The solution space of sorting by reversals

Marília D. V. Braga

marilia@biomserv.univ-lyon1.fr

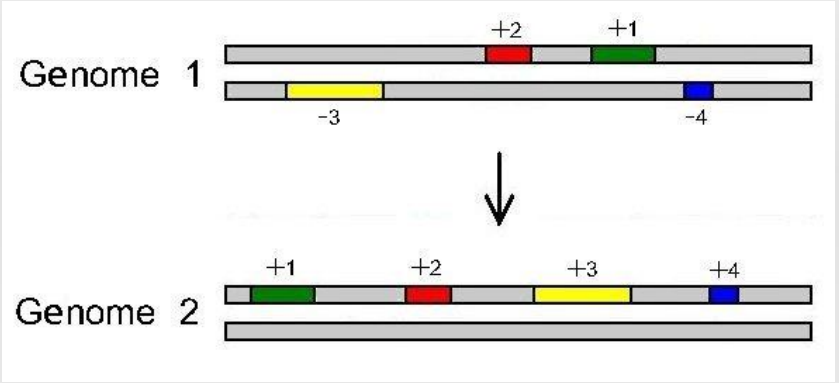
Marie-France Sagot
Celine Scornavacca
Eric Tannier

University of Lyon (France)

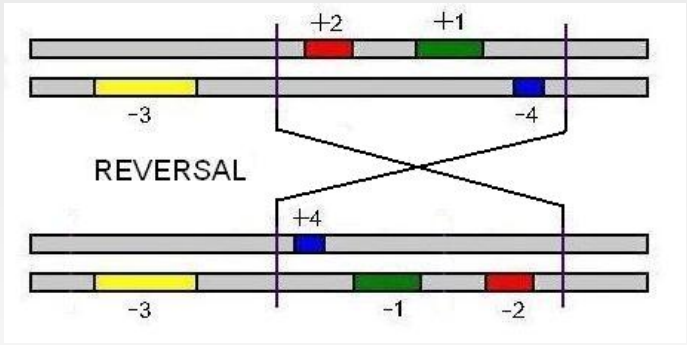
INRIA / CNRS / Program Alßan

Motivation

Genome rearrangement studies: when **comparing the contents of two different genomes**, try to identify the **mutation events** (reversals, insertions, deletions, transpositions...) that have **transformed one genome into the other**.



Identify all parsimonious scenarios of rearrangement restricted to **reversal** events (**sorting by reversals**)



(a **reversal** reverts the order and orientation of the genes in an interval of the genome)

Sorting by reversals (1)

- A **signed permutation** π represents a genome (each value represents a marker and its orientation; duplications are not allowed)

- Only reversal operations are considered (a **reversal** ρ reverts the order and orientation of the values in an interval of the permutation)

- Example:
  Sorting the permutation
  π = (-3, +2, +1, -4):

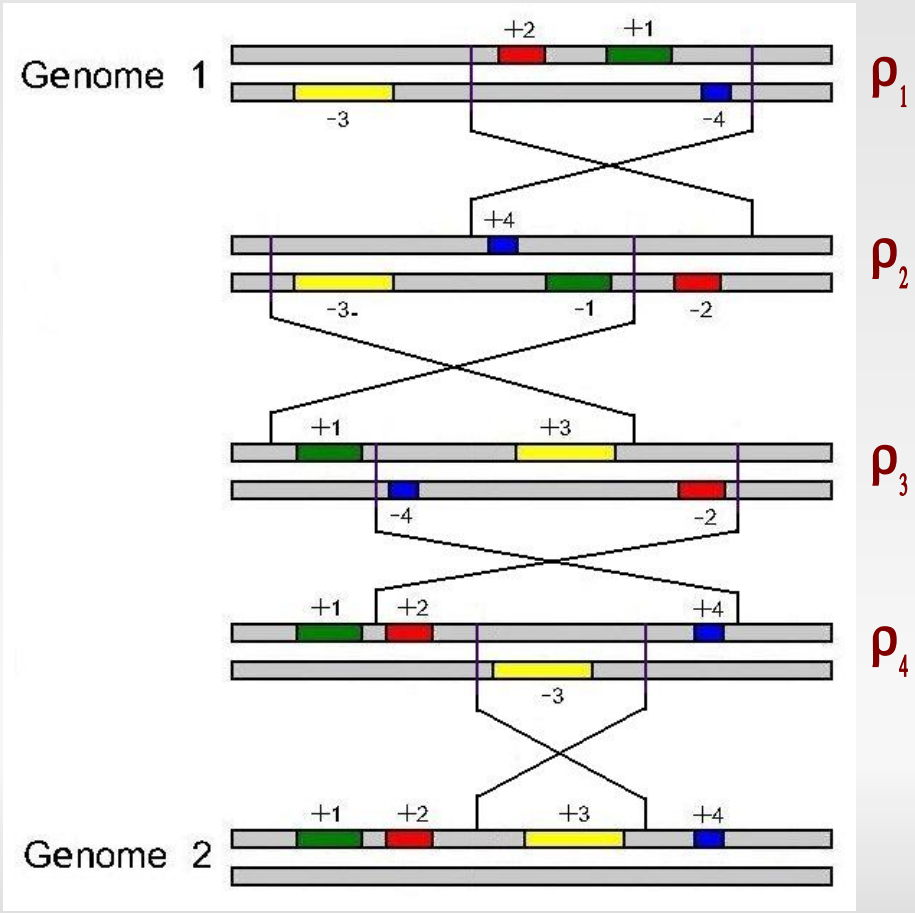
(-3, +2, +1, -4) $\rho_1 = \{1, 2, 4\}$

(-3, +4, -1, -2) $\rho_2 = \{1, 3, 4\}$

(+1, -4, +3, -2) $\rho_3 = \{2, 3, 4\}$

(+1, +2, -3, +4) $\rho_4 = \{3\}$

(+1, +2, +3, +4) (identity permutation)



Sorting by reversals (2)

- **d(π)** is the **reversal distance** for a permutation π
(the minimum number of reversals required to sort π)

- A sequence *S* of reversals which sorts π is an **optimal solution** for π if $|S| = d(\pi)$.

In the **previous example**, the sorting sequence of reversals
S = {1, 2, 4} {1, 3, 4} {2, 3, 4} {3}
is an **optimal solution** for π = **(-3, +2, +1, -4)**

- Given a permutation π, **calculating d(π)** and **finding one optimal solution for π** can be computed in **polynomial time** (Hannenhalli and Pevzner, 1995)

- Several other approaches find **one** optimal solution...

The solution space (1)

- The **number of optimal solutions** for the sorting by reversals problem is usually **huge**

- Some examples:

π = (-3, +2, +1, -4)
d = 4 ; s = 28

π = (-4, +1, -3, +6, -7, -5, +2)
d = 6 ; s = 204

π = (-6, +5, +7, -1, -4, +3, +2)
d = 6 ; s = 496

π = (-4, -3, +12, -11, -8, +10, +9, +7, -6, -5, +2, -1)
d = 8 ; s = 31 752

π = (-4, +3, +12, -11, -8, +10, +9, +7, -6, -5, +2, -1)
d = 9 ; s = 407 232

π = (-12, +11, -10, +6, +13, -5, +2, +7, +8, -9, +3, +4, +1)
d = 10 ; s = 8 278 540

π = (-12, +11, -10, -1, +16, -4, -3, +15, -14, +9, -8, -7, -2, -13, +5, -6)
d = 12 ; s = 505 634 256

π = (-12, +11, -10, +6, -5, +13, +2, +7, +8, -9, +14, -15, +3, +4, -16, +1)
d = 13 ; s = 40 313 272 766

The solution space (2)

Siepel (2003) proposed an algorithm that gives all optimal **next reversals** for a given permutation π.

Example:

For **(-3, +2, +1, -4)**, the possible next reversals are
{1}, {1,2,3}, {2}, {3}, **{1,2,4}**, {4}

After aplying **{1,2,4}** to **(-3, +2, +1, -4)**, we obtain **(-3, +4, -1, -2)**,
for which the possible next reversals are {3}, {1,3,4}

This algorithm allows the **enumeration of all existing optimal solutions** for π.

(but the **number of optimal solutions** for the sorting by reversals problem is usually **huge**)

Traces (1)

Bergeron et al (2002):

- Many optimal solutions are **equivalent**

{1, 2, 4} {1, 3, 4} {2, 3, 4} {3}
{1, 2, 4} {1, 3, 4} {3} {2, 3, 4}
{1, 2, 4} {3} {1, 3, 4} {2, 3, 4}
{3} {1, 2, 4} {1, 3, 4} {2, 3, 4}

π = (-3, +2, +1, -4)

- A **trace** is a set of **optimal solutions** composed by the **same reversals** but in **different orders**

- The **set of all optimal solutions** for a permutation π is a **union of traces**

Finding an element of **each trace without enumerating all solutions** was stated to be an **open problem**

Traces (2)

Two reversals ρ and θ **commute** if they are **disjoint sets** or if **one is a subset of the other**.

Examples:

{1,3,4} and {2,5} commute
{1,3,4} and {3} commute
{1,2,4} and {1,3,4} do not commute

If two reversals ρ and θ commute, then any optimal sequence of reversals containing $\rho\theta$ as a substring is equivalent to the same sequence, replacing $\rho\theta$ by $\theta\rho$

{1, 2, 4} **{1, 3, 4} {3}** {2, 3, 4} is equivalent to
{1, 2, 4} **{3} {1, 3, 4}** {2, 3, 4}

A **trace** is a set of **optimal sequences** which are all **equivalent** under the transitive closure of this **commuting relation**.

i-traces and prefixes

All **elements** of a trace have the **same number of reversals**.
We call **i-trace** a trace which **elements** have ***i*** **reversals**.

Example of a **4-trace** (each element has 4 reversals):

{1, 2, 4} {1, 3, 4} {2, 3, 4} {3}
{1, 2, 4} {1, 3, 4} {3} {2, 3, 4}
{1, 2, 4} {3} {1, 3, 4} {2, 3, 4}
{3} {1, 2, 4} {1, 3, 4} {2, 3, 4}

(this **4-trace** has **four** elements; a coincidence!)

The **k-prefixes** of an **i-trace** ***t*** are the traces that are formed by sequences which correspond to the **first** ***k*** **reversals** of the elements of ***t***.

The previous **4-trace** has **two 3-prefixes** (3-traces):

{1, 2, 4} {1, 3, 4} {2, 3, 4}

(a **3-trace** with only **one** element)

{1, 2, 4} {1, 3, 4} {3}
{1, 2, 4} {3} {1, 3, 4}
{3} {1, 2, 4} {1, 3, 4}

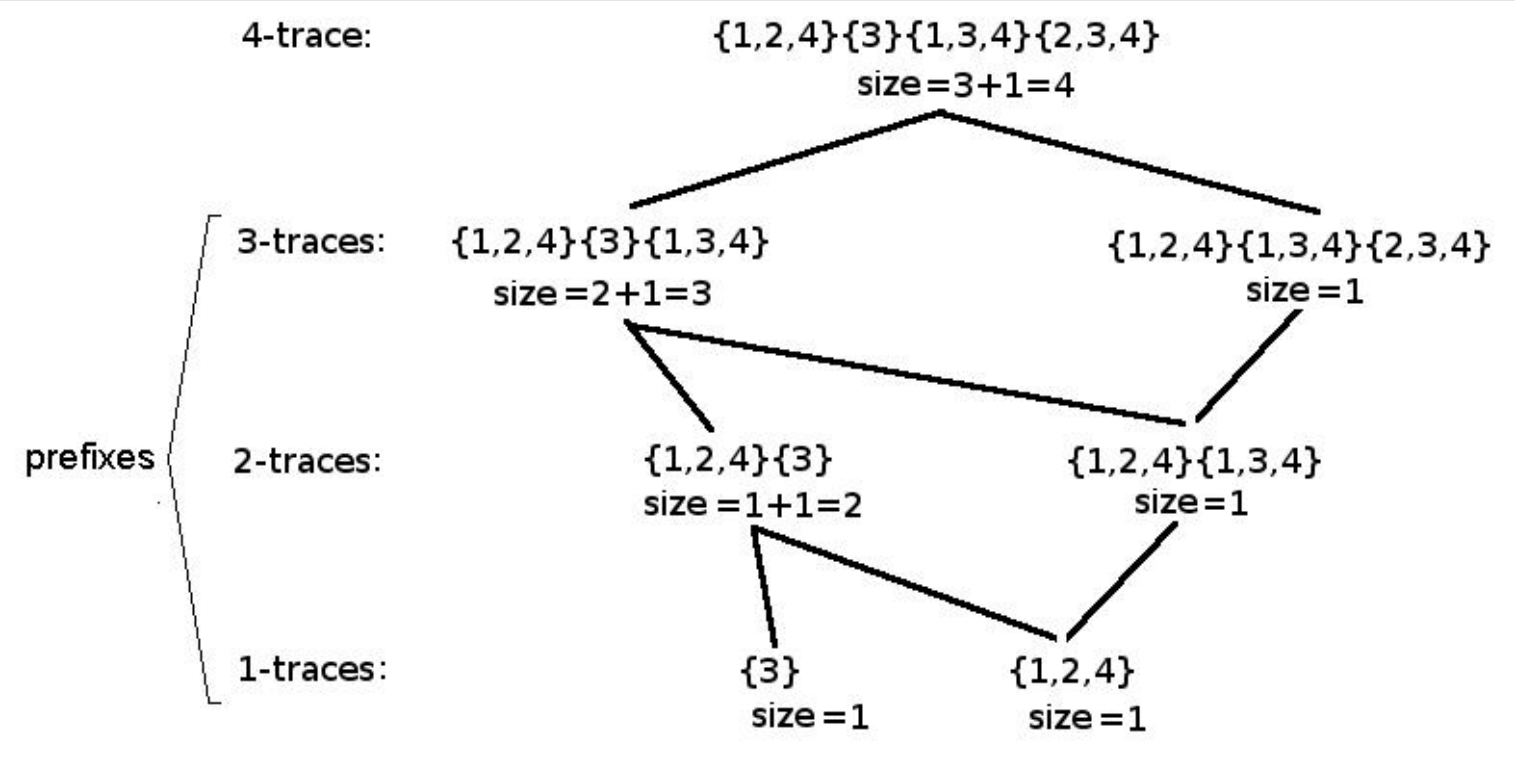
(a **3-trace** with **three** elements)

[The **size** (number of elements) of the **4-trace** is the **sum of the sizes** of its **3-prefixes**]

Decomposing a trace

The **size** (number of elements) of an **i-trace** is the **sum of the sizes** of its **(i-1)-prefixes**.

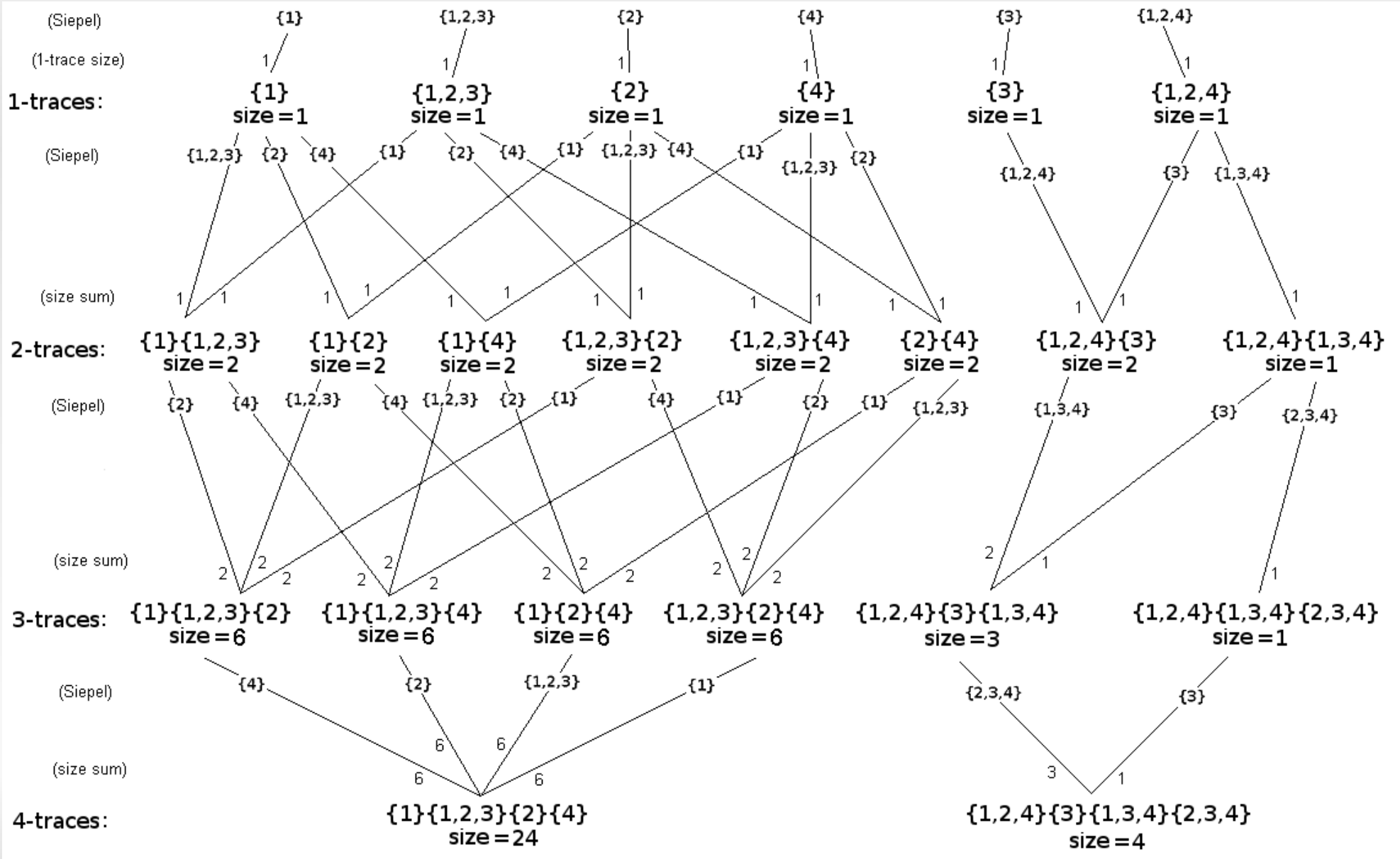
The **size** of a **1-trace** is **1** (trivial)



Constructing all the traces

$\pi = (-3,+2,+1,-4)$

ISBRA 2007
Marília D. V. Braga

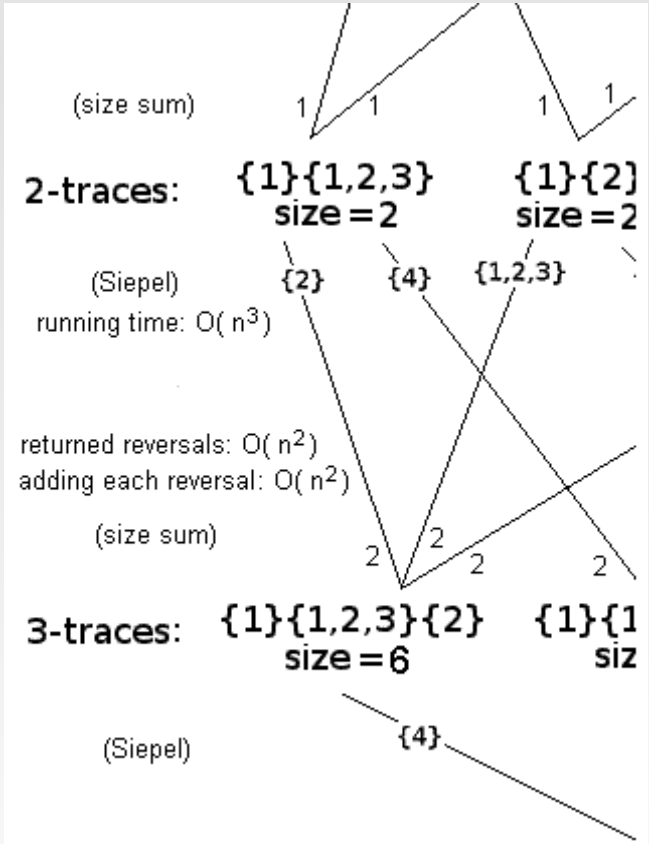


Theoretical complexity (1)

. n = size of the input permutation

. N = number of final traces

For **each partial trace (prefix)**, we run the algorithm of **Siepel**, which complexity is **$O(n^3)$**, and then **add each of the $O(n^2)$** returned reversals to the partial trace in **$O(n^2)$**. Thus, for **each partial trace**, the additional processing time is $O(n^3 + n^2 . n^2) = \mathbf{O(n^4)}$.



The theoretical complexity depends on the **total number of partial traces**, which are prefixes of final traces.

Theoretical complexity (2)

The **number of prefixes** of an i-trace is bounded by $i^k = O(n^k)$, where **k is the width** of the i-trace (Steiner, 1986)

The **width** of a trace *t* is defined as the **biggest subset** of reversals of *t* such that **every pair of reversals** in this subset **commutes**.

Example:

Trace: {1, 2, 4}{1, 3, 4}{2, 3, 4}{3}

Subsets:

{ {1, 2, 4}{3} }, size = 2
{ {1, 3, 4}{3} }, size = 2
{ {2, 3, 4}{3} }, size = 2

Width = 2

The width of a trace can be calculated in polynomial time (Fulkerson, 1956)

Theoretical complexity (3)

. n = size of the input permutation

. N = number of final traces

The **total number of partial traces** (prefixes of final traces)

is $\sum_1^N n^k = \mathbf{O(N.\ n^{kmax})}$ (kmax is the maximum width of a final trace)

For **each partial trace**, the additional processing time is

$O(\ n^3 + n^2\ .\ n^2\ ) = \mathbf{O(\ n^4\ )}$

Theoretical complexity: $N\ .\ n^{kmax}\ .\ n^4\ = O(\ N.\ n^{kmax+4}\ )$

(For our example, we have n = 4, N = 2 and kmax = 4)

Algorithm implementation

The algorithm has been implemented using the Java Technology* , integrated to the **baobabLuna** framework.

On-line download:

http://biomserv.univ-lyon1.fr/~marilia/baobabLuna.html

baobabLuna is a java framework to deal with permutations - A collection of classes for building breakpoint graphs (the basic structures behind the work on genome rearrangements), performing reversals, calculating reversal distances, sorting permutations.

* java.sun.com

Experimental results

Experiences were made in a personal computer with a 1.8GHz CPU and 1GB of RAM

PERMUTATION	# Solutions	# Traces	Enum. solutions	Constr. traces
rat / human chr X n=16 d=10	2 419 750	418	≃10 min	≃10 s
mouse / human chr X n=16 d=10	3 362 310	218	≃12 min	≃10 s
random n=17 d=11	57 019 369	18 255	≃ 4 h	≃ 5 min
random n=18 d=12	327 905 046	34 317	≃ 24 h	≃18 min
human chr X / chr Y n=30 d=12	207 600 628	115 512	> 24 h	≃ 28 min

Both algorithms have been implemented in baobabLuna framework:

http://biomserv.univ-lyon1.fr/~marilia/baobabLuna.html

Conclusions

Our algorithm **gives a representation of all solutions** of sorting signed permutations by reversals, **without enumerating all solutions**

The **solution space is dramatically reduced** when dealing with traces

An **implementation** of this algorithm is available **on-line**, integrated to the **baobabLuna** framework.

But unfortunately...

The **solution space** represented by traces **is still too big** for direct human interpretation

The **algorithm implementation is limited** to permutations with reversal distance bounded by 20

Thank you !

References:

Hannenhalli S. and Pevzner P., Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals), In Proceedings of the 27th Annual ACM-SIAM Symposium on the Theory of Computing, pages 178-189, 1995.

Hannenhalli, S. and Pevzner, P., Transforming men into mice (polynomial algorithm for genomic distance problem), In Proceedings of the IEEE 36th Annual Symposium on Foundations of Computer Science, pages 581-592, 1995.

Bergeron A., Chauve C., Hartmann T., St-Onge K., On the properties of sequences of reversals that sort a signed permutation. JOBIM 2002, 99-108.

Siepel A. An algorithm to enumerate sorting reversals for signed permutations. J Comput Biol 10:575-597, 2003.

Steiner G., An algorithm to generate the ideals of a partial order. Operations Research Letters, 5(6):317-320, 1986.

Fulkerson D.R., Note on Dilworth's decomposition theorem for partially ordered sets, Proc. Amer. Math. Soc. 7 (1956), 701-702