Genetic Algorithms for Syntactic Parsing

Sergio Penkale

July 24^{th}, 2008

What is a Genetic Algorithm?

- Heuristic search algorithm
- Based on analogy to biological evolution

1. We randomly build an *Initial Population*
2. While finalisation criteria doesn't hold:
   1. A percentage of individuals is selected
   2. Individuals are recombined
   3. mutations are introduced
3. the individual with the greatest fitness is returned

What is a Genetic Algorithm?

- Heuristic search algorithm
- Based on analogy to biological evolution

1. We randomly build an *Initial Population*
2. While finalisation criteria doesn't hold:
   1. A percentage of individuals is selected
   2. Individuals are recombined
   3. mutations are introduced
3. the individual with the greatest fitness is returned

Obtaining the initial population

- We will use a traditional parser and ask it to return the best k trees according to its model
- This way we will obtain a set of possible parses of great quality

Mutation Operation (Mutations 1 and 2)

- It models a *reattachment*

Example: Original

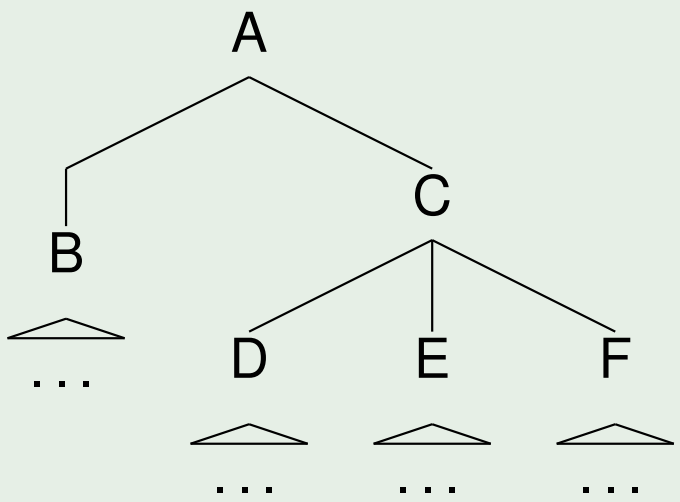
Example: Mutation

- The inverse operation is also allowed

Mutation Operation (Mutations 1 and 2)

- It models a *reattachment*

Example: Original



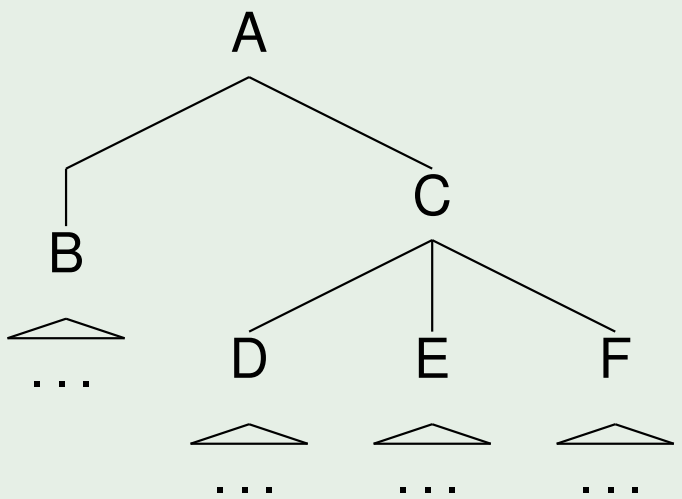
Example: Mutation

- The inverse operation is also allowed

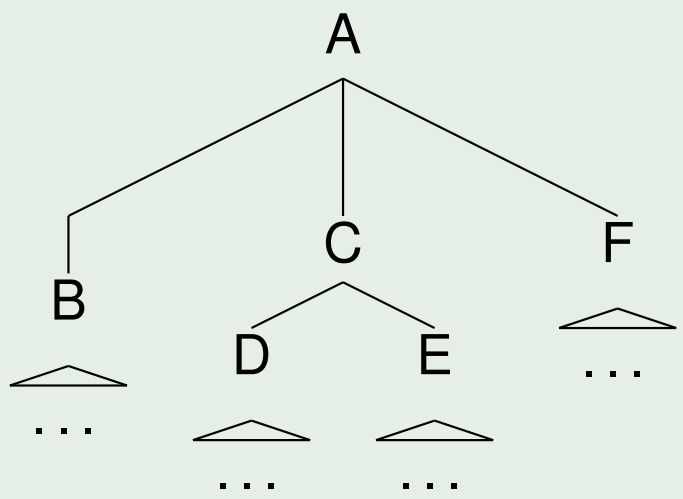
Mutation Operation (Mutations 1 and 2)

- It models a *reattachment*

Example: Original



Example: Mutation

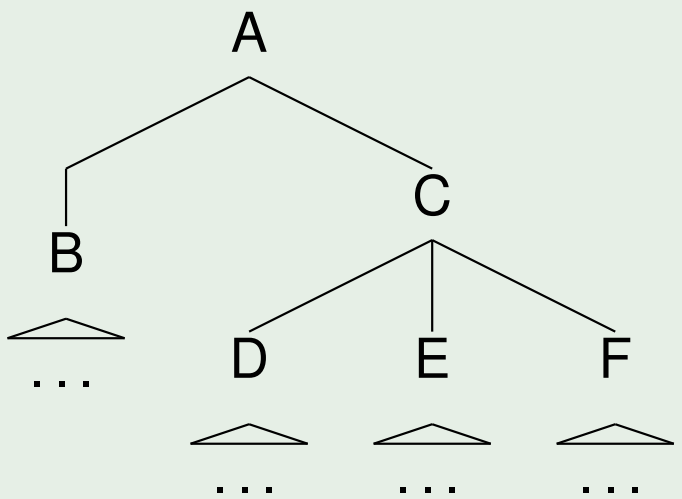


- The inverse operation is also allowed

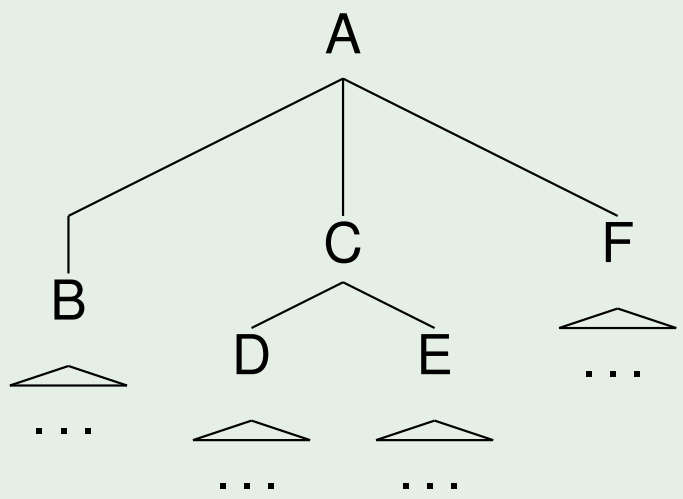
Mutation Operation (Mutations 1 and 2)

- It models a *reattachment*

Example: Original



Example: Mutation



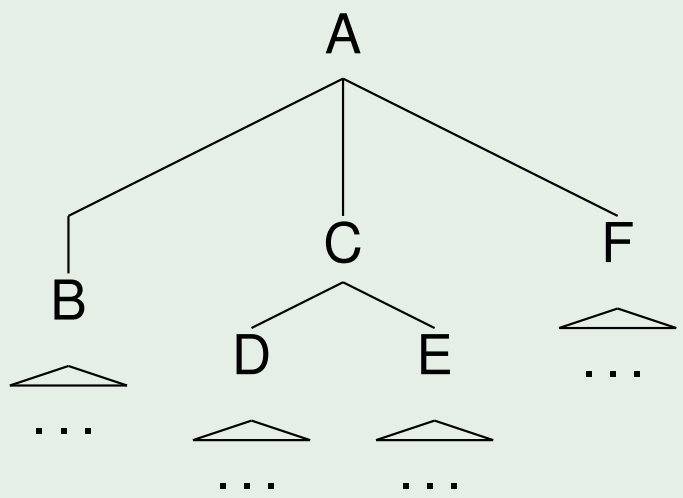
- The inverse operation is also allowed

Mutation Operation (Mutations 1 and 2)

- It models a *reattachment*

Example: Original

Example: Mutation

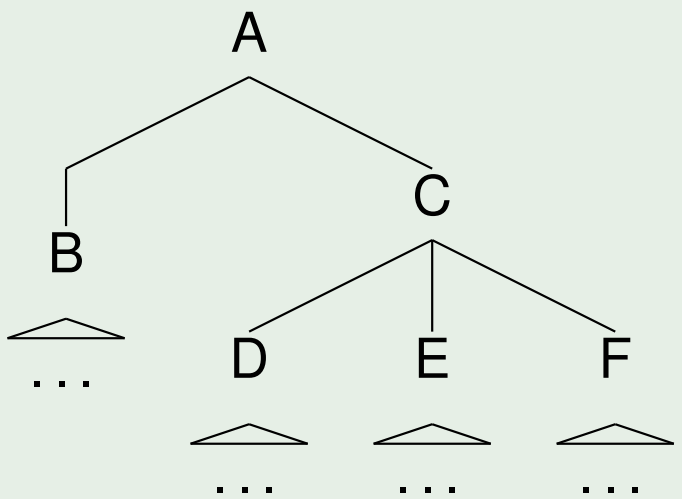


- The inverse operation is also allowed

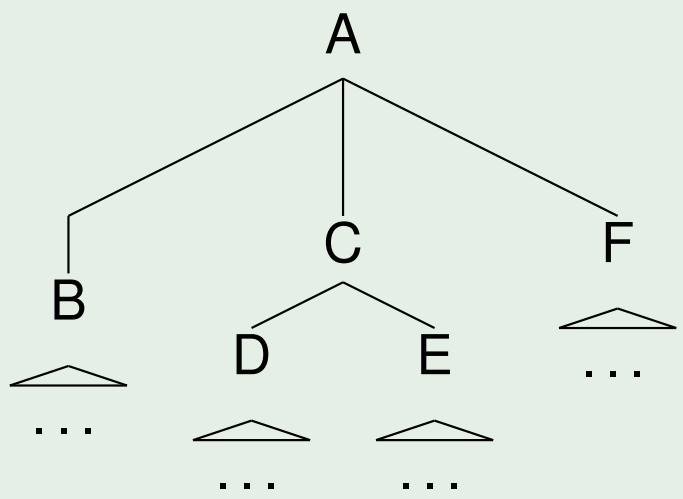
Mutation Operation (Mutations 1 and 2)

- It models a *reattachment*

Example: Original



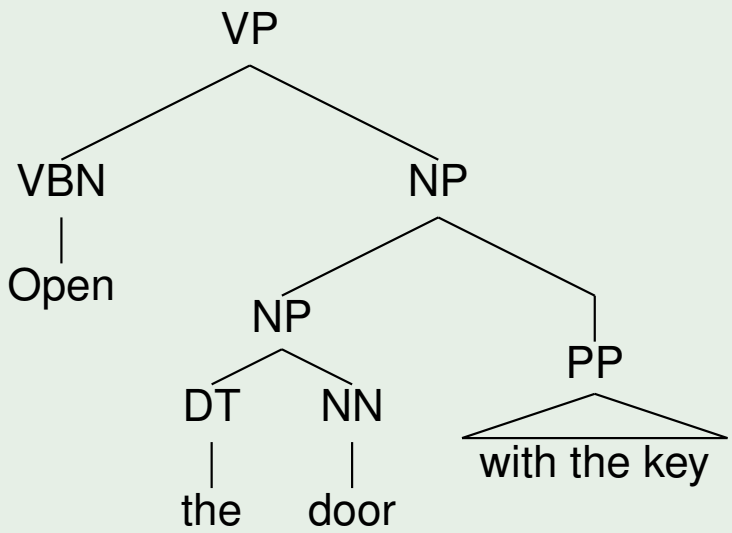
Example: Mutation



- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

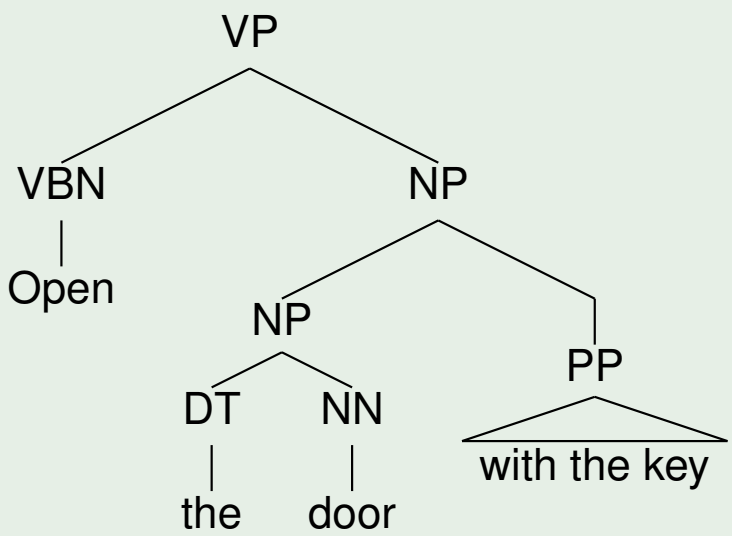
Original



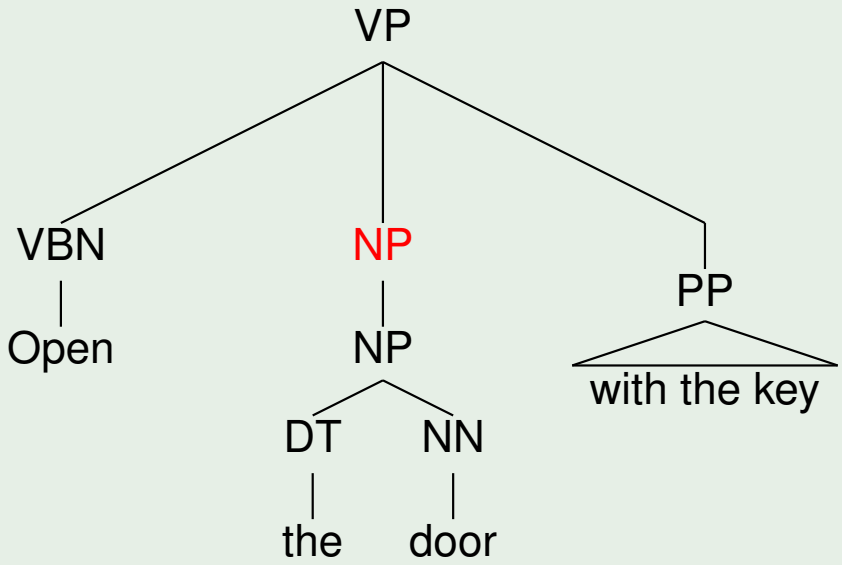
- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

Original



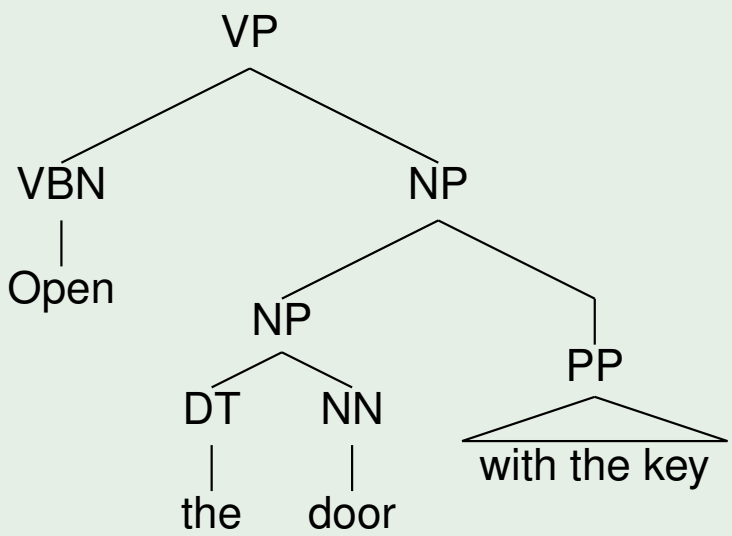
Mutation



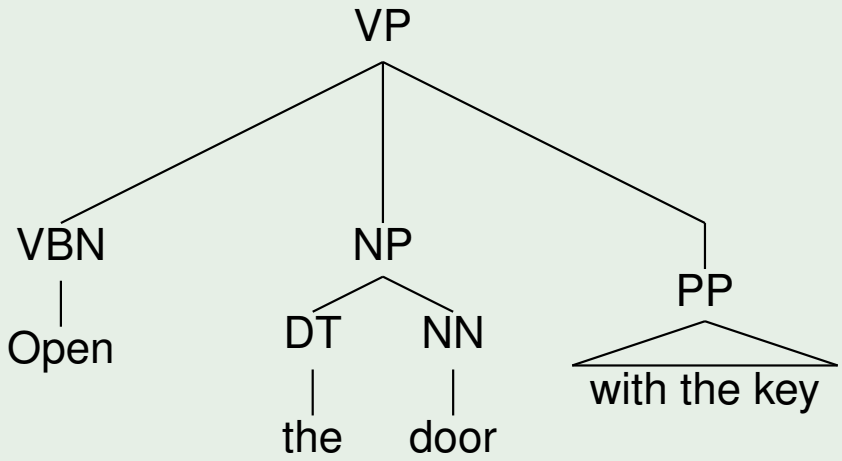
- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

Original



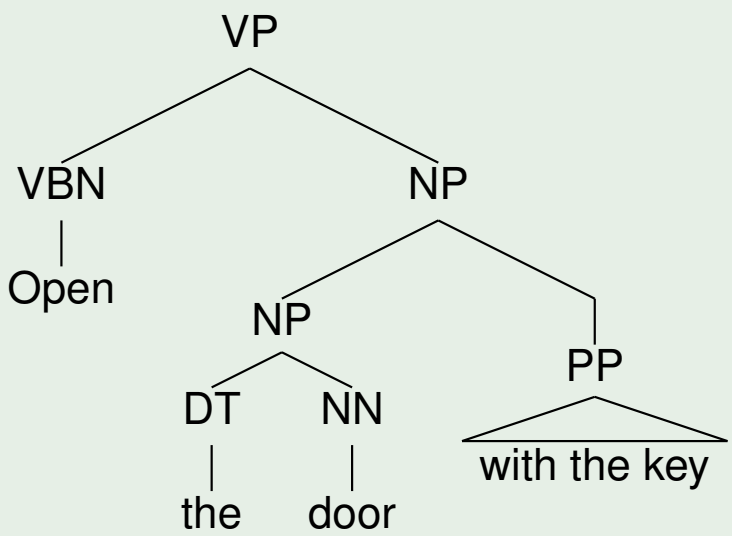
Mutation



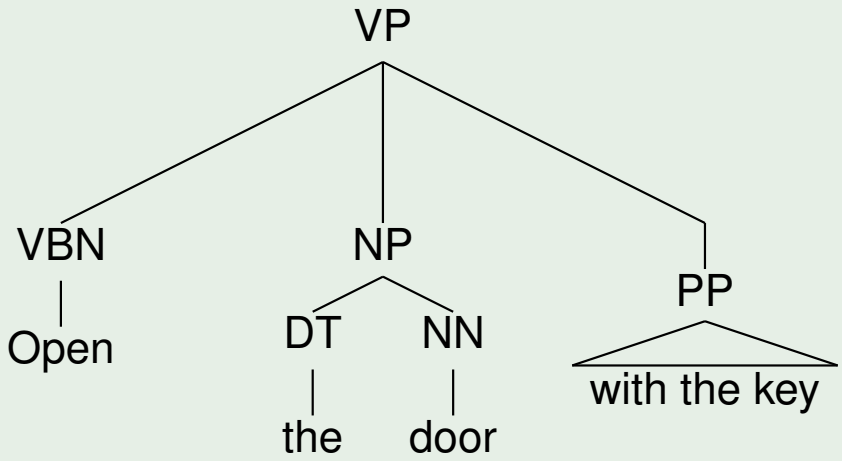
- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

Original



Mutation

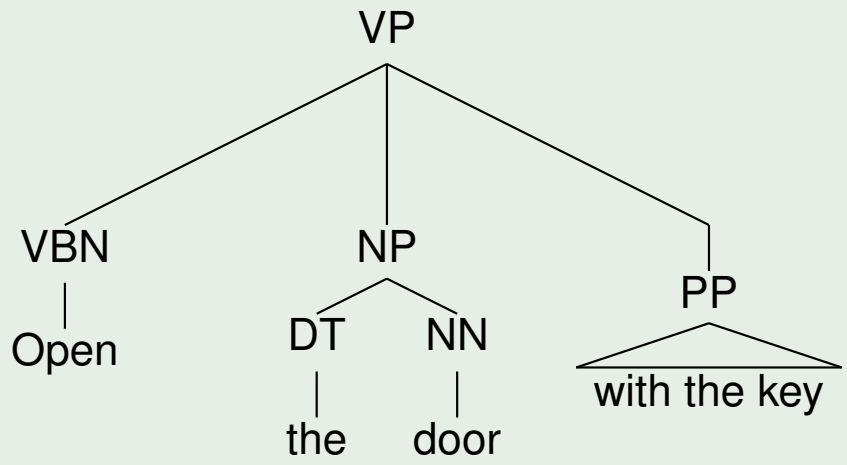


- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

Original

Mutation

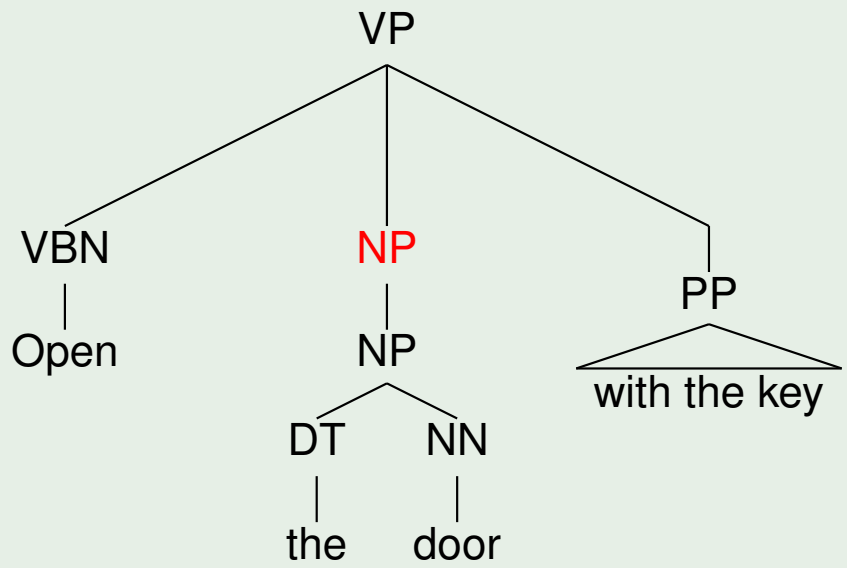


- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

Original

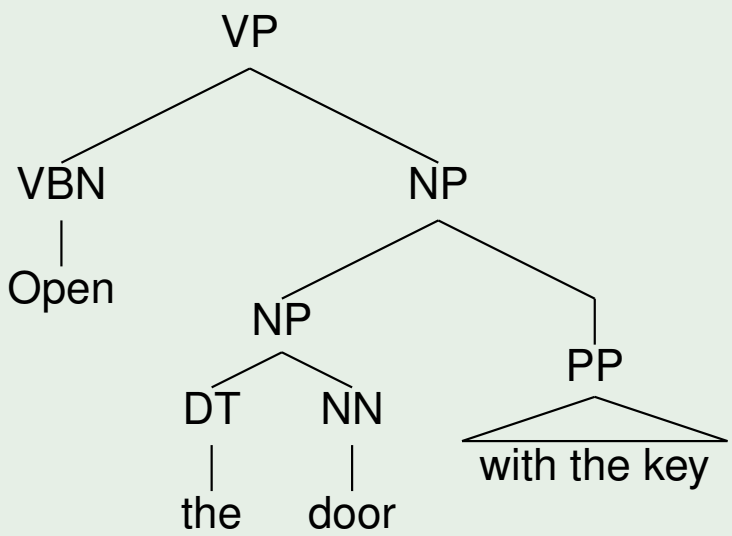
Mutation



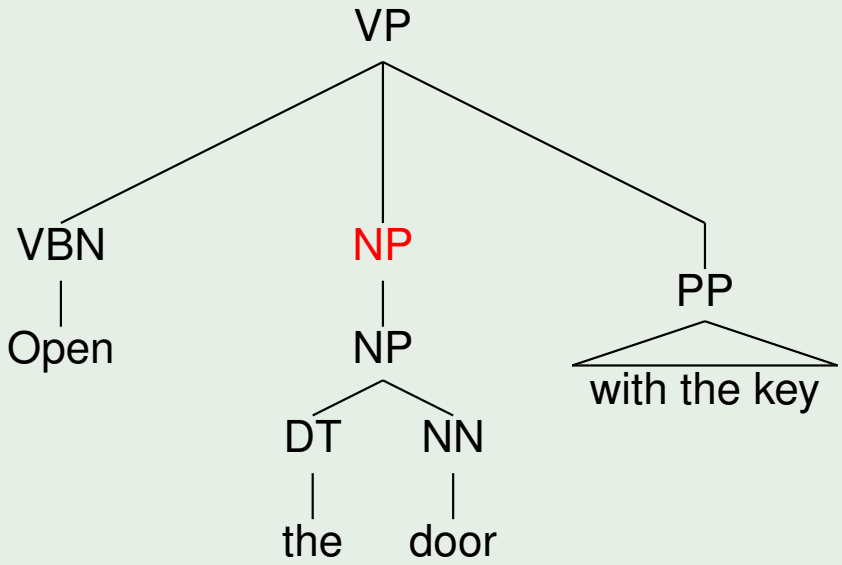
- The inverse operation is also allowed

Mutations 3/4: Eliminating/Creating nodes

Original



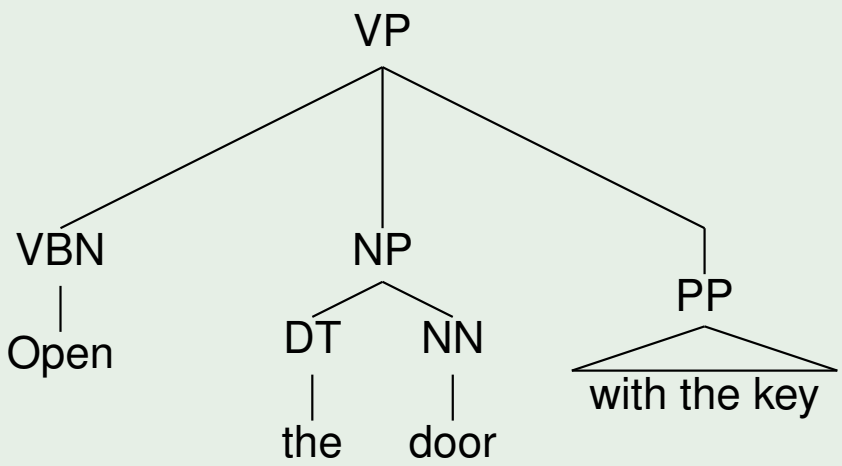
Mutation



- The inverse operation is also allowed

Mutation 5: Upwards reattach creating node

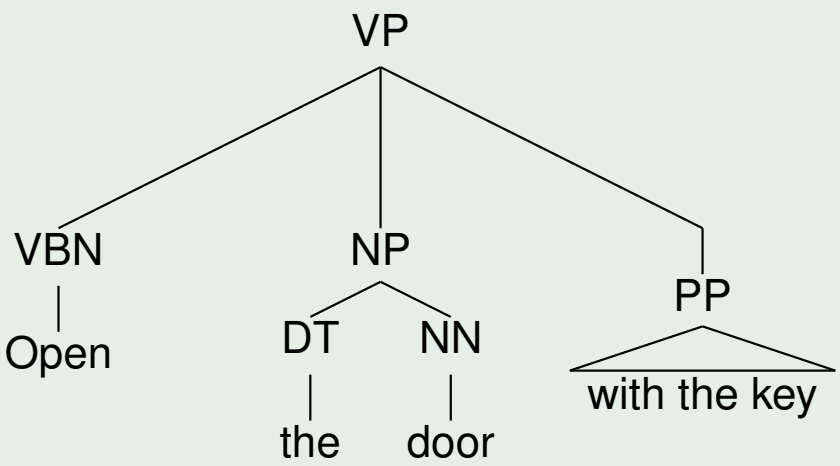
Original



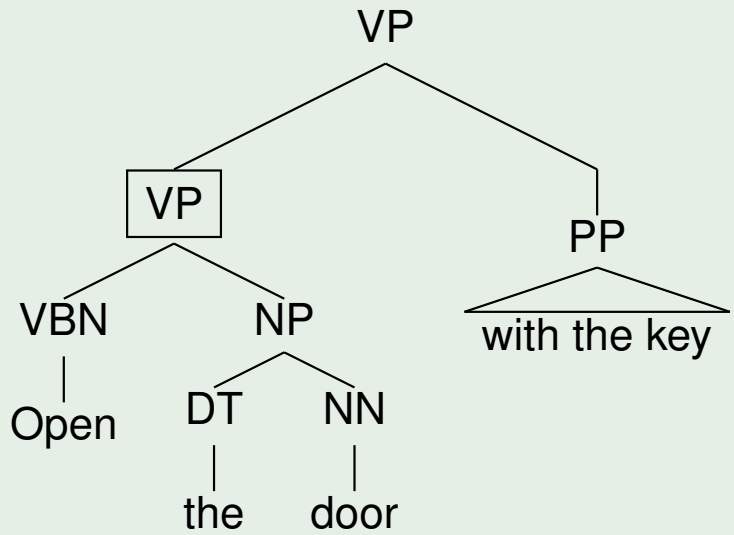
Mutation

Mutation 5: Upwards reattach creating node

Original



Mutation



The crossover operation

- It takes two “parents” syntactic trees
- Subfragments are recombined
- It returns two “child” syntactic trees

The crossover operation

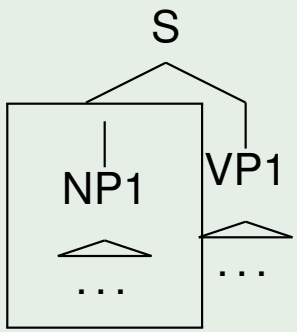
- It takes two “parents” syntactic trees
- Subfragments are recombined
- It returns two “child” syntactic trees

The crossover operation

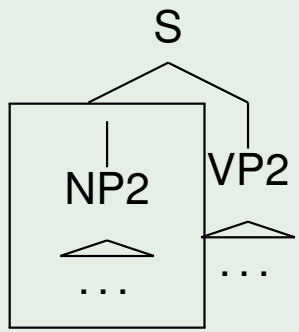
- It takes two “parents” syntactic trees
- Subfragments are recombined
- It returns two “child” syntactic trees

Crossover: example

Parent 1



Parent 2

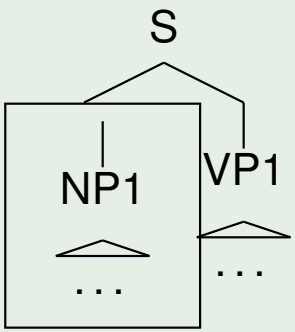


Child 1

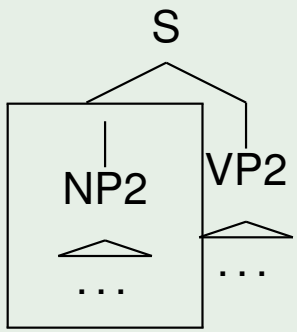
Child 2

Crossover: example

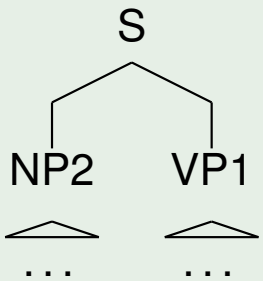
Parent 1



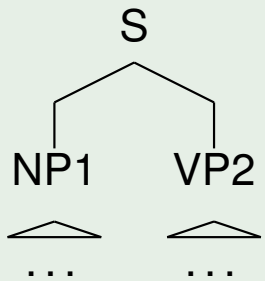
Parent 2



Child 1



Child 2



The Fitness Function

- Quantifies how well an individual adapts to its environment
- Determines the probability for selecting individuals
- Determines which features from the individuals the GA will try to maximise
- We will define a few fitness functions and will evaluate their influence in the behaviour of the algorithm

The Fitness Function

- Quantifies how well an individual adapts to its environment
- Determines the probability for selecting individuals
- Determines which features from the individuals the GA will try to maximise
- We will define a few fitness functions and will evaluate their influence in the behaviour of the algorithm

The Fitness Function

- Quantifies how well an individual adapts to its environment
- Determines the probability for selecting individuals
- Determines which features from the individuals the GA will try to maximise
- We will define a few fitness functions and will evaluate their influence in the behaviour of the algorithm

The Fitness Function

- Quantifies how well an individual adapts to its environment
- Determines the probability for selecting individuals
- Determines which features from the individuals the GA will try to maximise
- We will define a few fitness functions and will evaluate their influence in the behaviour of the algorithm

CheaterFitness

- We will define a fitness function that cheats: CheaterFitness
- It has access to the reference tree created by a human

$$\text{CheaterFitness} = \text{F-Measure}$$

- It maximises exactly the criteria we will use to evaluate
- It's purpose is to verify that the defined operators have sufficient expressive power

CheaterFitness

- We will define a fitness function that cheats: CheaterFitness
- It has access to the reference tree created by a human

$$\text{CheaterFitness} = \text{F-Measure}$$

- It maximises exactly the criteria we will use to evaluate
- It's purpose is to verify that the defined operators have sufficient expressive power

CheaterFitness

- We will define a fitness function that cheats: CheaterFitness
- It has access to the reference tree created by a human

$$\text{CheaterFitness} = \text{F-Measure}$$

- It maximises exactly the criteria we will use to evaluate
- It's purpose is to verify that the defined operators have sufficient expressive power

CheaterFitness

- We will define a fitness function that cheats: CheaterFitness
- It has access to the reference tree created by a human

$$\text{CheaterFitness} = \text{F-Measure}$$

- It maximises exactly the criteria we will use to evaluate
- It's purpose is to verify that the defined operators have sufficient expressive power

CheaterFitness

- We will define a fitness function that cheats: CheaterFitness
- It has access to the reference tree created by a human

$$\text{CheaterFitness} = \text{F-Measure}$$

- It maximises exactly the criteria we will use to evaluate
- It's purpose is to verify that the defined operators have sufficient expressive power

The ChunkFitness function

- We will define a fitness function that assigns high values to trees whose structures are “typical”
- This is determined by subtrees contained in a set `chunks`, built during training
- We add the lengths of the spans of subtrees contained in `chunks`
- We divide by the sum of the lengths of the spans of all subtrees

$$\text{ChunkFtiness}(\tau) = \frac{\sum_{\{\rho \in ST(\tau):\, \rho \in \text{chunks}\}} \text{len}(\rho)}{\sum_{\{\rho \in ST(\tau)\}} \text{len}(\rho)}$$

The ChunkFitness function

- We will define a fitness function that assigns high values to trees whose structures are “typical”
- This is determined by subtrees contained in a set `chunks`, built during training
- We add the lengths of the spans of subtrees contained in `chunks`
- We divide by the sum of the lengths of the spans of all subtrees

$$\text{ChunkFtiness}(\tau) = \frac{\sum_{\{\rho \in ST(\tau):\, \rho \in \text{chunks}\}} \text{len}(\rho)}{\sum_{\{\rho \in ST(\tau)\}} \text{len}(\rho)}$$

The ChunkFitness function

- We will define a fitness function that assigns high values to trees whose structures are “typical”
- This is determined by subtrees contained in a set `chunks`, built during training
- We add the lengths of the spans of subtrees contained in `chunks`
- We divide by the sum of the lengths of the spans of all subtrees

$$\text{ChunkFtiness}(\tau) = \frac{\sum_{\{\rho \in ST(\tau):\, \rho \in \text{chunks}\}} \text{len}(\rho)}{\sum_{\{\rho \in ST(\tau)\}} \text{len}(\rho)}$$

The ChunkFitness function

- We will define a fitness function that assigns high values to trees whose structures are “typical”
- This is determined by subtrees contained in a set `chunks`, built during training
- We add the lengths of the spans of subtrees contained in `chunks`
- We divide by the sum of the lengths of the spans of all subtrees

$$\text{ChunkFtiness}(\tau) = \frac{\sum_{\{\rho \in ST(\tau):\, \rho \in \text{chunks}\}} \text{len}(\rho)}{\sum_{\{\rho \in ST(\tau)\}} \text{len}(\rho)}$$

The ChunkFitness function

- We will define a fitness function that assigns high values to trees whose structures are “typical”
- This is determined by subtrees contained in a set `chunks`, built during training
- We add the lengths of the spans of subtrees contained in `chunks`
- We divide by the sum of the lengths of the spans of all subtrees

$$\text{ChunkFtiness}(\tau) = \frac{\sum_{\{\rho \in ST(\tau):\ \rho \in \text{chunks}\}} \text{len}(\rho)}{\sum_{\{\rho \in ST(\tau)\}} \text{len}(\rho)}$$

The PCFGFitness Function

- We obtain the PCFG that covers the treebank
- We smooth the probabilities of unseen rules with the function:

$$\text{unseen}(A) = \text{lowest probability among rules with } A \text{ on the left side}$$

Then

$$\text{PCFGFitness}(\tau) = 1 - \frac{\log_{10}(\prod_{\{r \in \text{rules}(\tau)\}} \Theta(r))}{\log_{10}(\prod_{\{r \in \text{rules}(\tau)\}} \text{unseen}(r))}$$

The PCFGFitness Function

- We obtain the PCFG that covers the treebank
- We smooth the probabilities of unseen rules with the function:

$\text{unseen}(A) =$ lowest probability among rules with A on the left side

Then

$$\text{PCFGFitness}(\tau) = 1 - \frac{\log_{10}(\prod_{\{r \in \text{rules}(\tau)\}} \Theta(r))}{\log_{10}(\prod_{\{r \in \text{rules}(\tau)\}} \text{unseen}(r))}$$

The PCFGFitness Function

- We obtain the PCFG that covers the treebank
- We smooth the probabilities of unseen rules with the function:

$\text{unseen}(A) =$ lowest probability among rules with A on the left side

Then

$$\text{PCFGFitness}(\tau) = 1 - \frac{\log_{10}(\prod_{\{r \in \text{rules}(\tau)\}} \Theta(r))}{\log_{10}(\prod_{\{r \in \text{rules}(\tau)\}} \text{unseen}(r))}$$

Grammatical Rules

- We defined a few grammatical rules we wish the tree satisfy
- They are not fitness functions. They modify values returned by fitness functions
- They don't use lexical information

Grammatical Rules

- We defined a few grammatical rules we wish the tree satisfy
- They are not fitness functions. They modify values returned by fitness functions
- They don't use lexical information

Grammatical Rules

- We defined a few grammatical rules we wish the tree satisfy
- They are not fitness functions. They modify values returned by fitness functions
- They don't use lexical information

Evaluation

We made several experiments to evaluate the behaviour of each fitness function

- We used the Wall Street Journal section of the Penn Treebank
- We used sections 02-21 for training and evaluated using section 22
- For initial populations we used Bikel's parser with settings set to emulate Collins '99

Evaluation

We made several experiments to evaluate the behaviour of each fitness function

- We used the Wall Street Journal section of the Penn Treebank
- We used sections 02-21 for training and evaluated using section 22
- For initial populations we used Bikel's parser with settings set to emulate Collins '99

Evaluation

We made several experiments to evaluate the behaviour of each fitness function

- We used the Wall Street Journal section of the Penn Treebank
- We used sections 02-21 for training and evaluated using section 22
- For initial populations we used Bikel's parser with settings set to emulate Collins '99

Results

Sentences with 1 to 35 words

	PREC	REC	F-Measure
CheaterFitness	99.51	99.02	99.19
PCFGFitness w/GR	74.73	83.77	78.63
PCFGFitness	74.36	83.82	78.43
ChunkFitness w/GR	70.97	67.03	68.43
ChunkFitness	71.03	66.74	68.29

Conclusions

- The excellent behaviour of the cheater fitness shows that the defined operations have the necessary expressive power to achieve good results
- The implementations made during this work can be used as a framework to investigate fitness functions in future work
- The inclusion of a few grammatical rules improved the results

Thank you