Exploring Phylogenetic Relationships in *Drosophila* with Ciliate Operations

Jacob Herlin, Anna Nelson and Marion Scheepers

February 24, 2012

What is a phylogenetic relationship?

Phylogenetics is the study of evolutionary relationships between groups of organisms. We define the phylogenetic distance between species to be the time since they diverged into distinct species. For this research, we focused on several organisms in the genus *Drosophila*.

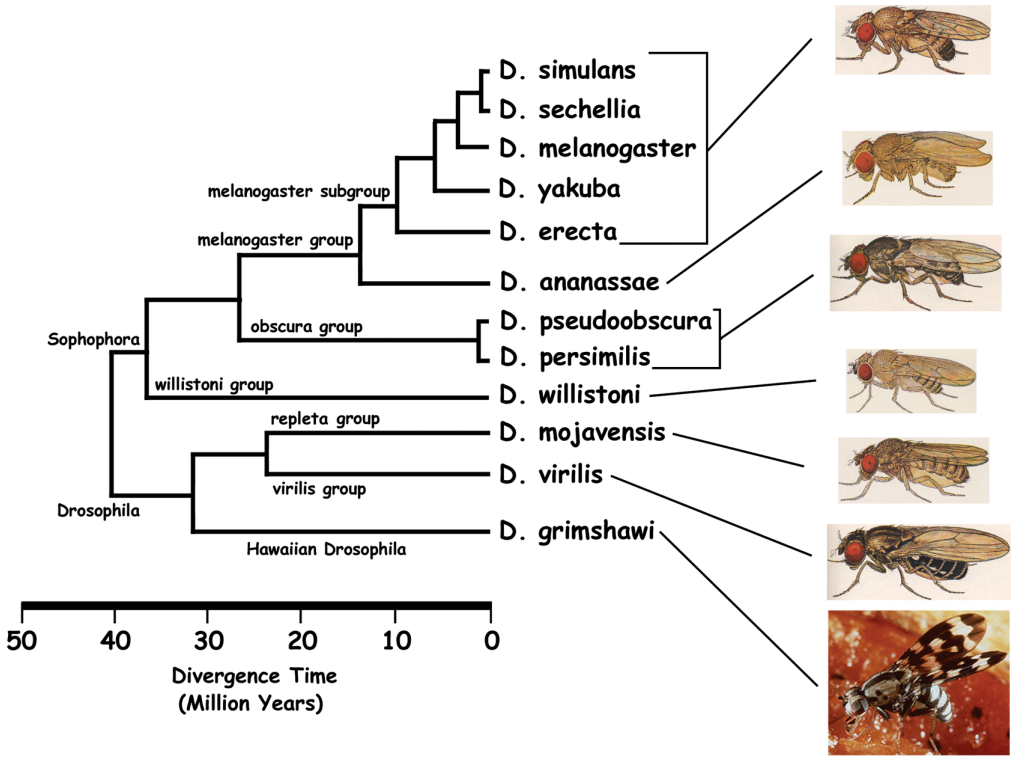


Image courtesy of DroSpeGe: Drosophila Species Genomes. http://insects.eugenes.org/DroSpeGe

Using reversals to determine phylogeny

Genetic operations that result from breaking and rejoining the chromosome cause the DNA to be scrambled

Using reversals to determine phylogeny

Genetic operations that result from breaking and rejoining the chromosome cause the DNA to be scrambled

- In 1938, Dobzhansky and Sturtevant discovered that the gene arrangements occurred in *D. pseudoobscura* were reversals. It has been conjectured that reversals are the main operation for genetic rearrangement in evolution.

Using reversals to determine phylogeny

Genetic operations that result from breaking and rejoining the chromosome cause the DNA to be scrambled

- In 1938, Dobzhansky and Sturtevant discovered that the gene arrangements occurred in *D. pseudoobscura* were reversals. It has been conjectured that reversals are the main operation for genetic rearrangement in evolution.
- Using *Drosophila melanogaster* as the canonical reference species, one can use reversals as a measure of evolutionary distance.

Using reversals to determine phylogeny

Genetic operations that result from breaking and rejoining the chromosome cause the DNA to be scrambled

- In 1938, Dobzhansky and Sturtevant discovered that the gene arrangements occurred in *D. pseudoobscura* were reversals. It has been conjectured that reversals are the main operation for genetic rearrangement in evolution.
- Using *Drosophila melanogaster* as the canonical reference species, one can use reversals as a measure of evolutionary distance.
- From Hannenhalli and Pevzner, it is known that the shortest signed reversal path can be found in polynomial time.

Cilliates

Ciliates are multinuclear protozoans found in aqueous environments. Cillies have two nuclei:

Cilliates

Ciliates are multinuclear protozoans found in aqueous environments. Cillies have two nuclei:

- The micronucleus contains very long strands of DNA that are encrypted versions of the macronuclear DNA.

Cilliates

Ciliates are multinuclear protozoans found in aqueous environments. Cillies have two nuclei:

- The micronucleus contains very long strands of DNA that are encrypted versions of the macronuclear DNA.
- The macronucleus is larger than the micronucleus and contains short strands of DNA that have been multiplied.

Cilliates

Ciliates are multinuclear protozoans found in aqueous environments. Cillies have two nuclei:

- The micronucleus contains very long strands of DNA that are encrypted versions of the macronuclear DNA.
- The macronucleus is larger than the micronucleus and contains short strands of DNA that have been multiplied.
- During reproduction, the micronucleus decrypts the DNA using three ciliate operations (hi, dlad, ld) to form macronuclear DNA.

Cilliates

Ciliates are multinuclear protozoans found in aqueous environments. Cillies have two nuclei:

- The micronucleus contains very long strands of DNA that are encrypted versions of the macronuclear DNA.
- The macronucleus is larger than the micronucleus and contains short strands of DNA that have been multiplied.
- During reproduction, the micronucleus decrypts the DNA using three ciliate operations (hi, dlad, ld) to form macronuclear DNA.
- We want to use these operations to relate the DNA of fruitflies in *Drosophila*, and look for a correlation with their known phylogenetic relationsips.

Macronuclear vs. Micronuclear DNA

Micronuclear DNA has three elements:

Macronuclear vs. Micronuclear DNA

Micronuclear DNA has three elements:

1. Macronuclear destined sequences (MDSs) are the important genes that will be present in the macronuclear DNA

Macronuclear vs. Micronuclear DNA

Micronuclear DNA has three elements:

1. Macronuclear destined sequences (MDSs) are the important genes that will be present in the macronuclear DNA
2. Internal eliminated sequences (IESs) are segments between MDSs that will be removed using the operations

Macronuclear vs. Micronuclear DNA

Micronuclear DNA has three elements:

1. Macronuclear destined sequences (MDSs) are the important genes that will be present in the macronuclear DNA
2. Internal eliminated sequences (IESs) are segments between MDSs that will be removed using the operations
3. Pointers occur on the flanks of the MDSs that preserve the orientation of the genes.

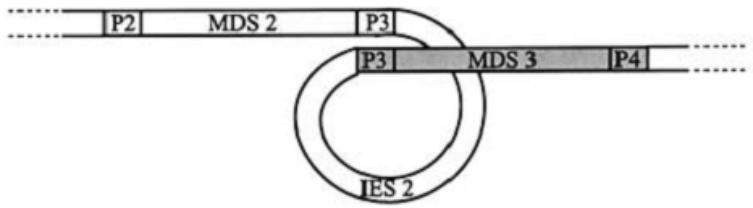
ld operation

The ld operation occurs when a pair of pointers flank an IES between two MDSs. The DNA molecular folds into a simple loop that brings the pairs together. The pairs then join the two MDSs and the removed IES is a closed circle.

$$\ldots\ P_{i,i+1}\ \ldots\ IES_i\ \ldots\ P_{i,i+1}\ \ldots$$

A special case of the ld operation is the boundary ld operation, where a pair of pointers occurs at the ends of a part of the molecule that contains all MDSs.

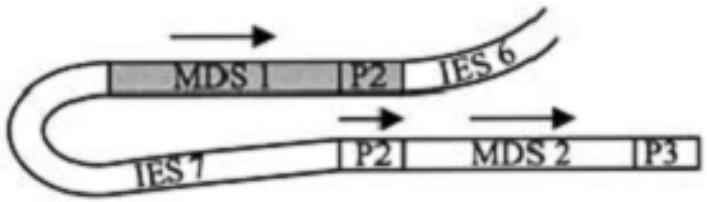
$$P_{i,i+1}\ \ldots\ MDS_i\ \ldots\ MDS_j\ \ldots\ P_{i,i+1}$$



hi operation

The hairpin (hi) operation occurs when a pair of pointers in which one of the pointers is the opposite sign of the other. The DNA molecule folds so that the two opposite-signed pointers are brought together with the same sign. This operation is analogous to a reversal.

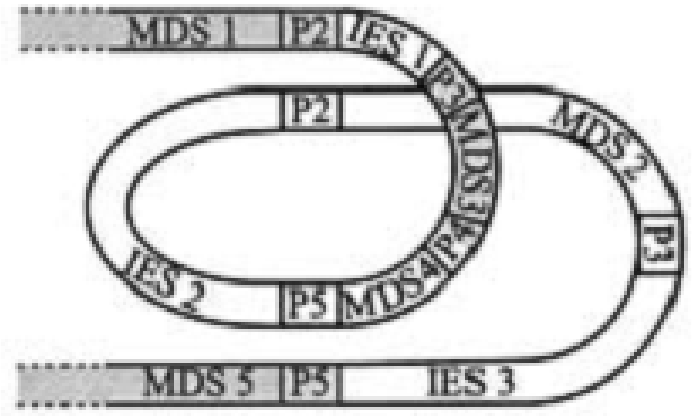
$$\ldots\ P_{i,i+1}\ \ldots\ P_{j,j+1}\ \ldots\ -P_{i,i+1}\ \ldots$$
$$\ldots\ P_{i,i+1}\ \ldots\ P_{i,i+1}\ \ldots\ -P_{j,j+1}\ \ldots$$



dlad operation

The double-loop (dlad) operation applies to a sequence of DNA when two pairs of pointers are staggered on the strand where one pair of pointers overlap another.

$$\ldots\ P_{i,i+1}\ \ldots\ P_{j,j+1}\ \ldots\ P_{i,i+1}\ \ldots\ P_{j,j+1}\ \ldots$$



DNA as a permutation

To compare the DNA of the flies in *Drosophila*, we looked at the set of all the genes that occur common in all the species. We call these genes orthologous.

DNA as a permutation

To compare the DNA of the flies in *Drosophila*, we looked at the set of all the genes that occur common in all the species. We call these genes orthologous.

- *D. melanogaster* was chosen to be the reference species. Each gene was numbered and given a sign, such that *D. melanogaster*'s DNA is described by the canonical permutation with positive signs.

DNA as a permutation

To compare the DNA of the flies in *Drosophila*, we looked at the set of all the genes that occur common in all the species. We call these genes orthologous.

- *D. melanogaster* was chosen to be the reference species. Each gene was numbered and given a sign, such that *D. melanogaster*'s DNA is described by the canonical permutation with positive signs.
- The sign represents the direction of a gene.

DNA as a permutation

To compare the DNA of the flies in *Drosophila*, we looked at the set of all the genes that occur common in all the species. We call these genes orthologous.

- *D. melanogaster* was chosen to be the reference species. Each gene was numbered and given a sign, such that *D. melanogaster*'s DNA is described by the canonical permutation with positive signs.
- The sign represents the direction of a gene.
- The genomes of the species of *Drosophila* are divided into six Muller elements, lettered A-F. We treated each muller element as a seperate permutation.

DNA as a permutation

To compare the DNA of the flies in *Drosophila*, we looked at the set of all the genes that occur common in all the species. We call these genes orthologous.

- *D. melanogaster* was chosen to be the reference species. Each gene was numbered and given a sign, such that *D. melanogaster*'s DNA is described by the canonical permutation with positive signs.
- The sign represents the direction of a gene.
- The genomes of the species of *Drosophila* are divided into six Muller elements, lettered A-F. We treated each muller element as a seperate permutation.
- We created an algorithm to sort signed permutations in the canonical permutation, and applied this to permutations representing the rest of *Drosophila*, to look for a correlation between the number of requisite steps and their phylogenetic distance from melanogaster.

The Algorithm

We start our algorithm by mapping a signed permutation where each elements represents a section of genome:

The Algorithm

We start our algorithm by mapping a signed permutation where each elements represents a section of genome:

$$[1, 4, -3, -2, 6, -5]$$

onto a list of pairs of pointers:

The Algorithm

We start our algorithm by mapping a signed permutation where each elements represents a section of genome:

$$[1, 4, -3, -2, 6, -5]$$

onto a list of pairs of pointers:

$$[(1, 2), (4, 5), (4, 3), (3, 2), (6, 7), (6, 5)]$$

where each pair (a, b) represents a section of genome spanning from a pointer a to a pointer b. In our algorithm, we give the pointers signs to represent the orientation of each section, in stead of keeping them in pairs:

The Algorithm

We start our algorithm by mapping a signed permutation where each elements represents a section of genome:

$$[1, 4, -3, -2, 6, -5]$$

onto a list of pairs of pointers:

$$[(1, 2), (4, 5), (4, 3), (3, 2), (6, 7), (6, 5)]$$

where each pair (a, b) represents a section of genome spanning from a pointer a to a pointer b. In our algorithm, we give the pointers signs to represent the orientation of each section, in stead of keeping them in pairs:

$$[1, 2, 4, 5, -4, -3, -3, -2, 6, 7, -6, -5]$$

We call this representation a *pointer list*.

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

1. n is even.

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

1. n is even.
2. There is a unique i with $\mu = |x_i| = min|x_j| : 1 \leq j \leq n$

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

1. n is even.
2. There is a unique i with $\mu = |x_i| = min|x_j| : 1 \leq j \leq n$
3. There is a unique j with $\lambda = |x_i| = max|x_j| : 1 \leq j \leq n$

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

1. n is even.
2. There is a unique i with $\mu = |x_i| = min|x_j| : 1 \leq j \leq n$
3. There is a unique j with $\lambda = |x_i| = max|x_j| : 1 \leq j \leq n$
4. For each $i \in \{1, ..., n\}$ with $\mu < |x_i| < \lambda$, there is a unique $j \in \{1, ..., n\} \backslash \{i\}$ with $|x_i| = |x_j|$.

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

1. n is even.
2. There is a unique i with $\mu = |x_i| = min|x_j| : 1 \leq j \leq n$
3. There is a unique j with $\lambda = |x_i| = max|x_j| : 1 \leq j \leq n$
4. For each $i \in \{1, ..., n\}$ with $\mu < |x_i| < \lambda$, there is a unique $j \in \{1, ..., n\} \backslash \{i\}$ with $|x_i| = |x_j|$.
5. For each odd $i \in \{1, ..., n\}, x_i \leq x_{i+1}$ and $x_i \cdot x_{i+1} > 0$.

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Pointer Lists

We define a pointer list formally as a list $L = [x_1, x_2, \ldots x_n]$ that satisfies the following six conditions:

1. n is even.
2. There is a unique i with $\mu = |x_i| = min|x_j| : 1 \leq j \leq n$
3. There is a unique j with $\lambda = |x_i| = max|x_j| : 1 \leq j \leq n$
4. For each $i \in \{1, ..., n\}$ with $\mu < |x_i| < \lambda$, there is a unique $j \in \{1, ..., n\} \backslash \{i\}$ with $|x_i| = |x_j|$.
5. For each odd $i \in \{1, ..., n\}, x_i \leq x_{i+1}$ and $x_i \cdot x_{i+1} > 0$.
6. For each odd i, $\nexists$ an odd j such that $x_i < x_j < x_{i+1} < x_{j+1}$.

$$[1, 2, 4, 5, -4, -3, -3, -2, 7, 9, -7, -5]$$

Id

The Id operation is represented by the removal of pairs of the same pointer that are adjacent. This is equivalent to joining two sections that are correctly adjacent.

Id

The Id operation is represented by the removal of pairs of the same pointer that are adjacent. This is equivalent to joining two sections that are correctly adjacent.

$$[1, 2, 4, 5, -4, -3, -3, -2, 6, 7, -6, 5]$$

Id

The Id operation is represented by the removal of pairs of the same pointer that are adjacent. This is equivalent to joining two sections that are correctly adjacent.

$$[1, 2, 4, 5, -4, -3, -3, -2, 6, 7, -6, 5]$$
$$[1, 2, 4, 5, -4, -2, 6, 7, -6, -5]$$

hi

The hi operation is represented by moving together two pointers of opposite orientation with a reversal, setting up an Id move. Note that reversing a section changes the signs of each element.

hi

The hi operation is represented by moving together two pointers of opposite orientation with a reversal, setting up an Id move. Note that reversing a section changes the signs of each element.

$$[1, 2, 4, 5, -4, -2, 6, 7, -6, -5]$$

hi

The hi operation is represented by moving together two pointers of opposite orientation with a reversal, setting up an Id move. Note that reversing a section changes the signs of each element.

$$[1, 2, 4, 5, -4, -2, 6, 7, -6, -5]$$
$$[1, 2, 4, 4, -5, -2, 6, 7, -6, -5]$$

dlad

The dlad operation is represented by finding a 4-tuple of pointers (x_i, x_j, x_k, x_l) where $x_i = x_j$, and $x_k = x_l$ and $i < k < j < l$. Then, you take the section $x_j x_l$, including the pointers, and the section in-between, but not including, the pointers x_i and x_k, and swapping them, setting up two ld-moves.

dlad

The dlad operation is represented by finding a 4-tuple of pointers (x_i, x_j, x_k, x_l) where $x_i = x_j$, and $x_k = x_l$ and $i < k < j < l$. Then, you take the section $x_j x_l$, including the pointers, and the section in-between, but not including, the pointers x_i and x_k, and swapping them, setting up two ld-moves.

$$[4, 5, 10, 11, -4, -3, 6, 7, 1, 2, 9, 10, -3, -2, 5, 6, -13, -11]$$

dlad

The dlad operation is represented by finding a 4-tuple of pointers (x_i, x_j, x_k, x_l) where $x_i = x_j$, and $x_k = x_l$ and $i < k < j < l$. Then, you take the section $x_j x_l$, including the pointers, and the section in-between, but not including, the pointers x_i and x_k, and swapping them, setting up two ld-moves.

$$[4, 5, 10, 11, -4, -3, 6, 7, 1, 2, 9, 10, -3, -2, 5, 6, -13, -11]$$
$$[4, 5, 10, 10, -3, -2, 5, 6, 6, 7, 1, 2, 9, 11, -4, -3, -13, -11]$$

dlad

The dlad operation is represented by finding a 4-tuple of pointers (x_i, x_j, x_k, x_l) where $x_i = x_j$, and $x_k = x_l$ and $i < k < j < l$. Then, you take the section $x_j x_l$, including the pointers, and the section in-between, but not including, the pointers x_i and x_k, and swapping them, setting up two ld-moves.

$$[4, 5, 10, 11, -4, -3, 6, 7, 1, 2, 9, 10, -3, -2, 5, 6, -13, -11]$$
$$[4, 5, 10, 10, -3, -2, 5, 6, 6, 7, 1, 2, 9, 11, -4, -3, -13, -11]$$

boundary-ld

The boundary ld move (b-ld) maps lists of length 4 onto lists of length 2.

boundary-ld

The boundary ld move (b-ld) maps lists of length 4 onto lists of length 2. It only operates on lists of the following form:

$$[x, m, m', x]$$

where $m, m' \in \{\pm\mu, \pm\lambda\}$ and $x \notin \{\pm\mu, \pm\lambda\}$ is some pointer.

boundary-ld

The boundary ld move (b-ld) maps lists of length 4 onto lists of length 2. It only operates on lists of the following form:

$$[x, m, m', x]$$

where $m, m' \in \{\pm\mu, \pm\lambda\}$ and $x \notin \{\pm\mu, \pm\lambda\}$ is some pointer. It maps as such:

$$[x, m, m', x] \Rightarrow [m', m]$$

boundary-ld

The boundary ld move (b-ld) maps lists of length 4 onto lists of length 2. It only operates on lists of the following form:

$$[x, m, m', x]$$

where $m, m' \in \{\pm\mu, \pm\lambda\}$ and $x \notin \{\pm\mu, \pm\lambda\}$ is some pointer. It maps as such:

$$[x, m, m', x] \Rightarrow [m', m]$$

For example,

$$[-2, -1, -3, -2] \Rightarrow [-3, -1]$$

.

A list is considered sorted if it is in the form $[\mu, \lambda]$ or $[-\lambda, -\mu]$ Thus, if a boundary-ld move is done, it will always be the final move in the sorting.

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

(3) Check if a boundary-Id can be done. If it can, do it.

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

(3) Check if a boundary-Id can be done. If it can, do it.

(4) Check if the list is sorted. If it is, end the program. Otherwise, go to (5)

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

(3) Check if a boundary-Id can be done. If it can, do it.

(4) Check if the list is sorted. If it is, end the program. Otherwise, go to (5)

(5) Search through the list, and keep memory of pairs of oppositely-oriented pointers and of pairs of equally-oriented pointers.

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

(3) Check if a boundary-Id can be done. If it can, do it.

(4) Check if the list is sorted. If it is, end the program. Otherwise, go to (5)

(5) Search through the list, and keep memory of pairs of oppositely-oriented pointers and of pairs of equally-oriented pointers.

(6) Search through the list of equally-oriented pointers for a possible dlad move. If one is found, do it and go to (2). If none is found, go to (7).

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

(3) Check if a boundary-Id can be done. If it can, do it.

(4) Check if the list is sorted. If it is, end the program. Otherwise, go to (5)

(5) Search through the list, and keep memory of pairs of oppositely-oriented pointers and of pairs of equally-oriented pointers.

(6) Search through the list of equally-oriented pointers for a possible dlad move. If one is found, do it and go to (2). If none is found, go to (7).

(7) Do the hi represented by the first element of the list of equally-oriented pairs, then go to (2).

The Algorithm

(1) Map the signed permutation onto a list of signed pointers.

(2) Search for and do the first possible Id. If one is done, go to (2). If no Id is found, go to (3).

(3) Check if a boundary-Id can be done. If it can, do it.

(4) Check if the list is sorted. If it is, end the program. Otherwise, go to (5)

(5) Search through the list, and keep memory of pairs of oppositely-oriented pointers and of pairs of equally-oriented pointers.

(6) Search through the list of equally-oriented pointers for a possible dlad move. If one is found, do it and go to (2). If none is found, go to (7).

(7) Do the hi represented by the first element of the list of equally-oriented pairs, then go to (2).

This algorithm runs in polynomial time in the length of the original permutation. Specifically, the complexity is $O(n^3)$

The Algorithm

We proved that a correctly-formed pointer list is always in the domain of an ld, hi, dlad or b-ld opedration.

The Algorithm

We proved that a correctly-formed pointer list is always in the domain of an ld, hi, dlad or b-ld opedration.

The Algorithm

We proved that a correctly-formed pointer list is always in the domain of an ld, hi, dlad or b-ld opedration.

- We proved this for lists of length 4 by cases

The Algorithm

We proved that a correctly-formed pointer list is always in the domain of an ld, hi, dlad or b-ld opedration.

- We proved this for lists of length 4 by cases
- For lists of length $n > 4$, we proved this by contradiction, assuming that none of the conditions are met for any of the moves, and showing that the list cannot still be well-formed.

The Algorithm

We proved that a correctly-formed pointer list is always in the domain of an ld, hi, dlad or b-ld opedration.

- We proved this for lists of length 4 by cases
- For lists of length $n > 4$, we proved this by contradiction, assuming that none of the conditions are met for any of the moves, and showing that the list cannot still be well-formed.
- follows that our algorithm will never crash, and will always find a path to $[\mu, \lambda]$ or $[-\lambda, -\mu]$.

Data Analysis

We obtained, from *Flybase*, data for the locations of every gene on the genomes of the following species of the *Drosophila* genus:

1. Erecta
2. Grimshawi
3. Melanogaster
4. Mojavensis
5. Sechellia
6. Simulans
7. Virillis
8. Yakuba

The algorithm produced these numbers of each move for the Muller *A* element, shown with the known time since their diverenge from *D. melanogaster*.

species	hi	dlad	boundary ld	known divergence time
D. erecta	3	170	0	12.6 mya
D. grimshawi	36	381	1	62 mya
D. sechellia	7	488	0	5.4 mya
D. simulans	8	268	0	5.4 mya
D. yakuba	11	105	1	12.6 mya
D. mojavensis	41	407	0	62 mya
D. virilis	33	403	0	62 mya

The algorithm produced these total numbers for every Muller element added together:

species	hi	dlad	boundary ld	known divergence time
D. erecta	30	658	0	12.6 mya
D. grimshawi	197	1459	3	62 mya
D. sechellia	15	1460	3	5.4 mya
D. simulans	24	508	0	5.4 mya
D. yakuba	47	438	1	12.6 mya
D. mojavensis	189	986	2	62 mya
D. virilis	194	1278	2	62 mya

While this algorithm does not necessarily produce the shortest overall path in the number of combined hi, dlad and ld moves, we conjecture that this is the shortest possible number of hi moves.

The Future

Here are some further areas we have to explore:

1. We conjecture that our algorithm minimises the number of hi moves in its reversal path. We have yet to prove this
2. Develop an algorithm to find the shortest possible cilliate operation path.
3. Look at more species in the *Drosophila* genus and see if the correlation between cilliate operation path length and divergence time holds.
4. Produce permutations for each species as the reference, and use the results from those to construct a phylogenetic tree.

Bibliography

- Sridhar Hannenhalli, Pavel A. Pevzner *Transforming Cabbahe into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals*. Journal of the ACM, Vol. 46, No. 1, 1999.
- Pavel Pevzner, Glenn Tesler *genome rearrangements in mammalian evolution: lessons from human and mouse genomes*. Genome Research, Vol. 13, 2003.
- Arjun Bhutkar, Stephen W. Schaeffer, Susan M. Russo, Mu Xu, Temple F. Smith, William M. Gelbart *Chromosomal Rearrangement Inferred From Comparisons of 12 Drosophila Genomes*. Genetics, Vol 197, 2008.

Bibliography (cont.)


Jose M. Ranz, Damien Maurin, Yuk S. Chan, Marchin Von Grotthuss, LeDeana W. Hillier, John Roote, Michael Ashburner, Casey M. Bergman *Principles of Genome Evolution in the* Drosophila melanogaster *Species Group*. PLoS Biology, Vol. 5, Issue 6, 2007.


Andrzej Ehrenfeucht, Tero Harju, Ion Petre, David M. Prescott, Grzegorz Rozenberg *Computation in Living Cells*. Springer-Verlag Berlin Heidelberg, 2004.


S. Tweedie, M. Ashburner, K. Falls, P. Leyland, P. McQuilton, S. Marygold, G. Millburn, D. Osumi-Sutherland, A. Schroeder, R. Seal, H. Zhang and The FlyBase Consortium *FlyBase: enhancing Drosophila Gene Ontology annotations*. Nucleic Acids Research, Vol. 37, 2009.