Sorting Genomes by Prefix Double-Cut-and-Joins

Guillaume Fertin Géraldine Jean **Anthony Labarre**

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

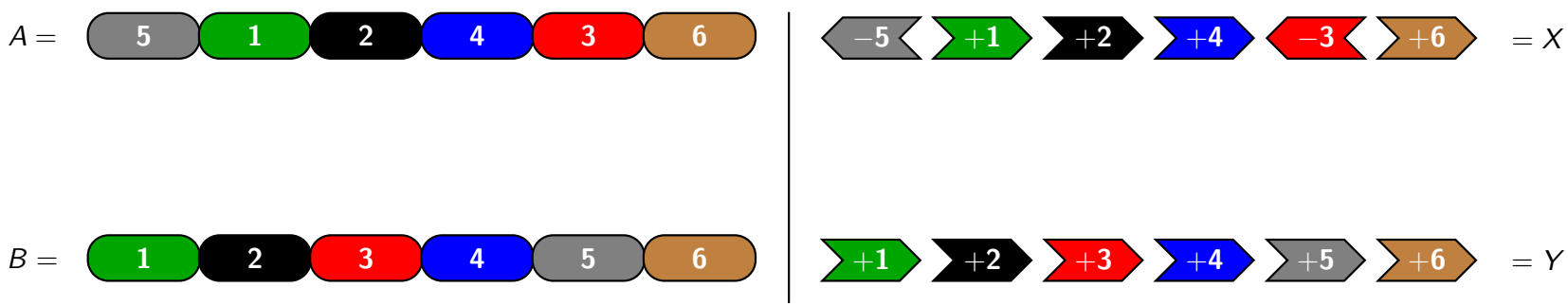
March 23rd, 2023

UNIVERSITÉ DE NANTES

Genome rearrangements for permutations

- (Signed) permutations model duplication-free genomes with the same contents;
- The actual numbering is irrelevant, so we assume either genome is the **identity** $\mathrm{Id} = \langle 1\ 2\ \cdots\ n \rangle$;

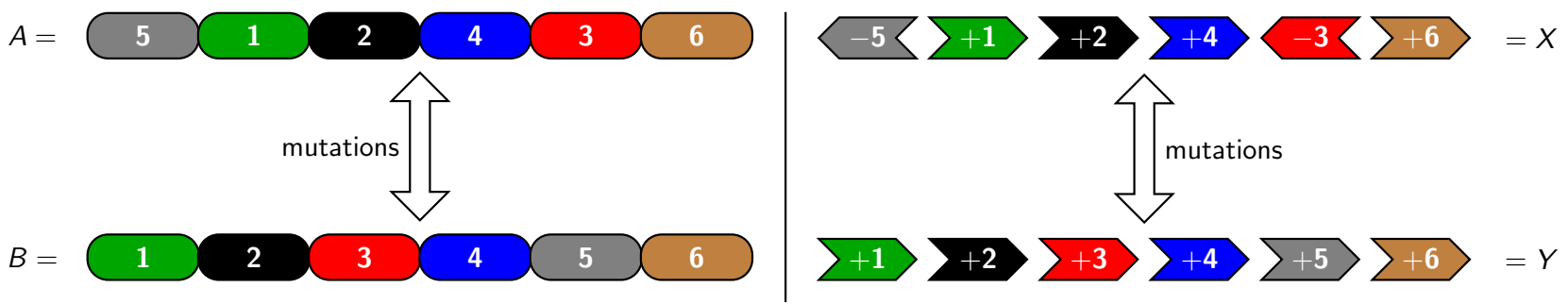
Example (disregarding / considering gene orientation)



Genome rearrangements for permutations

- (Signed) permutations model duplication-free genomes with the same contents;
- The actual numbering is irrelevant, so we assume either genome is the **identity** $\mathrm{Id} = \langle 1\ 2\ \cdots\ n \rangle$;
- We aim to reconstruct evolutionary scenarios between species;

Example (disregarding / considering gene orientation)



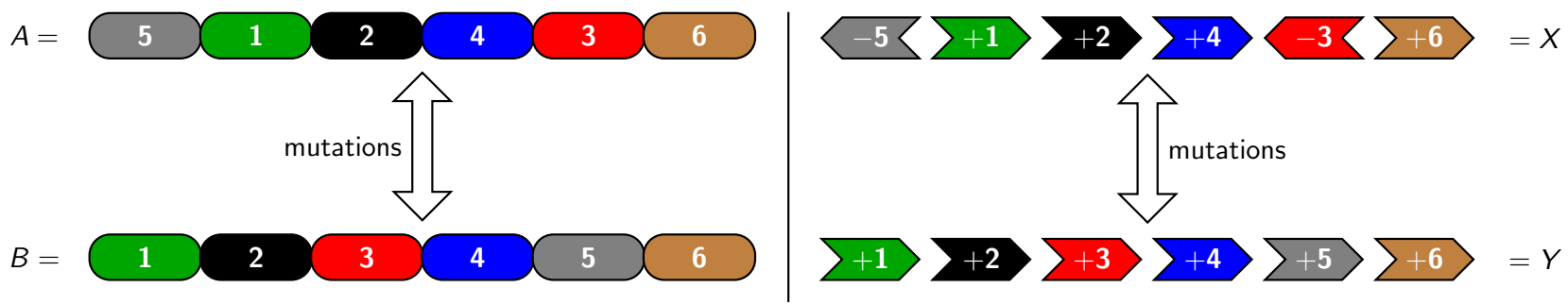
Genome rearrangements for permutations

- (Signed) permutations model duplication-free genomes with the same contents;
- The actual numbering is irrelevant, so we assume either genome is the **identity** $\mathrm{Id} = \langle 1\ 2\ \cdots\ n \rangle$;
- We aim to reconstruct evolutionary scenarios between species;

GENOME SORTING (PERMUTATIONS)

Input: a (signed) permutation π, a set S of (per)mutations;
Goal: find a shortest sorting sequence of elements of S for π.
(the length of that sequence is the **distance** of π)

Example (disregarding / considering gene orientation)



Modelling genomes

A more unified treatment is provided by:

1. **unsigned genomes**: paths on $\{0, 1, 2, \ldots, n+1\}$;
2. **signed genomes**: perfect matchings on $\{0, 1, 2, \ldots, 2n+1\}$;

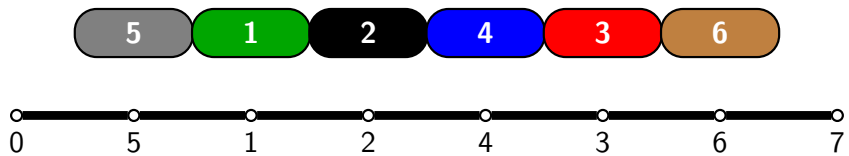
Example (from permutations to genomes)

Modelling genomes

A more unified treatment is provided by:

1. **unsigned genomes**: paths on $\{0, 1, 2, \ldots, n+1\}$;
2. **signed genomes**: perfect matchings on $\{0, 1, 2, \ldots, 2n+1\}$;

Example (from permutations to genomes)

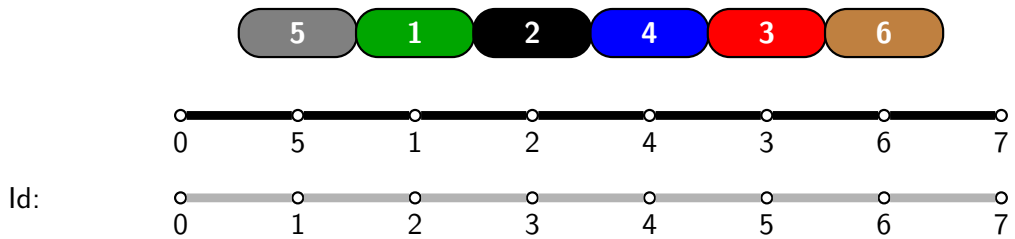


Modelling genomes

A more unified treatment is provided by:

1. **unsigned genomes**: paths on $\{0, 1, 2, \ldots, n+1\}$;
2. **signed genomes**: perfect matchings on $\{0, 1, 2, \ldots, 2n+1\}$;

Example (from permutations to genomes)

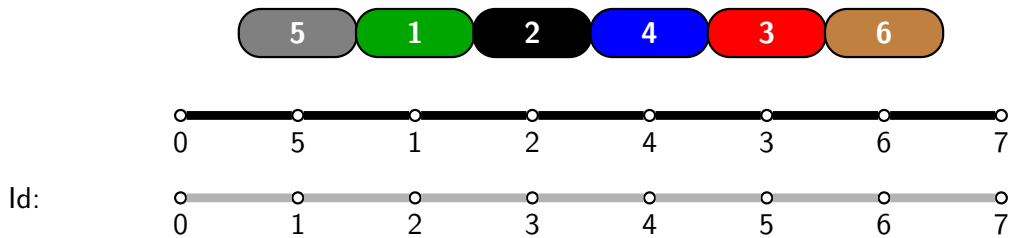


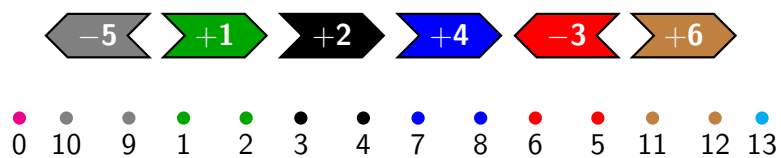
Modelling genomes

A more unified treatment is provided by:

1. **unsigned genomes**: paths on $\{0, 1, 2, \ldots, n+1\}$;
2. **signed genomes**: perfect matchings on $\{0, 1, 2, \ldots, 2n+1\}$;

Example (from permutations to genomes)





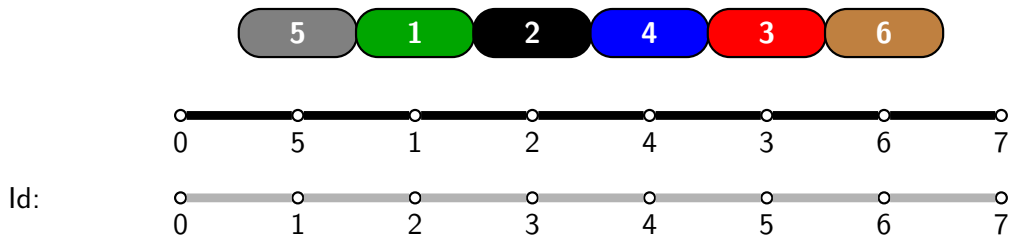
$x < 0 \mapsto (2|x|, 2|x| - 1)$; $x > 0 \mapsto (2|x| - 1, 2|x|)$;

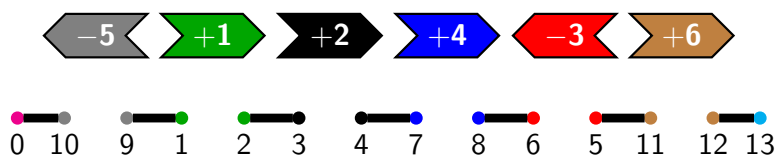
Modelling genomes

A more unified treatment is provided by:

1. **unsigned genomes**: paths on $\{0, 1, 2, \ldots, n+1\}$;
2. **signed genomes**: perfect matchings on $\{0, 1, 2, \ldots, 2n+1\}$;

Example (from permutations to genomes)





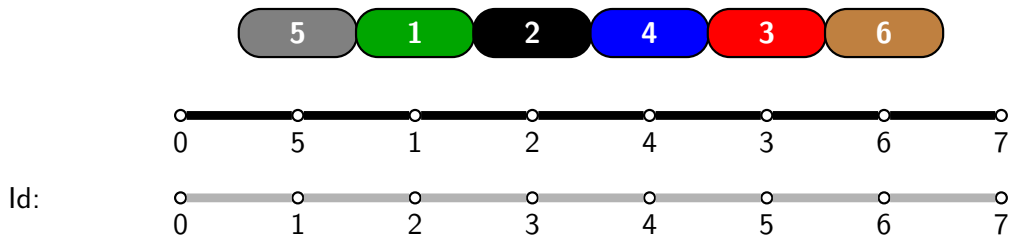
$x < 0 \mapsto (2|x|, 2|x| - 1)$; $x > 0 \mapsto (2|x| - 1, 2|x|)$;

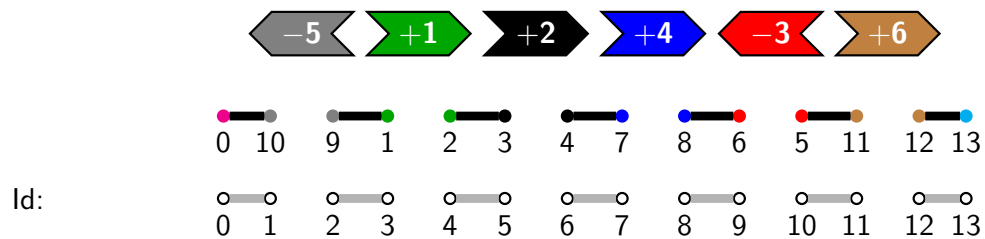
Modelling genomes

A more unified treatment is provided by:

1. **unsigned genomes**: paths on $\{0, 1, 2, \ldots, n+1\}$;
2. **signed genomes**: perfect matchings on $\{0, 1, 2, \ldots, 2n+1\}$;

Example (from permutations to genomes)



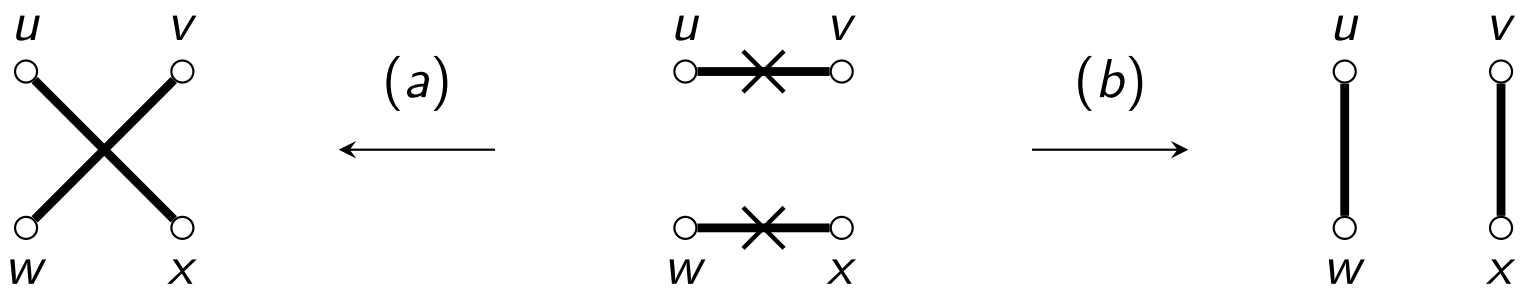


$x < 0 \mapsto (2|x|, 2|x| - 1)$; $x > 0 \mapsto (2|x| - 1, 2|x|)$;

The double cut-and-join (DCJ) operation

A **double cut-and-join** (DCJ) removes two edges $\{u, v\}$ and $\{w, x\}$ from a graph, then connects the four endpoints in one of two ways.

Example



The graph might be directed, belong to a particular class, ... which may restrict our options for reconnecting the endpoints (see examples later on).

DCJs in a biological setting

- DCJs generalise several well-studied mutations, e.g.:
  - transpositions; $3\ \boxed{1}\ 5\ \boxed{4}\ 2\ 6 \rightarrow 3\ \boxed{4}\ 5\ \boxed{1}\ 2\ 6$
  - reversals; $3\ \underline{1\ 5\ 4\ 2}\ 6 \rightarrow 3\ 2\ 4\ 5\ 1\ 6$
  - signed reversals; $3\ \underline{-1\ 5\ -4\ 2}\ 6 \rightarrow 3\ -2\ 4\ -5\ 1\ 6$
  - block-transpositions; $3\ \boxed{1\ 5}\ \boxed{4\ 2}\ 6 \rightarrow 3\ 4\ 2\ 1\ 5\ 6$
  - block-interchanges; $3\ \boxed{1\ 5}\ 4\ \boxed{2\ 6} \rightarrow 3\ 2\ 6\ 4\ 1\ 5$

DCJs in a biological setting

- DCJs generalise several well-studied mutations, e.g.:
  - transpositions; $3\ \boxed{1}\ 5\ \boxed{4}\ 2\ 6 \rightarrow 3\ \boxed{4}\ 5\ \boxed{1}\ 2\ 6$
  - reversals; $3\ \underline{1\ 5\ 4\ 2}\ 6 \rightarrow 3\ 2\ 4\ 5\ 1\ 6$
  - signed reversals; $3\ \underline{-1\ 5\ -4\ 2}\ 6 \rightarrow 3\ -2\ 4\ -5\ 1\ 6$
  - block-transpositions; $3\ \boxed{1\ 5}\ \boxed{4\ 2}\ 6 \rightarrow 3\ 4\ 2\ 1\ 5\ 6$
  - block-interchanges; $3\ \boxed{1\ 5}\ 4\ \boxed{2\ 6} \rightarrow 3\ 2\ 6\ 4\ 1\ 5$
- Sorting genomes by DCJs is:
  - in P in the signed case [7];
  - NP-hard in the unsigned case [5];

The prefix constraint

- We study **prefix DCJs**: one of the cut edges must be incident with 0;
- The constraint has no biological relevance: it originates from interconnection network design;
- Theoretical interest: many "unrestricted" problems remain open under the prefix constraint;

Results

We obtain:

- new lower bounds for sorting by prefix reversals or DCJs (signed or unsigned);
- a polynomial time algorithm for sorting by signed prefix DCJs;
- a 3/2-approximation for sorting by unsigned prefix DCJs;

To the best of our knowledge, this is the first $(2-\varepsilon)$-approximation for a prefix sorting problem not known to be in P.

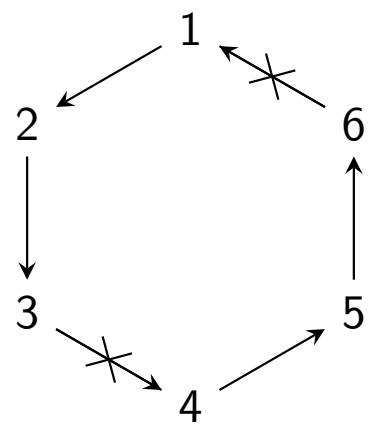
Mimicking other rearrangements using DCJs

Algebraic transpositions as DCJs

Let π be a permutation and $\Gamma(\pi)$ be its graph; i.e., the cycles of π are exactly those of $\Gamma(\pi)$.

Example

Let us compute $(1,2,3)(4,5,6) = (1,4) \circ (1,2,3,4,5,6)$.

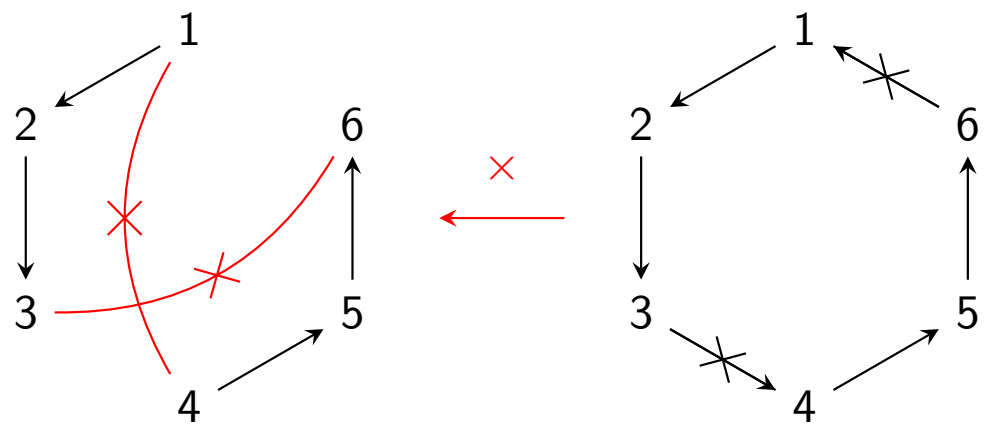


Algebraic transpositions as DCJs

Let π be a permutation and $\Gamma(\pi)$ be its graph; i.e., the cycles of π are exactly those of $\Gamma(\pi)$.

Example

Let us compute $(1,2,3)(4,5,6) = (1,4) \circ (1,2,3,4,5,6)$.



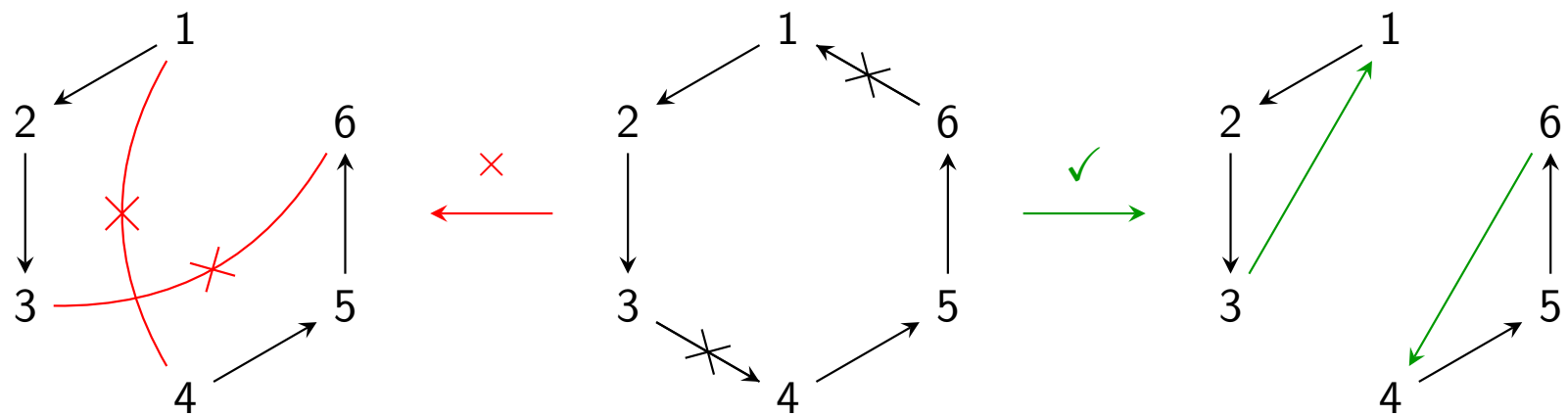
We must obtain a collection of cycles, so the red option is invalid.

Algebraic transpositions as DCJs

Let π be a permutation and $\Gamma(\pi)$ be its graph; i.e., the cycles of π are exactly those of $\Gamma(\pi)$.

Example

Let us compute $(1,2,3)(4,5,6) = (1,4) \circ (1,2,3,4,5,6)$.

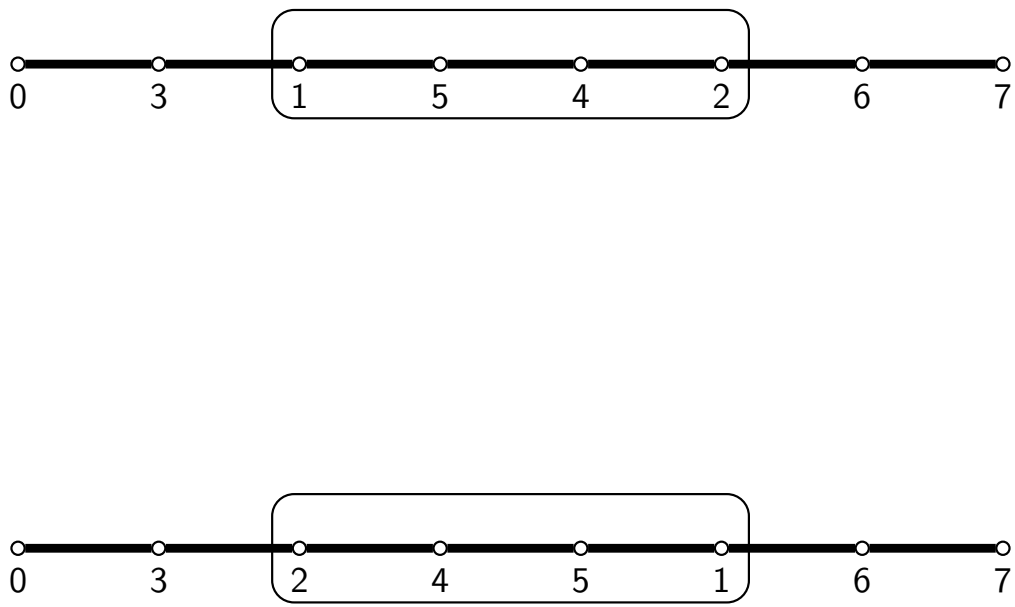


We must obtain a collection of cycles, so the red option is invalid.

Reversals as DCJs

Viewing permutations of $\{1, 2, \ldots, n\}$ as paths on $\{0, 1, 2 \ldots, n, n+1\}$ allows us to express reversals as DCJs.

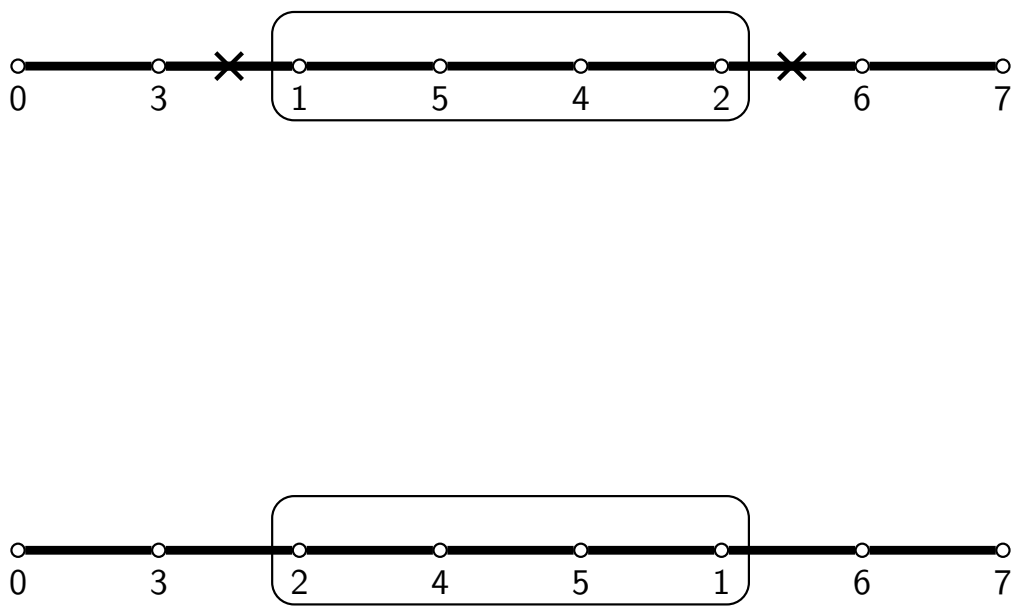
Example



Reversals as DCJs

Viewing permutations of $\{1, 2, \ldots, n\}$ as paths on $\{0, 1, 2 \ldots, n, n+1\}$ allows us to express reversals as DCJs.

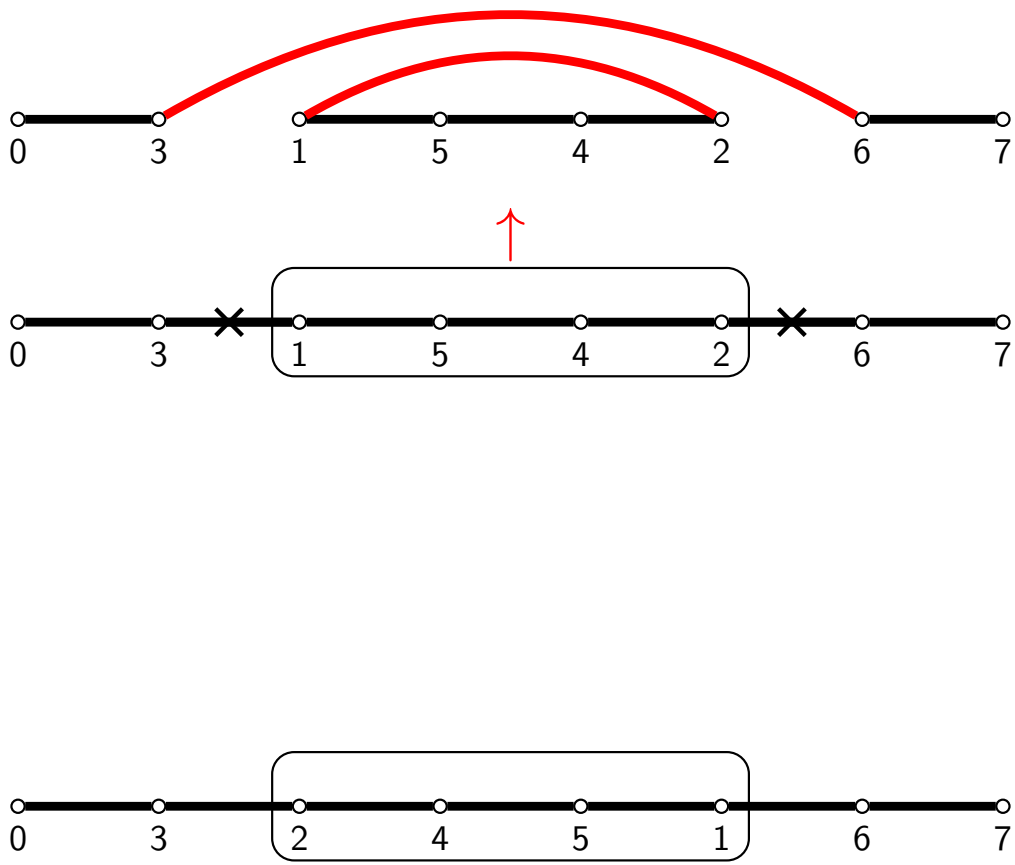
Example



Reversals as DCJs

Viewing permutations of $\{1, 2, \ldots, n\}$ as paths on $\{0, 1, 2 \ldots, n, n+1\}$ allows us to express reversals as DCJs.

Example

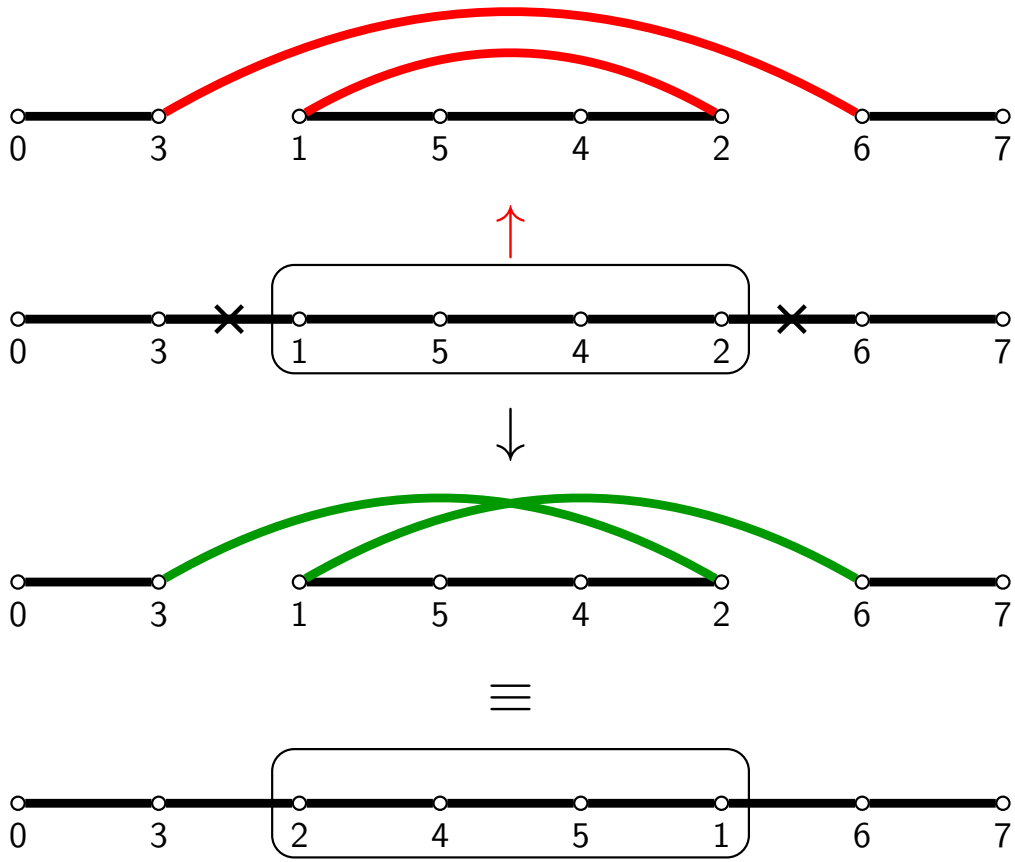


We must obtain a path, so the red option is forbidden.

Reversals as DCJs

Viewing permutations of $\{1, 2, \ldots, n\}$ as paths on $\{0, 1, 2 \ldots, n, n+1\}$ allows us to express reversals as DCJs.

Example

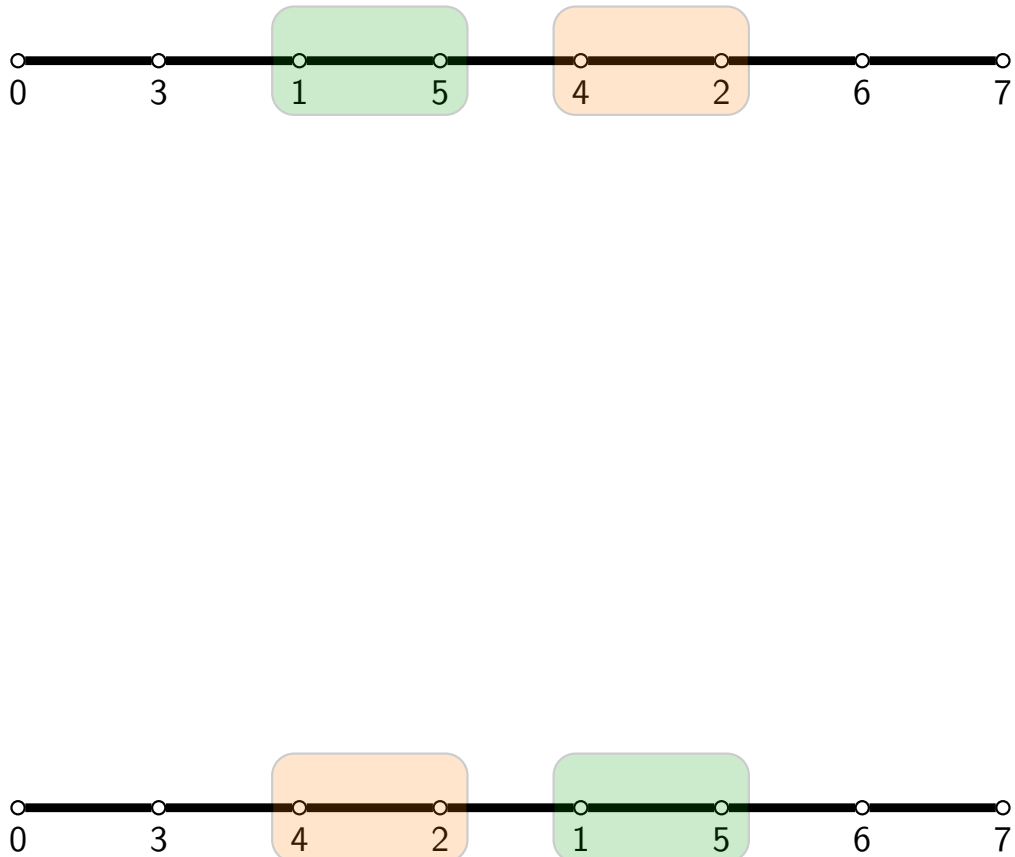


We must obtain a path, so the red option is forbidden.

Block-transpositions as DCJs

We can also simulate block-transpositions using *two* DCJs.

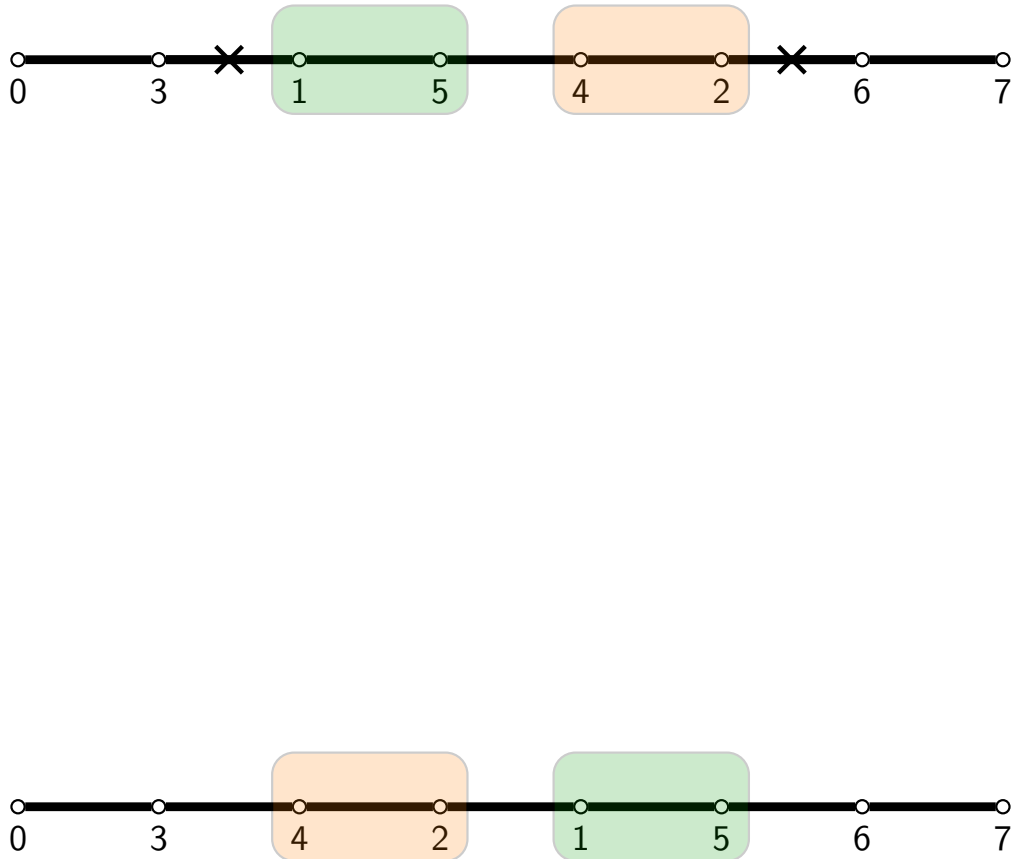
Example



Block-transpositions as DCJs

We can also simulate block-transpositions using *two* DCJs.

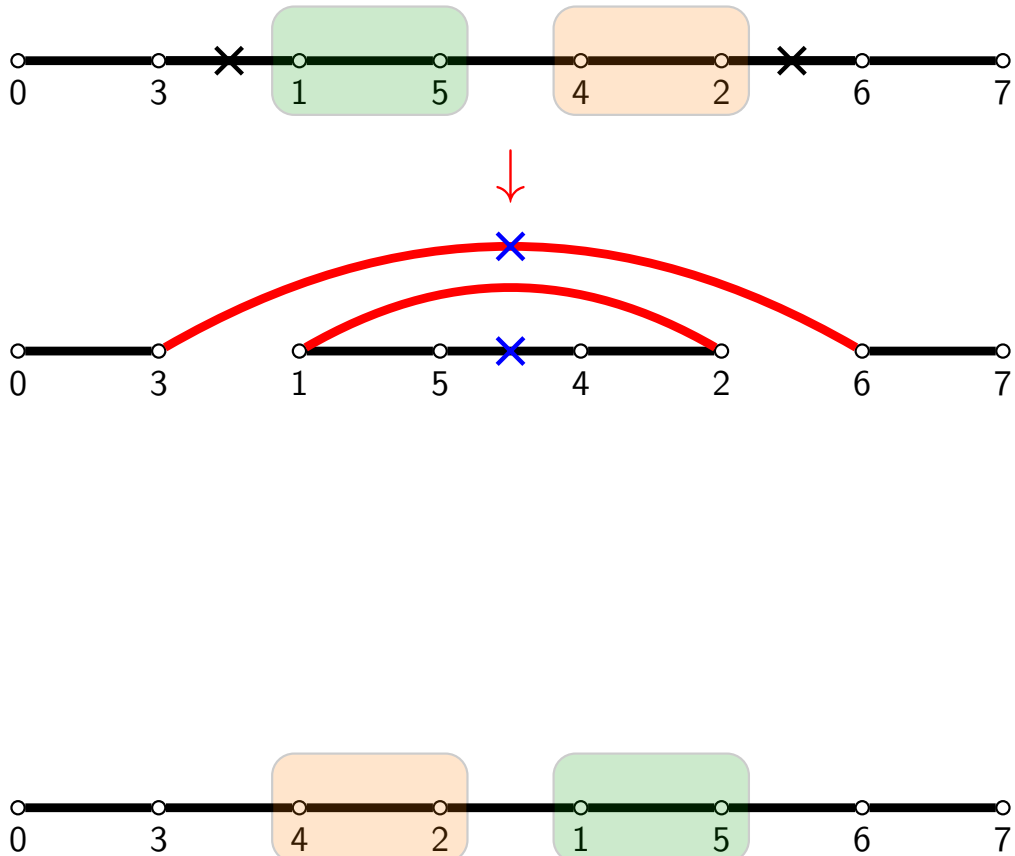
Example



Block-transpositions as DCJs

We can also simulate block-transpositions using *two* DCJs.

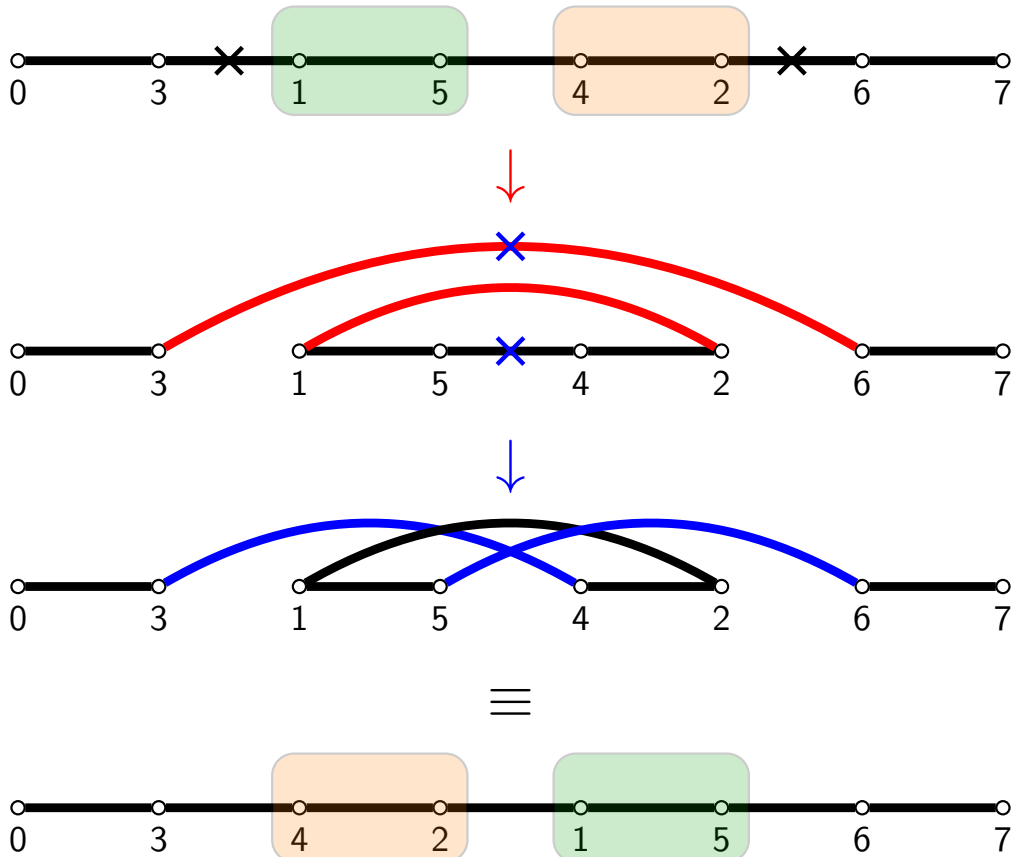
Example



Block-transpositions as DCJs

We can also simulate block-transpositions using *two* DCJs.

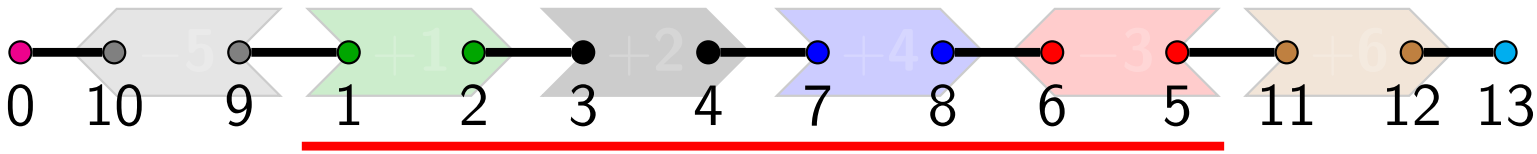
Example

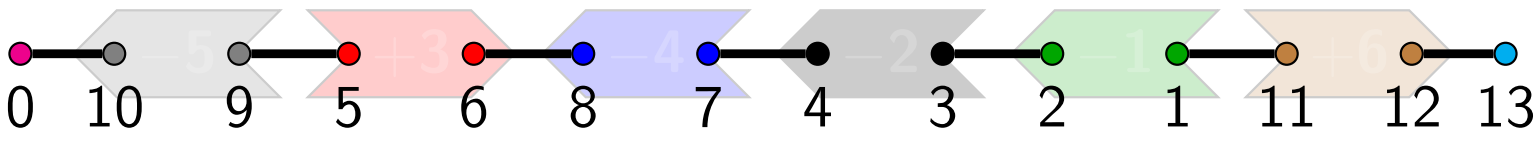


Signed reversals as DCJs

Likewise, we can represent *signed permutations* and mimick *signed reversals* using DCJs.

Example

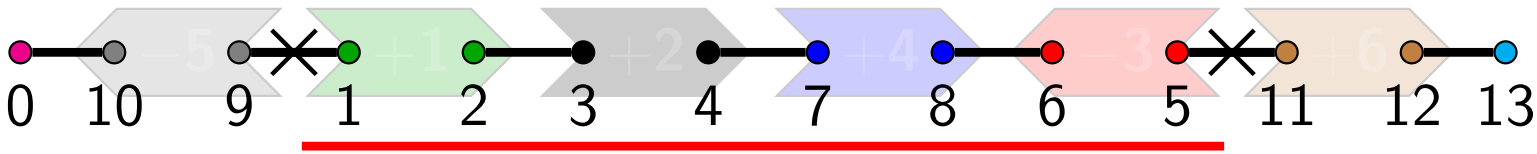


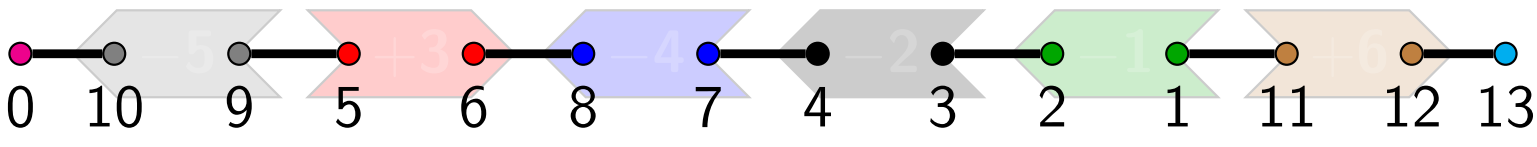


Signed reversals as DCJs

Likewise, we can represent *signed permutations* and mimick *signed reversals* using DCJs.

Example

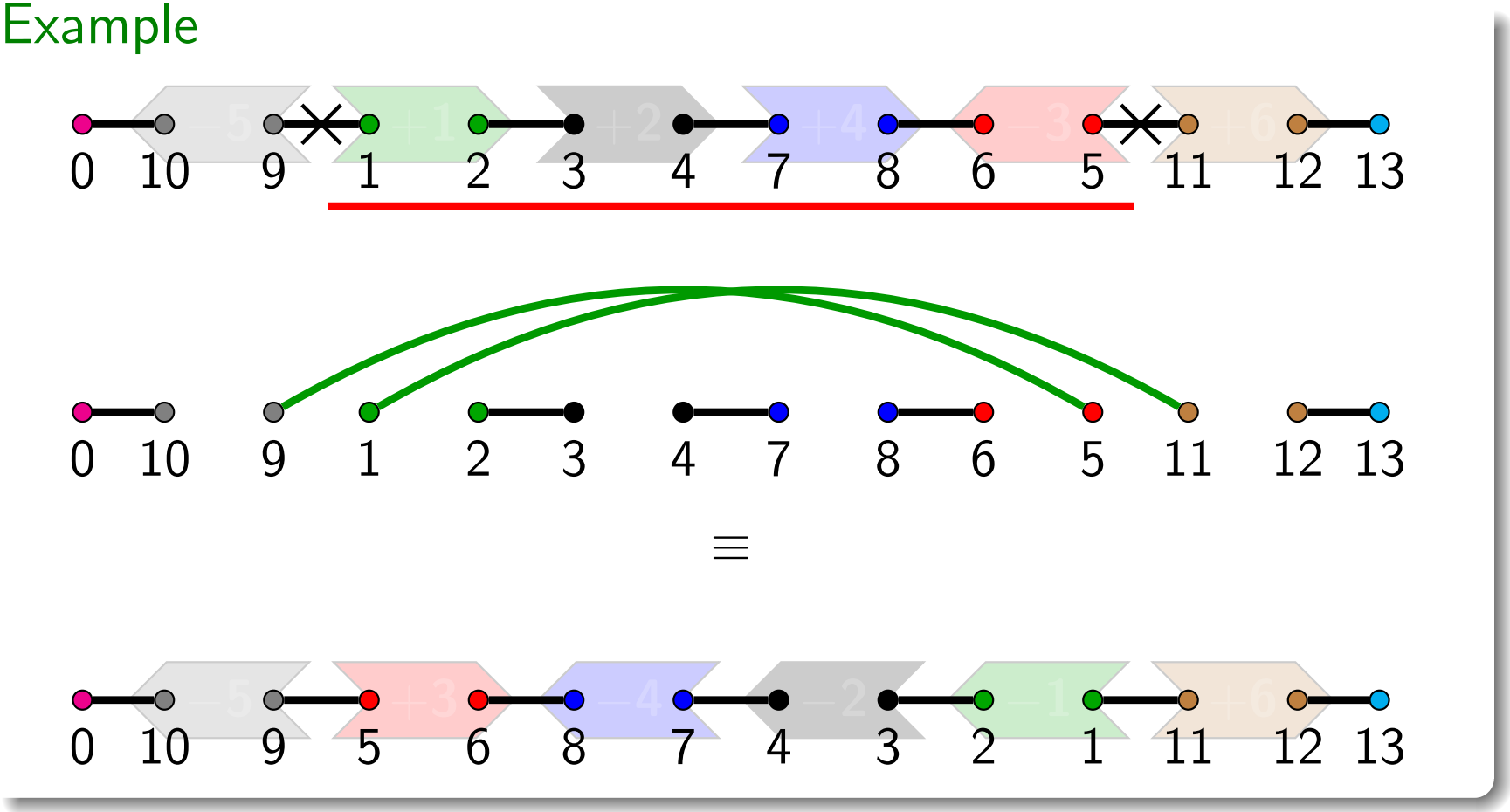




Signed reversals as DCJs

Likewise, we can represent *signed permutations* and mimick *signed reversals* using DCJs.

Example



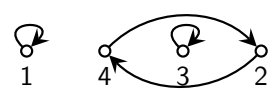
Results

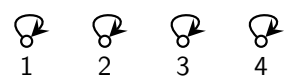
Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

Example





Preliminary results

Theorem (Cayley distance)

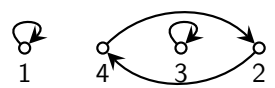
Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

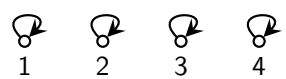
Theorem ("Prefix" Cayley distance)

[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly $(c_1(\pi) = \text{number of trivial cycles})$

$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Example





Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

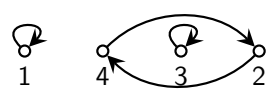
Theorem ("Prefix" Cayley distance)

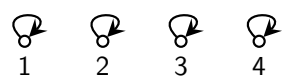
[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly $(c_1(\pi) = \text{number of trivial cycles})$

$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Intuition

Example





Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

Theorem ("Prefix" Cayley distance)

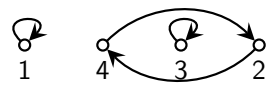
[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly *($c_1(\pi)$ = number of trivial cycles)*

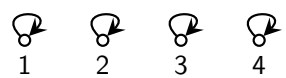
$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Intuition

- we can only split the cycle that contains π_1;

Example





Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

Theorem ("Prefix" Cayley distance)

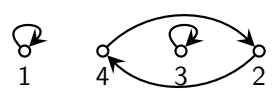
[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly *($c_1(\pi)$ = number of trivial cycles)*

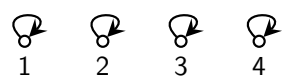
$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Intuition

- we can only split the cycle that contains π_1;
- if $\pi_1 = 1$ but π is not sorted, we must waste one operation to access another nontrivial cycle.

Example





Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

Theorem ("Prefix" Cayley distance)

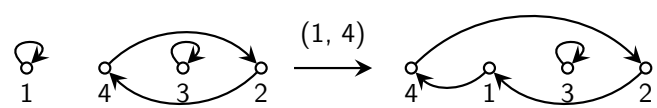
[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly *($c_1(\pi)$ = number of trivial cycles)*

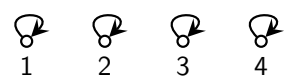
$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Intuition

- we can only split the cycle that contains π_1;
- if $\pi_1 = 1$ but π is not sorted, we must waste one operation to access another nontrivial cycle.

Example





Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

Theorem ("Prefix" Cayley distance)

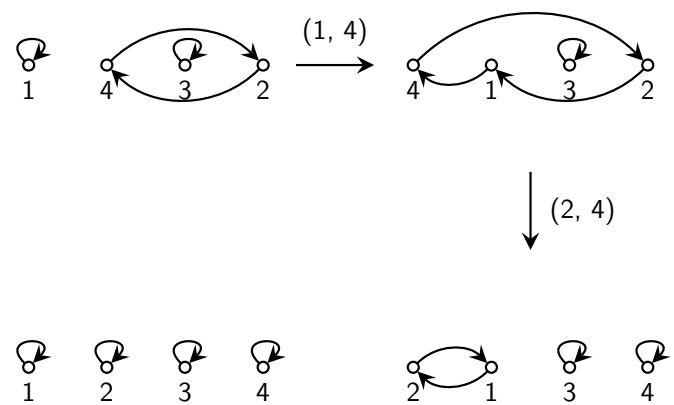
[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly *($c_1(\pi)$ = number of trivial cycles)*

$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Intuition

- we can only split the cycle that contains π_1;
- if $\pi_1 = 1$ but π is not sorted, we must waste one operation to access another nontrivial cycle.

Example



Preliminary results

Theorem (Cayley distance)

Sorting any permutation π in S_n requires $n - c(\pi)$ transpositions.

Theorem ("Prefix" Cayley distance)

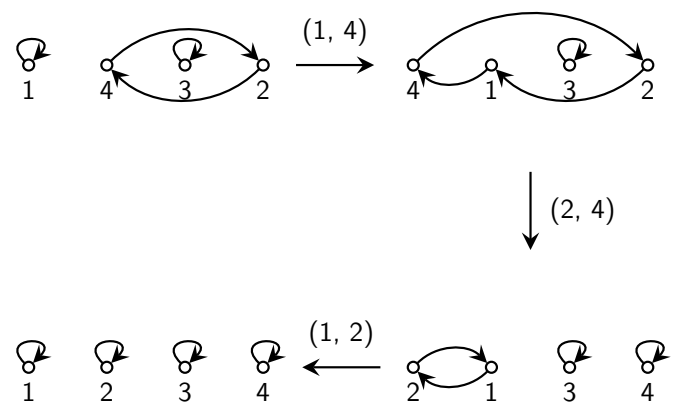
[1] For any permutation π in S_n, the number of prefix transpositions required to sort π is exactly *($c_1(\pi)$ = number of trivial cycles)*

$$n + c(\pi) - 2c_1(\pi) - \begin{cases} 0 & \text{if } \pi_1 = 1, \\ 2 & \text{otherwise.} \end{cases}$$

Intuition

- we can only split the cycle that contains π_1;
- if $\pi_1 = 1$ but π is not sorted, we must waste one operation to access another nontrivial cycle.

Example



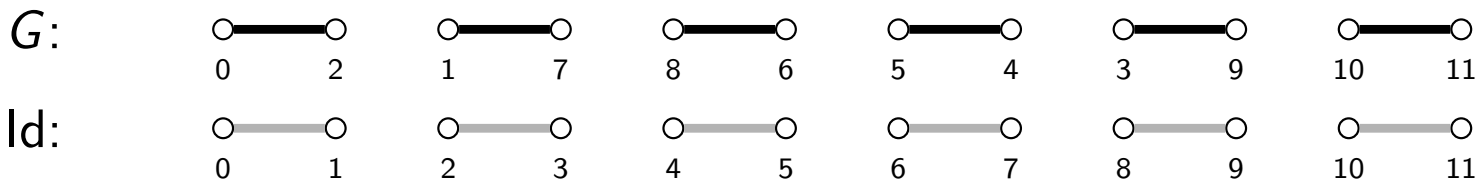
Approach

- As we have seen, (prefix) transpositions are (prefix) DCJs;
- Strategy:
  - find "the right graph" representation for pairs of genomes, depending on the mutations we want to use;
  - rely on the prefix Cayley distance to obtain bounds based on that graph;

Signed prefix DCJs

- A **signed genome** is a perfect matching G over $\{0, 1, \ldots, 2n+1\}$;
- We want to obtain $\mathrm{Id} = \{\{0,1\},\{2,3\},\ldots,\{2n,2n+1\}\}$;

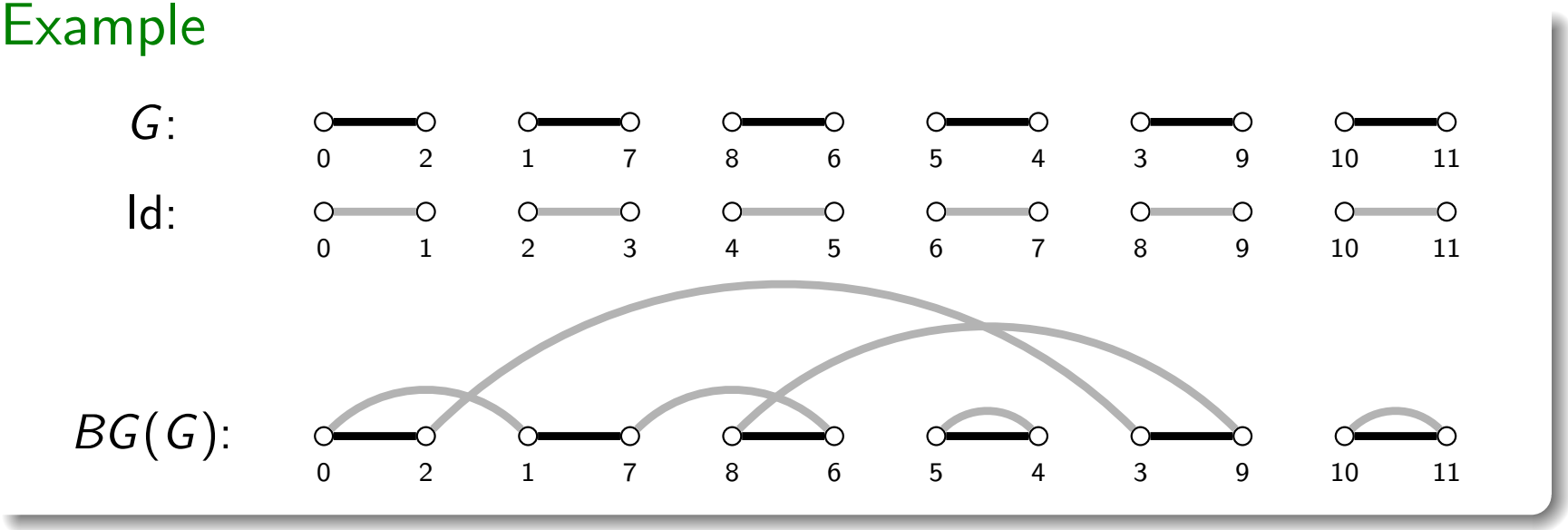
Example



Signed prefix DCJs

- A **signed genome** is a perfect matching G over $\{0, 1, \ldots, 2n+1\}$;
- We want to obtain $\mathrm{Id} = \{\{0,1\},\{2,3\},\ldots,\{2n,2n+1\}\}$;
- The **breakpoint graph** $BG(G)$ is the union of G and Id;

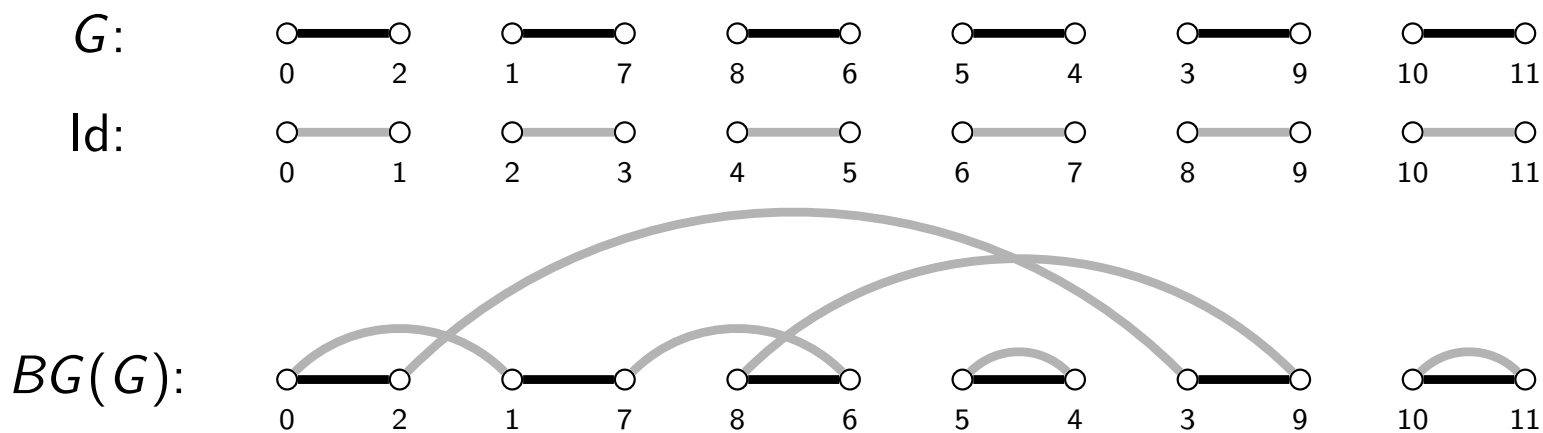
Example



Signed prefix DCJs

- A **signed genome** is a perfect matching G over $\{0, 1, \ldots, 2n+1\}$;
- We want to obtain $\mathrm{Id} = \{\{0,1\},\{2,3\},\ldots,\{2n,2n+1\}\}$;
- The **breakpoint graph** $BG(G)$ is the union of G and Id;

Example



Every vertex has degree 2 $\Rightarrow$ collection of cycles.

Signed prefix DCJs

- Prefix DCJs have the same effect on $BG(G)$ as on the cycles of a permutation; therefore:

Signed prefix DCJs

- Prefix DCJs have the same effect on $BG(G)$ as on the cycles of a permutation; therefore:

Theorem

For any signed genome G, we have

$$psdcj(G) \geq n+1 + c(BG(G)) - 2c_1(BG(G)) - \begin{cases} 0 & \text{if } \{0,1\} \in G, \\ 2 & \text{otherwise.} \end{cases}$$

Signed prefix DCJs

- Prefix DCJs have the same effect on $BG(G)$ as on the cycles of a permutation; therefore:

Theorem

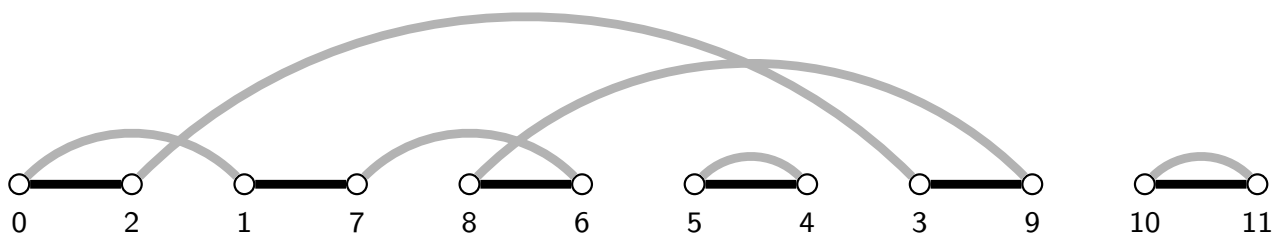
For any signed genome G, we have

$$psdcj(G) \geq n+1 + c(BG(G)) - 2c_1(BG(G)) - \begin{cases} 0 & \text{if } \{0,1\} \in G, \\ 2 & \text{otherwise.} \end{cases}$$

Example

With G as in the previous slide:

$BG(G)$:



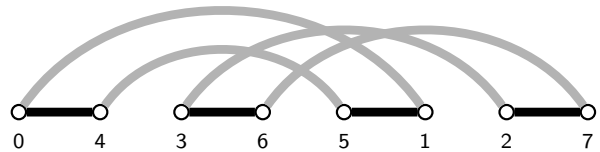
we have $psdcj(G) \geq 6 + 3 - 2 \times 2 - 2 = 3$.

Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

Example



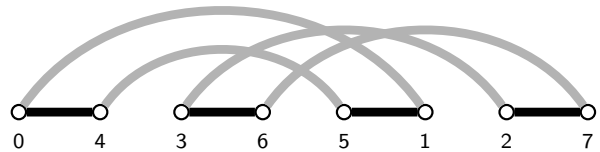
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;

Example



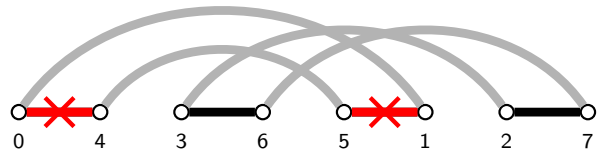
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;

Example



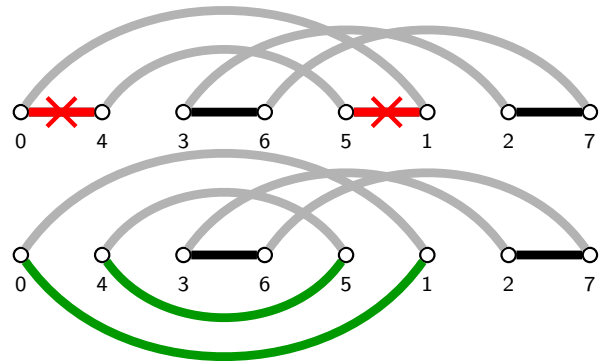
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;

Example



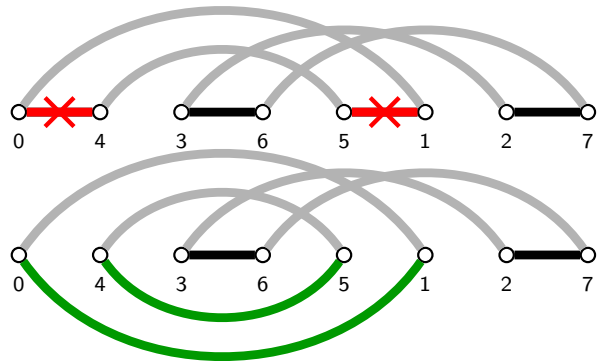
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



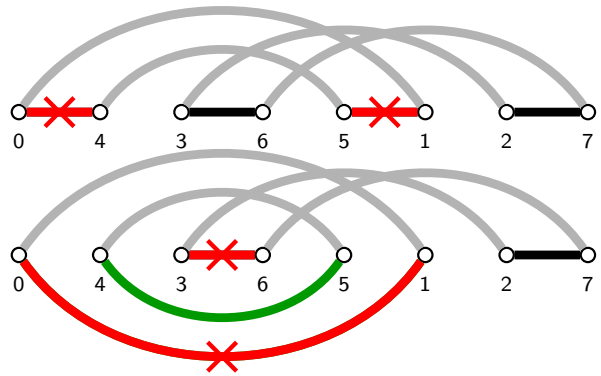
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



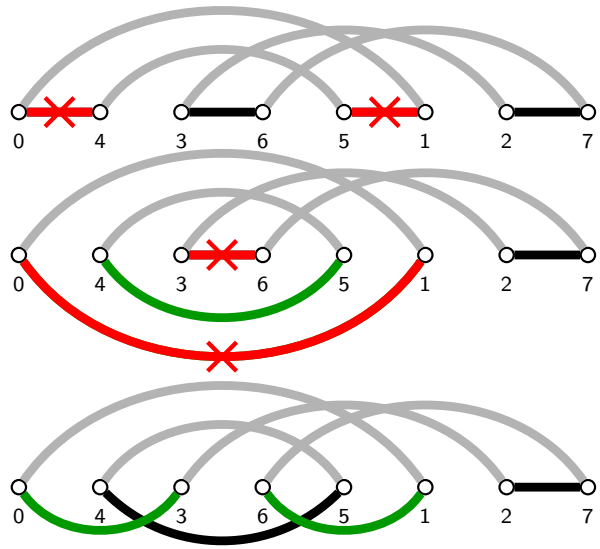
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



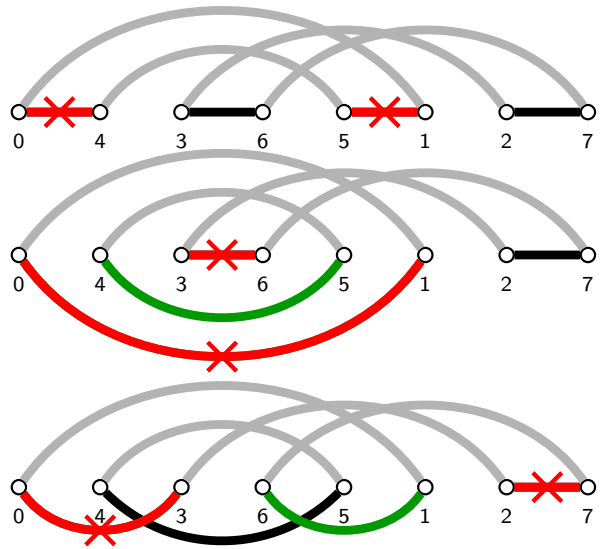
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



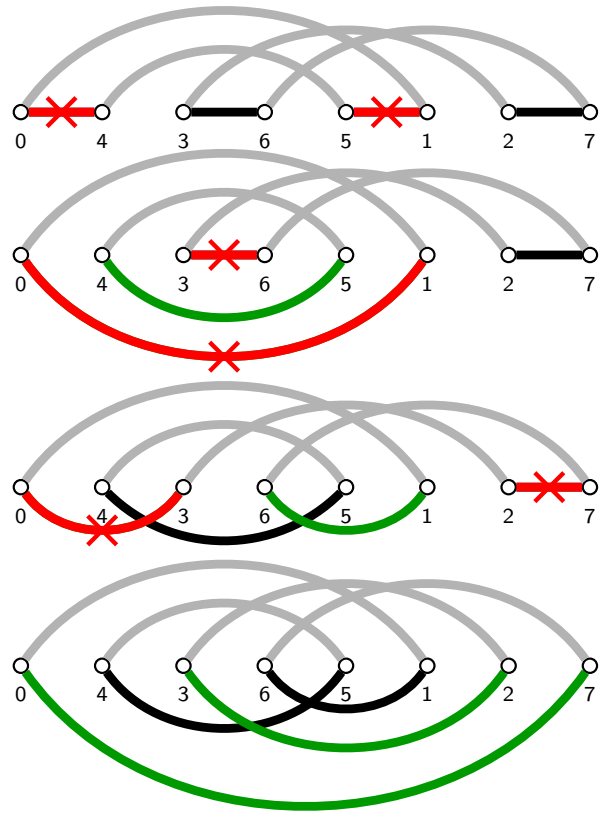
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



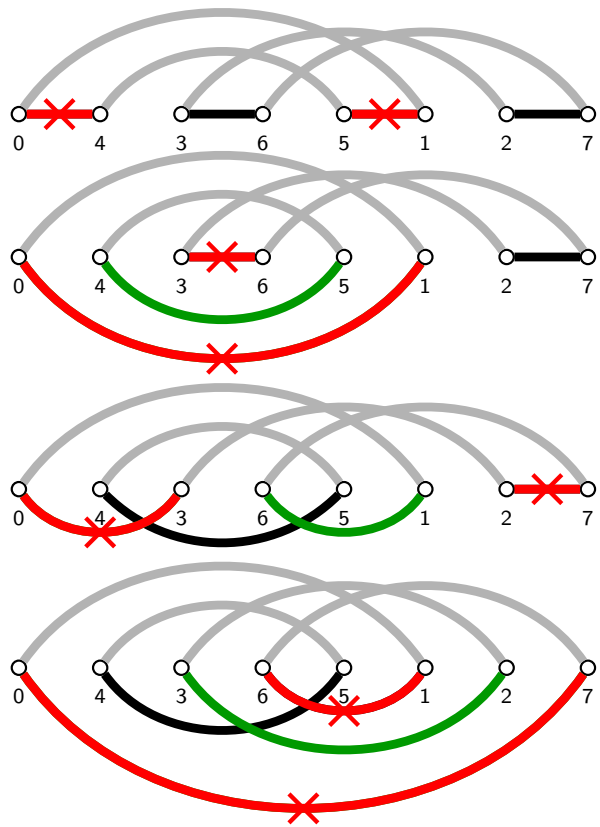
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



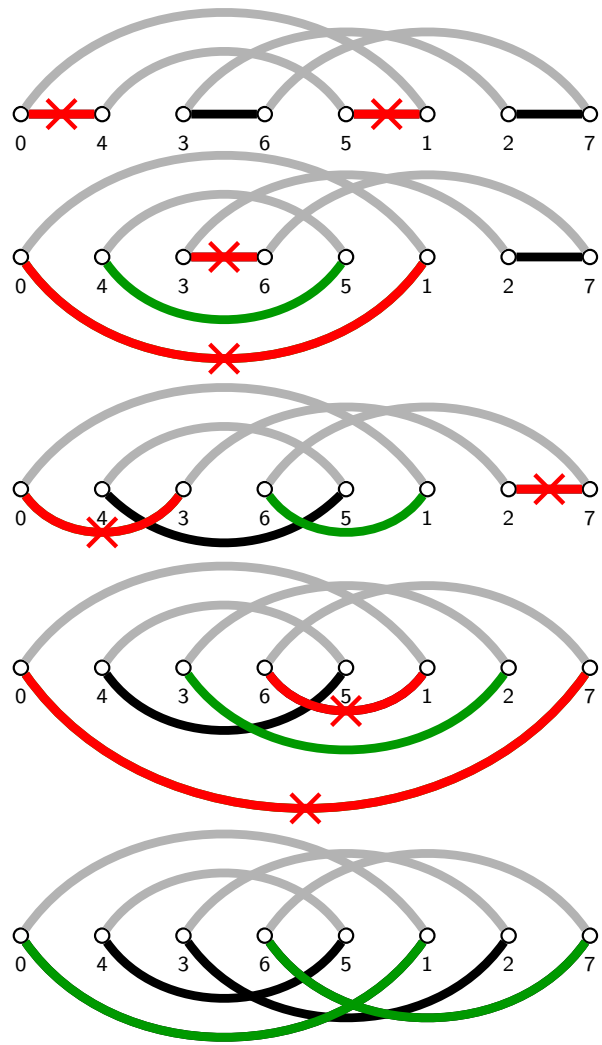
Sorting by signed prefix DCJs is in P

Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Example



Sorting by signed prefix DCJs is in P

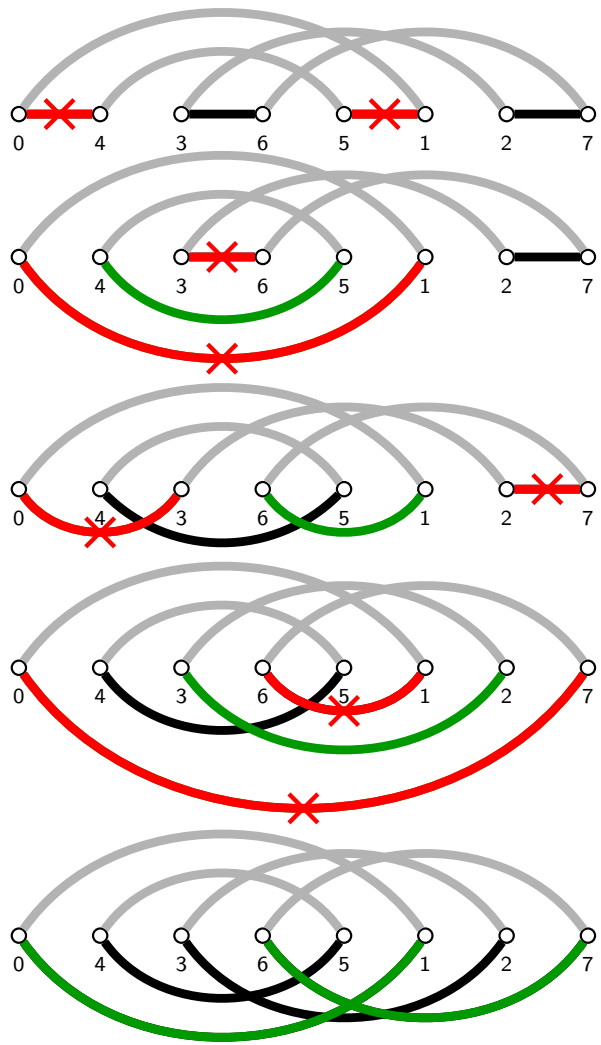
Algorithm outline

Until $G = \mathrm{Id}$, check edge $\{0, v\} \in G$:

1. if $v \neq 1$: connect v to its "grey neighbour" in Id;
2. otherwise $v = 1$: apply any prefix DCJ that involves a black edge from a nontrivial cycle.

Every operation decreases the value of our lower bound by 1 $\Rightarrow$ algorithm is optimal.

Example



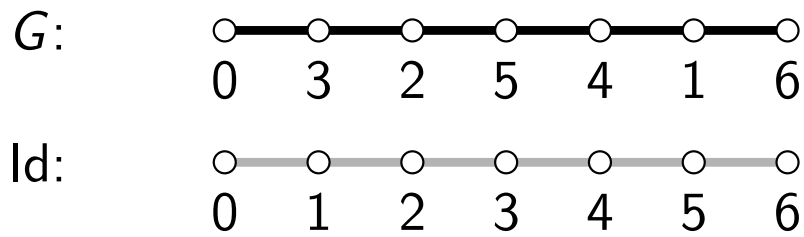
Signed prefix reversals

- Signed prefix reversals are signed prefix DCJs that must preserve an additional structural constraint (details omitted);
  - ✓ therefore, $psrd(G) \geq psdcj(G)$;
  - × but previous algorithm cannot be used;

Unsigned prefix DCJs

- An **unsigned genome** is a path G over $\{0, 1, \ldots, n+1\}$;
- We want to obtain the path $\mathrm{Id} = (0, 1, \ldots, n+1)$;

Example



- An unsigned version of the breakpoint graph yields a similar lower bound to the signed case (no time for details);

A lower bound for sorting by unsigned prefix DCJs

Theorem

For any genome G, we have:

$$pdcj(G) \geq n + 1 + c^*(UBG(G)) - 2c_1^*(UBG(G)) - \begin{cases} 0 & \text{if } \{0,1\} \in G \text{ and } \{1,2\} \in G, \\ 1 & \text{if } \{0,1\} \in G \text{ and } \{1,2\} \notin G, \\ 2 & \text{otherwise.} \end{cases}$$

where $c^*(\cdot)$ (resp. $c_1^*(\cdot)$) is the number of (trivial) cycles **in an optimal decomposition** *of $UBG(G)$.*

An optimal decomposition can be computed as follows:

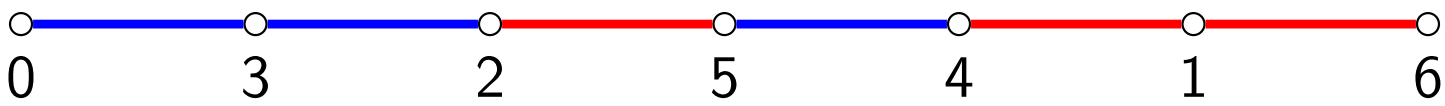
1. remove all edges that belong to trivial cycles;
2. each connected component that remains is Eulerian and therefore constitutes a nontrivial cycle.

Approximating the unsigned prefix DCJ distance

An edge $e \in G$ is a **breakpoint** if $0 \notin e$ and $e \notin \mathrm{Id}$, and an **adjacency** otherwise;

Example

The following genome has 3 breakpoints and 3 adjacencies:

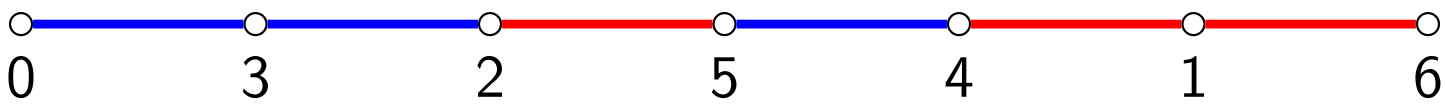


Approximating the unsigned prefix DCJ distance

An edge $e \in G$ is a **breakpoint** if $0 \notin e$ and $e \notin \mathrm{Id}$, and an **adjacency** otherwise;

Example

The following genome has 3 breakpoints and 3 adjacencies:



Lemma

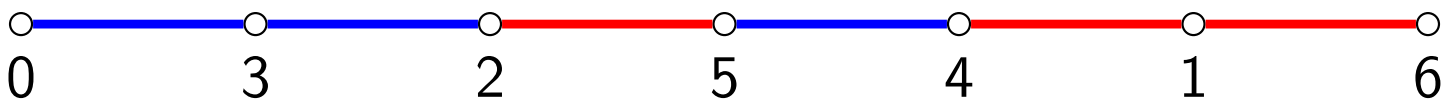
For any genome G, we have $pdcj(G) \geq b(G)$.

Approximating the unsigned prefix DCJ distance

An edge $e \in G$ is a **breakpoint** if $0 \notin e$ and $e \notin \mathrm{Id}$, and an **adjacency** otherwise;

Example

The following genome has 3 breakpoints and 3 adjacencies:



Lemma

For any genome G, we have $pdcj(G) \geq b(G)$.

Proof.

A prefix DCJ cuts $\{0, v\}$ and another edge, then reconnects their endpoints. But $\{0, v\}$ is never a breakpoint, so $b(G)$ can only decrease by 1. □

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$,

Example

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$,

2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$ $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$: extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$,

2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$ $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$: extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example

A 3/2-approximation algorithm for unsigned prefix DCJs

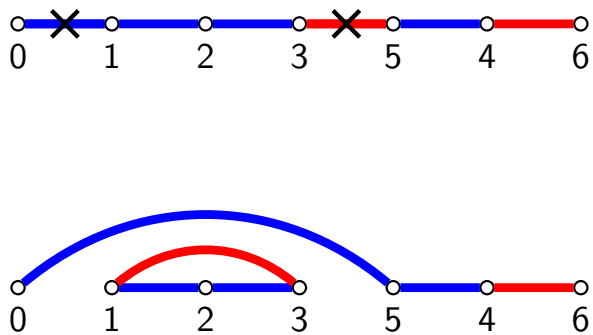
Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$,

2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$ $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$: extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example



A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

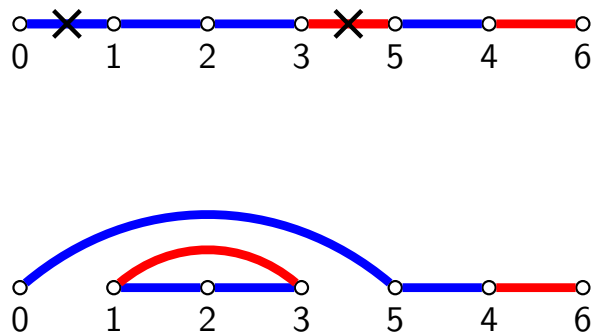
1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;

   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;

2. otherwise $v = 1$:

   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example



A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

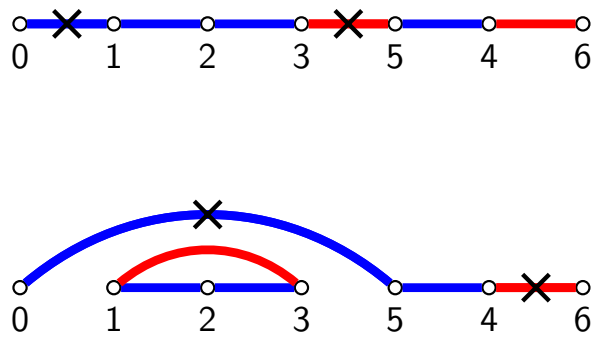
1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;

   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;

2. otherwise $v = 1$:

   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example



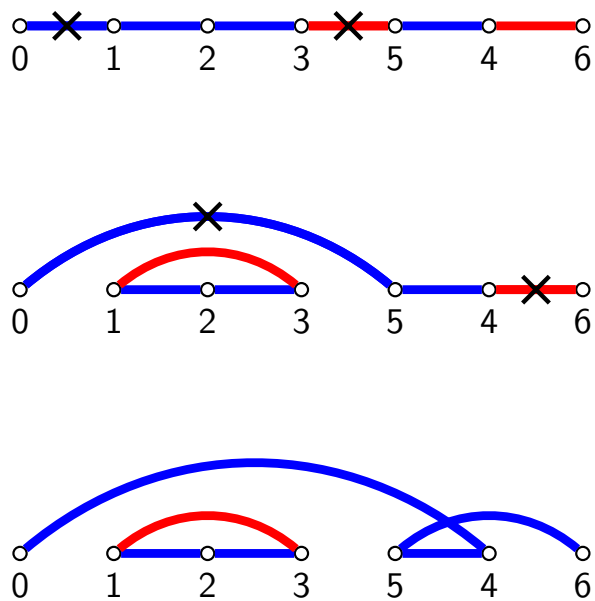
A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;
   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;
2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example



A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

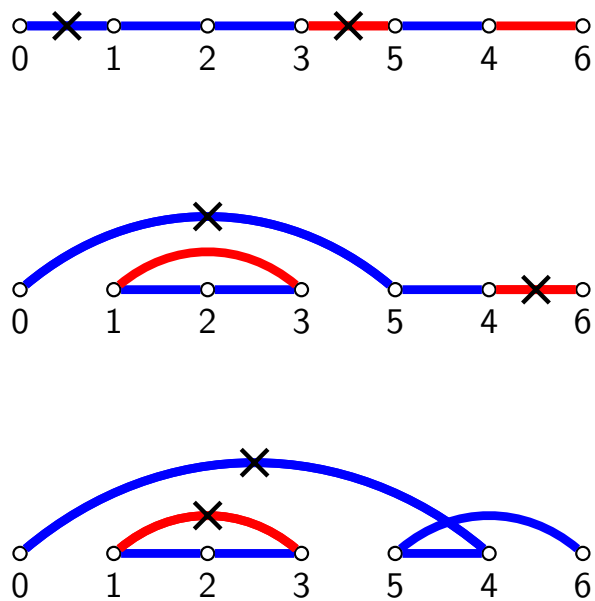
1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;

   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;

2. otherwise $v = 1$:

   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example



A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

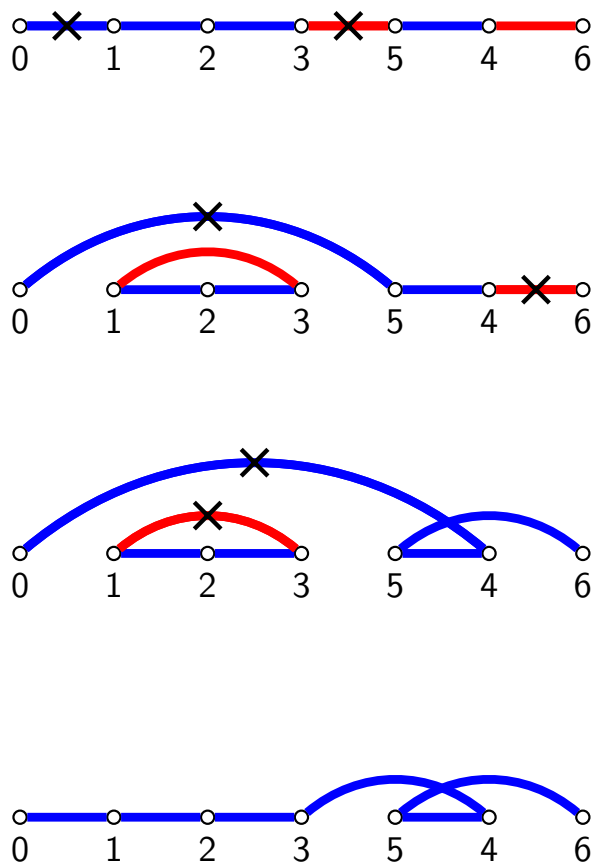
1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;

   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;

2. otherwise $v = 1$:

   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Example



A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;

   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;

2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Approximation guarantee

- Case 1: $b(G)$ decreases by 1;

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;
   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;
2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Approximation guarantee

- Case 1: $b(G)$ decreases by 1;
- Case 2.1: $b(G)$ decreases by 1;

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;
   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;
2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Approximation guarantee

- Case 1: $b(G)$ decreases by 1;
- Case 2.1: $b(G)$ decreases by 1;
- Case 2.2: $b(G)$ decreases by 0, then by 2 (case 1);

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;
   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;
2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Approximation guarantee

- Case 1: $b(G)$ decreases by 1;
- Case 2.1: $b(G)$ decreases by 1;
- Case 2.2: $b(G)$ decreases by 0, then by 2 (case 1);

$\Rightarrow$ Worst case: $b(G)$ decreases by 2 in 3 steps;

A 3/2-approximation algorithm for unsigned prefix DCJs

Algorithm outline

Until $G = \mathrm{Id}$, consider edge $\{0, v\} \in G$:

1. if $v \neq 1$, then at least one of $\{v-1, x\}$ or $\{v+1, y\}$ is a breakpoint;
   $\Rightarrow$ create $\{\{0, x\}, \{v-1, v\}\}$ or $\{\{0, y\}, \{v, v+1\}\}$;
2. otherwise $v = 1$:
   1. if $\{1, 2\} \notin G$: $\exists$ breakpoint $\{2, z\}$
      $\Rightarrow$ create $\{\{0, z\}, \{1, 2\}\}$
   2. otherwise $\{1, 2\} \in G$:
      extract the longest run of adjacencies from 1; and then we can apply case 1 twice.

Approximation guarantee

- Case 1: $b(G)$ decreases by 1;
- Case 2.1: $b(G)$ decreases by 1;
- Case 2.2: $b(G)$ decreases by 0, then by 2 (case 1);

$\Rightarrow$ Worst case: $b(G)$ decreases by 2 in 3 steps;

$\Rightarrow$ $\boxed{3/2\text{-approximation}}$

Unsigned prefix reversals

- Unsigned prefix reversals are unsigned prefix DCJs that must yield a path at each step;
  - ✓ therefore, $prd(G) \geq pdcj(G)$;
  - × but previous algorithm cannot be used;

Open problems

- Complexity issues:

	reversals		DCJs	
	unsigned	signed	unsigned	signed
unrestricted	NP-hard [4]	in P [6]	NP-hard [5]	in P [7]
prefix	NP-hard [3]	???	???	in P(here)

Open problems

- Complexity issues:

	reversals		DCJs	
	unsigned	signed	unsigned	signed
unrestricted	NP-hard [4]	in P [6]	NP-hard [5]	in P [7]
prefix	NP-hard [3]	**???**	**???**	in P(here)

- Approximability: is there a better guarantee than:
  - 2 for prefix reversals (signed or unsigned)?
  - 3/2 for unsigned prefix DCJs?

Open problems

- Complexity issues:

	reversals		DCJs	
	unsigned	signed	unsigned	signed
unrestricted	NP-hard [4]	in P [6]	NP-hard [5]	in P [7]
prefix	NP-hard [3]	???	???	in P(here)

- Approximability: is there a better guarantee than:
  - 2 for prefix reversals (signed or unsigned)?
  - 3/2 for unsigned prefix DCJs?
- Exploring (prefix) DCJs on other graph classes;
  - finding a shortest scenario is NP-hard [2];
  - there is a 7/4-approximation [2];

Open problems

- Complexity issues:

	reversals		DCJs	
	unsigned	signed	unsigned	signed
unrestricted	NP-hard [4]	in P [6]	NP-hard [5]	in P [7]
prefix	NP-hard [3]	???	???	in P(here)

- Approximability: is there a better guarantee than:
  - 2 for prefix reversals (signed or unsigned)?
  - 3/2 for unsigned prefix DCJs?
- Exploring (prefix) DCJs on other graph classes;
  - finding a shortest scenario is NP-hard [2];
  - there is a 7/4-approximation [2];

Thanks!

References I

Sheldon B. Akers, Balakrishnan Krishnamurthy, and Dov Harel.

The star graph: An attractive alternative to the n-cube.

In *Proceedings of the Fourth International Conference on Parallel Processing*, pages 393–400. Pennsylvania State University Press, August 1987.

Daniel Bienstock and Oktay Günlük.

A degree sequence problem related to network design.

Networks, 24(4):195–205, 1994.

Laurent Bulteau, Guillaume Fertin, and Irena Rusu.

Pancake flipping is hard.

Journal of Computer and System Sciences, 81(8):1556–1574, 2015.

Alberto Caprara.

Sorting permutations by reversals and Eulerian cycle decompositions.

SIAM Journal on Discrete Mathematics, 12(1):91–110 (electronic), January 1999.

Xin Chen.

On sorting unsigned permutations by double-cut-and-joins.

Journal of Combinatorial Optimization, 25(3):339–351, April 2013.

Sridhar Hannenhalli and Pavel A. Pevzner.

Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals.

Journal of the ACM, 46(1):1–27, 1999.

Sophia Yancopoulos, Oliver Attie, and Richard Friedberg.

Efficient sorting of genomic permutations by translocation, inversion and block interchange.

Bioinformatics, 21(16):3340–3346, 2005.