Algorithmic Approaches for Biological Data, Lecture #23

Katherine St. John

City University of New York
American Museum of Natural History

2 May 2016

Outline

- Project & Last Day Notes

Outline

- Project & Last Day Notes
- Complexity Revisited: NP-hardness, Time & Space Complexity

Outline

- Project & Last Day Notes
- Complexity Revisited: NP-hardness, Time & Space Complexity
- Searching for Optimal Trees

Outline

- Project & Last Day Notes
- Complexity Revisited: NP-hardness, Time & Space Complexity
- Searching for Optimal Trees
- Edit distances between trees

Outline

- Project & Last Day Notes
- Complexity Revisited: NP-hardness, Time & Space Complexity
- Searching for Optimal Trees
- Edit distances between trees
- Tree Vectors

Project & Last Day Notes

- Last week of lectures (and lab).

Project & Last Day Notes

- Last week of lectures (and lab).
- Next Monday (last day of class):

Project & Last Day Notes

- Last week of lectures (and lab).
- Next Monday (last day of class):
  - Project Presentations (about 5 minutes): brief overview of biological question, techniques used, and results)

Project & Last Day Notes

- Last week of lectures (and lab).
- Next Monday (last day of class):
  - Project Presentations (about 5 minutes): brief overview of biological question, techniques used, and results)
  - Open Lab for questions about labs, Rosalind questions, etc.

Project & Last Day Notes

- Last week of lectures (and lab).
- Next Monday (last day of class):
  - Project Presentations (about 5 minutes): brief overview of biological question, techniques used, and results)
  - Open Lab for questions about labs, Rosalind questions, etc.
- For those who cannot make Monday, possible to do presentation this Wednesday (see me).

Complexity Revisited: Running Times

- Theoretical Estimates on Running Time

Complexity Revisited: Running Times

- Theoretical Estimates on Running Time
- big-Oh notation

Complexity Revisited: Running Times

- Theoretical Estimates on Running Time
- big-Oh notation
- Complexity Classes

Complexity Revisited: Running Times

- Theoretical Estimates on Running Time
- big-Oh notation
- Complexity Classes
- P vs. NP

- Measure the size of the problem, usually called n.

Comparing Algorithms

- Measure the size of the problem, usually called n.
- Example: for sorting cards, n is the number of cards.

Comparing Algorithms

- Measure the size of the problem, usually called n.
- Example: for sorting cards, n is the number of cards.
- Different approaches can take different amounts of time.

Comparing Algorithms

- Measure the size of the problem, usually called n.
- Example: for sorting cards, n is the number of cards.
- Different approaches can take different amounts of time.
- How long does the algorithm take proportional to n?

Comparing Algorithms

- Measure the size of the problem, usually called n.
- Example: for sorting cards, n is the number of cards.
- Different approaches can take different amounts of time.
- How long does the algorithm take proportional to n?
- *Sorting Algorithms demo*

  Not in demo is the built-in Python sort: timSort (invented by Tim Peters in 2002) that is hybrid of merge sort and insertion sort.

Analysis of Algorithms

- How long does the algorithm take proportional to n?

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.
- Would write "the running time is $O(n)$."
  ("big-Oh" notation).

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.
- Would write "the running time is $O(n)$." ("big-Oh" notation).
- Usually measure the worst-case running time.

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.
- Would write "the running time is $O(n)$."
  ("big-Oh" notation).
- Usually measure the worst-case running time.
- Formally: If the running time of an algorithm is $f(n)$ for n items, then we $f(n) = O(g(n))$

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.
- Would write "the running time is $O(n)$."
  ("big-Oh" notation).
- Usually measure the worst-case running time.
- Formally: If the running time of an algorithm is $f(n)$ for n items, then we $f(n) = O(g(n))$
  if there exists $N, c > 0$ such that for all $n > N$,

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.
- Would write "the running time is $O(n)$." ("big-Oh" notation).
- Usually measure the worst-case running time.
- Formally: If the running time of an algorithm is $f(n)$ for n items, then we $f(n) = O(g(n))$ if there exists $N, c > 0$ such that for all $n > N$,

$$f(n) < c \cdot g(n)$$

Analysis of Algorithms

- How long does the algorithm take proportional to n?
- If an algorithm looks at each element once (or a constant number of times), the running time is proportional to n, the number of elements.
- Then, the algorithm runs in linear time.
- Would write "the running time is $O(n)$." ("big-Oh" notation).
- Usually measure the worst-case running time.
- Formally: If the running time of an algorithm is $f(n)$ for n items, then we $f(n) = O(g(n))$ if there exists $N, c > 0$ such that for all $n > N$,

$$f(n) < c \cdot g(n)$$

(That is, after some point, $f(n)$ is smaller than $c \cdot g(n)$.)

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
```

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
```

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
        for i in range(1,len(a)):  #Will go through this loop n times.
```

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
        for i in range(1,len(a)):  #Will go through this loop n times.
            if a[i-1] > a[i]:  #Takes constant time.
```

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
        for i in range(1,len(a)):  #Will go through this loop n times.
            if a[i-1] > a[i]:  #Takes constant time.
                a[i-1], a[i] = a[i], a[i-1] #Takes constant time.
```

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
        for i in range(1,len(a)):  #Will go through this loop n times.
            if a[i-1] > a[i]:  #Takes constant time.
                a[i-1], a[i] = a[i], a[i-1] #Takes constant time.
```

- The lines in the if statement take constant time, but are performed $n \cdot n$ time.

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
        for i in range(1,len(a)):  #Will go through this loop n times.
            if a[i-1] > a[i]:  #Takes constant time.
                a[i-1], a[i] = a[i], a[i-1] #Takes constant time.
```

- The lines in the if statement take constant time, but are performed $n \cdot n$ time.
- Upper bound on running time is $O(c \cdot n \cdot n) = O(n^2)$.

Analyzing Sorts by Running Times

- The sorting algorithms vary in running time, depending on number of elements and type of data.
- *Sorting Demo*
- Thinking about the worst-case, how many operations are performed in bubbleSort?

```
def bubbleSort(a):  #Let n be # of elements in a.
    for j in range(1,len(a)):  #Will go through this loop n times.
        for i in range(1,len(a)):  #Will go through this loop n times.
            if a[i-1] > a[i]:  #Takes constant time.
                a[i-1], a[i] = a[i], a[i-1] #Takes constant time.
```

- The lines in the if statement take constant time, but are performed $n \cdot n$ time.
- Upper bound on running time is $O(c \cdot n \cdot n) = O(n^2)$.

  (For big-Oh notation, drop constants and keep only largest terms.)

Complexity Classes: What is NP-hardness?

CLAY
MATHEMATICS
INSTITUTE

- $P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?

Complexity Classes: What is NP-hardness?

CLAY
MATHEMATICS
INSTITUTE

- $P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?
- P stands for problems that can be computed quickly (polynomial time).
- NP stands for problems that can be checked quickly (nondeterministic polynomial time).

Complexity Classes: What is NP-hardness?

CLAY MATHEMATICS INSTITUTE

- $P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?
- P stands for problems that can be computed quickly (polynomial time).
- NP stands for problems that can be checked quickly (nondeterministic polynomial time).
- Example: given a geometry proof, you can check if its correct quickly, but knowing that, is there a quick way to find proofs?

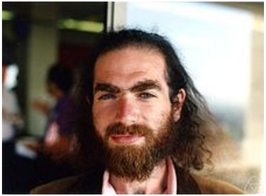
Millennium Prize Problems

CLAY MATHEMATICS INSTITUTE

In 2000, the Clay Mathematics Institute announced million dollar prizes for:

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

Millenium Prize Problems

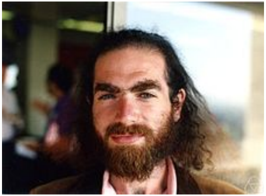


Grigori Perelman, 1993

In 2010, CMI announced Grigori Perelman solved:

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

Millenium Prize Problems



Grigori Perelman, 1993

In 2010, CMI announced Grigori Perelman solved:

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

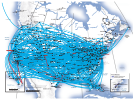
He turned down the prize.

P vs. NP

CLAY
MATHEMATICS
INSTITUTE

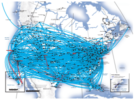
- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs. NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

More Examples of NP Problems

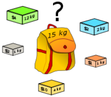


Traveling Salesman Problem (TSP): find the shortest path that visits all cities.

More Examples of NP Problems

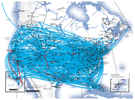


Traveling Salesman Problem (TSP): find the shortest path that visits all cities.

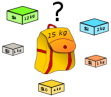


Knapsack Problem: fill your backpack with the most valuable objects without exceeding weight restrictions.

More Examples of NP Problems



Traveling Salesman Problem (TSP): find the shortest path that visits all cities.



Knapsack Problem: fill your backpack with the most valuable objects without exceeding weight restrictions.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

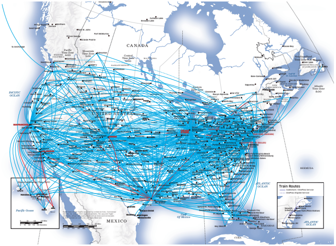
Sudoku: find a solution to a (large) Sudoku puzzle.

$P \stackrel{?}{=} NP$: Why Does It Matter?

wiki

If you could quickly find solutions to NP-hard problems (i.e. P=NP), then

- Many security systems (such as the Data Encryption Standard (DES) used to send ATM/bank data) would be easily breached.

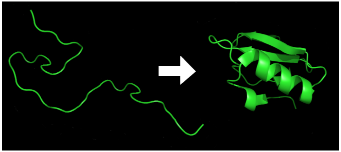


United Airlines

If you could quickly find solutions to NP-hard problems (i.e. P=NP), then

- Scheduling and routing questions (such as the Knapsack question and Traveling Salesman Problem) could be done efficiently.

$P \stackrel{?}{=} NP$: Why Does It Matter?



wiki

If you could quickly find solutions to NP-hard problems (i.e. P=NP), then

- Some hard biological questions (such as protein folding) would be tractable.

CLAY
MATHEMATICS
INSTITUTE

$P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?

CLAY MATHEMATICS INSTITUTE

$P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?

Solving this, will bring

- fame,

$P \stackrel{?}{=} NP$

CLAY MATHEMATICS INSTITUTE

$P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?

Solving this, will bring

- fame,
- fortune, and

CLAY MATHEMATICS INSTITUTE

$P \stackrel{?}{=} NP$: Roughly, if the answer to a problem can be checked quickly, can it be computed quickly?

Solving this, will bring

- fame,
- fortune, and
- change how algorithms are designed

In Pairs: Analyze Running Time

Give upper bounds on the worst case running time of:

1.
```
def double(n):
    d = 2*n
    return d
```

2.
```
def sum(n):
    s = 0
    for i in range(n):
        s += i
    return s
```

3.
```
def sum2(n):
    return n*(n+1)/2
```

4.
```
def findMin(numList):
    m = numList[0]
    for i in range(1,len(numList)):
        if numList[i] < m:
            m = numList[i]
    return m
```

5. (Code from text):
```
def selectionSort(alist):
   for fillslot in range(len(alist)-1,0,-1):
       positionOfMax=0
       for location in range(1,fillslot+1):
           if alist[location]>alist[positionOfMax]:
               positionOfMax = location

       temp = alist[fillslot]
       alist[fillslot] = alist[positionOfMax]
       alist[positionOfMax] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)
```

6. (Code from text):
```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

Analyze Space Requirements

- How much space an algorithm uses can matter.

Analyze Space Requirements

- How much space an algorithm uses can matter.
- If you want to align two long sequences (say 1 million bp each). The dynamic programming will require a matrix with 1 million $\times$ 1 million entries, requiring $10^6 \times 10^6 = 10^{12}$ places of storage.

Analyze Space Requirements

- How much space an algorithm uses can matter.
- If you want to align two long sequences (say 1 million bp each). The dynamic programming will require a matrix with 1 million $\times$ 1 million entries, requiring $10^6 \times 10^6 = 10^{12}$ places of storage.
- Can easily overwhelm the memory on your computer.

Analyze Space Requirements

- How much space an algorithm uses can matter.
- If you want to align two long sequences (say 1 million bp each). The dynamic programming will require a matrix with 1 million $\times$ 1 million entries, requiring $10^6 \times 10^6 = 10^{12}$ places of storage.
- Can easily overwhelm the memory on your computer.
- Can measure space usage as we did for time complexity.

Analyze Space Requirements

```python
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

- insertionSort sorts “in place”, and use no additional space.

Analyze Space Requirements

```python
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

- insertionSort sorts "in place", and use no additional space.
- Our sequence alignment under Needleman-Wunsch used an additional $O(n^2)$ space to store the dynamic programming array.

Analyze Space Requirements

```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

- insertionSort sorts “in place”, and use no additional space.
- Our sequence alignment under Needleman-Wunsch used an additional $O(n^2)$ space to store the dynamic programming array.

In Pairs: Analyze Space Requirement

The running time of bubblesort is analyzed above. Give upper bounds on the worst case space needed for:

1.
```
def double(n):
    d = 2*n
    return d
```

2.
```
def sum(n):
    s = 0
    for i in range(n):
        s += i
    return s
```

3.
```
def sum2(n):
    return n*(n+1)/2
```

4.
```
def findMin(numList):
    m = numList[0]
    for i in range(1,len(numList)):
        if numList[i] < m:
            m = numList[i]
    return m
```

5.
```
def findMaxDist(numList):
    n = len(numList)
    d = np.zeros(n,n)
    for i in range(1,n):
        for j in range(1,n)
            d[i,j] = abs(i,j)
    return np.amax(d)
```

6.
```
def findMaxDist2(numList):
    n = len(numList)
    m = 0
    for i in range(1,n):
        for j in range(1,n)
            if abs(i,j) > m:
                m = abs(i,j)
    return m
```

7. (Code from text):
```
def selectionSort(alist):
   for fillslot in range(len(alist)-1,0,-1):
       positionOfMax=0
       for location in range(1,fillslot+1):
           if alist[location]>alist[positionOfMax]:
               positionOfMax = location

       temp = alist[fillslot]
       alist[fillslot] = alist[positionOfMax]
       alist[positionOfMax] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)
```

More Complexity: Fixed Parameter Tractability

- Roughly, the ability to efficiently calculate instances that are small with respect to some parameter is called fixed parameter tractability.

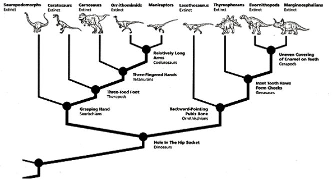
- Roughly, the ability to efficiently calculate instances that are small with respect to some parameter is called fixed parameter tractability.
- Though NP-hard, some problems can be solved in time polynomial in the size of the input size but exponential in the size of a fixed parameter.

More Complexity: Fixed Parameter Tractability

- Roughly, the ability to efficiently calculate instances that are small with respect to some parameter is called fixed parameter tractability.
- Though NP-hard, some problems can be solved in time polynomial in the size of the input size but exponential in the size of a fixed parameter.
- Often, the parameter, k, will be the distance between the trees.

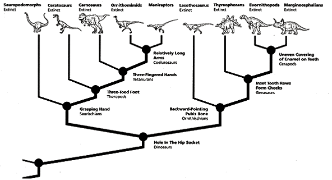
- Roughly, the ability to efficiently calculate instances that are small with respect to some parameter is called fixed parameter tractability.
- Though NP-hard, some problems can be solved in time polynomial in the size of the input size but exponential in the size of a fixed parameter.
- Often, the parameter, k, will be the distance between the trees.
- For example, the distance between the two trees can be calculated by shrinking the common regions and focusing on the differences, which can be bounded by k.

Recap: Small Parsimony Problem



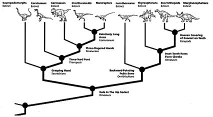
- Last Week: given a tree with leaves labeled by sequences, computed the parsimony score of the tree.

Recap: Small Parsimony Problem



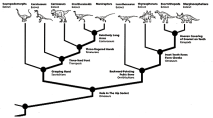
- Last Week: given a tree with leaves labeled by sequences, computed the parsimony score of the tree.
- Thinking in terms of time complexity: How long does it take?

- How do you code this?



AMNH

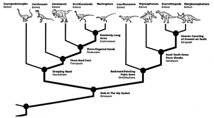
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.

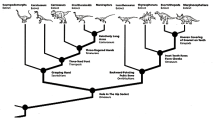
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).

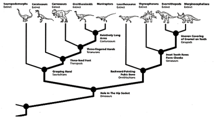
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?

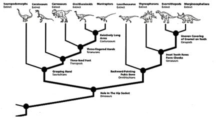
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure

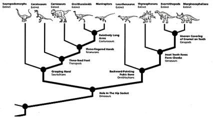
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure
  - Count of the number changes

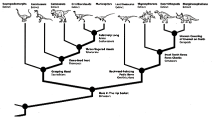
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure
  - Count of the number changes
- Algorithm:
  - First pass: Starting at the leaves, label the internal leaves (with possible multiple labels).

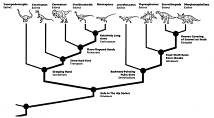
Algorithm Design: Scoring Trees Under Parsimony



AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure
  - Count of the number changes
- Algorithm:
  - First pass: Starting at the leaves, label the internal leaves (with possible multiple labels).
  - Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.

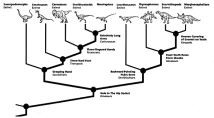
Fitch's Algorithm: Pseudocode



AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):

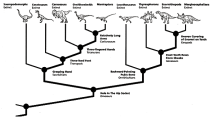
Fitch's Algorithm: Pseudocode



AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:
    `A T A T G`

Fitch's Algorithm: Pseudocode



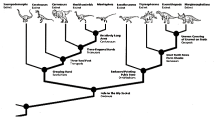
AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    `A T A T G`

    `A A T T G` $\rightarrow$ `A AT AT T G`
  - Go position by position:

Fitch's Algorithm: Pseudocode



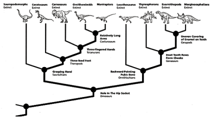
AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    `A T A T G`

    `A A T T G` $\rightarrow$ `A AT AT T G`
  - Go position by position:
    - ★ If overlap, use that label.

Fitch's Algorithm: Pseudocode

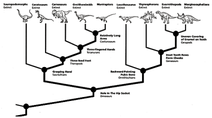


AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    ```
    A T A T G
    A A T T G  →  A AT AT T G
    ```
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.

Fitch's Algorithm: Pseudocode



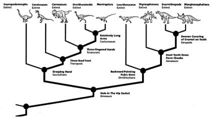
AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    `A T A T G`

    `A A T T G` $\rightarrow$ `A AT AT T G`
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`

Fitch's Algorithm: Pseudocode



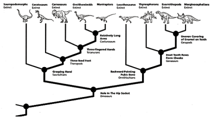
AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    `A T A T G`

    `A A T T G` $\rightarrow$ `A AT AT T G`
  - Go position by position:
    - If overlap, use that label.
    - If no overlap, use the union.
  - Useful Python container type: `set`
    - Has functions for union and intersection of sets.

Fitch's Algorithm: Pseudocode



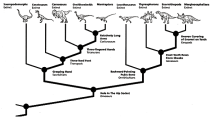
AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    ```
    A T A T G
    A A T T G   →   A AT AT T G
    ```

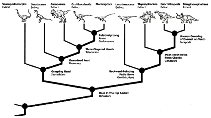
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.
    - ★ `s1 = set(l1)`
      `s2 = set(l2)`

Fitch's Algorithm: Pseudocode

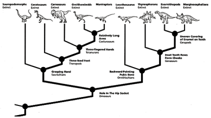


AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    ```
    A T A T G
    A A T T G   →   A AT AT T G
    ```
  - Go position by position:
    - If overlap, use that label.
    - If no overlap, use the union.
  - Useful Python container type: `set`
    - Has functions for union and intersection of sets.
    - ```
      s1 = set(l1)
      s2 = set(l2)
      print s1.intersection(s2)
      ```
      ```

Fitch's Algorithm: Pseudocode



AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    `A T A T G`

    `A A T T G` $\rightarrow$ `A AT AT T G`
  - Go position by position:
    - If overlap, use that label.
    - If no overlap, use the union.
  - Useful Python container type: `set`
    - Has functions for union and intersection of sets.
    - ```
      s1 = set(l1)
      s2 = set(l2)
      print s1.intersection(s2)
      print s1.union(s2)
      ```
      ```

Fitch's Algorithm: Pseudocode



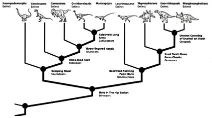
AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    ```
    A T A T G
    A A T T G   →   A AT AT T G
    ```
  - Go position by position: for-loop
    - ★ If overlap, use that label. if-statement
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.
    - ★ set operations

      ```
      s1 = set(l1)
      s2 = set(l2)
      print s1.intersection(s2)
      print s1.union(s2)
      ```

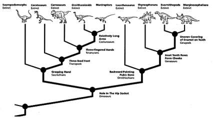
Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.

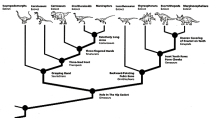
Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:

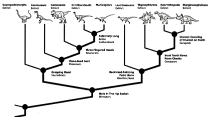
Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` $\rightarrow$ `A A T T G`
  - For all other nodes, compare to the parent:

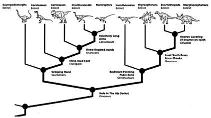
Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` $\rightarrow$ `A A T T G`
  - For all other nodes, compare to the parent:
    `A T A T G`
    `AT AT G ACT G`

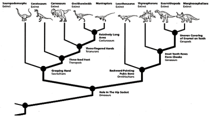
Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` → `A A T T G`
  - For all other nodes, compare to the parent:
    `A T A T G`
    `AT AT G ACT G` → `A T G T G`

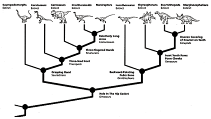
Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` $\rightarrow$ `A A T T G`
  - For all other nodes, compare to the parent:
    `A T A T G`
    `AT AT G ACT G` $\rightarrow$ `A T G T G`
  - Go position by position:

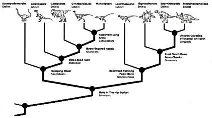
Fitch's Algorithm: Pseudocode



AMNH

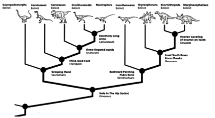
- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` $\rightarrow$ `A A T T G`
  - For all other nodes, compare to the parent:
    `A T A T G`
    `AT AT G ACT G` $\rightarrow$ `A T G T G`
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, choose label from child.

Fitch's Algorithm: Pseudocode



AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` $\rightarrow$ `A A T T G`
  - For all other nodes, compare to the parent:
    `A T A T G`
    `AT AT G ACT G` $\rightarrow$ `A T G T G`
  - Go position by position: for-loop
    - ★ If overlap, use that label. if-statement
    - ★ If no overlap, choose label from child.

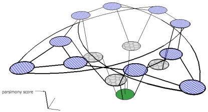


AMNH

What is the running time and space requirements for:

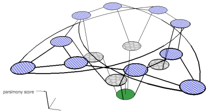
- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels).
- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
- Print out all tree on *n* leaves.

Searching for Optimal Trees



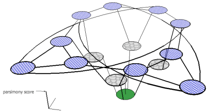
- Large Parsimony Problem: Given sequences for leaves, find the optimal scoring tree.

Searching for Optimal Trees



- Large Parsimony Problem: Given sequences for leaves, find the optimal scoring tree.
- Visually: think of the trees on a 2D map and the height above sea level is the score.

Searching for Optimal Trees



- Large Parsimony Problem: Given sequences for leaves, find the optimal scoring tree.
- Visually: think of the trees on a 2D map and the height above sea level is the score.
- Works for any optimality criteria (i.e. same analogy works for maximum likelihood).

Analogy: Find the Highest Point

polymaps.org

Analogy: Find the Highest Point

Sampling:

- Choose 1000 random points.

Analogy: Find the Highest Point

Sampling:

- Choose 1000 random points.
- Find height at each point.

Analogy: Find the Highest Point

Sampling:

- Choose 1000 random points.
- Find height at each point.
- Output the sampled point with largest height.

Analogy: Find the Highest Point

Sampling:

- Choose 1000 random points.
- Find height at each point.
- Output the sampled point with largest height.
- Will you reach the highest point?

Sampling:

- Choose 1000 random points.
- Find height at each point.
- Output the sampled point with largest height.
- Will you reach the highest point?
- Only if very lucky or a very dense sample.

Hill Climbing:

- Start at the harbor.

Analogy: Find the Highest Point

Hill Climbing:

- Start at the harbor.
- Can see 25 meters in all directions.

Analogy: Find the Highest Point

Hill Climbing:

- Start at the harbor.
- Can see 25 meters in all directions.
- Walk upwards, repeat.

Analogy: Find the Highest Point

Hill Climbing:

- Start at the harbor.
- Can see 25 meters in all directions.
- Walk upwards, repeat.
- Will you reach the highest point?

Hill Climbing:

- Start at the harbor.
- Can see 25 meters in all directions.
- Walk upwards, repeat.
- Will you reach the highest point?
- Maybe, but maybe not.

Analogy: Find the Highest Point

Hill Climbing:

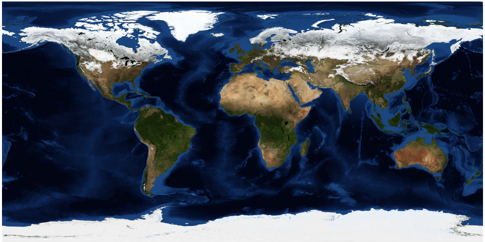
- Start at the harbor.
- Can see 25 meters in all directions.
- Walk upwards, repeat.
- Will you reach the highest point?
- Maybe, but maybe not.
  - Could reach small peaks, but miss the larger ones.

Analogy: Find the Highest Point

Hill Climbing:

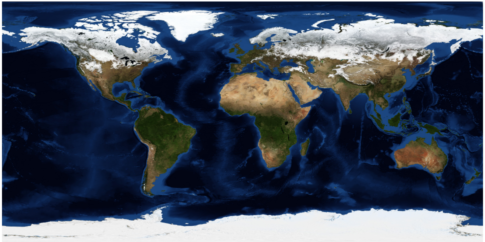
- Start at the harbor.
- Can see 25 meters in all directions.
- Walk upwards, repeat.
- Will you reach the highest point?
- Maybe, but maybe not.
  - Could reach small peaks, but miss the larger ones.
  - Start in multiple places to see more.

Local Search Techniques



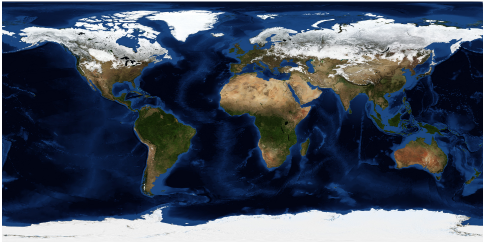
- **Goal:** Find the point with the optimal score

Local Search Techniques



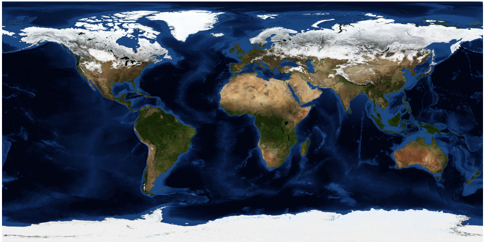
- **Goal:** Find the point with the optimal score
- Local search techniques prevail:

Local Search Techniques



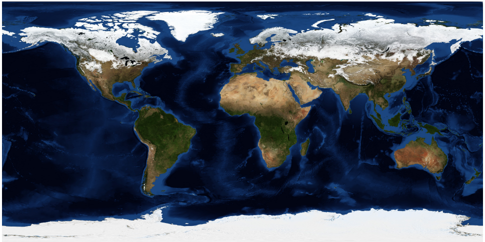
- **Goal:** Find the point with the optimal score
- Local search techniques prevail:
  - Begin with a point

Local Search Techniques



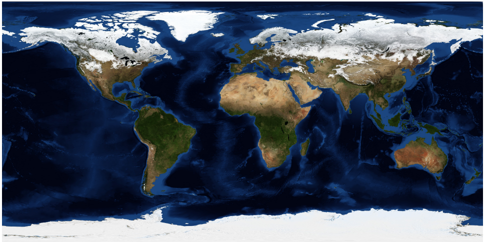
- **Goal:** Find the point with the optimal score
- Local search techniques prevail:
  - Begin with a point
  - Choose the next point from its neighbors (e.g. best scoring)

Local Search Techniques



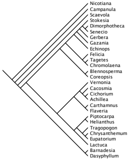
- **Goal:** Find the point with the optimal score
- Local search techniques prevail:
  - Begin with a point
  - Choose the next point from its neighbors (e.g. best scoring)
  - Repeat

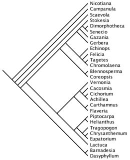
Local Search Techniques



- **Goal:** Find the point with the optimal score
- Local search techniques prevail:
  - Begin with a point
  - Choose the next point from its neighbors (e.g. best scoring)
  - Repeat
- Many variations on the theme: branch-and-bound, MCMC, genetic algorithms,...

Popular Tree Metrics

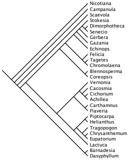


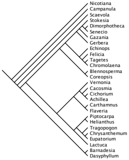


Those based on tree rearrangements:

- Subtree Prune and Regraft (SPR)
- Tree Bisection and Reconnection (TBR)
- Nearest Neighbor Interchange (NNI)

Popular Tree Metrics

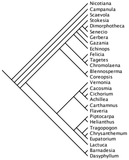




Those based on tree rearrangements:

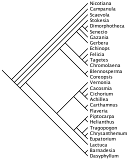
- Subtree Prune and Regraft (SPR)
- Tree Bisection and Reconnection (TBR)
- Nearest Neighbor Interchange (NNI)
- Used for Searching for Optimal Trees, NP-hard

Popular Tree Metrics



Those based on tree rearrangements:

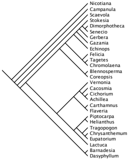
- Subtree Prune and Regraft (SPR)
- Tree Bisection and Reconnection (TBR)
- Nearest Neighbor Interchange (NNI)
- Used for Searching for Optimal Trees, NP-hard



Those based on comparing tree vectors:

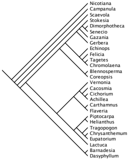
- Robinson-Foulds (RF)
- Rooted Triples (RT)
- Quartet Distance
- Billera-Holmes-Vogtmann (BHV or geodesic))

Popular Tree Metrics



Those based on tree rearrangements:

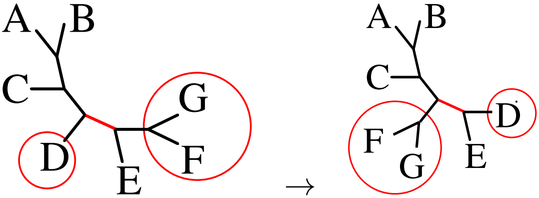
- Subtree Prune and Regraft (SPR)
- Tree Bisection and Reconnection (TBR)
- Nearest Neighbor Interchange (NNI)
- Used for Searching for Optimal Trees, NP-hard



Those based on comparing tree vectors:

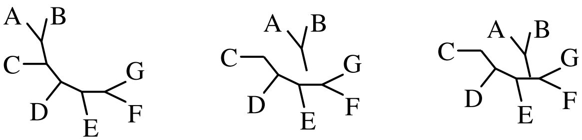
- Robinson-Foulds (RF)
- Rooted Triples (RT)
- Quartet Distance
- Billera-Holmes-Vogtmann (BHV or geodesic))
- Used for comparing trees, poly time

NNI Metric



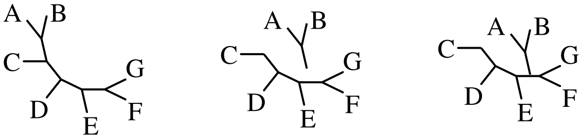
The NNI distance between two trees is the minimal number of moves needed to transform one to the other (NP-hard, DasGupta *et al.* '97).

SPR Distance



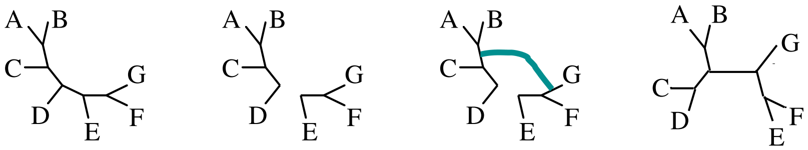
- SPR distance is the minimal number of moves that transforms one tree into the other.

SPR Distance



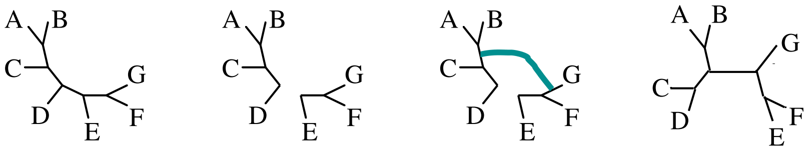
- SPR distance is the minimal number of moves that transforms one tree into the other.
- SPR for rooted trees is NP-hard (Bordewich & Semple '05).
- SPR for unrooted trees is NP-hard (Hickey *et al.* '08).
- SAT-based heuristic (Bonet & S '09).

TBR Distance



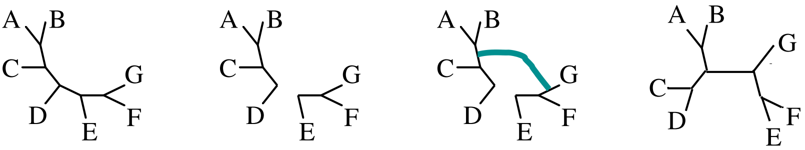
- TBR distance is the minimal number of moves that transforms one tree into the other.

TBR Distance



- TBR distance is the minimal number of moves that transforms one tree into the other.
- TBR is NP-hard and FPT. (Allen & Steel '01)

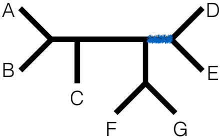
TBR Distance



- TBR distance is the minimal number of moves that transforms one tree into the other.
- TBR is NP-hard and FPT. (Allen & Steel '01)
- TBR has a linear time 5-approximation and a polynomial time 3-approximation (Amenta, Bonet, Mahindru, & S '06; Bordewich, McCartin, & Semple '08)

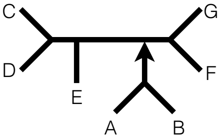
Tree Rearrangements

Tree Rearrangements



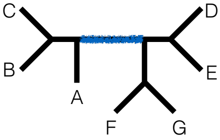
Nearest Neighbor Interchange

(NNI)



Subtree Prune & Regraft

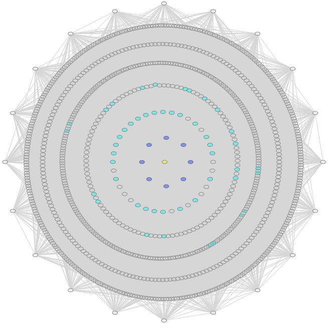
(SPR)



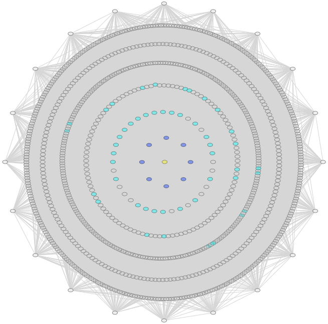
Tree Bisection & Reconnection

(TBR)

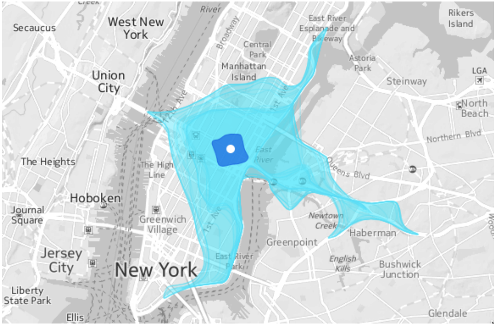
Metrics Matter



Metrics Matter

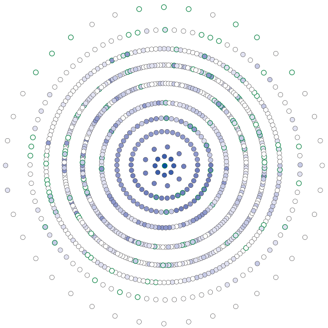


NNI & SPR



Isoscope

Landscapes



Parsimony score for compatible characters for $n = 7$ (Urheim, Ford, & S, submitted)

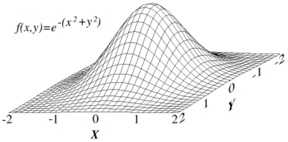
A treespace with assigned scores is often called a landscape.

What does the landscape look like?

Each landscape depends on the number of taxa and the score of each tree (usually derived from the inputted character sequences).

What does the landscape look like?

Each landscape depends on the number of taxa and the score of each tree (usually derived from the inputted character sequences).

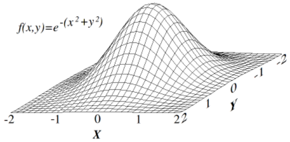


(from wikipedia)

If very smooth, 'hill climbing' will work well.

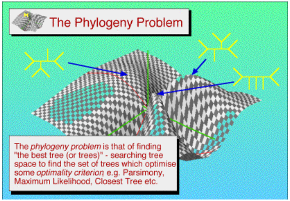
What does the landscape look like?

Each landscape depends on the number of taxa and the score of each tree (usually derived from the inputted character sequences).



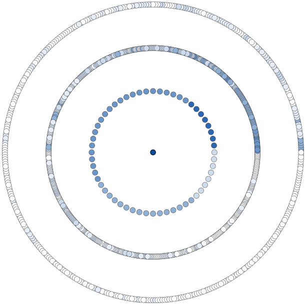
(from wikipedia)

If very smooth, ‘hill climbing’ will work well.

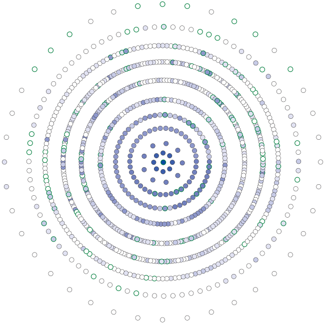


If very rugged, need more sophisticated searches that use the underlying structure of the space.

Adjusting Search Space



SPR metric

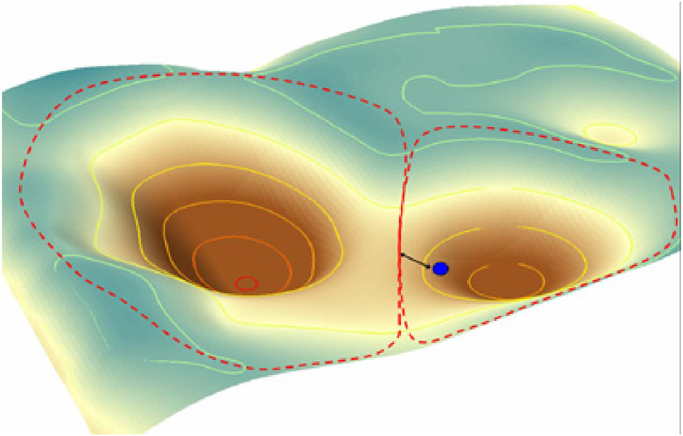


NNI metric

Parsimony score for compatible characters for $n = 7$ (Urheim, Ford, & S, submitted)

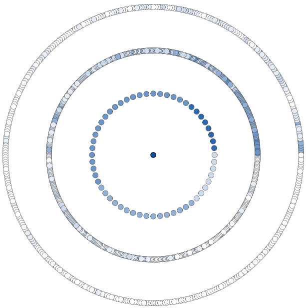
The same data, organized by different tree metrics.

Attraction Basins

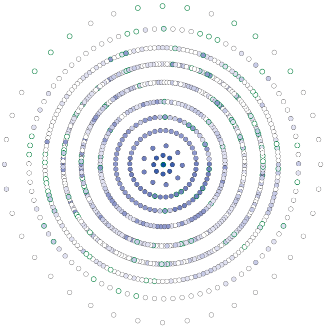


resalliance.org

Adjusting Search Space



SPR metric



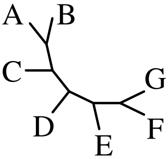
NNI metric

Parsimony score for compatible characters for $n = 7$ (Urheim, Ford, & S, submitted)

Simplest Case: for compatible character sequences ('perfect data'):

- Under SPR, there is a single attraction basin.
- Under NNI, multiple attraction basins occur even for perfect data.

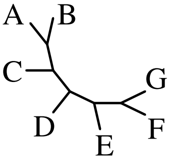
In Pairs: Tree Rearrangements



For the tree on the left:

1. What are the NNI neighbors?

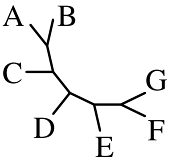
In Pairs: Tree Rearrangements



For the tree on the left:

1. What are the NNI neighbors?
2. Give a SPR neighbor that is not a NNI neighbor.

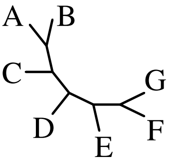
In Pairs: Tree Rearrangements



For the tree on the left:

1. What are the NNI neighbors?
2. Give a SPR neighbor that is not a NNI neighbor.
3. Give a TBR neighbor that is not an SPR neighbor.

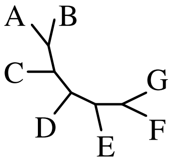
In Pairs: Tree Rearrangements



For the tree on the left:

1. What are the NNI neighbors?
2. Give a SPR neighbor that is not a NNI neighbor.
3. Give a TBR neighbor that is not an SPR neighbor.
4. Find a tree that is NNI distance 3 but SPR distance 1.

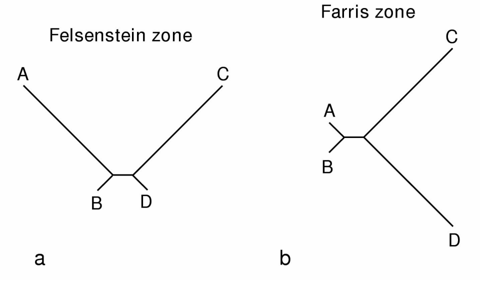
In Pairs: Tree Rearrangements



For the tree on the left:

1. What are the NNI neighbors?
2. Give a SPR neighbor that is not a NNI neighbor.
3. Give a TBR neighbor that is not an SPR neighbor.
4. Find a tree that is NNI distance 3 but SPR distance 1.
5. Write an algorithm for given a tree T and a specific edge/node, the two NNI neighbors around that edge.

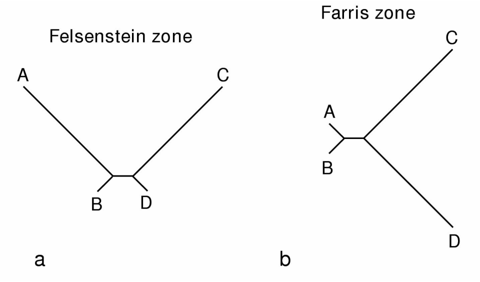
Weighted Trees



Philippe *et al.*, '05

- Branch weights are part of the model.

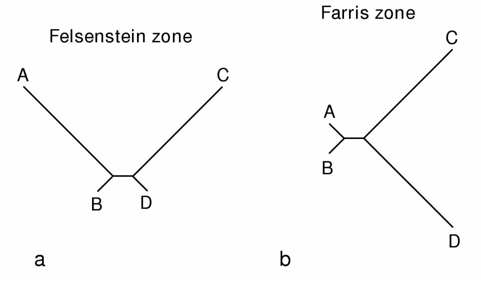
Weighted Trees



Philippe *et al.*, '05

- Branch weights are part of the model.
- Indicated by length of edges in drawing.

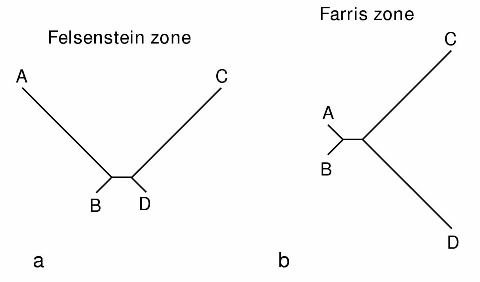
Weighted Trees



Philippe *et al.*, '05

- Branch weights are part of the model.
- Indicated by length of edges in drawing.
- Two classic trees with same underlying topology.

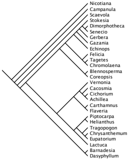
Weighted Trees

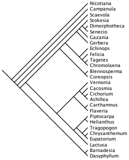


Philippe *et al.*, '05

- Branch weights are part of the model.
- Indicated by length of edges in drawing.
- Two classic trees with same underlying topology.
- The metrics and search spaces above treat them as identical.

Popular Tree Metrics

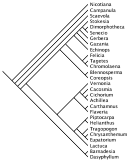




Those based on tree rearrangements:

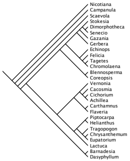
- Subtree Prune and Regraft (SPR)
- Tree Bisection and Reconnection (TBR)
- Nearest Neighbor Interchange (NNI)

Popular Tree Metrics



Those based on tree rearrangements:

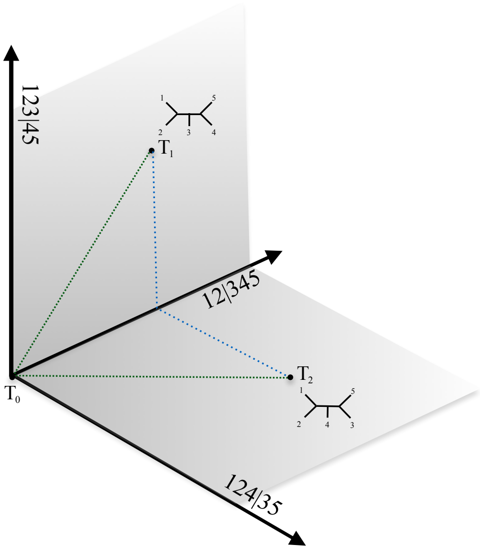
- Subtree Prune and Regraft (SPR)
- Tree Bisection and Reconnection (TBR)
- Nearest Neighbor Interchange (NNI)
- Used for Searching for Optimal Trees, NP-hard



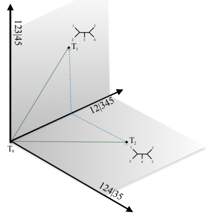
Those based on comparing tree vectors:

- Robinson-Foulds (RF)
- Rooted Triples (RT)
- Quartet Distance
- Billera-Holmes-Vogtmann (BHV or geodesic))
- Used for comparing trees, poly time

Trees as Vectors

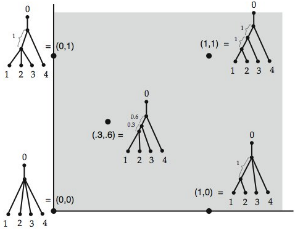


Trees as Vectors



	1|2345	2|1345	3|1245	4|1235	5|1234	12|345	13|245	14|235	15|234	23|145	24|135	25|134	34|125	35|124	45|123
$T_0 = (1,2,3,4,5)$	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$T_1 = ((1,2),(3,(4,5))$	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1
$T_2 = ((1,2),(4,(3,5))$	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0

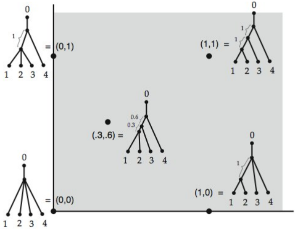
BHV Distance



Billera, Holmes, Vogtmann '01

- Billera, Holmes, and Vogtmann '01 have a continuous metric space of trees.

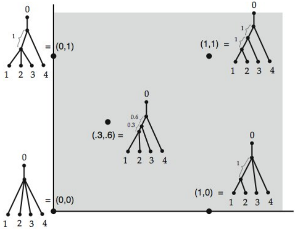
BHV Distance



Billera, Holmes, Vogtmann '01

- Billera, Holmes, and Vogtmann '01 have a continuous metric space of trees.
- View each split in a tree as a coordinate in the space.

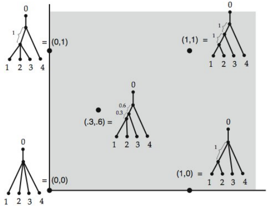
BHV Distance



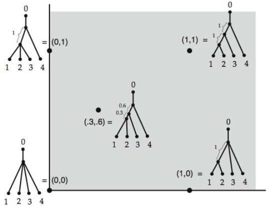
Billera, Holmes, Vogtmann '01

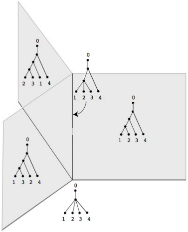
- Billera, Holmes, and Vogtmann '01 have a continuous metric space of trees.
- View each split in a tree as a coordinate in the space.
- Identify edges of orthants to form space

Identify Edges of Orthants

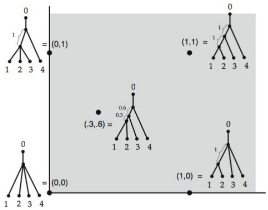


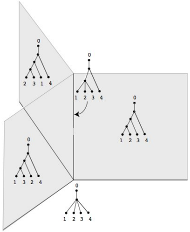
Identify Edges of Orthants

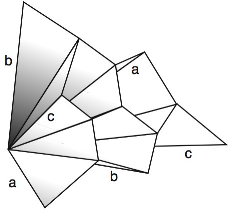




Identify Edges of Orthants

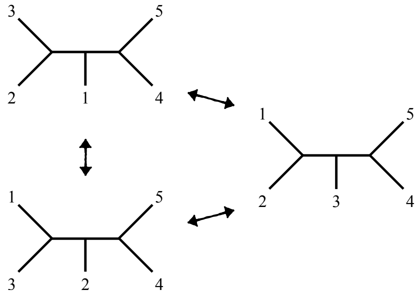


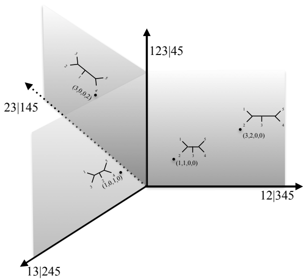




(All images from Billera, Holmes, Vogtmann '01)

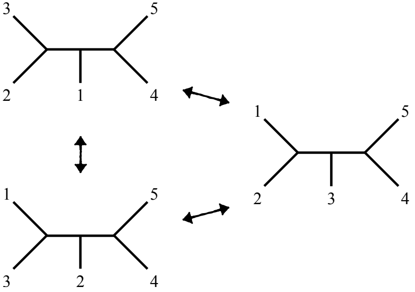
BHV & NNI Distances

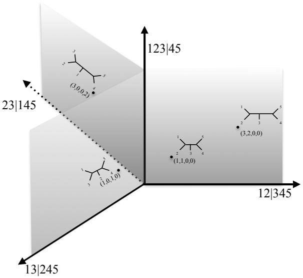




- Simplest move between orthants: shrink coordinate/edge and expand.

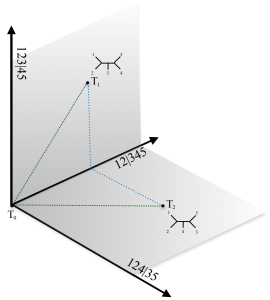
BHV & NNI Distances

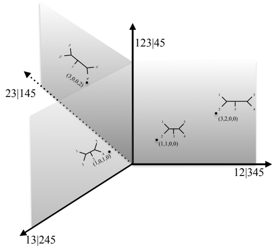




- Simplest move between orthants: shrink coordinate/edge and expand.
- Corresponds to a Nearest Neighbor Interchange (NNI) move on the topology.

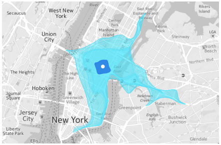
In Pairs: Continuous Treespace





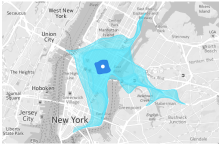
1. What is the distance between T_1 and T_2?
2. What is the average (tree at the midpoint) of T_1 and T_2?
3. Give three trees that have distance 1 to the origin, T_0 (star tree).
4. Lower figure: what is the distance between (3,0,0,2) and (1,1,0,0)?
5. What is a good average/consensus for the four trees?

Recap



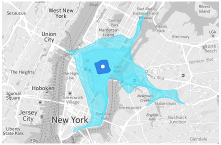
- Efficiency (in time and space) matters when data gets large.

Recap



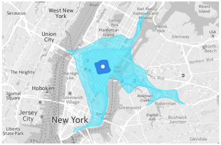
- Efficiency (in time and space) matters when data gets large.
- More on tree searching next lecture.

Recap



- Efficiency (in time and space) matters when data gets large.
- More on tree searching next lecture.
- Email lab reports to `kstjohn@amnh.org`.

Recap



- Efficiency (in time and space) matters when data gets large.
- More on tree searching next lecture.
- Email lab reports to `kstjohn@amnh.org`.
- Challenges available at `rosalind.info`.