Comparing Genomes with Rearrangements and Segmental Duplications

Mingfu Shao    Bernard M.E. Moret

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

June 8, 2014
Model of Genomes

- **Genome**: A set of chromosomes.
- **Chromosome**: A linear/circular list of genes (synteny blocks).
**Model of Genomes**

- **Genome**: A set of chromosomes.
- **Chromosome**: A linear/circular list of genes (synteny blocks).

![Diagram of Genome and Chromosome with labeled genes and arrows indicating transcriptional direction.](image)

Each gene has a sign, indicating its transcriptional direction. All genes are grouped into gene families.
**Model of Genomes**

- **Genome**: A set of chromosomes.
- **Chromosome**: A linear/circular list of genes (synteny blocks).

Each gene has a **sign**, indicating its transcriptional direction.
Model of Genomes

- **Genome:** A set of chromosomes.

- **Chromosome:** A linear/circular list of genes (synteny blocks).

Each gene has a **sign**, indicating its transcriptional direction.

All genes are grouped into **gene families**.
Comparing Genomes

Different copy numbers for some gene families.
Different number of chromosomes.
Different gene orders and orientations.
Comparing Genomes

Different copy numbers for some gene families.

Different number of chromosomes.

Different gene orders and orientations.
Comparing Genomes

- Different copy numbers for some gene families.
- Different number of chromosomes.
- Different gene orders and orientations.

(G1) 

(G2)
Genome Evolution

- **Content-modifying events** (segmental duplication, lateral transfer, etc): change copy numbers

- **Rearrangements** (inversion, translocation, fission, fusion, etc): change number of chromosomes, gene orders and orientations.
**Genome Evolution**

- **Content-modifying events** (segmental duplication, lateral transfer, etc): change copy numbers

- **Rearrangements** (inversion, translocation, fission, fusion, etc): change number of chromosomes, gene orders and orientations.

![Diagram](image)

- **Segmental duplication**
- **Inversion**
- **Chromosomal fission**

- **(G₁)**
- **(G₂)**
Parsimony Reconstruction

- **Edit Distance Problem**: to compute a set of *minimum* number of operations that can explain the difference of the two given genomes.
**Edit Distance Problem:** to compute a set of minimum number of operations that can explain the difference of the two given genomes.

**Evolutionary Model (possible operations):**
- Segmental duplications
- DCJ (double-cut-and-join) operations

\[
\text{Inversion}
\quad \text{Translocation}
\quad \text{Chromosomal fission}
\quad \text{Chromosomal fusion}
\quad \Rightarrow \quad \text{DCJ operation}
\]
DCJ (double-cut-and-join) Operation

Some definitions

Extremities: two ends (head and tail) of a gene

Adjacency: two consecutive extremities

Null extremity: special extremity 0 added to each end of the linear chromosomes

DCJ operation:

\[
\{p, q\} + \{r, s\} \Rightarrow \{p, r\} + \{q, s\}
\]

page 6 of 23
DCJ (double-cut-and-join) Operation

Some definitions

Extremities: two ends (head and tail) of a gene

Adjacency: two consecutive extremities

Null extremity: special extremity 0 added to each end of the linear chromosomes

DCJ operation: \{p, q\} + \{r, s\} = \Rightarrow \{p, r\} + \{q, s\}
Some definitions

- **Extremities:** two ends (head and tail) of a gene

The diagram shows the DCJ operation with the following labels:
- $a$, $b$, $c$, $d$ represent genes
- $a_t$ and $a_h$ denote extremities (head and tail) of a gene.
DCJ (double-cut-and-join) Operation

- Some definitions

  - Extremities: two ends (head and tail) of a gene
  - Adjacency: two consecutive extremities
Some definitions

- **Extremities:** two ends (head and tail) of a gene
- **Adjacency:** two consecutive extremities
- **Null extremity:** special extremity 0 added to each end of the linear chromosomes
Some definitions

- **Extremities:** two ends (head and tail) of a gene
- **Adjacency:** two consecutive extremities
- **Null extremity:** special extremity 0 added to each end of the linear chromosomes

**DCJ operation:** \( \{p, q\} + \{r, s\} \implies \{p, r\} + \{q, s\} \)
Various Cases of DCJ Operation

\[
\begin{align*}
\{a_h, b_t\} & \quad \{c_h, d_t\} \\
\text{inversion} \\
\{a_h, c_h\} & \quad \{b_t, d_t\}
\end{align*}
\]
Various Cases of DCJ Operation

inversion

\[
\begin{align*}
& a \quad b \quad c \quad d \\
\{a_h, b_t\} & \quad \{c_h, d_t\}
\end{align*}
\]

\[
\begin{align*}
& a \quad -c \quad -b \quad d \\
\{a_h, c_h\} & \quad \{b_t, d_t\}
\end{align*}
\]

translocation

\[
\begin{align*}
& a^1 \quad b^1 \quad c^1 \quad d^1 \\
\{b_h^1, c_t^1\}
\end{align*}
\]

\[
\begin{align*}
& a^2 \quad b^2 \quad c^2 \quad d^2 \\
\{b_h^2, c_t^2\}
\end{align*}
\]

\[
\begin{align*}
& a^1 \quad b^1 \quad c^2 \quad d^2 \\
\{b_h^1, c_t^2\}
\end{align*}
\]

\[
\begin{align*}
& a^2 \quad b^2 \quad c^1 \quad d^1 \\
\{b_h^2, c_t^1\}
\end{align*}
\]
Various Cases of DCJ Operation

**Inversion**

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
\{a_h, b_t\} & \quad \{c_h, d_t\}
\end{align*}
\]

\[
\begin{align*}
\text{a} & \quad \text{c} & \quad \text{b} & \quad \text{d} \\
\{a_h, c_h\} & \quad \{b_t, d_t\}
\end{align*}
\]

**Translocation**

\[
\begin{align*}
\text{a}^1 & \quad \text{b}^1 & \quad \text{c}^1 & \quad \text{d}^1 \\
\{b_h^1, c_t^1\}
\end{align*}
\]

\[
\begin{align*}
\text{a}^2 & \quad \text{b}^2 & \quad \text{c}^2 & \quad \text{d}^2 \\
\{b_h^2, c_t^2\}
\end{align*}
\]

\[
\begin{align*}
\text{a}^1 & \quad \text{b}^1 & \quad \text{c}^2 & \quad \text{d}^2 \\
\{b_h^1, c_t^2\}
\end{align*}
\]

\[
\begin{align*}
\text{a}^2 & \quad \text{b}^2 & \quad \text{c}^1 & \quad \text{d}^1 \\
\{b_h^2, c_t^1\}
\end{align*}
\]

**Fusion**

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
\{b_h, 0\} & \quad \{d_h, 0\}
\end{align*}
\]

**Fission**

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
\{b_h, d_h\}
\end{align*}
\]
The adjacency graph consists of vertex-disjoint cycles.

DCJ distance: $d(B) = \#(\text{adjacencies}) - \#(\text{cycles})$. 
Adjacency Graph (w.r.t. a given bijection $B$)
Edit Distance under DCJ Operation

The adjacency graph consists of vertex-disjoint cycles.

Adjacency Graph (w.r.t. a given bijection $B$)
Edit Distance under DCJ Operation

The adjacency graph consists of vertex-disjoint cycles. DCJ distance:
\( d(B) = \#(\text{adjacencies}) - \#(\text{cycles}) \).

Adjacency Graph (w.r.t. a given bijection \( B \))
The adjacency graph consists of vertex-disjoint cycles. DCJ distance:
\[ d(B) = \#(\text{adjacencies}) - \#(\text{cycles}). \]

Adjacency Graph (w.r.t. a given bijection \( B \))
The adjacency graph consists of vertex-disjoint cycles. DCJ distance: $d(B) = \#(\text{adjacencies}) - \#(\text{cycles})$. 

Adjacency Graph (w.r.t. a given bijection $B$)
Adjacency Graph (w.r.t. a given bijection $B$)

The adjacency graph consists of vertex-disjoint cycles.
Adjacency Graph (w.r.t. a given bijection $B$)

The adjacency graph consists of vertex-disjoint cycles.

DCJ distance: $d(B) = \#(\text{adjacencies}) - \#(\text{cycles})$. 
List all candidate segmental duplications.

\[ S_1 = \{(a_1, -c_1), (a_2, -c_2), (b_1), (b_2), (a_1), (c_1), (a_2), (c_2)\} \]

\[ S_2 = \{(b_1), (b_2), (b_3)\} \]

**Edit Distance Problem:** to compute \( X_1 \subset S_1 \) and \( X_2 \subset S_2 \), and a bijection \( B \) between non-duplicated genes (those genes not in \( X_1 \cup X_2 \)), such that 

\[ |X_1| + |X_2| + d(B) \] 

is minimized.
List all candidate segmental duplications.

\[ S_1 = \{ (a_1, -c_1), (a_2, -c_2), (b_1), (b_2), (a_1), (c_1), (a_2), (c_2) \} \]

\[ S_2 = \{ (b_1), (b_2), (b_3) \} \]

Edit Distance Problem: to compute \( X_1 \subset S_1 \) and \( X_2 \subset S_2 \), and a bijection \( B \) between non-duplicated genes (those genes not in \( X_1 \cup X_2 \)), such that

\[ |X_1| + |X_2| + d(B) \]

is minimized.
List all candidate segmental duplications.

\[
S_1 = \{(a^1, -c^1), (a^2, -c^2), (b^1), (b^2), (a^1), (c^1), (a^2), (c^2)\}
\]

\[
S_2 = \{(b^1), (b^2), (b^3)\}
\]
Edit Distance under SDs and DCJs

List all candidate segmental duplications.

\[ S_1 = \{ (a^1, -c^1), (a^2, -c^2), (b^1), (b^2), (a^1), (c^1), (a^2), (c^2) \} \]
\[ S_2 = \{ (b^1), (b^2), (b^3) \} \]

Edit Distance Problem: to compute \( X_1 \subset S_1 \) and \( X_2 \subset S_2 \), and a bijection \( B \) between non-duplicated genes (those genes not in \( X_1 \cup X_2 \)), such that \( |X_1| + |X_2| + d(B) \) is minimized.
List all candidate segmental duplications.

\[ S_1 = \{(a^1, -c^1), (a^2, -c^2), (b^1), (b^2), (a^1), (c^1), (a^2), (c^2)\} \]

\[ S_2 = \{(b^1), (b^2), (b^3)\} \]

Edit Distance Problem: to compute \( X_1 \subset S_1 \) and \( X_2 \subset S_2 \), and a bijection \( B \) between non-duplicated genes (those genes not in \( X_1 \cup X_2 \)), such that \(|X_1| + |X_2| + d(B)\) is minimized.
List all candidate segmental duplications.

\[ S_1 = \{(a^1, -c^1), (a^2, -c^2), (b^1), (b^2), (a^1), (c^1), (a^2), (c^2)\} \]

\[ S_2 = \{(b^1), (b^2), (b^3)\} \]

**Edit Distance Problem:** to compute \( X_1 \subset S_1 \) and \( X_2 \subset S_2 \), and a bijection \( B \) between non-duplicated genes (those genes not in \( X_1 \cup X_2 \)), such that \(|X_1| + |X_2| + d(B)\) is minimized.
This problem is NP-hard.

Previous work: MSOAR (Chen et al., 2005, Fu et al., 2007, Shi et al., 2009) Approximation algorithm (Shao et al., 2012)

Our Contribution: an exact and practical algorithm

- An ILP formulation that gives the optimal solution.
- An algorithm that can iteratively identify optimal substructures of the problem.
This problem is NP-hard.

Previous work:
- MSOAR (Chen et al., 2005, Fu et al., 2007, Shi et al., 2009)
- Approximation algorithm (Shao et al., 2012)

Our Contribution:
- An exact and practical algorithm
  - An ILP formulation that gives the optimal solution.
  - An algorithm that can iteratively identify optimal substructures of the problem.
This problem is NP-hard.

Previous work:
- MSOAR (Chen et al., 2005, Fu et al., 2007, Shi et al., 2009)
- Approximation algorithm (Shao et al., 2012)
This problem is NP-hard.

Previous work:
- MSOAR (Chen et al., 2005, Fu et al., 2007, Shi et al., 2009)
- Approximation algorithm (Shao et al., 2012)

Our Contribution: an exact and practical algorithm
This problem is NP-hard.

Previous work:
- MSOAR (Chen et al., 2005, Fu et al., 2007, Shi et al., 2009)
- Approximation algorithm (Shao et al., 2012)

**Our Contribution:** an exact and practical algorithm
- An ILP formulation that gives the optimal solution.
This problem is NP-hard.

Previous work:
- MSOAR (Chen et al., 2005, Fu et al., 2007, Shi et al., 2009)
- Approximation algorithm (Shao et al., 2012)

Our Contribution: an exact and practical algorithm
- An ILP formulation that gives the optimal solution.
- An algorithm that can iteratively identify optimal substructures of the problem.
ILP Formulation

\[
\begin{align*}
\text{min} & \quad \sum_{s \in S_1 \cup S_2} x_s + |X| - \sum_{a \in X} y_a - \sum_{e \in E(X) \cup E(Y)} w_e \\
\text{s.t.} & \quad y_a \geq x_s, \quad \forall s \in S_1 \cup S_2 \text{ and } \forall a \in s \\
& \quad y_a \leq \sum_{s \in S_1 \cup S_2 : a \in s} x_s, \quad \forall a \in X \cup Y \\
& \quad \sum_{a \in F(X,f)} (1 - y_a) = \sum_{b \in F(Y,f)} (1 - y_b), \quad \forall f \in A(X) \\
& \quad \sum_{a \in F(X,f)} (1 - y_a) \geq 1, \quad \forall f \in A(X) \\
& \quad \sum_{b \in F(Y,f)} (1 - y_b) \geq 1, \quad \forall f \in A(Y) \\
& \quad \sum_{b \in F(Y,f_a)} z_{a,b} = 1 - y_a, \quad \forall a \in X \\
& \quad \sum_{a \in F(X,f_b)} z_{a,b} = 1 - y_b, \quad \forall b \in Y \\
& \quad 0 \leq l_e, \quad \forall e \in E(X) \cup E(Y) \\
& \quad l_e \leq U_e, \quad \forall e \in E(X) \cup E(Y) \\
& \quad l_{e_i} = l_{e_j}, \quad \forall \{e_i, e_j\} \\
& \quad l_{a_h} \leq l_{b_h} + (1 - z_{a,b}) \cdot U_{a_h} \\
& \quad l_{b_h} \leq l_{a_h} + (1 - z_{a,b}) \cdot U_{b_h} \\
& \quad \ldots
\end{align*}
\]
Key Point of the ILP Formulation

Key technique:
to count the number of cycles with variables.

Problem:
given an undirected graph \( G = (V, E) \), to choose \( E' \subset E \), such that in \( G' = (V, E') \) every vertex has exact degree 2 and the number of connected components in \( G' \) is maximized. (Formulate this problem as an ILP.)
Key Point of the ILP Formulation

- **Key technique:** to count the number of cycles with variables.
**Key Point of the ILP Formulation**

- **Key technique:** to count the number of cycles with variables.

- **Problem:** given an undirected graph $G = (V, E)$, to choose $E' \subset E$, such that in $G' = (V, E')$ every vertex has exact degree 2 and the number of connected components in $G'$ is maximized. (Formulate this problem as an ILP.)
Identify Optimal Substructure

Intuition:
shared segment is more likely to be in the optimal solution (non-duplicated and mapped to each other).

Problem:
to decide whether a given shared segment is in some optimal solution.

Algorithm:
first reduce the problem based on the given shared segment, then perform the enumeration.
Identify Optimal Substructure

Intuition: shared segment is more likely to be in the optimal solution (non-duplicated and mapped to each other).

Problem: to decide whether a given shared segment is in some optimal solution.

Algorithm: first reduce the problem based on the given shared segment, then perform the enumeration.
Identify Optimal Substructure

\[ G_1 \quad \begin{align*} &c^1 \quad b^1 \quad a^1 \quad b^2 \quad -e^1 \quad d^1 \end{align*} \]

\[ G_2 \quad \begin{align*} &-e^1 \quad -c^1 \quad a^1 \quad b^1 \quad d^1 \quad -a^2 \end{align*} \]

- **Intuition**: shared segment is more likely to be in the optimal solution (non-duplicated and mapped to each other).
**Identify Optimal Substructure**

- **Intuition:** shared segment is more likely to be in the optimal solution (non-duplicated and mapped to each other).

- **Problem:** to decide whether a given shared segment is in some optimal solution.
Identify Optimal Substructure

**Intuition:** shared segment is more likely to be in the optimal solution (non-duplicated and mapped to each other).

**Problem:** to decide whether a given shared segment is in some optimal solution.

**Algorithm:** first reduce the problem based on the given shared segment, then perform the enumeration.
Build Reduced Adjacency Graph

1. Core extremities: those extremities that are in the same family with the given segment.
2. Boundary extremities: forming adjacencies with core extremities.
3. Reduce the graph only on those core and boundary extremities.
4. For each pair of boundary extremities, check possible alternating path.
1 Core extremities: those extremities that are in the same family with the given segment.
1 Core extremities: those extremities that are in the same family with the given segment.

2 Boundary extremities: forming adjacencies with core extremities.
Build Reduced Adjacency Graph

1. **Core extremities**: those extremities that are in the same family with the given segment.
2. **Boundary extremities**: forming adjacencies with core extremities.
3. Reduce the graph only on those core and boundary extremities.

Reduced adjacency graph
Build Reduced Adjacency Graph

1. **Core extremities:** those extremities that are in the same family with the given segment.

2. **Boundary extremities:** forming adjacencies with core extremities.

3. Reduce the graph only on those core and boundary extremities.

4. For each pair of boundary extremities, check possible alternating path.
Build Reduced Adjacency Graph

1 Core extremities: those extremities that are in the same family with the given segment.

2 Boundary extremities: forming adjacencies with core extremities.

3 Reduce the graph only on those core and boundary extremities.

4 For each pair of boundary extremities, check possible alternating path.
Build Reduced Adjacency Graph

1. **Core extremities:** those extremities that are in the same family with the given segment.

2. **Boundary extremities:** forming adjacencies with core extremities.

3. Reduce the graph only on those core and boundary extremities.

4. For each pair of boundary extremities, check possible alternating path.
Build Reduced Adjacency Graph

1. **Core extremities**: those extremities that are in the same family with the given segment.

2. **Boundary extremities**: forming adjacencies with core extremities.

3. Reduce the graph only on those core and boundary extremities.

4. For each pair of boundary extremities, check possible alternating path.
Build Reduced Adjacency Graph

1. **Core extremities**: those extremities that are in the same family with the given segment.

2. **Boundary extremities**: forming adjacencies with core extremities.

3. Reduce the graph only on those core and boundary extremities.

4. For each pair of boundary extremities, check possible alternating path.
Build Reduced Adjacency Graph

1. **Core extremities**: those extremities that are in the same family with the given segment.

2. **Boundary extremities**: forming adjacencies with core extremities.

3. Reduce the graph only on those core and boundary extremities.

4. For each pair of boundary extremities, check possible alternating path.
Algorithm to Decide Optimal Substructure

- **Input:** a shared segment $s$.
- **Output:** whether or not $s$ is in the optimal solution.
Algorithm to Decide Optimal Substructure

- **Input**: a shared segment $s$.
- **Output**: whether or not $s$ is in the optimal solution.

**Algorithm**

1. Build the reduced adjacency graph $R(s) = (V, E)$ w.r.t. $s$.
2. If $|E| \geq c \cdot \log(n)$ return **false**.
3. Enumerate all the possible bijections in $R(s)$.
4. If the bijection induced by $s$ gives the maximum number of cycles, return **true**.
5. Return **false**.
Algorithm to Decide Optimal Substructure

- **Input**: a shared segment \( s \).
- **Output**: whether or not \( s \) is in the optimal solution.

**Algorithm**

1. Build the reduced adjacency graph \( R(s) = (V, E) \) w.r.t. \( s \).
2. If \( |E| \geq c \cdot \log(n) \) return **false**.
3. Enumerate all the possible bijections in \( R(s) \).
4. If the bijection induced by \( s \) gives the maximum number of cycles, return **true**.
5. Return **false**.

This algorithm runs in polynomial-time.
Application: Infer Paralogs and Orthologs

Input:
Two genomes $G_1$ and $G_2$.

Output:
$(X_1, X_2, B)$, which predict duplicated segments in $G_1$ and $G_2$, and the one-to-one correspondence between non-duplicated genes, respectively.

$X_1$ and $X_2$ infer paralogs in each given genomes.

$B$ infers orthologous pairs between the given genomes.
Application: Infer Paralogs and Orthologs

\[(G_1)\]

\[b^1 \quad a^1 \quad -c^1 \quad -b^2 \quad a^2 \quad -c^2 \quad d^1\]

\[(G_2)\]

\[b^1 \quad d^1 \quad a^1 \quad b^2 \quad c^1 \quad b^3 \quad d^2\]

- **Input:** Two genomes \(G_1\) and \(G_2\).
**Application: Infer Paralogs and Orthologs**

**Input:** Two genomes $G_1$ and $G_2$.

**Output:** $(X_1, X_2, B)$, which predict duplicated segments in $G_1$ and $G_2$, and the one-to-one correspondence between non-duplicated genes, respectively.
Application: Infer Paralogs and Orthologs

Input: Two genomes $G_1$ and $G_2$.

Output: $(X_1, X_2, B)$, which predict duplicated segments in $G_1$ and $G_2$, and the one-to-one correspondence between non-duplicated genes, respectively.

$X_1$ and $X_2$ infer paralogs in each given genomes.
Application: Infer Paralogs and Orthologs

Input: Two genomes $G_1$ and $G_2$.

Output: $(X_1, X_2, B)$, which predict duplicated segments in $G_1$ and $G_2$, and the one-to-one correspondence between non-duplicated genes, respectively.

- $X_1$ and $X_2$ infer **paralogs** in each given genomes.
- $B$ infers **orthologous pairs** between the given genomes.
Simulation Experiments

Simulation parameters

- \( L \in \{1, 2, 5\} \): length of duplicated segments
- \( D \in [200, 2000] \): the number of DCJs in each branch.

Compare our method with MSOAR, the only software that predicts orthlogs based on gene-order. Outperforms sequence-based methods (INPARANOID).

Evaluation of predicted paralogs and orthologs

- \( X_1 \) and \( X_2 \): sensitivity and specificity (classification problem).
- \( B \): accuracy (ratio between correct pairs and true pairs).
Simulation Experiments

$$L \in \{1, 2, 5\}$$: length of duplicated segments

$$D \in [200, 2000]$$: the number of DCJs in each branch.

Compare our method with MSOAR.

The only software that predicts orthlogs based on gene-order. Outperforms sequence-based methods (INPARANOID).

Evaluation of predicted paralogs and orthologs $X_1$ and $X_2$: sensitivity and specificity (classification problem).

$B$: accuracy (ratio between correct pairs and true pairs).
Simulation Experiments

SDs and DCJs

\[ b^1 \rightarrow a^1 \rightarrow -c^1 \rightarrow -b^2 \rightarrow d^1 \] (A)

SDs and DCJs

\[ b^1 \rightarrow a^1 \rightarrow -c^1 \rightarrow -b^2 \]
\[ a^2 \rightarrow -c^2 \rightarrow d^1 \] (G_1)

\[ b^1 \rightarrow a^1 \] (G_2)
\[ b^2 \rightarrow c^1 \rightarrow b^3 \rightarrow d^1 \]

**Evaluation of predicted paralogs and orthologs**

- \( X_1 \) and \( X_2 \): sensitivity and specificity (classification problem).
- \( B \) : accuracy (ratio between correct pairs and true pairs).

**Simulation parameters**

- \( L \in \{1, 2, 5\} \): length of duplicated segments
- \( D \in [200, 2000] \): the number of DCJs in each branch.

Compare our method with MSOAR.

The only software that predicts orthlogs based on gene-order. Outperforms sequence-based methods (INPARANOID).
**Simulation Experiments**

![Diagram of SDs and DCJs](image)

**Simulation parameters**
- $L \in \{1, 2, 5\}$: length of duplicated segments
- $D \in [200, 2000]$: the number of DCJs in each branch.
Simulation Experiments

SDs and DCJs

(A)

SDs and DCJs

(b_1, a_1, -c_1, -b_2, d_1)

(b_1, a_1, -c_1, -b_2, a_2, -c_2, d_1)

(b_1, a_1, b_2, c_1, b_3, d_1)

Simulation parameters

- \( L \in \{1, 2, 5\} \): length of duplicated segments
- \( D \in [200, 2000] \): the number of DCJs in each branch.

Compare our method with MSOAR.

- The only software that predicts orthlogs based on gene-order.
- Outperforms sequence-based methods (INPARANOID).
Simulation Experiments

Simulation parameters
- \( L \in \{1, 2, 5\} \): length of duplicated segments
- \( D \in [200, 2000] \): the number of DCJs in each branch.

Compare our method with MSOAR.
- The only software that predicts orthlogs based on gene-order.
- Outperforms sequence-based methods (INPARANOID).

Evaluation of predicted paralogs and orthologs
- \( X_1 \) and \( X_2 \): sensitivity and specificity (classification problem).
- \( B \): accuracy (ratio between correct pairs and true pairs).
Simulation Results (Sensitivity)

![Graph showing simulation results for different parameters.](graph_image)

- **Sensitivity (%)**
  - True DCJ Operations (Parameter D)
  - ILP, L=1
  - ILP, L=2
  - ILP, L=5
  - MSOAR, L=1
  - MSOAR, L=2
  - MSOAR, L=5

- **X-axis**: True DCJ Operations (Parameter D)
- **Y-axis**: Sensitivity (%)
Simulation Results (Specificity)

![Simulation Results Graph]

- True DCJ Operations (Parameter D)
- Specificity (%)

- ILP, L=1
- ILP, L=2
- ILP, L=5
- MSOAR, L=1
- MSOAR, L=2
- MSOAR, L=5

- 500 1000 1500 2000
- 40 50 60 70 80 90 100

- True DCJ Operations (Parameter D) vs. Specificity (%) graph with different markers representing various algorithms and parameter values.
Simulation Results (Accuracy)

Accuracy (%) vs. True DCJ Operations (Parameter D)

- ILP, L=1
- ILP, L=2
- ILP, L=5
- MSOAR, L=1
- MSOAR, L=2
- MSOAR, L=5
We choose 5 well-annotated mammalian genomes: human \((H.s.)\), gorilla \((G.g.)\), orangutan \((P.a.)\), mouse \((M.m.)\) and rat \((R.n.)\).

For each species, we collect all the protein-coding genes, and download their positions and gene families from Ensembl.

**True orthologs:** genes pairs with the same gene symbol.

Only accuracy is compared.
## Biological Results

<table>
<thead>
<tr>
<th>species pairs</th>
<th>true</th>
<th>accuracy</th>
<th>MSOAR</th>
<th>Our Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.g. &amp; H.s.</td>
<td>14911</td>
<td>98.9%</td>
<td>99.1%</td>
<td></td>
</tr>
<tr>
<td>G.g. &amp; M.m.</td>
<td>12989</td>
<td>98.7%</td>
<td>99.0%</td>
<td></td>
</tr>
<tr>
<td>G.g. &amp; P.a.</td>
<td>11328</td>
<td>98.7%</td>
<td>99.0%</td>
<td></td>
</tr>
<tr>
<td>G.g. &amp; R.n.</td>
<td>10848</td>
<td>97.2%</td>
<td>98.0%</td>
<td></td>
</tr>
<tr>
<td>H.s. &amp; M.m.</td>
<td>14077</td>
<td>99.1%</td>
<td>99.3%</td>
<td></td>
</tr>
<tr>
<td>H.s. &amp; P.a.</td>
<td>12028</td>
<td>99.1%</td>
<td>99.3%</td>
<td></td>
</tr>
<tr>
<td>H.s. &amp; R.n.</td>
<td>11768</td>
<td>97.5%</td>
<td>98.1%</td>
<td></td>
</tr>
<tr>
<td>M.m. &amp; P.a.</td>
<td>10586</td>
<td>98.9%</td>
<td>99.3%</td>
<td></td>
</tr>
<tr>
<td>M.m. &amp; R.n.</td>
<td>12316</td>
<td>97.7%</td>
<td>98.2%</td>
<td></td>
</tr>
<tr>
<td>R.n. &amp; P.a.</td>
<td>8797</td>
<td>97.6%</td>
<td>98.2%</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

We proposed an exact and practical algorithm to compare two genomes with segmental duplications and DCJ operations. The "ILP + Identify Optimal Substructure" framework has potential to be applied in other optimization problems. The proposed algorithms can be used to annotate genomes—the accuracy on inferring orthologs and paralogs was shown very high and outperformed the state-of-the-art algorithms.

Thank You!
Conclusion

- We proposed an exact and practical algorithm to compare two genomes with segmental duplications and DCJ operations.

Thank You!
Conclusion

- We proposed an exact and practical algorithm to compare two genomes with segmental duplications and DCJ operations.

- The “ILP + Identify Optimal Substructure” framework has potential to be applied in other optimization problems.

Thank You!
Conclusion

- We proposed an exact and practical algorithm to compare two genomes with segmental duplications and DCJ operations.

- The “ILP + Identify Optimal Substructure” framework has potential to be applied in other optimization problems.

- The proposed algorithms can be used to annotate genomes—the accuracy on inferring orthologs and paralogs was shown very high and outperformed the state-of-the-art algorithms.
We proposed an exact and practical algorithm to compare two genomes with segmental duplications and DCJ operations.

The “ILP + Identify Optimal Substructure” framework has potential to be applied in other optimization problems.

The proposed algorithms can be used to annotate genomes—the accuracy on inferring orthologs and paralogs was shown very high and outperformed the state-of-the-art algorithms.

Thank You!