Otto-Friedrich Universität Bamberg
Cognitive Systems Master Project
SS 2012

Learning Grammars with Swarm Genetic Approach

Submitted by:
Fco. Javier Cano
Fernando Macías
Martina Milovec

Content:

- Swarm Intelligence and Grammars
- Genetic Algorithms and Context Independent Grammars
- Program Structure

1. The Swarm Algorithm
2. Fitness Evaluation
3. Mutation

- Conclusion

Swarm Intelligence and Grammars

- A single ant is not smart, but ants colonies are - what they do is called swarm intelligence
- Swarm intelligence works on a principle of simple rules being applied on local information, so called self-organized system
- SI helps humans to manage complex systems, e.g. in Multi-Robot Systems

Swarm Intelligence and Grammars

- Dealing with problem of pain grammars using principles of swarm intelligence - generating short and good grammars
- In project:

1. the population - individual grammars that are exploring through hyperspace
2. the population is initialized
3. individual grammars are given initial empty values
4. fitness function chooses the best individual
5. others individuals from the rest of population converge to the best individual

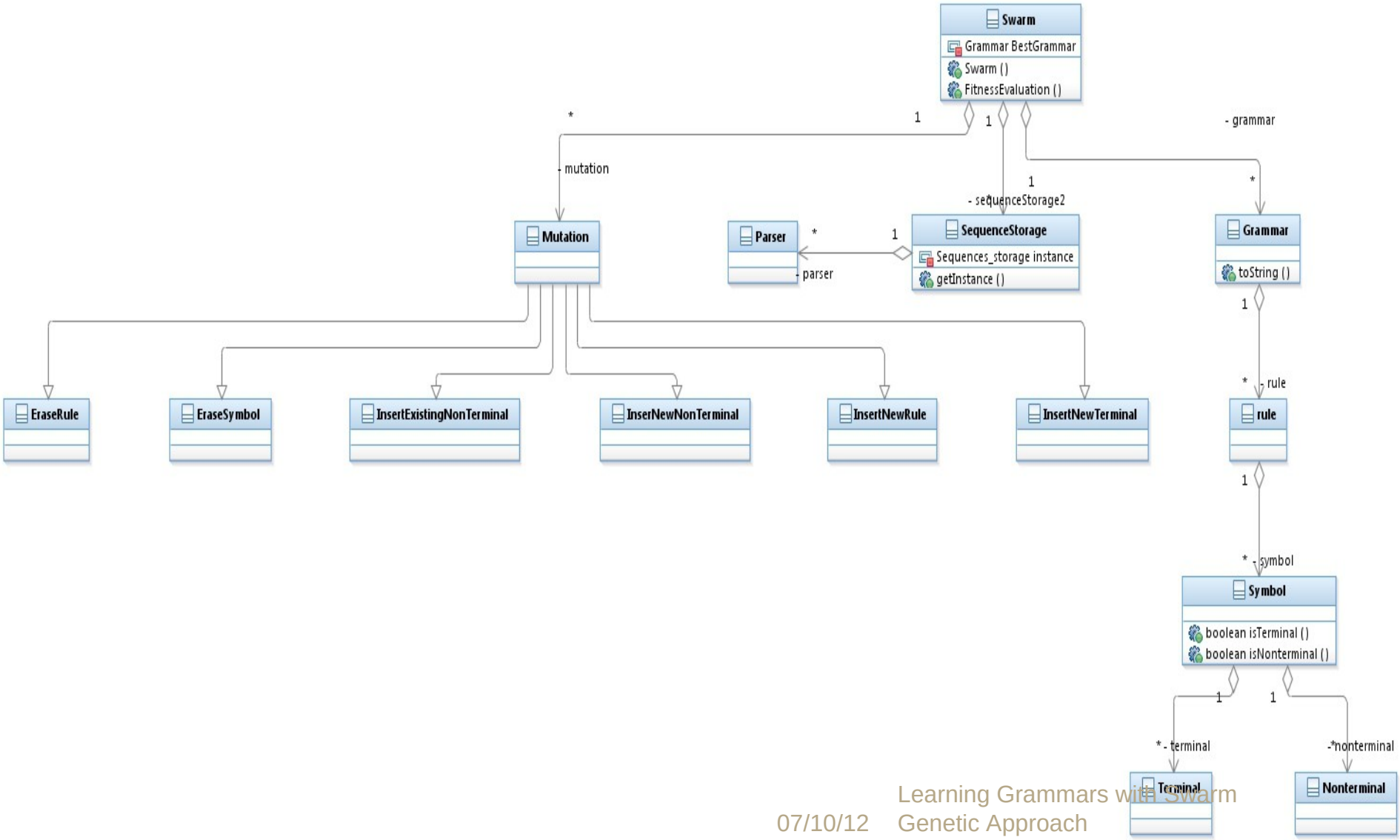
Genetic Algorithms and Context Independent Grammars

- Genetic Algorithms:

1. define elements e.g population of independent grammars
2. selection according to fitness
3. and random mutation

- GA follow rules to generate useful solutions for optimization and search problems, by an heuristic search that mimics the process of natural evolution

Program Structure



1. The Swarm Algorithm

INPUT:

Number of generations ***ngen, Offspring frequency ofreq, List of sequences S***

G <− List of the individuals (grammars), initialized empty;

M <− Array of all the possible mutations;

while i<*ngen do*

for each grammar g in G

select pseudo randomly one mutation m from M;

apply m to g;

end

if i%*ofreq=0 then*

select grammar *bestg from G with highest fitness value (best-so-far individual)*

for half of the remaining grammars g in G

g <− *bestg*

end

end

end

return ***bestg from G***

2. Fitness Evaluation

Input: Grammar G(N,T,S,P) and List of Words

Initialize variables

calulate value of maxLength

Add to Queue G(S) // Initial Symbol of the Grammar

while (Queue not empty) and List of Words not empty

s = popQueue

Add to seen s

if *CheckTerminals (s) then*

while remove from List of Words do

increase matches

end

else

if |s| <*maxLength then*

split in prefix, first and suffix

if prefix is contained in List of Words then

for each rule in the Grammar do

if first is the nonTerminal in the rule then

add to list: preffix , rule rightSide and suffix

if list has never been seen and is not in Queue then

add list to Queue

end *// all corresponding ends*

return matches

3. Mutation

- **New rule :** inserting a valid new rule
- **Erase rule :** erasing an existing rule
- **New symbol :** adding a new terminal or non terminal (not existing ones) and adding an existing terminal and non terminal to the right side of an existing rule
- **Erase symbol :** erasing a terminal on the right side of the rule or non terminal also on the right side of the rule

Conclusion

- Memory problem
- Very long execution time problem
- Consistency

Thank you for your attention!

?