Genetic Programming and its use for learning Concepts in Description Logics

Jens Lehmann

Artificial Intelligence Institute
Computer Science Department
Dresden Technical University

May 29, 2006

Outline

Outline:

- brief introduction to Evolutionary Computing
- explanation of the workings of a Genetic Programming algorithm
- application of Genetic Programming to Description Logic learning including experimental results

Evolutionary Computing

Evolutionary Computing - Introduction

- Machine Learning algorithm
- inspired by biological evolution in the real world
- uses Darwin's theory of evolution (natural selection, survival of the fittest)

nature	evolutionary computing
individual	problem solution
fitness	quality of a solution
crossover, mutation	genetic search operators
natural selection	reuse of good solutions

Evolutionary Computing - Generic Algorithm

Generic Evolutionary Algorithm:

- create population
- while the termination criterion is not met:
  - select a subset of the population based on their fitness
  - produce offspring using genetic operators on selected individuals
  - create a new population from the old one and the offspring

Genetic Programming

GP - Introduction

- Genetic Programming is a special kind of Evolutionary Computing algorithm
- distinctive feature: individuals are represented as variable length programs
- programs can have a linear (like code in imperative languages) or tree structure (like LISP)
- we focus on tree structures

Tree Representation and Alphabet

- alphabet consists of terminal set T and function set F
- terminal set: set of all leaf nodes, which can occur
- function set: set of all internal nodes, which can occur

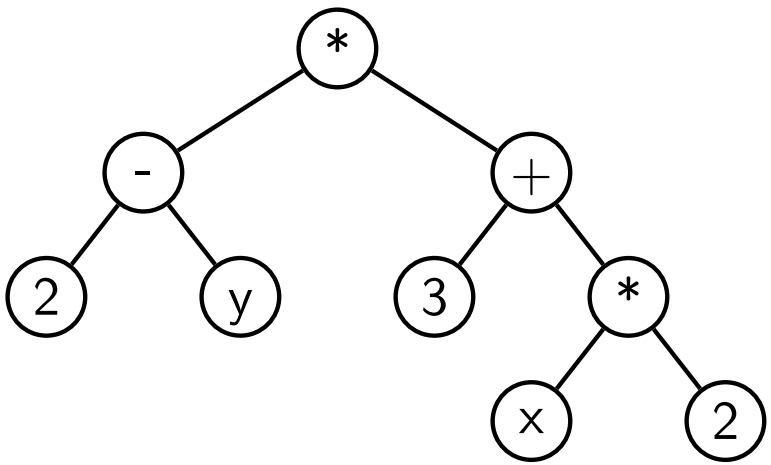


Figure: possible tree for $T = \{0, 1, 2, 3, x, y\}$ and $F = \{+, -, *\}$

Creating an initial Population

- all methods have as parameter a maximum depth d
- methods to create a single tree:
  - full method:
    - if $d > 0$ randomly select a function symbol and call the method recursively for all children, which need to be generated, with parameter $d - 1$
    - if $d = 0$ return a random terminal symbol
  - grow method: modify full method to select an alphabet symbol – not necessarily a function symbol – in the $d > 0$ case
- full method creates complete trees (i.e. all leaf nodes have equal depth)

Creating an initial Population

- ramped half and half:
  - divide population in $d-1$ parts, where each part has its own depth parameter (from 2 to d)
  - half of the trees in each part are generated using the full and the other half using the grow method with the part-specific depth parameter
- grow and full method can be used to create the population, but often ramped half and half is used to increase diversity

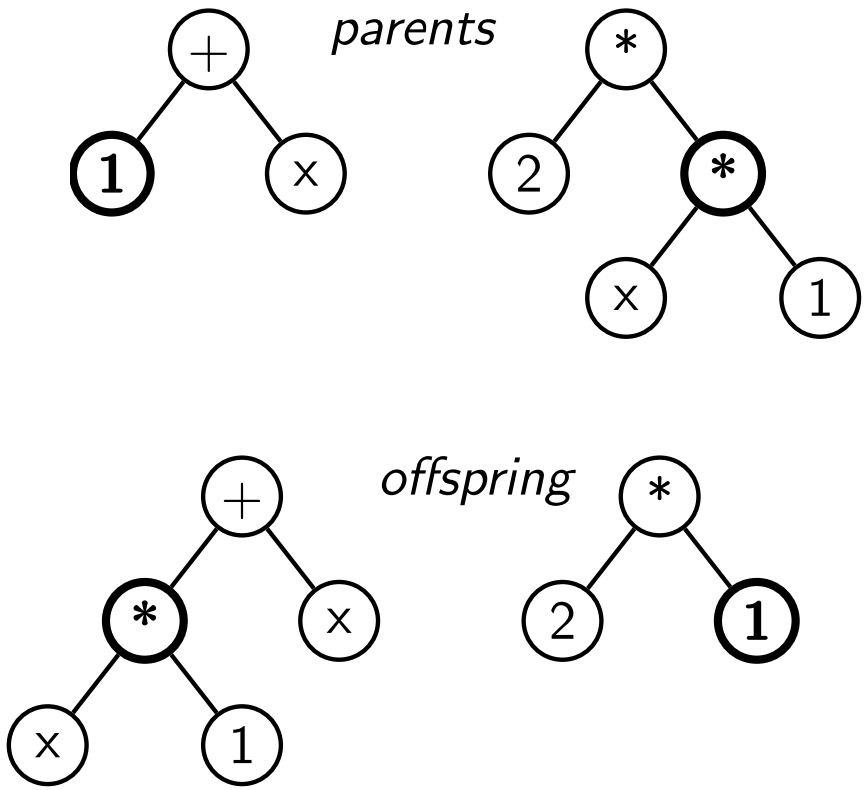
Genetic Operators

- it is unlikely that tree initialisation alone yields a good solution
- genetic operators aim at finding and combining pieces of a good solution
- simplest genetic operator is reproduction: copies one individual in the next generation

Crossover

Crossover:

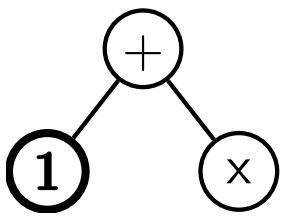
- selects two parents
- randomly selects a node in each parent, the *crossover point*
- subtrees rooted by these two points are swapped

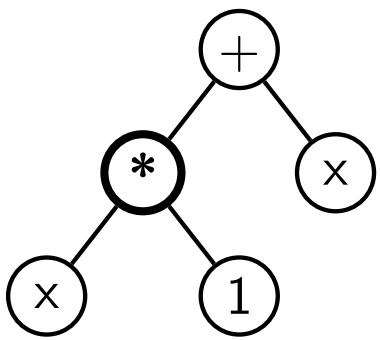


Mutation

Mutation:

- selects one individual from the population
- randomly chooses a node in the inidividual, the *mutation point*
- subtree rooted by this node is replaced by another tree (e.g. using the grow method)

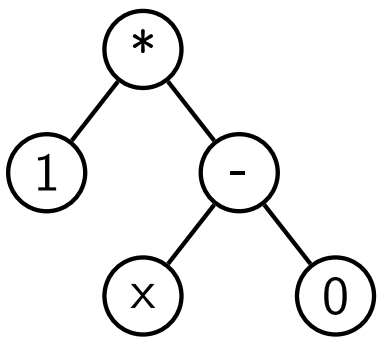




Editing

Editing:

- selects one individual from the population
- applies simplification rules to the individual



Selection

- selection methods in GPs simulate natural selection ("survival of the fittest")
- fitness of an individual is usually encoded in a single number
- measuring process can be very complex and take multiple objectives into account

FPS

Fitness Proportionate Selection (FPS):

- probability of an individual being selected is proportional to its fitness
- *stagnation* problem can arise
- stagnation: in a population of individuals with similar fitness, there is almost no *selective pressure*, because all individuals have approximately the same probability of being selected
- modified versions of FPS (e.g. sigma truncation) avoid the stagnation problem

Rank and Tournament Selection

Rank Selection:

- individuals are sorted according to their fitness, i.e. each individual has a rank within the population
- probability of being selected is a function of the rank (any sensible function can be used)

Tournament Selection:

- takes m individuals from the population and selects the best one (m is typically a number between 2 and 8)

Termination

Termination Criteria:

- when optimal solution is found
- when solution is within an ϵ-range of an optimal solution
- manual abort
- fixed number of generations
- no better solution has been found for a fixed number of generations
- population does not have sufficient genetic diversity to produce new solutions

Overall Algorithm

- create initial population
- while the termination criterion is not met:
  - evaluate the fitness of all individuals in the population
  - while the number of individuals to generate is not reached:
    - choose a genetic operator
    - select (by one of the selection methods) the appropriate numbers of individuals for this operator and generate new individuals
    - copy these individuals into the new population

Human competitive GP Results

- John Koza, a respectable GP researcher, has created a list of 36 instances of "human competitive intelligence" produced by genetic programming
- some of these are:
  - creation of quantum algorithms (better than previously published results)
  - creation of soccer playing programs for Robo Cup
  - synthesis of analog circuits

Learning in Description Logics

Introduction

- goal: learn a concept definition from positive examples + negative examples + background knowledge
- Why is it useful to learn in DLs?
  - may have similar applications like ILP (Inductive Logic Programming) approaches for learning horn clauses e.g. in biology and medicine
  - incremental ontology learning in context of OWL and the Semantic Web (can help to build ontologies)

Using GP for Learning in DLs

- Why use GP?
  - flexible: can (probably) learn in any description language
  - robust: can handle noise
  - GP has shown promising results in practise
  - GP usually performs good when approximate solutions are acceptable
  - in description languages, which allow all boolean operations it becomes difficult to apply standard ILP appraches
  - makes use of computational resources (the more resources the better the result)
  - can be fine-tuned in various ways to improve results
- Why not?
  - too general: there may be better special purpose methods, especially for less expressive description languages
  - stochastic outcome: makes analysis difficult and good solution quality cannot be guaranteed
  - positive effect of natural selection and standard genetic operators is not obvious when learning DLs

ALC

ALC concepts:

$$C, D \rightarrow A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall r.C \mid \exists r.C$$

Example TBox $\mathcal{T}$:

$$\texttt{Man} \equiv \neg \texttt{Woman}$$
$$\texttt{Mother} \equiv \texttt{Woman} \sqcap \exists \texttt{hasChild}.\top$$

Example ABox $\mathcal{A}$:

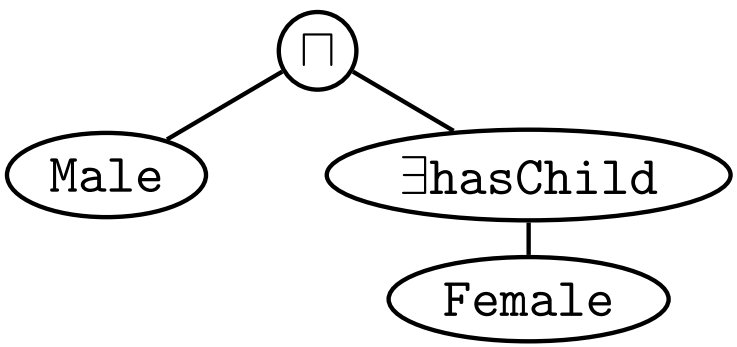
$$\texttt{Man}(\texttt{STEPHEN}).$$
$$\neg \texttt{Man}(\texttt{MONICA}).$$
$$\texttt{Woman}(\texttt{JESSICA}).$$
$$\texttt{hasChild}(\texttt{STEPHEN}, \texttt{JESSICA}).$$

Learning Problem

- we have a target concept name $\texttt{Target}$ and a knowledge base $\mathcal{K}$ as background knowledge
- examples are of the form $\texttt{Target}(a)$ (positive) or $\neg\texttt{Target}(a)$ (negative), where a is an object
- let E^+ be the set of positive examples, E^- the set of negative examples and $E = E^+ \cup E^-$
- we want to find an acyclic definition Def of the form $\texttt{Target} \equiv C$ such that for $\mathcal{K}' = \mathcal{K} \cup \{Def\}$ we have $\mathcal{K}' \models E$

Tree Representation of $\mathcal{ALC}$ concepts

- alphabet
  - let N_C be the set of concept names and N_R the set of role names
  - terminal set: $T = N_C \cup \{\top, \bot\}$
  - function set: $F = \{\sqcup, \sqcap, \neg\} \cup \{\forall r \mid r \in N_R\} \cup \{\exists r \mid r \in N_R\}$
- example of a tree representation of the $\mathcal{ALC}$ concept $\texttt{Male} \sqcap \exists \texttt{hasChild}.\texttt{Female}$:



Fitness Measurement

How to measure fitness?

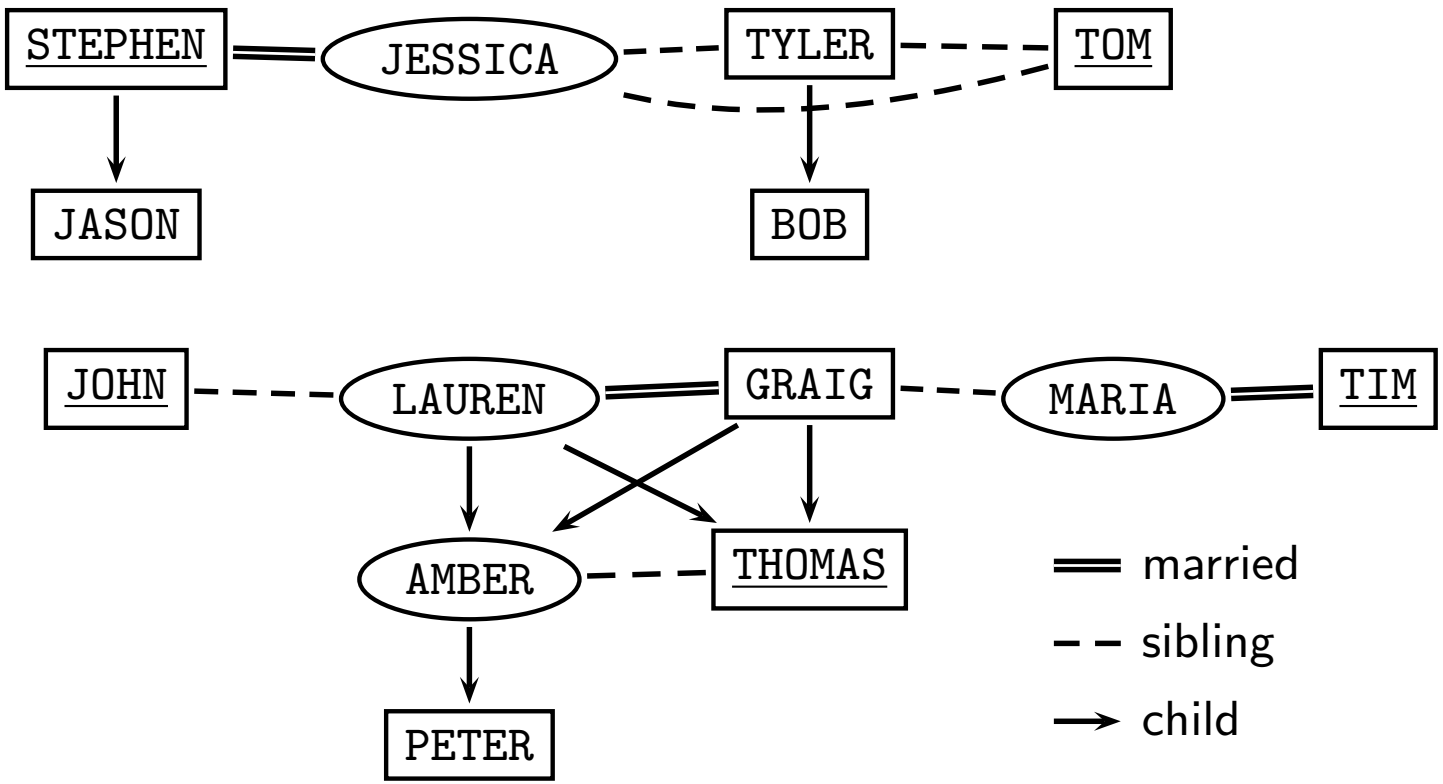
- an example $a \in E$ is in one of the three following classes:
  - positively classified: $\mathcal{K} \models C(a)$
  - negatively classified: $\mathcal{K} \models \neg C(a)$
  - neutrally classified: $\mathcal{K} \not\models C(a)$ and $\mathcal{K} \not\models \neg C(a)$
- high penalties for classification errors (positive example classified as negative or negative example classified as positive)
- smaller penalty if classification is not provably (in)correct (positive or negative example classified as neutral)

Fitness Measurement

- primary learning goal is correct classification, secondary goal is small concept length (smaller concepts usually generalize better to unseen examples)
- length $|C|$ of a concept C is defined inductively in the obvious way:

$$
\begin{aligned}
|A| = |\top| = |\bot| &= 1 \\
|\neg C| &= |C| + 1 \\
|C \sqcap D| = |C \sqcup D| &= 1 + |C| + |D| \\
|\exists r.C| = |\forall r.C| &= 2 + |C|
\end{aligned}
$$

Uncle Problem



$$\texttt{Uncle} \equiv \texttt{Male} \sqcap (\exists \texttt{hasSibling}.\exists \texttt{hasChild}.\top \sqcup \exists \texttt{married}.\exists \texttt{hasSibling}.\exists \texttt{hasChild}.\top)$$

Uncle Problem

GP:

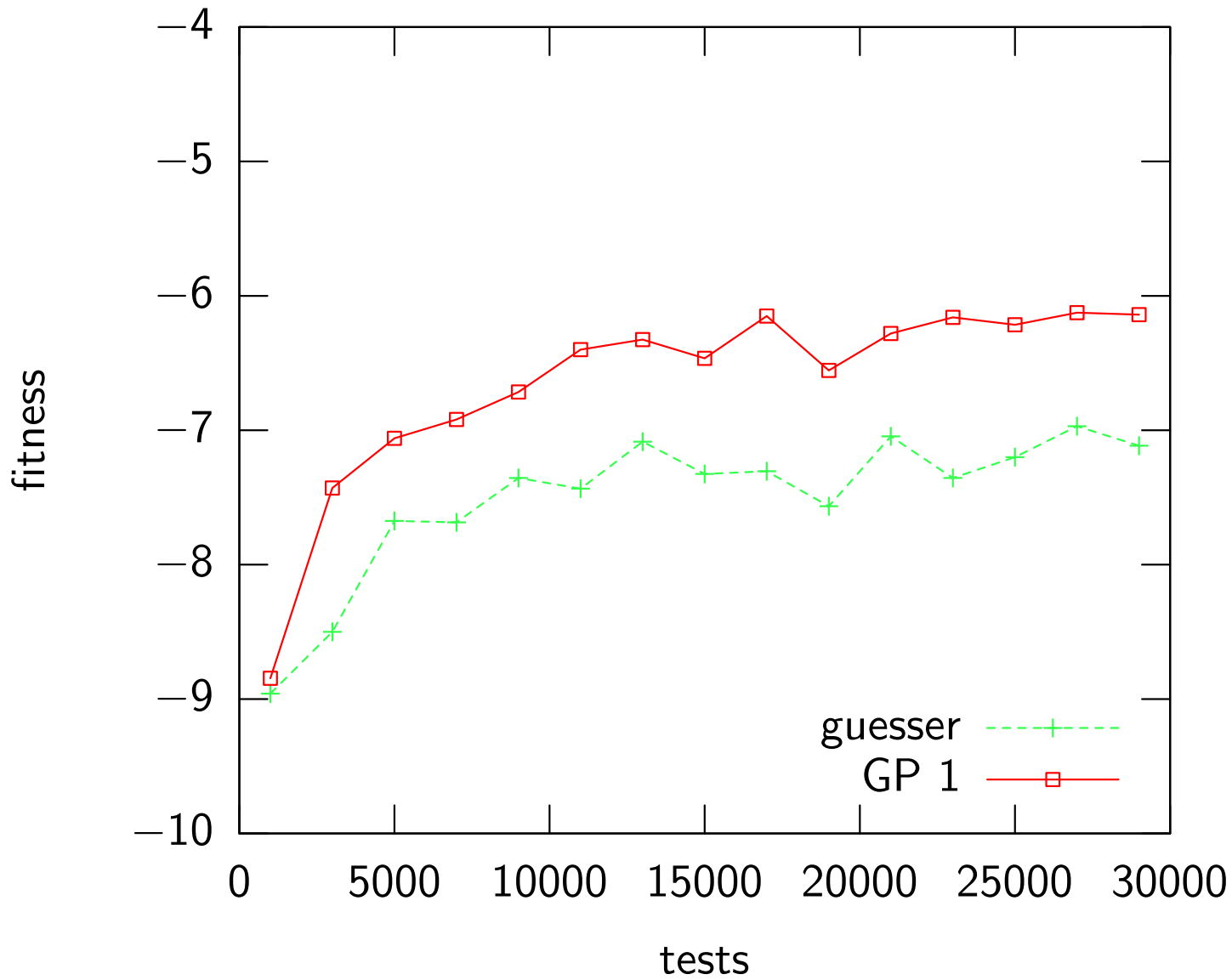
- 10 generations
- crossover probability: 90%, mutation probability: 3%
- selection: tournament (tournament-size 3)

Random Guesser:

- uses grow method to generate trees, which amounts to guessing a solution
- for technical reasons a depth limit of 10 is used (otherwise trees are too big to be processed in reasonable time)

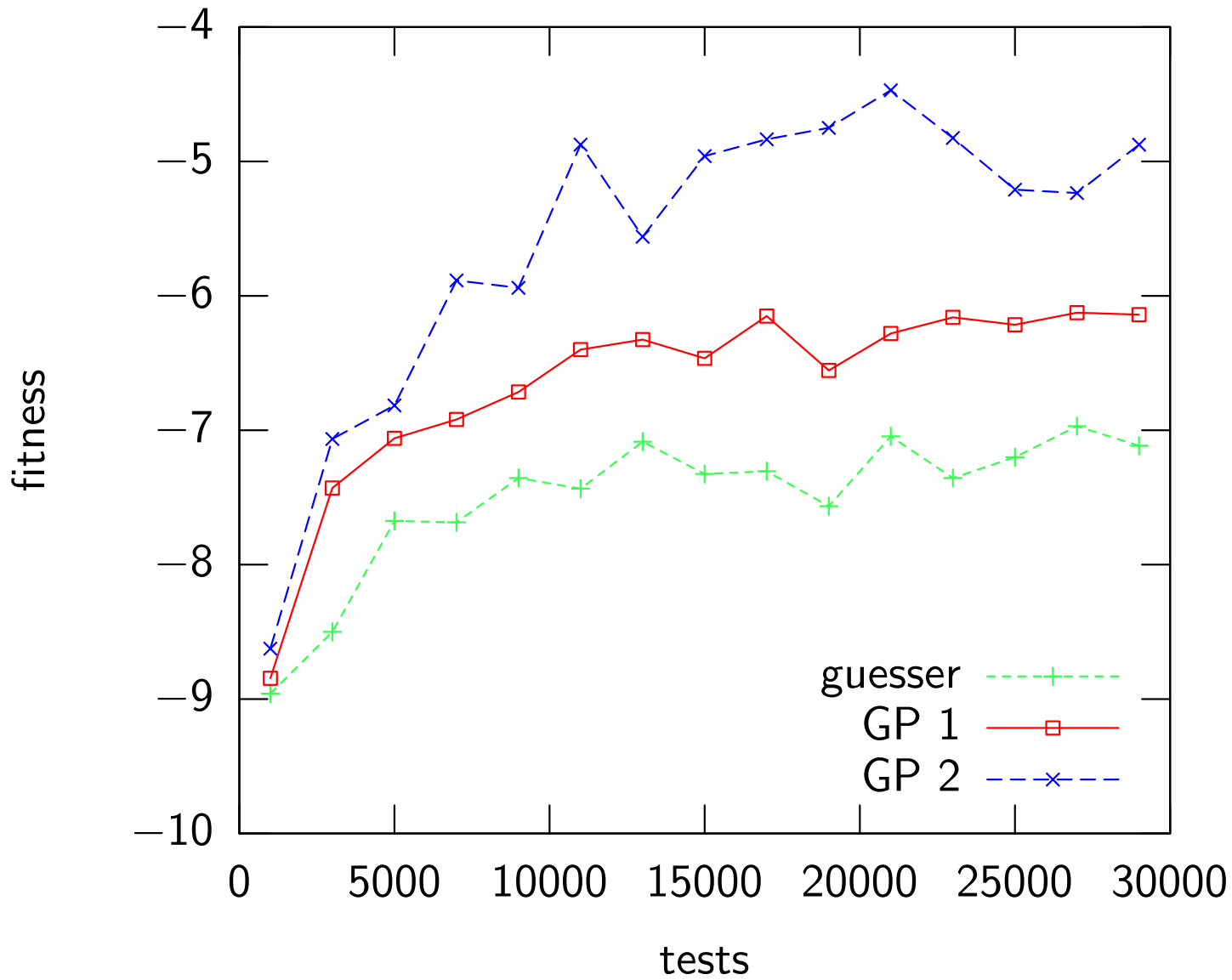
Fitness Function:

- error penalty: 3 points, neutral classification penalty: 1 point, length penalty: $\frac{1}{10}$ points



Hill Climbing Operator

- introduce a new genetic operator, which takes one individual and produces a new one:
  - given a concept C we can look at the "neighbourhood" $N(C)$ of C, i.e. concepts of the form $C \sqcup A$, $C \sqcap A$, $\exists r.C$, $\forall r.C$
  - from these concepts we can compute a subset, which are better classifiers than C and for which there is no better classifier in $N(C)$
  - if this set is empty we return C (= reproduction)
  - otherwise we randomly select one element of this set and return it
- operator can be seen as one-step-hill-climbing
- evaluation of neighbour concepts can be approximated efficiently using the fact that we already evaluated C (details omitted)
- new algorithm is a combination of GP and hill climbing
- to test it on the uncle problem we use 15% hill climbing, 80% crossover (see next slide)



Automatically Defined Concepts (ADC)

- idea: inventing new concepts, which are useful for defining the target concept
- instead of one tree, we evolve two trees
- alphabet of main tree is extended by a terminal symbol "ADC"
- ADCs are a modification of the idea of automatically defined functions (ADFs) already known in genetic programming
- advantage: solution more modular, easier to read
- disadvantage: larger search space