Minimum Mosaic Inference of a Set of Recombinants

Guillaume Blin\textsuperscript{1}  Florian Sikora\textsuperscript{1}  Romeo Rizzi\textsuperscript{2}  Stéphane Vialette\textsuperscript{1}

\textsuperscript{1}LIGM, Université Paris-Est Marne-la-Vallée - France

\textsuperscript{2}DIMI, Università di Udine - Italy
Minimum Mosaic Inference of a Set of Recombinants

Guillaume Blin\textsuperscript{1}  Florian Sikora\textsuperscript{1}  Romeo Rizzi\textsuperscript{2}  Stéphane Vialette\textsuperscript{1}

\textsuperscript{1}LIGM, Université Paris-Est Marne-la-Vallée - France

\textsuperscript{2}DIMI, Università di Udine - Italy
Minimum Mosaic Inference of a Set of Recombinants

Guillaume Blin¹  Florian Sikora¹  Romeo Rizzi²  Stéphane Vialette¹

¹LIGM, Université Paris-Est Marne-la-Vallée - France
²DIMI, Università di Udine - Italy
Minimum Mosaic Inference of a Set of Recombinants

Guillaume Blin\textsuperscript{1} \hspace{0.5cm} Florian Sikora\textsuperscript{1} \hspace{0.5cm} Romeo Rizzi\textsuperscript{2} \\
Stéphane Vialette\textsuperscript{1}

\textsuperscript{1}LIGM, Université Paris-Est Marne-la-Vallée - France

\textsuperscript{2}DIMI, Università di Udine - Italy
Outline

Introduction

NP-Hardness

Exact Algorithms

Conclusion
Outline

Introduction

NP-Hardness

Exact Algorithms

Conclusion
SNPs

- SNP (Single Nucleotide Polymorphism)

- When a single nucleotide (A,C,G,T) differs in the genome of two members of a specie (or paired chromosome in an individual)

Figure by D. Hall
SNPs

- SNP (Single Nucleotide Polymorphism)
- When a single nucleotide (A,C,G,T) differs in the genome of two members of a specie (or paired chromosome in a individual)
- Represents 90% of the human genetic variation
- Must cheaper to collect than full sequence data
SNPs

- In most SNPs, **two** (of four) different nucleotides occurs
SNPs

- In most SNPs, **two** (of four) different nucleotides occurs
- Can use **0** and **1** – **binary** data
Recombination

- Principal process inducing these genetic variations
Recombination

- Principal process inducing these genetic variations
- Two equal length sequences...
Recombination

- Principal process inducing these genetic variations
- Two equal length sequences...
- ...generates a third of same length

110001111111001

000110000001111

11000 000001111

000110000001111
Recombination

- Principal process inducing these genetic variations
- Two equal length sequences...
- ...generates a third of same length
- Concatenation of a prefix in the first one and a suffix in the second one [KOIVISTO ET AL. 04]
Recombination

- Principal process inducing these genetic variations
- Two equal length sequences...
- ...generates a third of same length
- Concatenation of a prefix in the first one and a suffix in the second one [KOIVISTO ET AL. 04]
Founders sequences

- Current sequences are descendant of a small number of **founders sequences**
- A current sequence is composed of **blocks from the founders**, due to recombination
Mosaic

- Really look like a mosaic!

Generated by RecBlock
Mosaic Problem [UKKONEN 02]

- **Input**: A set of *m* sequences (current population) of length *n*, an integer *K*

- **Output**: A set of *K* founders sequences that induce a minimum number of breakpoints
State of the art

- Polynomial in $\mathcal{O}(mn)$ if $K = 2$ [Ukkonen 02, Wu et al. 07]
- Exact exponentials algorithms [Ukkonen 02, Wu et al. 07]
- Heuristics [Roly & Blum 09]
- Lower bounds on the minimum number of breakpoints needed [Wu 10]
State of the art

- Polynomial in $O(mn)$ if $K = 2$ [Ukkonen 02, Wu et al. 07]
- Exact exponentials algorithms [Ukkonen 02, Wu et al. 07]
- Heuristics [Roly & Blum 09]
- Lower bounds on the minimum number of breakpoints needed [Wu 10]

- What about the complexity if $K > 2$?
Outline

Introduction

NP-Hardness

Exact Algorithms

Conclusion
Hardness

- A first step in an answer: the problem is NP-Complete if the number of founders is not bounded
- Just some tricks for the proof...
Tool 1: Using arbitrary string

- If the problem on arbitrary strings is NP-hard, then so is the problem on binary strings.
Tool 1: Using arbitrary string

- If the problem on *arbitrary strings* is NP-hard, then so is the problem on binary strings
  - Suppose an alphabet $\Sigma$

- Example
  - $\Sigma = A, B, C$
Tool 1: Using arbitrary string

- If the problem on **arbitrary strings** is NP-hard, then so is the problem on binary strings
  - Suppose an alphabet $\Sigma$
  - Take any encoding $\delta$ of symbols in $\Sigma$ by binary strings of length $\lceil \log_2 |\Sigma| \rceil$

- Example
  - $\Sigma = A, B, C$
  - $\delta(A) = 00$
  - $\delta(B) = 01$
  - $\delta(C) = 10$
Tool 1: Using arbitrary string

- \( \Leftrightarrow \) Any solution with strings over \( \Sigma \) maps into a solution for binary strings without changing the number of breakpoints.

**Example**

- \( \Sigma = A, B, C \)
- \( \delta(A) = 00 \)
- \( \delta(B) = 01 \)
- \( \delta(C) = 10 \)
Tool 1: Using arbitrary string

- \((\Rightarrow)\) Any solution with strings over \(\Sigma\) maps into a solution for binary strings without changing the number of breakpoints.

- \((\Leftarrow)\) If we cannot map the binary founders sequence to symbols of \(\Sigma\), then we can replace the missing "word" by its longest suffix in common in \(\Sigma\) without increasing the cost.

Example:
- \(\Sigma = A, B, C\)
- \(\delta(A) = 00\)
- \(\delta(B) = 01\)
- \(\delta(C) = 10\)
Tool 2: Forcing Founders

- One can force $K' < K$ founders to be part of the solution
- Add $nm$ copies of each forced founders in the input
Tool 2: Forcing Founders

- One can force $K' < K$ founders to be part of the solution
- **Add $nm$ copies of each forced founders** in the input
- If the ”forced founder” is not in the solution founders:
Tool 2: Forcing Founders

- One can force $K' < K$ founders to be part of the solution
- **Add $nm$ copies of each forced founders** in the input
- If the "forced founder" is not in the solution founders:
  - Induce at least 1 breakpoint for one sequence
Tool 2: Forcing Founders

- One can force $K' < K$ founders to be part of the solution
- Add $nm$ copies of each forced founders in the input
- If the ”forced founder” is not in the solution founders:
  - Induce at least 1 breakpoint for one sequence
  - Therefore induce $nm$ breakpoints on the whole...
Proof idea

- From the NP-Complete problem \textsc{Vertex Cover}
Vertex Cover
Vertex Cover
Reduction idea
Reduction idea

Input:

\[ ZZX_u X_u ZZZZZX_u X_u ZZ \]
\[ ZZX_v X_v ZZZZZX_w X_w ZZ \]

\[ 6.|E| \]
Reduction idea

Input:
ZZX_uX_uZZZZX_uX_uZZ
ZZX_vX_vZZZZX_wX_wZZ

6.|E|
Reduction idea

Input:
\[ Z \ Z X_u X_u Z \ Z \]
\[ Z \ Z X_v X_v Z \ Z \]
\[ X_u X_u X_u X_u X_u X_u (\times 6. |E| + 1 = 7) \]
\[ X_v X_v X_v X_v X_v X_v (\times 7) \]
Reduction idea

Input:
\[ Z \ Z \ X_u \ X_u \ Z \ Z \]
\[ Z \ Z \ X_v \ X_v \ Z \ Z \]
\[ X_u X_u X_u X_u X_u X_u \ (\times 7) \]
\[ X_v X_v X_v X_v X_v X_v \ (\times 7) \]

Forced founders:
\[ X_u X_u X_u \ Z \ Z \ Z \]
\[ Z \ Z \ Z \ X_u X_u X_u \]
\[ X_v X_v X_v \ Z \ Z \ Z \]
\[ Z \ Z \ Z \ X_v X_v X_v \]
Reduction idea

Input:
\[ZZ X_u X_u ZZ Z \]
\[ZZ X_v X_v ZZ \]
\[X_u X_u X_u X_u X_u X_u (\times 7) \]
\[X_v X_v X_v X_v X_v X_v (\times 7) \]

Forced founders:
\[X_u X_u X_u ZZ ZZ Z \]
\[ZZ ZZ X_u X_u X_u \]
\[X_v X_v X_v ZZ ZZ \]
\[ZZ ZZ X_v X_v X_v \]
Reduction idea

Input:

$$\begin{align*}
\text{Z Z } & X_u X_u Z Z \\
\text{Z Z } & X_v X_v Z Z \\
X_u X_u X_u & X_u X_u X_u (\times 7) \\
X_v X_v X_v & X_v X_v X_v (\times 7)
\end{align*}$$

Forced founders:

$$\begin{align*}
X_u X_u X_u & Z Z Z \\
Z Z Z & X_u X_u X_u \\
X_v X_v X_v & Z Z Z \\
Z Z Z & X_v X_v X_v
\end{align*}$$

- Remains |Vertex Cover| founders (here 1)

Mosaic Problem (18/25)
Reduction idea

Input:

\[
\begin{align*}
ZZX_uX_uZZ

ZZX_vX_vZZ

X_uX_uX_uX_uX_uX_u(\times 7)

X_vX_vX_vX_vX_vX_v(\times 7)
\end{align*}
\]

Forced founders:

\[
\begin{align*}
X_uX_uX_uZZZZ

ZZZZX_uX_uX_u

X_vX_vX_vZZZZ

ZZZZX_vX_vX_v
\end{align*}
\]

- Remains $|\text{Vertex Cover}|$ founders (here 1)
- Will be sequences "$X_iX_i...$" due to $(\times 7)$
- It is a vertex cover otherwise first sequences generate more breakpoints
Outline

Introduction

NP-Hardness

Exact Algorithms

Conclusion
Polynomial-time Algorithm

▶ Suppose one knows where the breakpoints are
Polynomial-time Algorithm

- Suppose one **knows where the breakpoints are**

Input:

![Input Diagram]

Each substring without breakpoints must by definition appear in the solution. Add the substring with the leftmost startpoint in the output.

\[ O(|\text{Breakpoints}| \times |\text{Output}| \times |\text{Longest block}|) \]
Polynomial-time Algorithm

▶ Suppose one **knows where the breakpoints are**

Input:

Each substring without breakpoints must by definition appears in the solution
Polynomial-time Algorithm

- Suppose one knows where the breakpoints are

Input:

Each substring without breakpoints must by definition appears in the solution

Add the substring with the leftmost startpoint in the output
Polynomial-time Algorithm

- Suppose one *knows where the breakpoints are*

  Input:

  ![Input Diagram]

  - Each substring without breakpoints must by definition appears in the solution
  - Add the substring with the leftmost startpoint in the output

  Output:

  ![Output Diagram]
Polynomial-time Algorithm

- Suppose one \textit{knows where the breakpoints are}

\begin{itemize}
  \item Input:
  \begin{itemize}
    \item Each substring without breakpoints must by definition appears in the solution
    \item Add the substring with the leftmost startpoint in the output
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item Output:
  \begin{itemize}
  \end{itemize}
\end{itemize}
Polynomial-time Algorithm

- Suppose one \textit{knews where the breakpoints are}

Input:

- Each substring without breakpoints must by definition appears in the solution
- Add the substring with the leftmost startpoint in the output

Output:
Polynomial-time Algorithm

- Suppose one **knows where the breakpoints are**

  ![Input Diagram]

- Each substring without breakpoints must by definition appears in the solution
- Add the substring with the leftmost startpoint in the output

![Output Diagram]
Polynomial-time Algorithm

- Suppose one **knows where the breakpoints are**

  ![Input Diagram]

  - Each substring without breakpoints must by definition appears in the solution
  - Add the substring with the leftmost startpoint in the output

  ![Output Diagram]
Polynomial-time Algorithm

- Suppose one **knows where the breakpoints are**

```
Input :

```

- Each substring without breakpoints must by definition appears in the solution

- Add the substring with the leftmost startpoint in the output

```
Output :

```
Polynomial-time Algorithm

- Suppose one knows where the breakpoints are

  Input:

  - Each substring without breakpoints must by definition appears in the solution
  - Add the substring with the leftmost startpoint in the output

  Output:
Polynomial-time Algorithm

- Suppose one *knows where the breakpoints are*

Input:

- Each substring without breakpoints must by definition appears in the solution
- Add the substring with the leftmost startpoint in the output

Output:
Polynomial-time Algorithm

- Suppose one **knows where the breakpoints are**

  Input:

  ![Input Diagram]

- Each substring without breakpoints must by definition appears in the solution

- Add the substring with the leftmost startpoint in the output

  Output:

  ![Output Diagram]
Polynomial-time Algorithm

▶ Suppose one knows where the breakpoints are

Input:

▶ Each substring without breakpoints must by definition appears in the solution
▶ Add the substring with the leftmost startpoint in the output
▶ $O(|\text{Breakpoints}| \times |\text{Output}| \times |\text{Longest block}|)$

Output:
If one only knows the number of breakpoints $B_i$ for each input sequence of size $n$:
Polynomial-time algorithm

- If one only \textbf{knows the number of breakpoints} $B_i$ for each input sequence of size $n$:
- One can "guess" where all breakpoints can be:

$$n$$

$$B_1 \Rightarrow \binom{n}{B_1} = \mathcal{O}(n^{B_1})$$
$$B_2 \Rightarrow \binom{n}{B_2} = \mathcal{O}(n^{B_2})$$
$$B_3 \Rightarrow \binom{n}{B_3} = \mathcal{O}(n^{B_3})$$
$$B_4 \Rightarrow \binom{n}{B_4} = \mathcal{O}(n^{B_4})$$

florian.sikora@univ-mlv.fr Mosaic Problem (21/25)
Polynomial-time algorithm

- If one only knows the number of breakpoints $B_i$ for each input sequence of size $n$:
- One can "guess" where all breakpoints can be:
- And launch the previous algorithm

$n$

$B_1 \Rightarrow \binom{n}{B_1} = \mathcal{O}(n^{B_1})$
$B_2 \Rightarrow \binom{n}{B_2} = \mathcal{O}(n^{B_2})$
$B_3 \Rightarrow \binom{n}{B_3} = \mathcal{O}(n^{B_3})$
$B_4 \Rightarrow \binom{n}{B_4} = \mathcal{O}(n^{B_4})$
Polynomial-time algorithm

- If one only **knows the number of breakpoints** $B_i$ for each input sequence of size $n$:
- One can "guess" where all breakpoints can be:
- And launch the previous algorithm
- Overall complexity: $\mathcal{O}(n^{B_1}.n^{B_2} \ldots n^{B_m}.BKn) = \mathcal{O}(n^B.BKn)$

\[
\begin{align*}
B_1 & \Rightarrow \binom{n}{B_1} = \mathcal{O}(n^{B_1}) \\
B_2 & \Rightarrow \binom{n}{B_2} = \mathcal{O}(n^{B_2}) \\
B_3 & \Rightarrow \binom{n}{B_3} = \mathcal{O}(n^{B_3}) \\
B_4 & \Rightarrow \binom{n}{B_4} = \mathcal{O}(n^{B_4})
\end{align*}
\]
Polynomial-time algorithm

- If one only knows the number of overall breakpoints $B$

Overall complexity: $O((K + B)BnK^2B)$. 

florian.sikora@univ-mlv.fr  
Mosaic Problem (22/25)
If one only **knows the number of overall breakpoints** $B$

- Maximum number of different input sequences
Polynomial-time algorithm

- If one only **knows the number of overall breakpoints** $B$
- Maximum number of different input sequences $= B$

$$B \quad \begin{array}{c} \end{array}$$
Polynomial-time algorithm

- If one only **knows the number of overall breakpoints** $B$
- Maximum number of different input sequences $= B + K$

$B$

$K$
Polynomial-time algorithm

- If one only knows the number of overall breakpoints $B$
- Maximum number of different input sequences $= B + K$

\[
\begin{array}{c}
\{ B \\
K \}
\end{array}
\]

- Decide which have the breakpoints: \( \binom{K+B}{B} = \mathcal{O}((K + B)^B) \)
Polynomial-time algorithm

- If one only **knows the number of overall breakpoints** $B$
- Maximum number of different input sequences $= B + K$
  
  $B \left\{ \begin{array}{c} \rule{0pt}{1.5ex} \\ \rule{0pt}{1.5ex} \end{array} \right. \left\{ \begin{array}{c} \rule{0pt}{1.5ex} \\ \rule{0pt}{1.5ex} \end{array} \right. \left\{ \begin{array}{c} \rule{0pt}{1.5ex} \\ \rule{0pt}{1.5ex} \end{array} \right.\left\{ \begin{array}{c} \rule{0pt}{1.5ex} \\ \rule{0pt}{1.5ex} \end{array} \right.
  
  $K \left\{ \begin{array}{c} \rule{0pt}{1.5ex} \\ \rule{0pt}{1.5ex} \end{array} \right.$

- Decide which have the breakpoints: $\binom{K+B}{B} = \mathcal{O}((K + B)^B)$
- For each, run the $\mathcal{O}(nK^{2m})$ Ukkonen’s algorithm
Polynomial-time algorithm

- If one only knows the number of overall breakpoints $B$
- Maximum number of different input sequences $= B + K$

\[
\begin{align*}
B & \quad \text{Decide which have the breakpoints} : \\ & \quad \binom{K+B}{B} = \mathcal{O}((K + B)^B) \\
K & \quad \text{For each, run the } \mathcal{O}(nK^2m) \text{ Ukkonen’s algorithm} \\
\end{align*}
\]

- Our sequences of interest are $m = B$
Polynomial-time algorithm

- If one only knows the number of overall breakpoints \( B \)
- Maximum number of different input sequences \( = B + K \)
  \[
  B \binom{K+B}{B} = \mathcal{O}((K+B)^B)
  \]
- Decide which have the breakpoints
- For each, run the \( \mathcal{O}(nK^{2m}) \) Ukkonen’s algorithm
  - Our sequences of interest are \( m = B \)
- Overall complexity: \( \mathcal{O}((K + B)^B.nK^{2B}) \)
Outline

Introduction

NP-Hardness

Exact Algorithms

Conclusion
If $K = 2$, Mosaic Problem is polynomial time solvable
If $K$ is not bounded, NP-Complete
Conclusion

- If $K = 2$, Mosaic Problem is polynomial time solvable
- If $K$ is not bounded, NP-Complete
- What about the complexity when $K$ is bounded? FPT?
- What about the existence of a PTAS?
Questions?

Guillaume Blin\textsuperscript{1}  \hspace{1em} Florian Sikora\textsuperscript{1}  \hspace{1em} Romeo Rizzi\textsuperscript{2}  
Stéphane Vialette\textsuperscript{1}

\textsuperscript{1}LIGM, Université Paris-Est Marne-la-Vallée - France

\textsuperscript{2}DIMI, Università di Udine - Italy