

# On the Locality of the Prüfer Code

Craig Lennon

November 3, 2008

# Talk Outline

- 1 Background
- 2 Results
- 3 Method

# Genetic/Evolutionary Algorithms

- Designing algorithms informed by principles of evolution has become a broad field in the last two decades.

# Genetic/Evolutionary Algorithms

- Designing algorithms informed by principles of evolution has become a broad field in the last two decades.
- My research involves investigating how useful the Prüfer code is as a genotype in a genetic algorithm.

# Genetic/Evolutionary Algorithms

- Designing algorithms informed by principles of evolution has become a broad field in the last two decades.
- My research involves investigating how useful the Prüfer code is as a genotype in a genetic algorithm.
- I will begin with some definitions.

# Genetic algorithms

- A genetic algorithm typically requires 4 things.

# Genetic algorithms

- A genetic algorithm typically requires 4 things.
  - A genotype which represents a possible solution to the problem we are attempting to solve.

# Genetic algorithms

- A genetic algorithm typically requires 4 things.
  - A genotype which represents a possible solution to the problem we are attempting to solve.
  - A “fitness function” which evaluates how well the genotype solves the problem.

# Genetic algorithms

- A genetic algorithm typically requires 4 things.
  - A genotype which represents a possible solution to the problem we are attempting to solve.
  - A “fitness function” which evaluates how well the genotype solves the problem.
  - A method for selecting an initial population of genotypes.

# Genetic algorithms

- A genetic algorithm typically requires 4 things.
  - A genotype which represents a possible solution to the problem we are attempting to solve.
  - A “fitness function” which evaluates how well the genotype solves the problem.
  - A method for selecting an initial population of genotypes.
  - A method of generating new genotypes from the current population.

# Example of a Genetic Algorithm

- Consider the search for a minimum weight spanning tree.

# Example of a Genetic Algorithm

- Consider the search for a minimum weight spanning tree.
- Genotype: a numeric string which represents a spanning tree.

# Example of a Genetic Algorithm

- Consider the search for a minimum weight spanning tree.
- Genotype: a numeric string which represents a spanning tree.
- Fitness function: the sum of the weights of the edges.

# Example of a Genetic Algorithm

- Consider the search for a minimum weight spanning tree.
- Genotype: a numeric string which represents a spanning tree.
- Fitness function: the sum of the weights of the edges.
- Choose strings uniformly at random for initial population.

# Example of a Genetic Algorithm

- Consider the search for a minimum weight spanning tree.
- Genotype: a numeric string which represents a spanning tree.
- Fitness function: the sum of the weights of the edges.
- Choose strings uniformly at random for initial population.
- Choose new genotypes by recombination and mutation.

# Recombination and mutation

- Recombination: choose two low weight strings  
(1, 2, 3, 4, 5, 6, 4, 5, 1, 2, 7, 12), (2, 5, 7, 7, 4, 2, 1, 3, 1, 1, 6, 6)

# Recombination and mutation

- Recombination: choose two low weight strings  
(1, 2, 3, 4, 5, 6, 4, 5, 1, 2, 7, 12), (2, 5, 7, 7, 4, 2, 1, 3, 1, 1, 6, 6)
- Our new strings are  
(1, 2, 3, 4, 5, 6, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 1, 2, 7, 12)

# Recombination and mutation

- Recombination: choose two low weight strings  
(1, 2, 3, 4, 5, 6, 4, 5, 1, 2, 7, 12), (2, 5, 7, 7, 4, 2, 1, 3, 1, 1, 6, 6)
- Our new strings are  
(1, 2, 3, 4, 5, 6, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 1, 2, 7, 12)
- Mutation:  
(1, 2, 3, 4, 5, 1, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 5, 2, 7, 12)

# Recombination and mutation

- Recombination: choose two low weight strings  
(1, 2, 3, 4, 5, 6, 4, 5, 1, 2, 7, 12), (2, 5, 7, 7, 4, 2, 1, 3, 1, 1, 6, 6)
- Our new strings are  
(1, 2, 3, 4, 5, 6, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 1, 2, 7, 12)
- Mutation:  
(1, 2, 3, 4, 5, 1, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 5, 2, 7, 12)
- Mutation is a small change which (ideally) helps us find a local optimum.

# Recombination and mutation

- Recombination: choose two low weight strings  
(1, 2, 3, 4, 5, 6, 4, 5, 1, 2, 7, 12), (2, 5, 7, 7, 4, 2, 1, 3, 1, 1, 6, 6)
- Our new strings are  
(1, 2, 3, 4, 5, 6, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 1, 2, 7, 12)
- Mutation:  
(1, 2, 3, 4, 5, 1, 1, 3, 1, 1, 6, 6), (2, 5, 7, 7, 4, 2, 4, 5, 5, 2, 7, 12)
- Mutation is a small change which (ideally) helps us find a local optimum.
- Recombination is a large change which (ideally) keeps us from getting stuck at one (or a few) local optimum.

# What encoding method?

- In our algorithm we need to decide what method we want to use to encode our trees as numeric strings.

# What encoding method?

- In our algorithm we need to decide what method we want to use to encode our trees as numeric strings.
- Maybe the best known method is the Prüfer code.

# What encoding method?

- In our algorithm we need to decide what method we want to use to encode our trees as numeric strings.
- Maybe the best known method is the Prüfer code.
- This was one of the first used for GAs.

# What encoding method?

- In our algorithm we need to decide what method we want to use to encode our trees as numeric strings.
- Maybe the best known method is the Prüfer code.
- This was one of the first used for GAs.
- For example it was suggested for a minimum weight spanning tree problem in
  - *A Note on Genetic Algorithms for Degree-Constrained Spanning Tree Problems*,
  - Authors Gengui Zhou and Mitsuo Gen,
  - The journal Network (April 1997).

# What encoding method?

- In our algorithm we need to decide what method we want to use to encode our trees as numeric strings.
- Maybe the best known method is the Prüfer code.
- This was one of the first used for GAs.
- For example it was suggested for a minimum weight spanning tree problem in
  - *A Note on Genetic Algorithms for Degree-Constrained Spanning Tree Problems*,
  - Authors Gengui Zhou and Mitsuo Gen,
  - The journal Network (April 1997).
- Problem: with the Prüfer code, a small change in the string may not correspond to a small change in the tree.

# The Prüfer code

- The Prüfer code is a bijection between (labeled) trees on the set  $[n]$  and strings of length  $n - 2$  on the set  $[n]$ .

# The Prüfer code

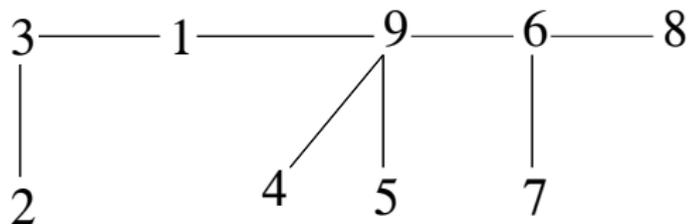
- The Prüfer code is a bijection between (labeled) trees on the set  $[n]$  and strings of length  $n - 2$  on the set  $[n]$ .
- To encode a tree as a string, we repeatedly remove the lowest number leaf and record its neighbor, stopping when we have a single edge remaining.

# The Prüfer code

- The Prüfer code is a bijection between (labeled) trees on the set  $[n]$  and strings of length  $n - 2$  on the set  $[n]$ .
- To encode a tree as a string, we repeatedly remove the lowest number leaf and record its neighbor, stopping when we have a single edge remaining.
- We will refer to the strings which represent trees as “P-strings.”

# The Encoding Process

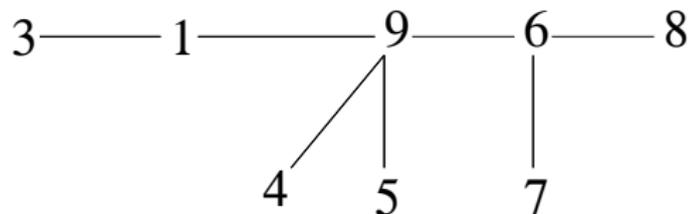
- As an example we consider a tree on 9 vertices which will be encoded as a length 7 P-string:
- The tree before step 1:



- The P-string before step 1:  $(\_, \_, \_, \_, \_, \_, \_)$

# The Encoding Process step 1

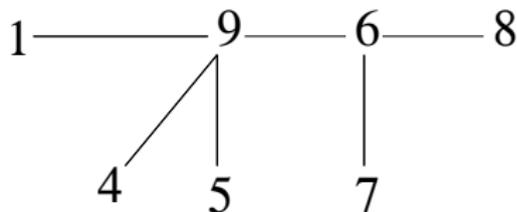
- The tree after step 1:



- The P-string after step 1:  $(3, \_, \_, \_, \_, \_, \_)$

# The Encoding Process step 2

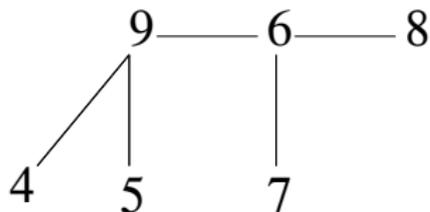
- The tree after step 2:



- The P-string after step 2: (3, 1, \_\_, \_\_, \_\_, \_\_, \_\_)

# The Encoding Process step 3

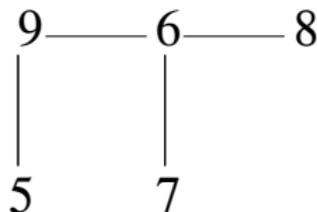
- The tree after step 3:



- The P-string after step 3: (3, 1, 9, \_\_, \_\_, \_\_, \_\_)

# The Encoding Process step 4

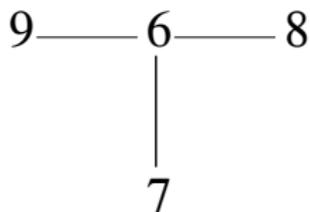
- The tree after step 4:



- The P-string after step 4: (3, 1, 9, 9, \_\_, \_\_, \_\_)

# The Encoding Process step 5

- The tree after step 5:



- The P-string after step 5: (3, 1, 9, 9, 9, \_\_, \_\_)

# The Encoding Process step 6

- The tree after step 6:

9——6——8

- The P-string after step 6: (3, 1, 9, 9, 9, 6, \_)

# The Encoding Process step 7

- The tree after step 7:

9——6

- The P-string after step 7: (3, 1, 9, 9, 9, 6, 6)

# Mutation

- A mutation is the change of a single entry in a P-string: for example changing (3, 1, 9, 9, 9, 6, 6) to (3, 1, 9, 7, 9, 6, 6)

# Mutation

- A mutation is the change of a single entry in a P-string: for example changing  $(3, 1, 9, 9, 9, 6, 6)$  to  $(3, 1, 9, 7, 9, 6, 6)$
- We are interested in how much a tree changes after a mutation.

# Mutation

- A mutation is the change of a single entry in a P-string: for example changing  $(3, 1, 9, 9, 9, 6, 6)$  to  $(3, 1, 9, 7, 9, 6, 6)$
- We are interested in how much a tree changes after a mutation.
- As a measure of change, we count the number of edges which must be changed to make one tree into another:

# Mutation

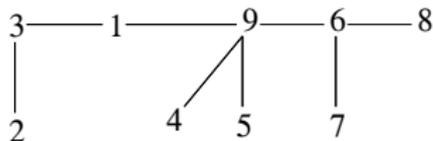
- A mutation is the change of a single entry in a P-string: for example changing  $(3, 1, 9, 9, 9, 6, 6)$  to  $(3, 1, 9, 7, 9, 6, 6)$
- We are interested in how much a tree changes after a mutation.
- As a measure of change, we count the number of edges which must be changed to make one tree into another:

$$\Delta^{(n)}(T, T^*) := n - 1 - |E(T) \cap E(T^*)|,$$

where  $E(T)$  is the edge set of tree  $T$ .

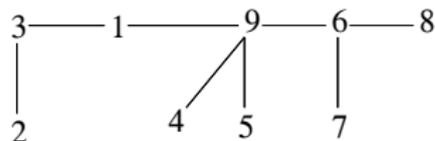
# Mutation example 1

- The tree  $T$  corresponding to  $(3, 1, 9, 9, 9, 6, 6)$  is

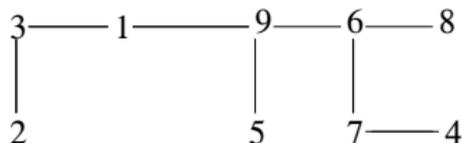


# Mutation example 1

- The tree  $T$  corresponding to  $(3, 1, 9, 9, 9, 6, 6)$  is

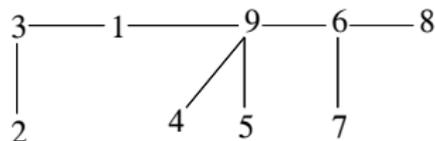


- The tree  $T^*$  corresponding to  $(3, 1, 9, 7, 9, 6, 6)$  is:

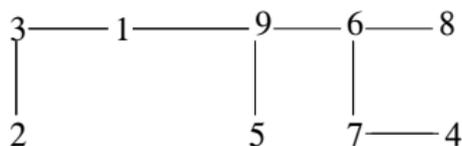


# Mutation example 1

- The tree  $T$  corresponding to  $(3, 1, 9, 9, 9, 6, 6)$  is



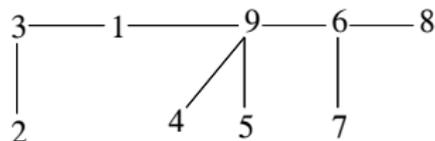
- The tree  $T^*$  corresponding to  $(3, 1, 9, 7, 9, 6, 6)$  is:



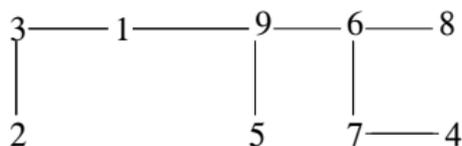
- Here  $\Delta^{(n)}(T, T^*) = 1$ , so the mutation resulted in a small change in the trees.

# Mutation example 1

- The tree  $T$  corresponding to  $(3, 1, 9, 9, 9, 6, 6)$  is



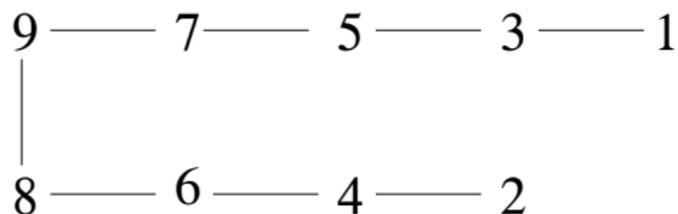
- The tree  $T^*$  corresponding to  $(3, 1, 9, 7, 9, 6, 6)$  is:



- Here  $\Delta^{(n)}(T, T^*) = 1$ , so the mutation resulted in a small change in the trees.
- Sometimes, a mutation can result in a large change in the trees.

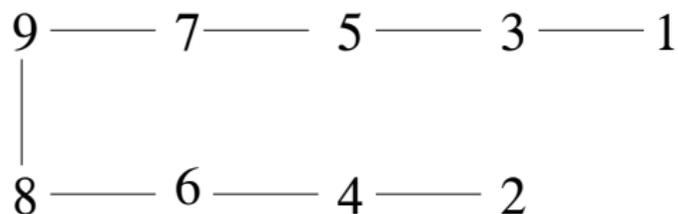
## Mutation example 2

- Consider the tree  $T$  corresponding to  $(3, 4, 5, 6, 7, 8, 9)$

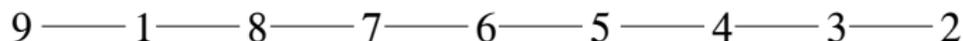


## Mutation example 2

- Consider the tree  $T$  corresponding to  $(3, 4, 5, 6, 7, 8, 9)$

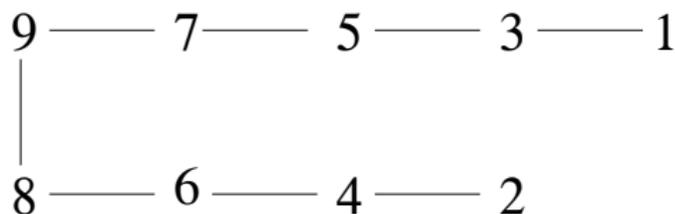


and the tree  $T^*$  corresponding to  $(3, 4, 5, 6, 7, 8, 1)$ :

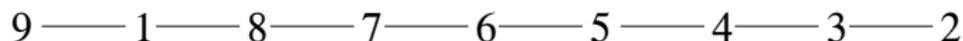


## Mutation example 2

- Consider the tree  $T$  corresponding to  $(3, 4, 5, 6, 7, 8, 9)$



and the tree  $T^*$  corresponding to  $(3, 4, 5, 6, 7, 8, 1)$ :



- Here  $\Delta^{(n)}(T, T^*) = 8$ , so the mutation resulted in a large change in the trees.

# Other tree representations

- The Prüfer code is not the only way to represent trees on  $[n]$  as strings of length  $n - 2$  on  $[n]$ .

# Other tree representations

- The Prüfer code is not the only way to represent trees on  $[n]$  as strings of length  $n - 2$  on  $[n]$ .
- Other codes include the Dandelion, Blob, Happy codes (which we won't describe here).

# Other tree representations

- The Prüfer code is not the only way to represent trees on  $[n]$  as strings of length  $n - 2$  on  $[n]$ .
- Other codes include the Dandelion, Blob, Happy codes (which we won't describe here).
- We want to select a code for which small changes in the representative string correspond to small changes in the represented tree.

## Other tree representations

- The Prüfer code is not the only way to represent trees on  $[n]$  as strings of length  $n - 2$  on  $[n]$ .
- Other codes include the Dandelion, Blob, Happy codes (which we won't describe here).
- We want to select a code for which small changes in the representative string correspond to small changes in the represented tree.
- A tree representation for which this holds is said to have high locality (and if it doesn't hold the locality is low).

# Locality

- We will say that a tree representation has high locality if mutations almost surely result in trees whose difference, in terms of  $\Delta^{(n)}$ , is 1.

# Locality

- We will say that a tree representation has high locality if mutations almost surely result in trees whose difference, in terms of  $\Delta^{(n)}$ , is 1.
- Formally, we choose a method for representing trees as strings.

# Locality

- We will say that a tree representation has high locality if mutations almost surely result in trees whose difference, in terms of  $\Delta^{(n)}$ , is 1.
- Formally, we choose a method for representing trees as strings.
  - Let  $\mathcal{S}$  represent the set of all ordered pairs of strings whose coordinates differ in exactly one position.

# Locality

- We will say that a tree representation has high locality if mutations almost surely result in trees whose difference, in terms of  $\Delta^{(n)}$ , is 1.
- Formally, we choose a method for representing trees as strings.
  - Let  $\mathcal{S}$  represent the set of all ordered pairs of strings whose coordinates differ in exactly one position.
  - We choose an element  $(S, S^*) \in \mathcal{S}$  uniformly at random with  $T, T^*$  being the trees corresponding to  $S, S^*$ .

# Locality

- We will say that a tree representation has high locality if mutations almost surely result in trees whose difference, in terms of  $\Delta^{(n)}$ , is 1.
- Formally, we choose a method for representing trees as strings.
  - Let  $\mathcal{S}$  represent the set of all ordered pairs of strings whose coordinates differ in exactly one position.
  - We choose an element  $(S, S^*) \in \mathcal{S}$  uniformly at random with  $T, T^*$  being the trees corresponding to  $S, S^*$ .
  - Then  $\Delta^{(n)}(T, T^*)$  is a random variable with sample space  $\mathcal{S}$ .

# Locality

- We will say that a tree representation has high locality if mutations almost surely result in trees whose difference, in terms of  $\Delta^{(n)}$ , is 1.
- Formally, we choose a method for representing trees as strings.
  - Let  $\mathcal{S}$  represent the set of all ordered pairs of strings whose coordinates differ in exactly one position.
  - We choose an element  $(S, S^*) \in \mathcal{S}$  uniformly at random with  $T, T^*$  being the trees corresponding to  $S, S^*$ .
  - Then  $\Delta^{(n)}(T, T^*)$  is a random variable with sample space  $\mathcal{S}$ .
- The tree representation under consideration has high locality if

$$P\left(\Delta^{(n)}(T, T^*) = 1\right) \rightarrow 1 \quad \text{as } n \rightarrow \infty.$$

# Condition on the location of the mutation

- We will find it convenient to condition on the entry of the  $P$ -string which is mutated.

# Condition on the location of the mutation

- We will find it convenient to condition on the entry of the  $P$ -string which is mutated.
- Let  $\mathcal{S}_\mu$  represent the set of all ordered pairs of strings whose coordinates differ in position  $\mu$ .

## Condition on the location of the mutation

- We will find it convenient to condition on the entry of the  $P$ -string which is mutated.
- Let  $\mathcal{S}_\mu$  represent the set of all ordered pairs of strings whose coordinates differ in position  $\mu$ .
- Then

$$P\left(\Delta^{(n)} = k \mid \mu\right) = \frac{|\{(S, S^*) : \Delta^{(n)}(T, T^*) = k\} \cap \mathcal{S}_\mu|}{|\mathcal{S}_\mu|}$$

# Numerical Evidence for the Locality of the Prüfer Code

- Numerical experiments for trees with a vertex size as large as  $n = 100$  conducted by Evan Thompson led him to conjecture that:

# Numerical Evidence for the Locality of the Prüfer Code

- Numerical experiments for trees with a vertex size as large as  $n = 100$  conducted by Evan Thompson led him to conjecture that:



$$\lim_{n \rightarrow \infty} \mathbb{P}(\Delta^{(n)} = 1) = \frac{1}{3},$$

# Numerical Evidence for the Locality of the Prüfer Code

- Numerical experiments for trees with a vertex size as large as  $n = 100$  conducted by Evan Thompson led him to conjecture that:

- 

$$\lim_{n \rightarrow \infty} \mathbf{P} \left( \Delta^{(n)} = 1 \right) = \frac{1}{3},$$

- and further, that if  $\mu/n \rightarrow \alpha$ , then

$$\lim_{n \rightarrow \infty} \mathbf{P} \left( \Delta^{(n)} = 1 \mid \mu \right) = (1 - \alpha)^2.$$

# Numerical Evidence for the Locality of the Prüfer Code

- Numerical experiments for trees with a vertex size as large as  $n = 100$  conducted by Evan Thompson led him to conjecture that:

- 

$$\lim_{n \rightarrow \infty} \mathbf{P} \left( \Delta^{(n)} = 1 \right) = \frac{1}{3},$$

- and further, that if  $\mu/n \rightarrow \alpha$ , then

$$\lim_{n \rightarrow \infty} \mathbf{P} \left( \Delta^{(n)} = 1 \mid \mu \right) = (1 - \alpha)^2.$$

- Later numerical experimentation led Tim Paulden and David Smith to conjecture that

$$\lim_{n \rightarrow \infty} \mathbf{P} \left( \Delta^{(n)} = \ell \right) = O \left( n^{-1} \right), \quad (\forall \ell \geq 2).$$

# The Results

- It is true that

$$P(\Delta = 1 \mid \mu) = (1 - \mu/n)^2 + O\left(n^{-1/3} \ln^2 n\right).$$

# The Results

- It is true that

$$P(\Delta = 1 \mid \mu) = (1 - \mu/n)^2 + O\left(n^{-1/3} \ln^2 n\right).$$

- Since  $\int_0^1 (1 - \alpha)^2 d\alpha = 1/3$ , this estimate implies

$$P(\Delta = 1) = 1/3 + O\left(n^{-1/3} \ln^2 n\right).$$

# The Results

- It is true that

$$P(\Delta = 1 \mid \mu) = (1 - \mu/n)^2 + O\left(n^{-1/3} \ln^2 n\right).$$

- Since  $\int_0^1 (1 - \alpha)^2 d\alpha = 1/3$ , this estimate implies

$$P(\Delta = 1) = 1/3 + O\left(n^{-1/3} \ln^2 n\right).$$

- For all  $\ell \geq 2$ ,

$$P(\Delta = \ell) = O\left(n^{-1/3} \ln^2 n\right).$$

# A decoding algorithm

- The proof relies on a decoding algorithm.

# A decoding algorithm

- The proof relies on a decoding algorithm.
  - We begin with a P-string  $(p_1, \dots, p_{n-2})$  and a tree which consists only of the vertex  $n$ .

# A decoding algorithm

- The proof relies on a decoding algorithm.
  - We begin with a P-string  $(p_1, \dots, p_{n-2})$  and a tree which consists only of the vertex  $n$ .
  - We read the P-string from right to left.

# A decoding algorithm

- The proof relies on a decoding algorithm.
  - We begin with a P-string  $(p_1, \dots, p_{n-2})$  and a tree which consists only of the vertex  $n$ .
  - We read the P-string from right to left.
  - If  $p_j$  is not a vertex of the tree we add it.

# A decoding algorithm

- The proof relies on a decoding algorithm.
  - We begin with a P-string  $(p_1, \dots, p_{n-2})$  and a tree which consists only of the vertex  $n$ .
  - We read the P-string from right to left.
  - If  $p_j$  is not a vertex of the tree we add it.
  - If  $p_j$  is a vertex of the tree, we add the largest (number) vertex not in the tree.

# A decoding algorithm

- The proof relies on a decoding algorithm.
  - We begin with a P-string  $(p_1, \dots, p_{n-2})$  and a tree which consists only of the vertex  $n$ .
  - We read the P-string from right to left.
  - If  $p_j$  is not a vertex of the tree we add it.
  - If  $p_j$  is a vertex of the tree, we add the largest (number) vertex not in the tree.
  - In either case we join the new vertex by an edge to  $p_{j+1}$  (where  $p_{n-1} := n$ ).

# A decoding algorithm

- The proof relies on a decoding algorithm.
  - We begin with a P-string  $(p_1, \dots, p_{n-2})$  and a tree which consists only of the vertex  $n$ .
  - We read the P-string from right to left.
  - If  $p_j$  is not a vertex of the tree we add it.
  - If  $p_j$  is a vertex of the tree, we add the largest (number) vertex not in the tree.
  - In either case we join the new vertex by an edge to  $p_{j+1}$  (where  $p_{n-1} := n$ ).
  - After placing the vertex corresponding to the reading of  $p_1$  we have one vertex remaining, which we join by an edge to  $p_1$ .

# An example of the decoding process

- We begin at step  $n - 1 = 8$  with:
  - the P-string  $(3, 1, 9, 9, 9, 6, 6)$ ,
  - the tree  $T_8$  consisting of the vertex  $n = 9$ ,
  - the set  $X_8$  of vertices that are not in our tree.

# An example of the decoding process

- We begin at step  $n - 1 = 8$  with:
  - the P-string  $(3, 1, 9, 9, 9, 6, 6)$ ,
  - the tree  $T_8$  consisting of the vertex  $n = 9$ ,
  - the set  $X_8$  of vertices that are not in our tree.
- Step 7: since  $p_7 = 6$  is not a vertex in the tree we add it, joining it to vertex 9 to obtain  $T_7$

$$9 \text{ --- } 6$$

# An example of the decoding process

- We begin at step  $n - 1 = 8$  with:
  - the P-string  $(3, 1, 9, 9, 9, 6, 6)$ ,
  - the tree  $T_8$  consisting of the vertex  $n = 9$ ,
  - the set  $X_8$  of vertices that are not in our tree.
- Step 7: since  $p_7 = 6$  is not a vertex in the tree we add it, joining it to vertex 9 to obtain  $T_7$

$$9 \text{ --- } 6$$

- At the end of step 7, we have  $X_7 = \{1, 2, 3, 4, 5, 7, 8\}$

# An example of the decoding process

- At step 6 we read  $p_6: (3, 1, 9, 9, 9, 6, 6)$  and look at  $T_7$

$$9 \text{ --- } 6$$

$$X_7 = \{1, 2, 3, 4, 5, 7, 8\}$$

# An example of the decoding process

- At step 6 we read  $p_6: (3, 1, 9, 9, 9, 6, 6)$  and look at  $T_7$

$$9 \text{ --- } 6$$

$$X_7 = \{1, 2, 3, 4, 5, 7, 8\}$$

- Since  $p_6 = 6$  is a vertex in the tree, we add 8 (which is the highest vertex not in the tree), joining it to  $p_7 = 6$  to obtain  $T_6$ .

$$9 \text{ --- } 6 \text{ --- } 8$$

# An example of the decoding process

- At step 6 we read  $p_6: (3, 1, 9, 9, 9, 6, 6)$  and look at  $T_7$

$$9 \text{ --- } 6$$

$$X_7 = \{1, 2, 3, 4, 5, 7, 8\}$$

- Since  $p_6 = 6$  is a vertex in the tree, we add 8 (which is the highest vertex not in the tree), joining it to  $p_7 = 6$  to obtain  $T_6$ .

$$9 \text{ --- } 6 \text{ --- } 8$$

$$X_6 = \{1, 2, 3, 4, 5, 7\}$$

# An example of the decoding process

- At step 5 we read  $p_5$ :  $(3, 1, 9, 9, 9, 6, 6)$  and look at our tree  $T_6$

$$9 \text{ --- } 6 \text{ --- } 8$$

$$X_6 = \{1, 2, 3, 4, 5, 7\}$$

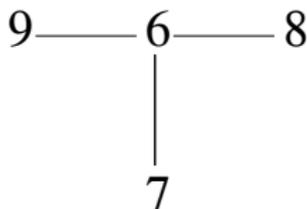
# An example of the decoding process

- At step 5 we read  $p_5$ :  $(3, 1, 9, 9, 9, 6, 6)$  and look at our tree  $T_6$



$$X_6 = \{1, 2, 3, 4, 5, 7\}$$

- Since  $p_5 = 9$  is a vertex in the tree, we add 7, joining it to  $p_6 = 6$  to get  $T_5$



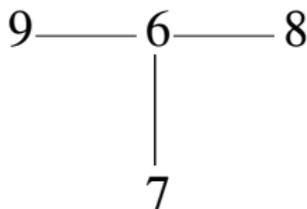
# An example of the decoding process

- At step 5 we read  $p_5$ :  $(3, 1, 9, 9, 9, 6, 6)$  and look at our tree  $T_6$



$$X_6 = \{1, 2, 3, 4, 5, 7\}$$

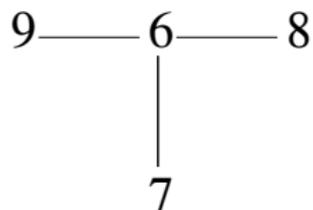
- Since  $p_5 = 9$  is a vertex in the tree, we add 7, joining it to  $p_6 = 6$  to get  $T_5$



$$X_5 = \{1, 2, 3, 4, 5\}$$

# An example of the decoding process

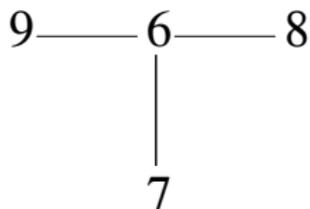
- At step 4 we read  $p_4$ : (3, 1, 9, 9, 9, 6, 6) and examine  $T_5$



$$X_5 = \{1, 2, 3, 4, 5\}$$

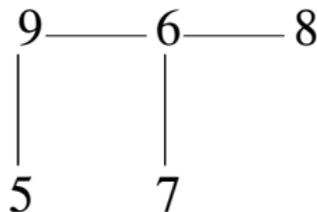
## An example of the decoding process

- At step 4 we read  $p_4$ : (3, 1, 9, **9**, 9, 6, 6) and examine  $T_5$



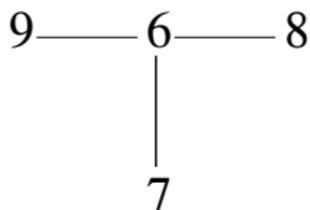
$$X_5 = \{1, 2, 3, 4, 5\}$$

- Since  $p_4 = 9$  is a vertex in the tree, we add 5, joining it to  $p_5 = 9$  to get  $T_4$



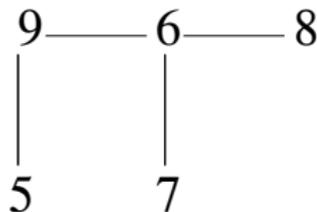
# An example of the decoding process

- At step 4 we read  $p_4$ : (3, 1, 9, 9, 9, 6, 6) and examine  $T_5$



$$X_5 = \{1, 2, 3, 4, 5\}$$

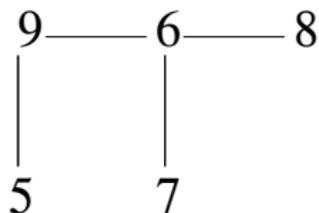
- Since  $p_4 = 9$  is a vertex in the tree, we add 5, joining it to  $p_5 = 9$  to get  $T_4$



$$X_4 = \{1, 2, 3, 4\}$$

# An example of the decoding process

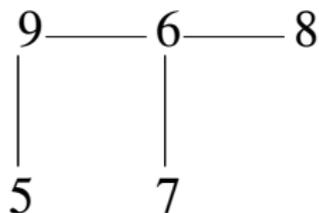
- At step 3 we read  $p_3: (3, 1, 9, 9, 9, 6, 6)$  and examine  $T_4$



$$X_4 = \{1, 2, 3, 4\}$$

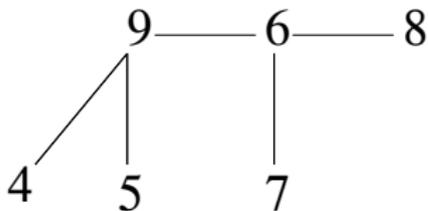
# An example of the decoding process

- At step 3 we read  $p_3$ : (3, 1, 9, 9, 9, 6, 6) and examine  $T_4$



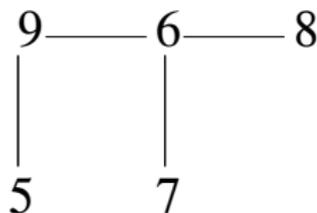
$$X_4 = \{1, 2, 3, 4\}$$

- Since  $p_3 = 9$  is a vertex in the tree, we add 4, joining it to  $p_4 = 9$  to obtain  $T_3$



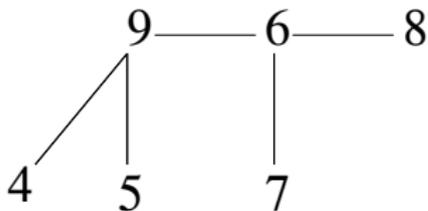
# An example of the decoding process

- At step 3 we read  $p_3$ : (3, 1, 9, 9, 9, 6, 6) and examine  $T_4$



$$X_4 = \{1, 2, 3, 4\}$$

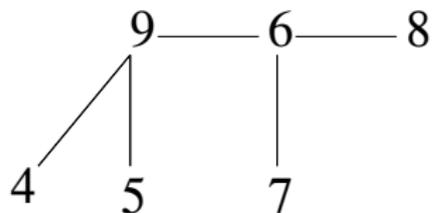
- Since  $p_3 = 9$  is a vertex in the tree, we add 4, joining it to  $p_4 = 9$  to obtain  $T_3$



$$X_3 = \{1, 2, 3\}$$

# An example of the decoding process

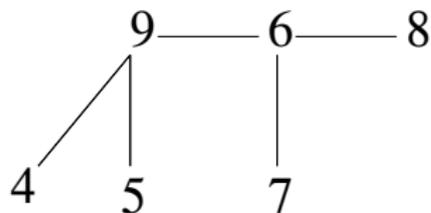
- At step 2 we read  $p_2: (3, 1, 9, 9, 9, 6, 6)$  and examine  $T_3$



$$X_3 = \{1, 2, 3\}$$

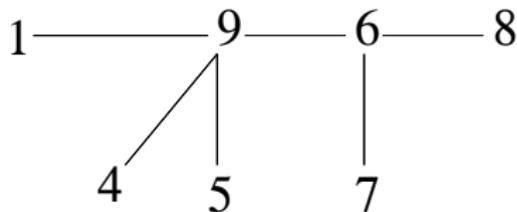
# An example of the decoding process

- At step 2 we read  $p_2$ : (3, 1, 9, 9, 9, 6, 6) and examine  $T_3$



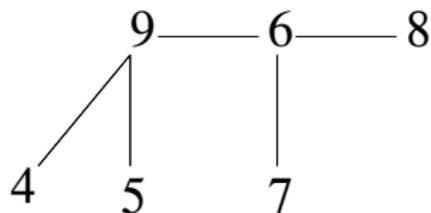
$$X_3 = \{1, 2, 3\}$$

- Since  $p_2 = 1$  is not a vertex in the tree, we add 1, joining it to  $p_3 = 9$  to obtain  $T_2$



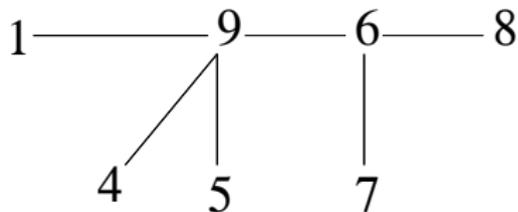
# An example of the decoding process

- At step 2 we read  $p_2$ : (3, 1, 9, 9, 9, 6, 6) and examine  $T_3$



$$X_3 = \{1, 2, 3\}$$

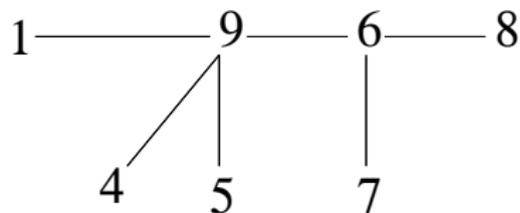
- Since  $p_2 = 1$  is not a vertex in the tree, we add 1, joining it to  $p_3 = 9$  to obtain  $T_2$



$$X_2 = \{2, 3\}$$

# An example of the decoding process

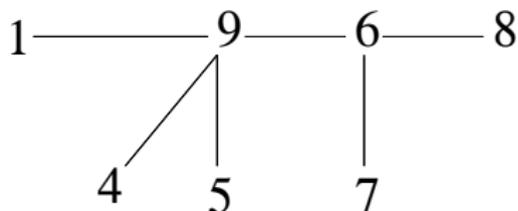
- At step 1 we read  $p_1: (3, 1, 9, 9, 9, 6, 6)$  and examine  $T_2$



$$X_2 = \{2, 3\}$$

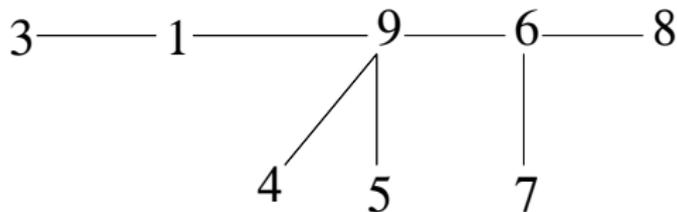
# An example of the decoding process

- At step 1 we read  $p_1: (3, 1, 9, 9, 9, 6, 6)$  and examine  $T_2$



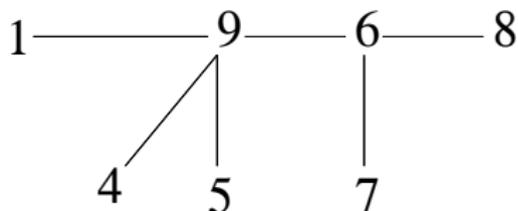
$$X_2 = \{2, 3\}$$

- Since  $p_1 = 3$  is not a vertex in the tree, we add 3, joining it to  $p_2 = 1$  to obtain  $T_1$



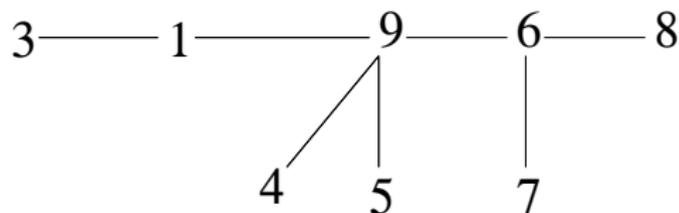
# An example of the decoding process

- At step 1 we read  $p_1: (3, 1, 9, 9, 9, 6, 6)$  and examine  $T_2$



$$X_2 = \{2, 3\}$$

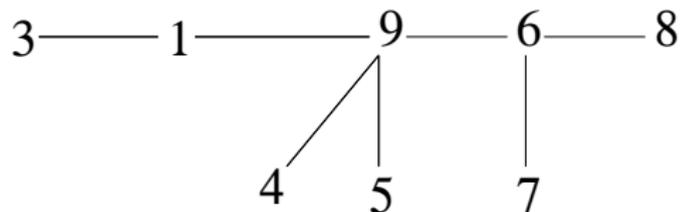
- Since  $p_1 = 3$  is not a vertex in the tree, we add 3, joining it to  $p_2 = 1$  to obtain  $T_1$



$$X_1 = \{2\}$$

# An example of the decoding process

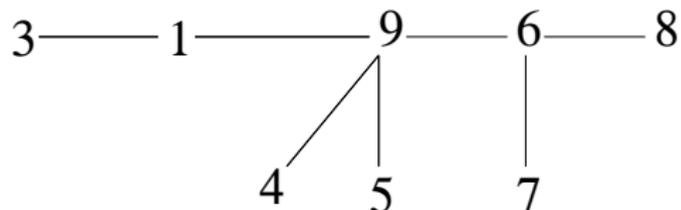
- At step 0 we add the last remaining vertex to  $T_1$



$$X_2 = \{2\}$$

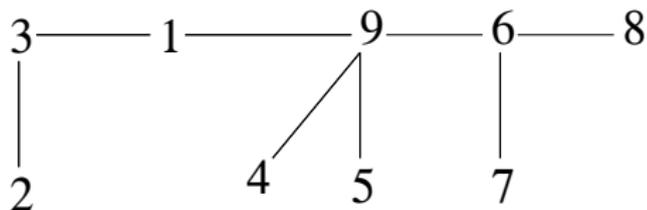
# An example of the decoding process

- At step 0 we add the last remaining vertex to  $T_1$



$$X_2 = \{2\}$$

- We join it to  $p_1 = 3$  to obtain  $T_0 = T$



## Benefits of the algorithm

- At each step  $j$ , the algorithm constructs a subtree  $T_j$  which depends only on  $p_j, \dots, p_{n-2}$ .

# Benefits of the algorithm

- At each step  $j$ , the algorithm constructs a subtree  $T_j$  which depends only on  $p_j, \dots, p_{n-2}$ .
- This lets us apply the principle of deferred decisions: we can let  $p_1, \dots, p_{j-1}$  remain unknown through step  $j$ .

## Benefits of the algorithm

- At each step  $j$ , the algorithm constructs a subtree  $T_j$  which depends only on  $p_j, \dots, p_{n-2}$ .
- This lets us apply the principle of deferred decisions: we can let  $p_1, \dots, p_{j-1}$  remain unknown through step  $j$ .
- We condition on the position of the mutation  $\mu$ , and consider two (random) elements of the set  $\mathcal{S}_\mu$

$$P = (p_1, \dots, p_{\mu-1}, p_\mu, p_{\mu+1}, \dots, p_{n-2})$$

$$P^* = (p_1, \dots, p_{\mu-1}, p_\mu^*, p_{\mu+1}, \dots, p_{n-2})$$

## Benefits of the algorithm

- At each step  $j$ , the algorithm constructs a subtree  $T_j$  which depends only on  $p_j, \dots, p_{n-2}$ .
- This lets us apply the principle of deferred decisions: we can let  $p_1, \dots, p_{j-1}$  remain unknown through step  $j$ .
- We condition on the position of the mutation  $\mu$ , and consider two (random) elements of the set  $\mathcal{S}_\mu$

$$P = (p_1, \dots, p_{\mu-1}, p_\mu, p_{\mu+1}, \dots, p_{n-2})$$

$$P^* = (p_1, \dots, p_{\mu-1}, p_\mu^*, p_{\mu+1}, \dots, p_{n-2})$$

- The first  $\mu - 1$  entries  $p_1, \dots, p_{\mu-1}$  remain unknown.

## Benefits of the algorithm

- At each step  $j$ , the algorithm constructs a subtree  $T_j$  which depends only on  $p_j, \dots, p_{n-2}$ .
- This lets us apply the principle of deferred decisions: we can let  $p_1, \dots, p_{j-1}$  remain unknown through step  $j$ .
- We condition on the position of the mutation  $\mu$ , and consider two (random) elements of the set  $\mathcal{S}_\mu$

$$P = (p_1, \dots, p_{\mu-1}, p_\mu, p_{\mu+1}, \dots, p_{n-2})$$

$$P^* = (p_1, \dots, p_{\mu-1}, p_\mu^*, p_{\mu+1}, \dots, p_{n-2})$$

- The first  $\mu - 1$  entries  $p_1, \dots, p_{\mu-1}$  remain unknown.
- The last  $n - \mu - 1$  entries  $p_{\mu+1}, \dots, p_{n-2}$  are known and the same.

## Benefits of the algorithm

- At each step  $j$ , the algorithm constructs a subtree  $T_j$  which depends only on  $p_j, \dots, p_{n-2}$ .
- This lets us apply the principle of deferred decisions: we can let  $p_1, \dots, p_{j-1}$  remain unknown through step  $j$ .
- We condition on the position of the mutation  $\mu$ , and consider two (random) elements of the set  $\mathcal{S}_\mu$

$$P = (p_1, \dots, p_{\mu-1}, p_\mu, p_{\mu+1}, \dots, p_{n-2})$$

$$P^* = (p_1, \dots, p_{\mu-1}, p_\mu^*, p_{\mu+1}, \dots, p_{n-2})$$

- The first  $\mu - 1$  entries  $p_1, \dots, p_{\mu-1}$  remain unknown.
- The last  $n - \mu - 1$  entries  $p_{\mu+1}, \dots, p_{n-2}$  are known and the same.
- What happens when we choose  $p_\mu, p_\mu^*$ ?

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )
- This event, which we denote by  $(P, P^*) \in \mathcal{E}$ , implies  $\Delta(T, T^*) = 1$ .

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )
- This event, which we denote by  $(P, P^*) \in \mathcal{E}$ , implies  $\Delta(T, T^*) = 1$ .
- Why?

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )
- This event, which we denote by  $(P, P^*) \in \mathcal{E}$ , implies  $\Delta(T, T^*) = 1$ .
- Why?
  - At step  $\mu$  we add the same vertex and edge to both trees  $T_{\mu+1}, T_{\mu+1}^*$ .

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )
- This event, which we denote by  $(P, P^*) \in \mathcal{E}$ , implies  $\Delta(T, T^*) = 1$ .
- Why?
  - At step  $\mu$  we add the same vertex and edge to both trees  $T_{\mu+1}, T_{\mu+1}^*$ .
  - At step  $\mu - 1$  we add the same vertex but a different edge because  $p_\mu \neq p_\mu^*$ .

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )
- This event, which we denote by  $(P, P^*) \in \mathcal{E}$ , implies  $\Delta(T, T^*) = 1$ .
- Why?
  - At step  $\mu$  we add the same vertex and edge to both trees  $T_{\mu+1}, T_{\mu+1}^*$ .
  - At step  $\mu - 1$  we add the same vertex but a different edge because  $p_\mu \neq p_\mu^*$ .
  - At every subsequent step we add the same vertex and edge.

# The key event

- The key event is the event that both  $p_\mu, p_\mu^*$  are in  $V_{\mu+1} \cup \{\max X_{\mu+1}\}$  (where  $V_j$  is the vertex set of  $T_j$ )
- This event, which we denote by  $(P, P^*) \in \mathcal{E}$ , implies  $\Delta(T, T^*) = 1$ .
- Why?
  - At step  $\mu$  we add the same vertex and edge to both trees  $T_{\mu+1}, T_{\mu+1}^*$ .
  - At step  $\mu - 1$  we add the same vertex but a different edge because  $p_\mu \neq p_\mu^*$ .
  - At every subsequent step we add the same vertex and edge.
- This is not the only way you can have  $\Delta(T, T^*) = 1$ .

# Examining the key event

- The key event is  $\mathcal{E} = \{p_\mu, p_\mu^* \in V_{\mu+1} \cup \{\max X_{\mu+1}\}\}$

# Examining the key event

- The key event is  $\mathcal{E} = \{p_\mu, p_\mu^* \in V_{\mu+1} \cup \{\max X_{\mu+1}\}\}$

$$P(\Delta = 1 \mid \mu) \geq P(\mathcal{E} \mid \mu)$$

# Examining the key event

- The key event is  $\mathcal{E} = \{p_\mu, p_\mu^* \in V_{\mu+1} \cup \{\max X_{\mu+1}\}\}$

$$\begin{aligned} P(\Delta = 1 \mid \mu) &\geq P(\mathcal{E} \mid \mu) \\ &\geq \frac{n - \mu}{n} \frac{n - \mu - 1}{n - 1} \end{aligned}$$

# Examining the key event

- The key event is  $\mathcal{E} = \{p_\mu, p_\mu^* \in V_{\mu+1} \cup \{\max X_{\mu+1}\}\}$

$$\begin{aligned} \mathbb{P}(\Delta = 1 \mid \mu) &\geq \mathbb{P}(\mathcal{E} \mid \mu) \\ &\geq \frac{n - \mu}{n} \frac{n - \mu - 1}{n - 1} \\ &= (1 - \mu/n)^2 + O(n^{-1}) \end{aligned}$$

# Examining the key event

- The key event is  $\mathcal{E} = \{p_\mu, p_\mu^* \in V_{\mu+1} \cup \{\max X_{\mu+1}\}\}$

$$\begin{aligned}
 \mathbb{P}(\Delta = 1 \mid \mu) &\geq \mathbb{P}(\mathcal{E} \mid \mu) \\
 &\geq \frac{n - \mu}{n} \frac{n - \mu - 1}{n - 1} \\
 &= (1 - \mu/n)^2 + O(n^{-1})
 \end{aligned}$$

- So we obtain our lower bound.

# Examining the event $\mathcal{E}^c$

- For an upper bound, we want to show that

$$\mathbb{P}(\{\Delta = \ell\} \cap \mathcal{E}^c | \mu) = O\left(n^{-1/3} \ln^2 n\right) \quad (\ell \geq 1).$$

# Examining the event $\mathcal{E}^c$

- For an upper bound, we want to show that

$$P(\{\Delta = \ell\} \cap \mathcal{E}^c | \mu) = O\left(n^{-1/3} \ln^2 n\right) \quad (\ell \geq 1).$$

- Consider all the vertices that need to be added to  $T_\mu, T_\mu^*$

$$\mathcal{X}_\mu = \{\text{vertices} < z_\mu < \text{vertices} < z_\mu^* < \text{more vertices}\}$$

- $z_\mu \in T_\mu$  but not  $T_\mu^*$
- $z_\mu^* \in T_\mu^*$  but not  $T_\mu$ .

# Examining the event $\mathcal{E}^c$

- For an upper bound, we want to show that

$$P(\{\Delta = \ell\} \cap \mathcal{E}^c | \mu) = O\left(n^{-1/3} \ln^2 n\right) \quad (\ell \geq 1).$$

- Consider all the vertices that need to be added to  $T_\mu, T_\mu^*$

$$\mathcal{X}_\mu = \{\text{vertices} < z_\mu < \text{vertices} < z_\mu^* < \text{more vertices}\}$$

- $z_\mu \in T_\mu$  but not  $T_\mu^*$
- $z_\mu^* \in T_\mu^*$  but not  $T_\mu$ .
- We choose entries  $p_j$  ( $j < \mu$ ) and see what happens.

## After choosing some vertices

- When an entry  $p_j$  is already in  $T_{j+1}$ , we add the largest vertex of  $X_{j+1}$ .

## After choosing some vertices

- When an entry  $p_j$  is already in  $T_{j+1}$ , we add the largest vertex of  $X_{j+1}$ .
- So it seems likely that we should at some step  $\tau$  have a set

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\}$$

- $z_\tau \in T_\tau$  but not  $T_\tau^*$
- $z_\tau^* \in T_\tau^*$  but not  $T_\tau$ .

# What happens after step $\tau$ ?

- Let us consider

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\}$$

- $z_\tau \in T_\tau$  but not  $T_\tau^*$ , and  $z_\tau^* \in T_\tau^*$  but not  $T_\tau$ .

# What happens after step $\tau$ ?

- Let us consider

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\}$$

- $z_\tau \in T_\tau$  but not  $T_\tau^*$ , and  $z_\tau^* \in T_\tau^*$  but not  $T_\tau$ .
- Now what happens if  $p_{\tau-1}$  is in both  $T_{\tau-1}$ ,  $T_{\tau-1}^*$ ?

# What happens after step $\tau$ ?

- Let us consider

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\}$$

- $z_\tau \in T_\tau$  but not  $T_\tau^*$ , and  $z_\tau^* \in T_\tau^*$  but not  $T_\tau$ .
- Now what happens if  $p_{\tau-1}$  is in both  $T_{\tau-1}, T_{\tau-1}^*$ ?
- We add  $\{z_\tau^*, p_\tau\}$  to  $T_{\tau-1}$ , and we add  $\{v, p_\tau\}$  to  $T_{\tau-1}^*$

# What happens after step $\tau$ ?

- Let us consider

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\}$$

- $z_\tau \in T_\tau$  but not  $T_\tau^*$ , and  $z_\tau^* \in T_\tau^*$  but not  $T_\tau$ .
- Now what happens if  $p_{\tau-1}$  is in both  $T_{\tau-1}, T_{\tau-1}^*$ ?
- We add  $\{z_\tau^*, p_\tau\}$  to  $T_{\tau-1}$ , and we add  $\{v, p_\tau\}$  to  $T_{\tau-1}^*$
- $\Delta$  will increase by one unless  $p_\tau$  was the neighbor of  $z_\tau^*$  when it was first added to  $T_{\tau-1}^*$ .

# What happens after step $\tau$ ?

- Let us consider

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\}$$

- $z_\tau \in T_\tau$  but not  $T_\tau^*$ , and  $z_\tau^* \in T_\tau^*$  but not  $T_\tau$ .
- Now what happens if  $p_{\tau-1}$  is in both  $T_{\tau-1}$ ,  $T_{\tau-1}^*$ ?
- We add  $\{z_\tau^*, p_\tau\}$  to  $T_{\tau-1}$ , and we add  $\{v, p_\tau\}$  to  $T_{\tau-1}^*$
- $\Delta$  will increase by one unless  $p_\tau$  was the neighbor of  $z_\tau^*$  when it was first added to  $T_{\tau-1}^*$ .
- This is an unlikely event, so we (probably) add different edges.

# The key to the proof

- The key to the proof is to show that we reach a point  $\tau$  where we have

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\},$$

# The key to the proof

- The key to the proof is to show that we reach a point  $\tau$  where we have

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\},$$

- and that there are “enough” vertices between  $z_\tau$  and  $z_\tau^*$ .

# The key to the proof

- The key to the proof is to show that we reach a point  $\tau$  where we have

$$\mathcal{X}_\tau = \{\text{vertices} < z_\tau < \text{vertices} < z_\tau^*\},$$

- and that there are “enough” vertices between  $z_\tau$  and  $z_\tau^*$ .
- Then for any fixed number  $\ell$ , it is likely that we choose many more than  $\ell$  values of  $p_j \in T_{j+1}, T_{j+1}$ , each time adding a different edge to the two trees.

# Conclusion

- It is true that

$$P(\Delta = 1 \mid \mu) = (1 - \mu/n)^2 + O\left(n^{-1/3} \ln^2 n\right).$$

# Conclusion

- It is true that

$$P(\Delta = 1 \mid \mu) = (1 - \mu/n)^2 + O\left(n^{-1/3} \ln^2 n\right).$$

- For all  $\ell \geq 2$ ,

$$P(\Delta = \ell) = O\left(n^{-1/3} \ln^2 n\right).$$

# Conclusion

- It is true that

$$P(\Delta = 1 \mid \mu) = (1 - \mu/n)^2 + O\left(n^{-1/3} \ln^2 n\right).$$

- For all  $\ell \geq 2$ ,

$$P(\Delta = \ell) = O\left(n^{-1/3} \ln^2 n\right).$$

- Questions?
- Thanks to Ohio State
- craig.lennon@usma.edu