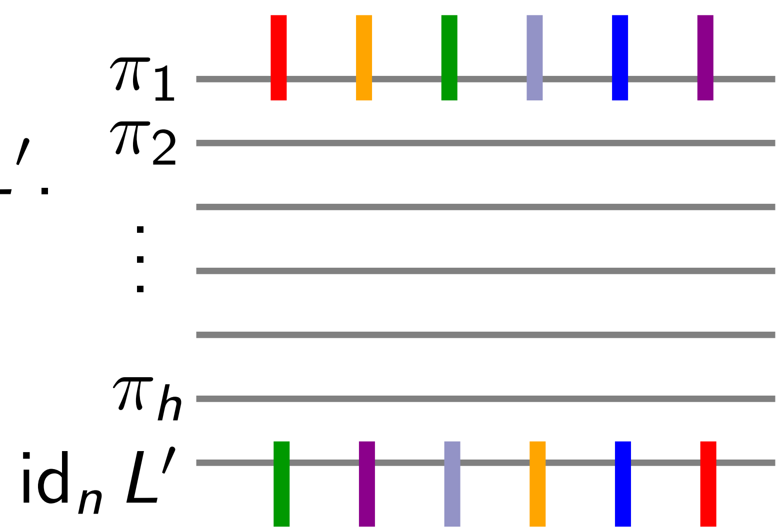
Let L' be the next list to consider.

Check its **consistency**.

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

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

Add the final permutation to its end.



Running time

$O(\qquad\qquad\qquad)$

Dynamic Programming Algorithm

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

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

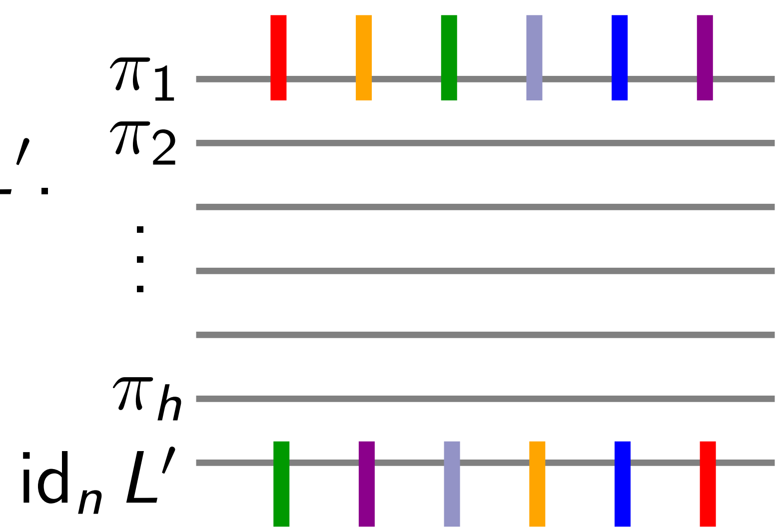
Let L' be the next list to consider.

Check its **consistency**.

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

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

Add the final permutation to its end.



Running time

$O(\lambda \cdot \qquad\qquad\qquad)$

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

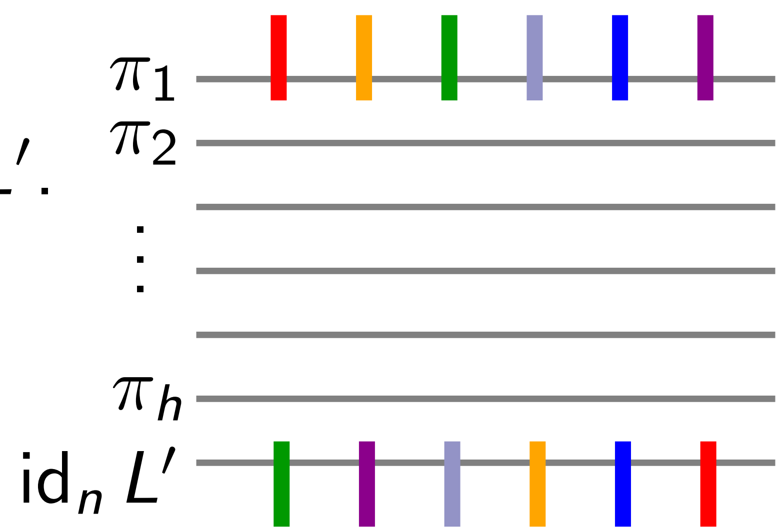
Let L' be the next list to consider.

Check its **consistency**.

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

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

Add the final permutation to its end.



Running time

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

F_n is the n-th Fibonacci number

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

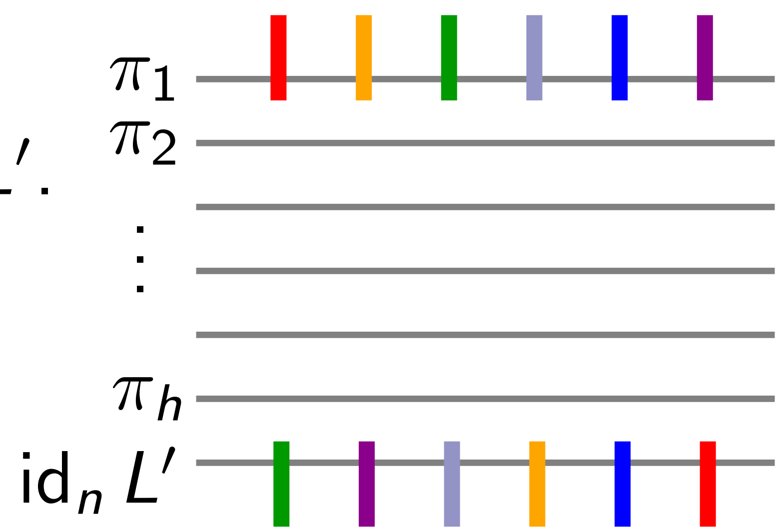
Let L' be the next list to consider.

Check its **consistency**.

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

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

Add the final permutation to its end.



Running time

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

$$\lambda = \prod_{i<j} (\ell_{ij} + 1) \leq \left(\frac{2|L|}{n^2} + 1\right)^{n^2/2}$$

$F_n \in O(\varphi^n)$

Dynamic Programming Algorithm

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

Let $L = (\ell_{ij})$ be the given list of swaps.

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

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

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

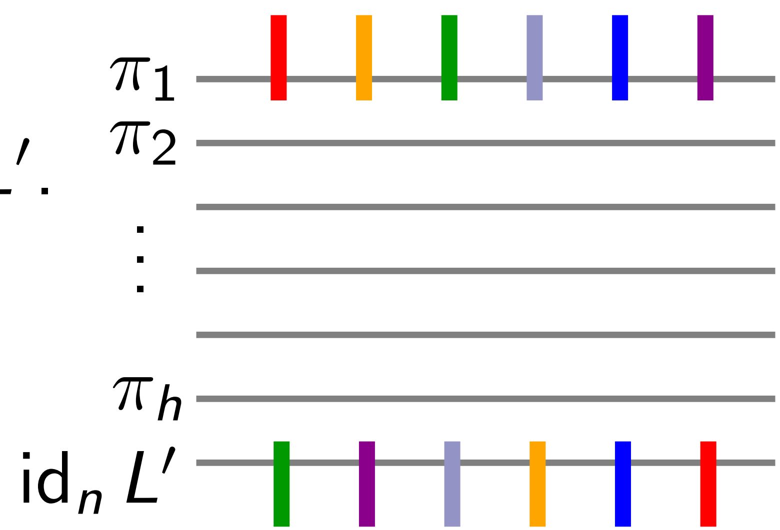
Let L' be the next list to consider.

Check its **consistency**.

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

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

Add the final permutation to its end.



Running time

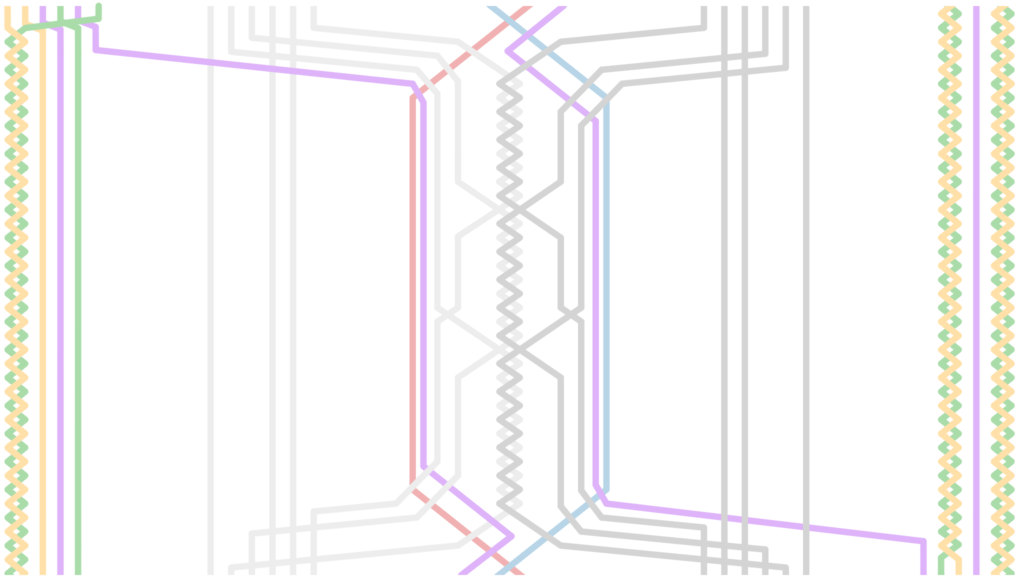
$O(\lambda \cdot (F_{n+1} - 1) \cdot n) \leq$

$\lambda = \prod_{i<j}(\ell_{ij}+1) \leq \left(\frac{2|L|}{n^2}+1\right)^{n^2/2}$

$F_n \in O(\varphi^n)$

Overview

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

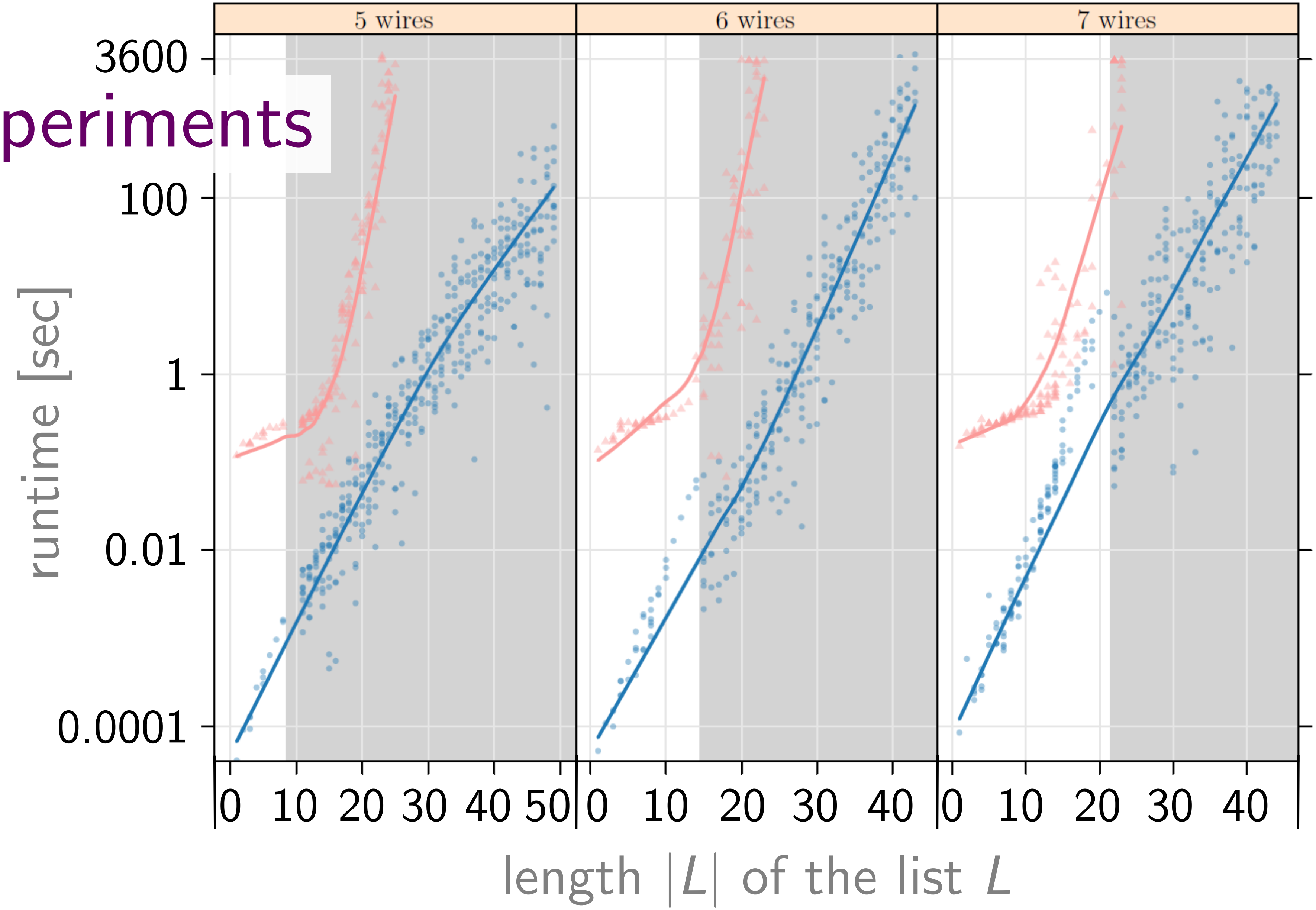


- **New algorithm:** using dynamic programming; asymptotically faster than [Olszewski et al., GD'18].

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

- **Experiments:** comparison with [Olszewski et al., GD'18]

Experiments



[Olszewski et al., GD'18]

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

Our algorithm

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

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

Can we decide a feasibility of a list faster than finding its optimal realization?

Open Problems

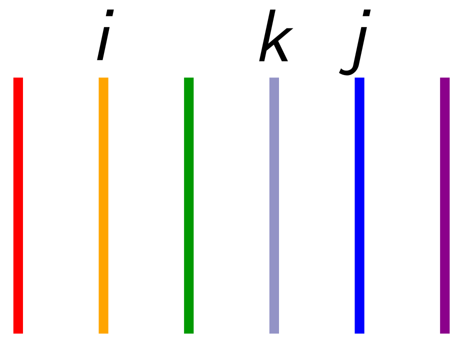
Problem 1

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

Problem 2

Can we decide a feasibility of a list faster than finding its optimal realization?

Problem 3



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

Open Problems

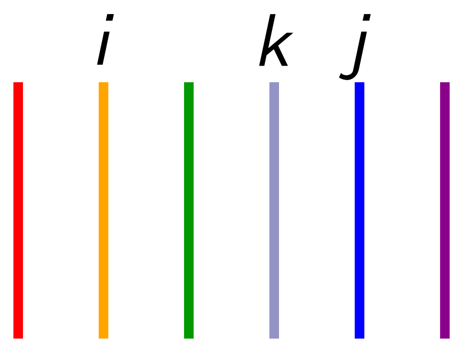
Problem 1

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

Problem 2

Can we decide a feasibility of a list faster than finding its optimal realization?

Problem 3



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

necessary

Open Problems

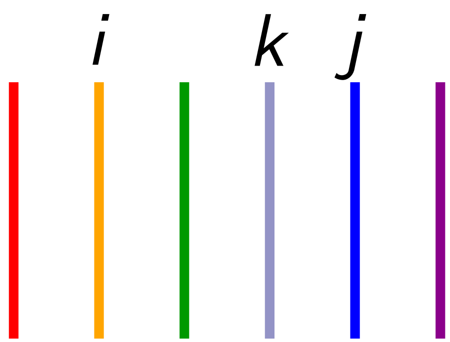
Problem 1

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

Problem 2

Can we decide a feasibility of a list faster than finding its optimal realization?

Problem 3



A list (ℓ_{ij}) is *non-separable* if $\forall i{<}k{<}j$: $\big(\ell_{ik} = \ell_{kj} = 0 \text{ implies } \ell_{ij} = 0\big)$.

necessary

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

Open Problems

Thank you!

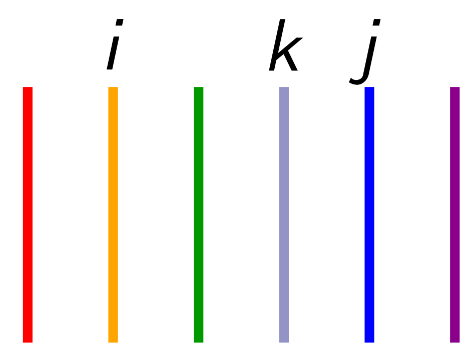
Problem 1

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

Problem 2

Can we decide a feasibility of a list faster than finding its optimal realization?

Problem 3



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

necessary

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