DeMeMech-Exchange Presentation

Research conducted by Alexander Taro Franke at Arai Lab,
School of Engineering, University of Tokyo

Table of Contents

- Introduction to GP
- Initial goal of this research
- Problem analysis of GP
- Example-based fitness functions
- Grammatical construction of GP
- Molecular program construction

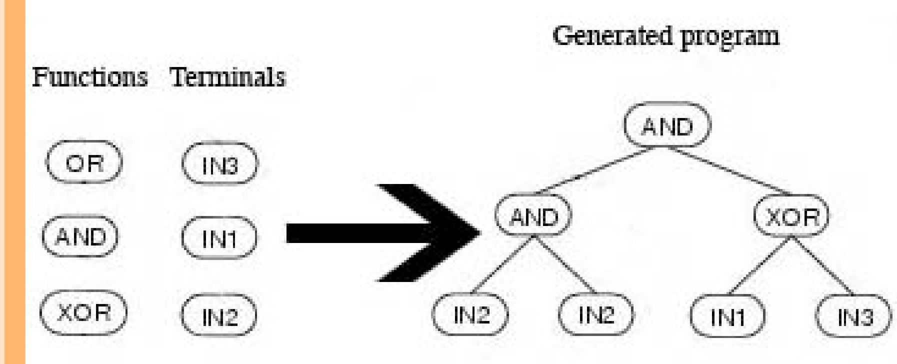
Introduction to GP

- Genetic Programming is an evolutionary search algorithm much like Genetic Algorithms
- It automatically finds solutions to a before specified problem
- Difference to GA: GP produces computer programs as solutions

Introduction to GP

- Classical GP produces programs with tree-structure called parse trees. The algorithms generates those trees by randomly choosing from a set of functions and terminals (non-terminals = nodes,terminals=leafs)

Introduction to GP



Construction of programs in GP

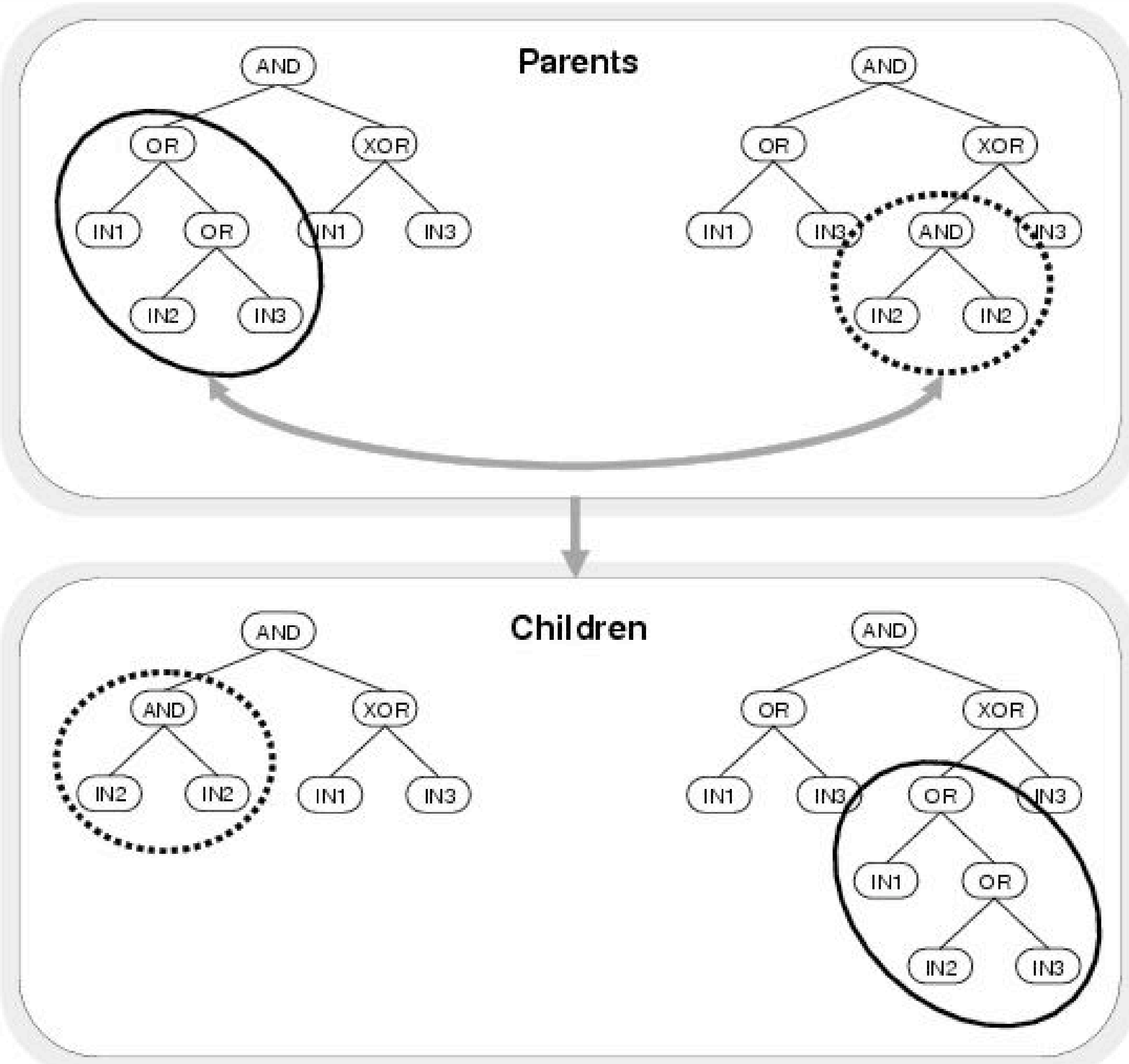
Introduction to GP

- Initially a fixed number of such programs (a generation) is generated
- Then every program is executed and a fitness is assigned which reflects how well the program accomplished the task

Introduction to GP

Cross-over operation

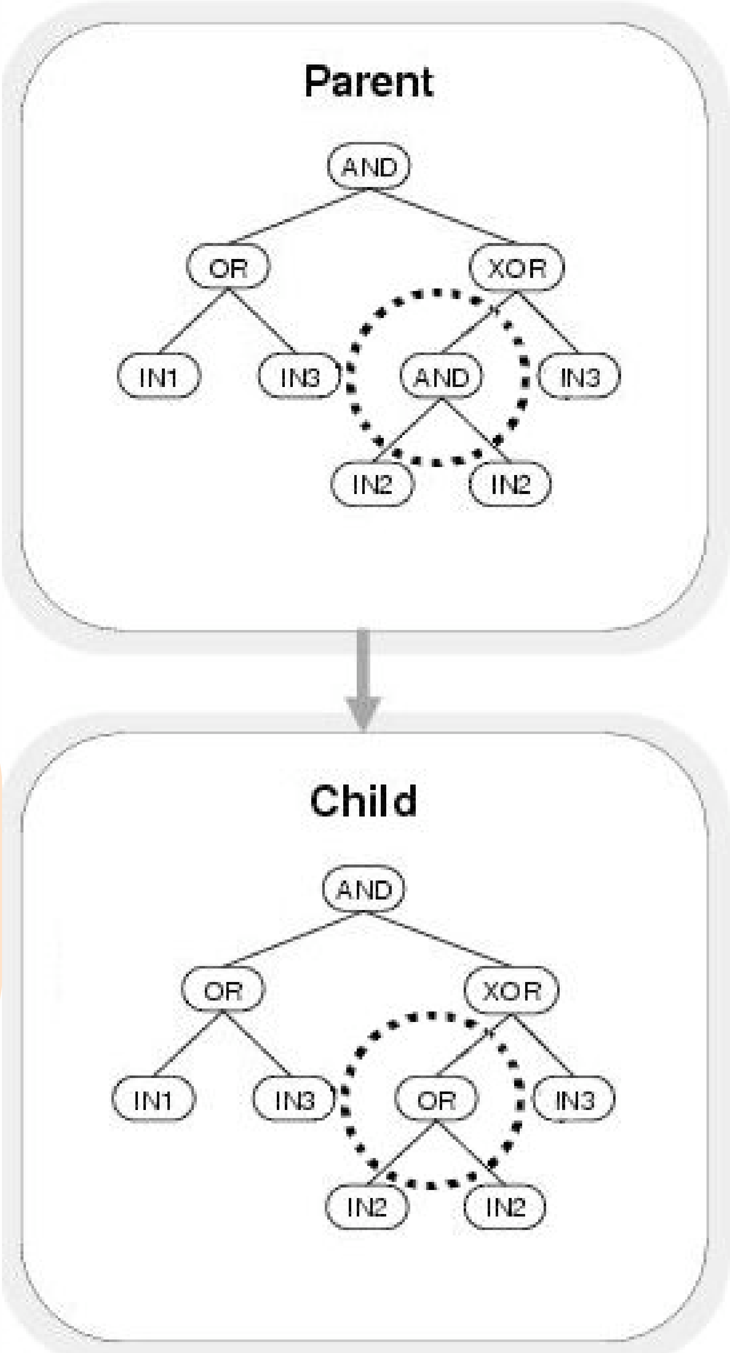
The cross-over operation is performed by taking subtrees of two parent programs and then exchanging those trees



Introduction to GP

Point-mutation

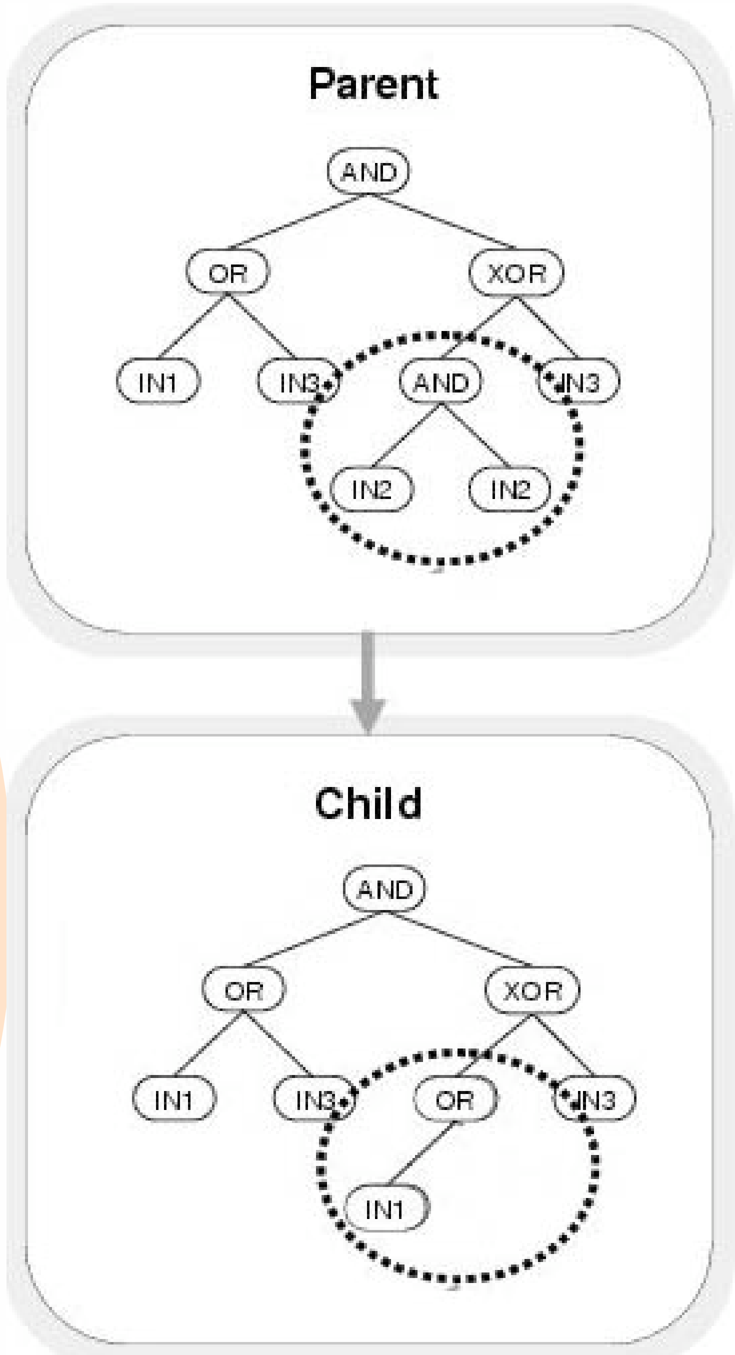
The point-mutation is performed by choosing a random node or leaf and changing it.



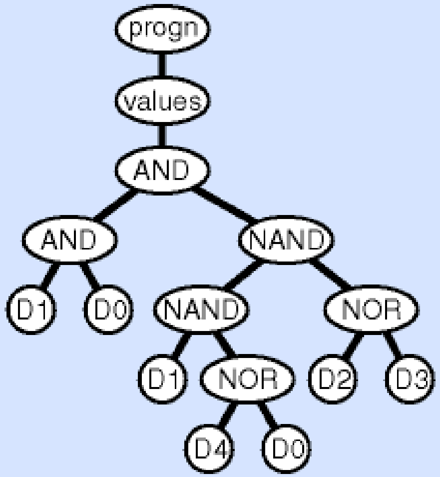
Introduction to GP

Subtree-mutation

The subtree-mutation is performed by choosing a random subtree and substituting it with another.

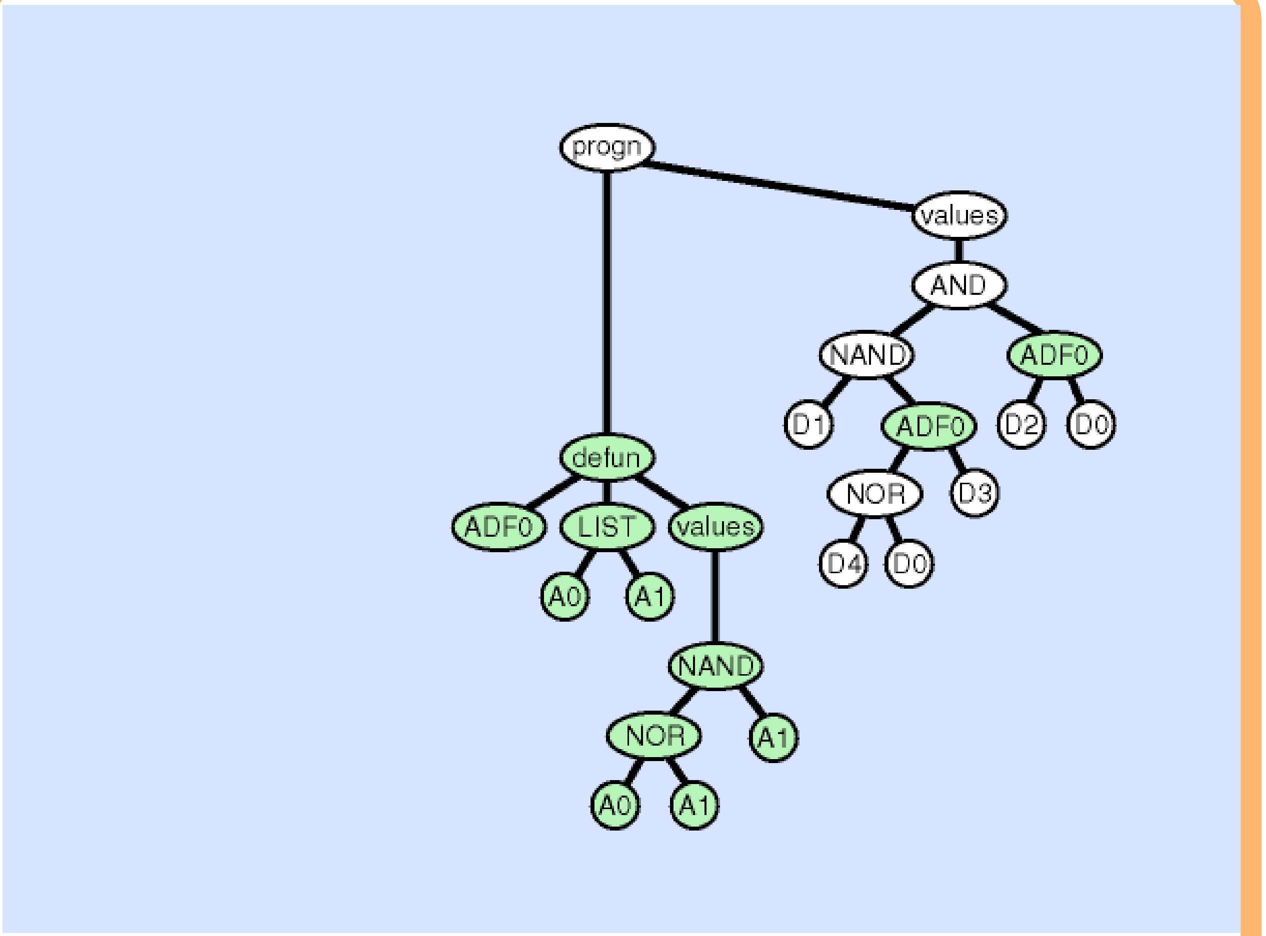


Construction of new ADFs



Source: http://www.genetic-programming.com/

Branch duplication ADFs



Source: http://www.genetic-programming.com/

Introduction to GP

- The process of generating and evaluating new program generations is repeated until a satisfying solution is found

Introduction to GP

- Recently GP has evolved new solutions for problems like electronic circuit design, quantum computing algorithms or optical lens system design
- This is due to the exponentially growing computing power and GP solutions often involved large clusters of computers (1000-node-clusters for example)

Initial goal of this research

- Use GP to derive a program for cooperative RoboCup Soccer Robots (4-legged league)

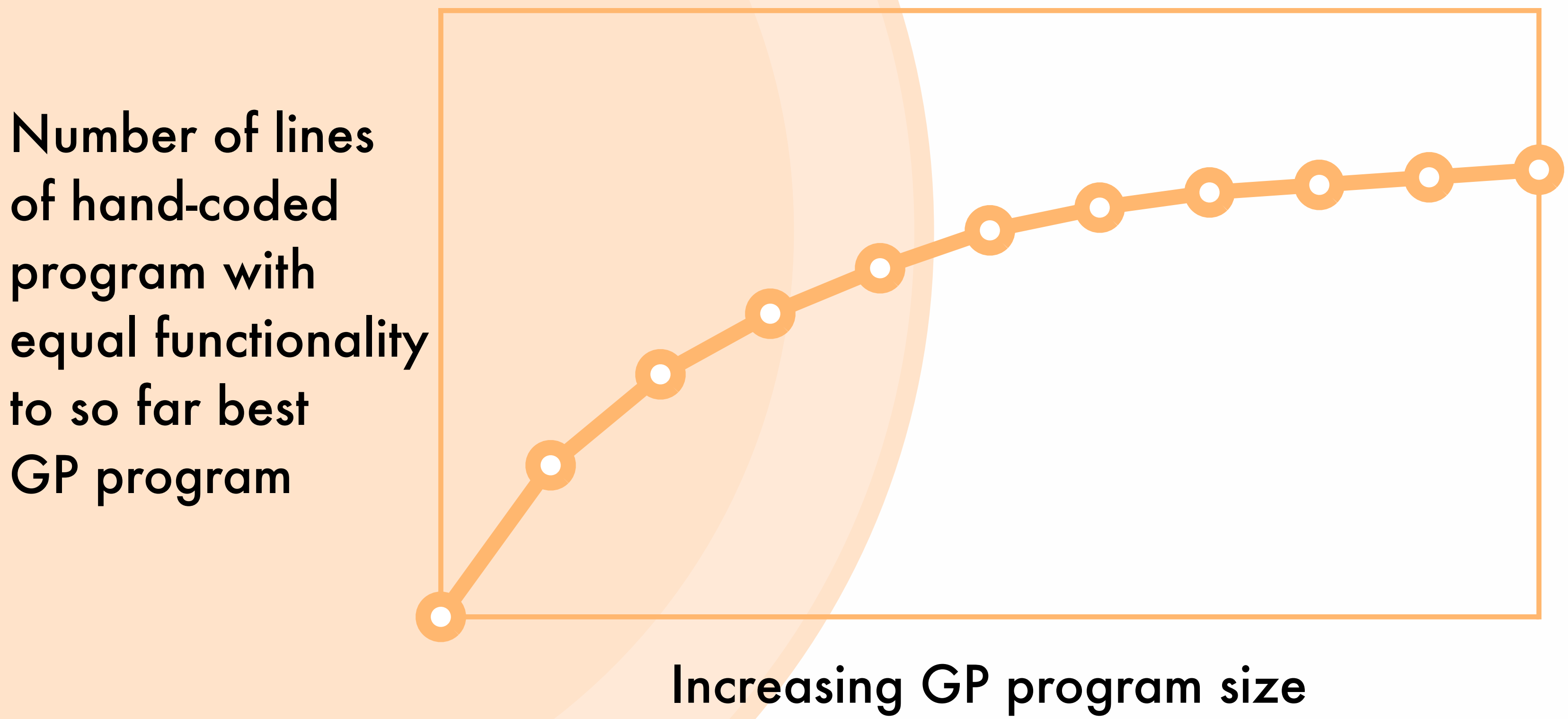
Initial goal of this research

- Existing code would allow construction of parse trees with little modification
- Use simulator to coevolve soccer teams using existing high-level functions

Problem analysis of GP

- GP is able to create simple solutions but fails to produce more complex solutions
- This is because by increasing the program size linearly the search space grows exponentially

→ Exponentially more programs with non-functional code

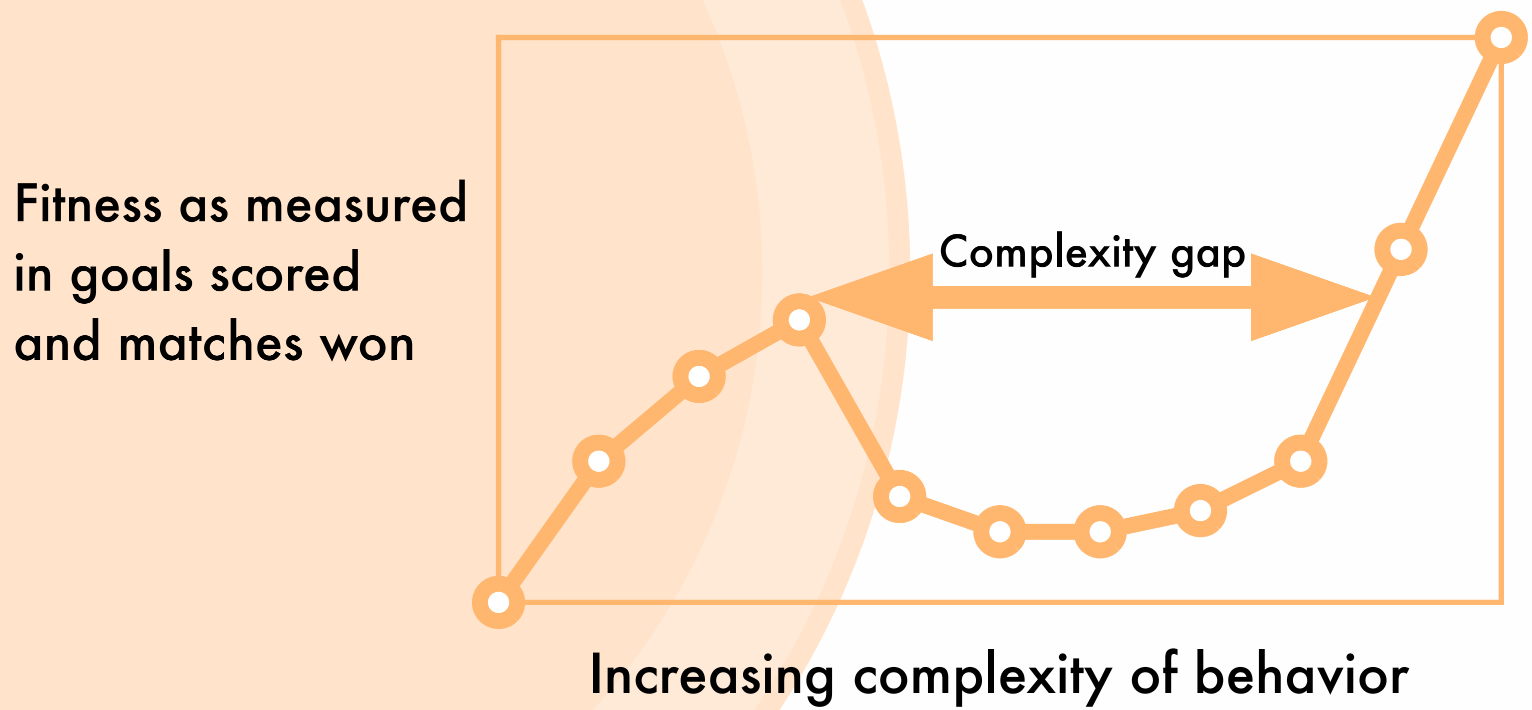


Problem analysis of GP

- This exponential grows in possible programs can be called the “curse of dimensionality”
- Therefore new ways to find useful programs with greater complexity have to be found, i.e. we have to restrict the search space to areas with possibly useful solutions without limiting it so much that the power of the evolutionary mechanism is lost

Problem analysis of GP

- Concerning the initial goal:
  Several attempts for using GP to derive soccer players for the Simulation League have been made, including evolution of homogeneous and heterogenous teams, multiple-goals fitness and competitive coevolution but the found solutions often show less complex behavior called "kiddie soccer"



Problem analysis of GP

- GP consists of the following problem areas:
  - 1. Terminal and Function set
  - 2. Fitness function
  - 3. Construction of program
- Terminals and function sets have been extensively researched as in automatically defined functions(ADFs) or macros(ADMs)

Example-based fitness functions

- Idea: Improve the fitness function
- Instead of having one fixed fitness function apply several fitness functions that compare the structure of the generated programs with the structure of handmade example programs using a metric measuring how similar the functionalities are

Example-based fitness functions

- Problem: Functional metric for tree structures
- Computational metrics can’t measure functionality or their computation is exponential and therefore not applicable
- Use a measure that compares the output of example programs with those of the programs using the same inputs

Example-based fitness functions

- Experiment:
  - Modification of TGP-code by Michai Oltean
    - Example-based fitness functions:
      Symbol Regression of a 10th-degree polynomial using lower-degree factors as example-functions
    - Usage of previous solved solutions as ADFs in the next case
    - Global elitist approach with cross-over rate of 20% and 5% mutation
    - 100 individuals per generations,100 generations, 10 fitness functions,10 sample-data,100 runs
    - Compare to classical GP with ADFs

Example-based fitness functions

- Results of experimenting on a symbolic regression problem showed no improvement to GP with ADFs.
- Conclusion: Either the given problem (symbolic regression) was too simple to be broken down to subtasks or the power of possible subtask accomplishment is annulled with the cross-over and mutation actions. These are believed to destroy previously found solutions to subtasks when a new fitness function is used
- Need for preserving functional entities

Grammatical construction of GP

- Since the improvement of the fitness function seemed not general and unfruitful

→ Improve the construction of programs in GP

→ Improve the structure of programs in GP

Grammatical construction of GP

- Use/evolve context-free grammars who should construct programs
- Grammars in this context are considered to be production rules for valid expressions

Production rules

$$S \longrightarrow Bcc$$
$$A \longrightarrow Aa$$
$$A \longrightarrow a$$
$$B \longrightarrow Ab$$

Example application

$$S \longrightarrow Bcc \longrightarrow Abcc \longrightarrow Aabcc \longrightarrow aabcc$$

Grammatical construction of GP

- Production rules are evolved which restrict the space of possible solutions
- Solutions are constructed with these rules which then are tested
- Reason for this approach: Possibly smaller dimension of search space for production rules then due to more abstract nature of production rules
- Approach of using grammars has already been applied with competitive but not superior performance

Molecular program construction

Idea:

To reduce the destructive power of crossover and to prevent
bloat, see genetic programs as collection of terminals and functions much like atoms with certain binding energies between them and maximize the energy to construct programs out of this configuraition

Table for binding energies between the components of the collection

	Fun1	Fun2	Fun3	Fun4	Fun5	Cons t1	Cons t2	Cons t3	Input 1	Input 2	Input 3	Input 4
Fun1			1	11			36		25		3	
Fun2	5	9		7								42
Fun3		7				62	3		6			23
Fun4			34	7				44	5	86	6	
Fun5				4	6	77			16			3
Const1												
Const2												
Const3												
Input1												
Input2												
Input3												
Input4												

Molecular program construction

Molecular program construction

Crossover operation would then simply be an exchange of components and mutation would also just be a replacing of certain components

The matrices would also be evolved and used to construct programs by optimizing the reward function

It is to be seen how effective a construction like this would be in means of computing necessary

Molecular program construction

Every matrix would be assigned a certain amount of component sets to construct programs out of them

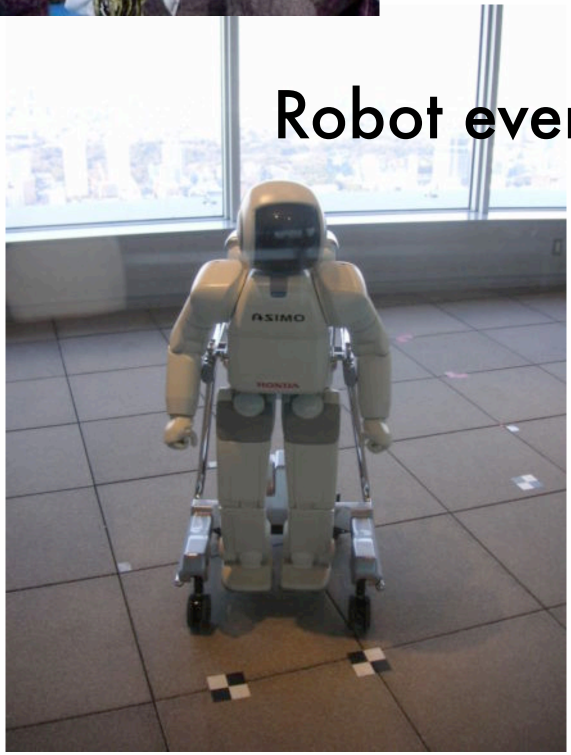
Those in turn would be evaluated and the average fitness would be assigned to the matrix

Then the matrices and the sets of components would undergo the evolutionary mechanisms and generate the next generation of matrices and sets

Impression of Japan

Cherry-blossom parties

Robot everywhere



Impressions of Japan

Making and eating soba

Yakiniku- Korean bbq

Suburban desert - never ends

Crazy video game devices

Rollercoasters through buildings

...and of course Karaoke