

Evolutionary Computation

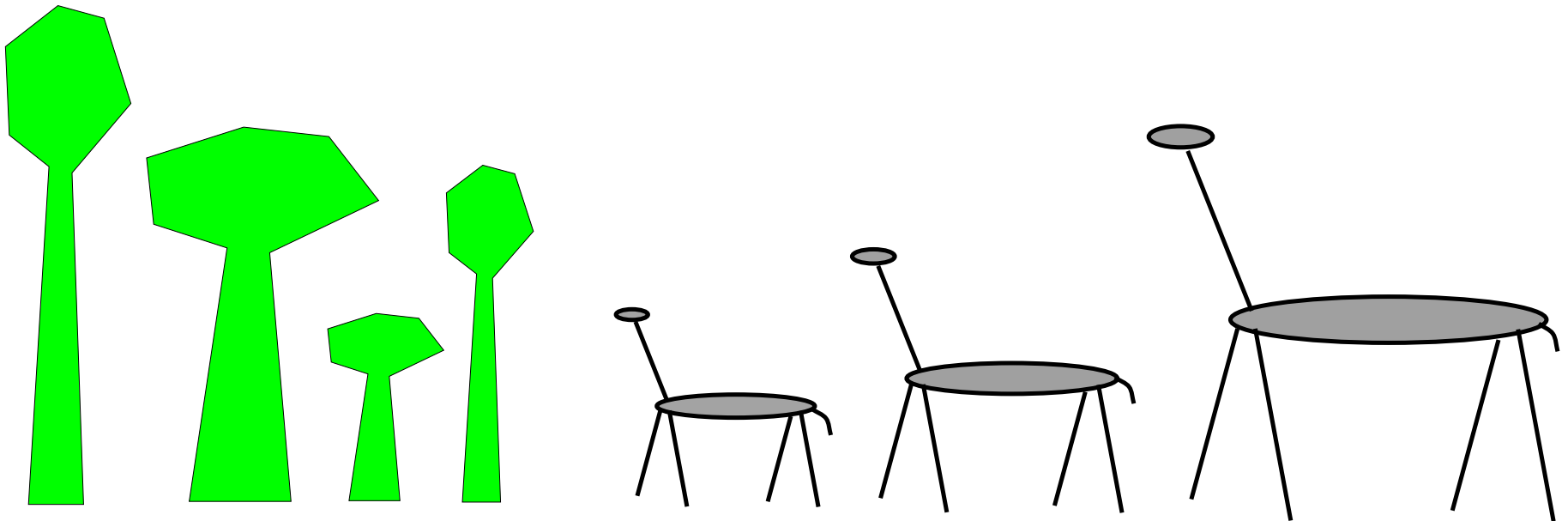
- Khaled Rasheed
- Computer Science Dept.
- University of Georgia
- <http://www.cs.uga.edu/~khaled>

Presentation outline

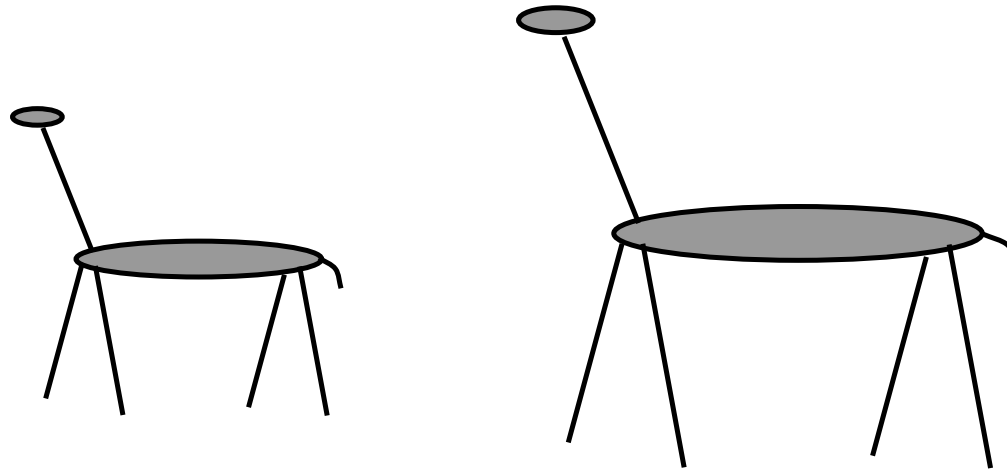
- Genetic algorithms
- Genetic programming
- Evolution strategies
- Classifier systems
- Evolution programming
- Conclusion

In the forest

- Fitness = Height
- Survival of the fittest



Reproduction



Genome: *ATTGCGCCATGAT*

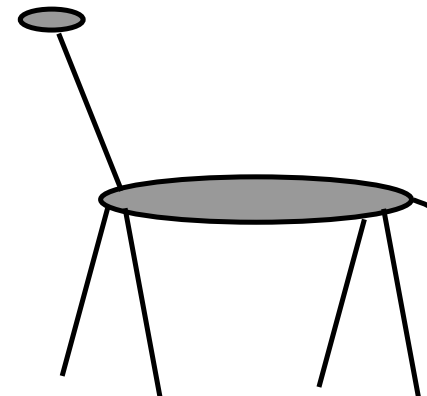
ATTAAACCATAGT

Crossover:

<i>ATTG</i>	<i>CGCCATGAT</i>
<i>ATTA</i>	<i>AACCATAGT</i>
<i>ATTG</i>	<i>AACCATAGT</i>

Mutation:

<i>ATTGAA</i>	C	<i>CATAGT</i>
<i>ATTGAA</i>	G	<i>CATAGT</i>

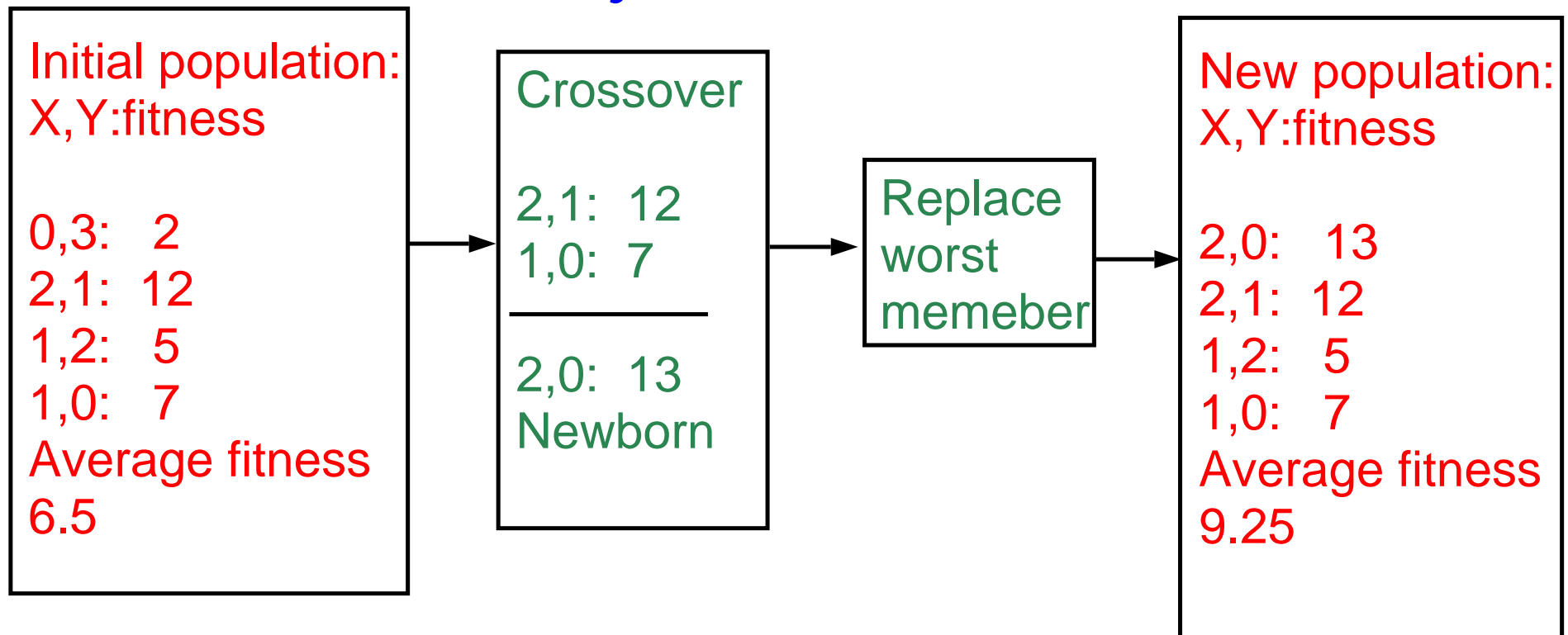


Genetic Algorithms

- Maintain population of potential solutions
- New solutions are generated by combining and modifying existing solutions
 - Crossover
 - Mutation
- Objective function = "Fitness function"

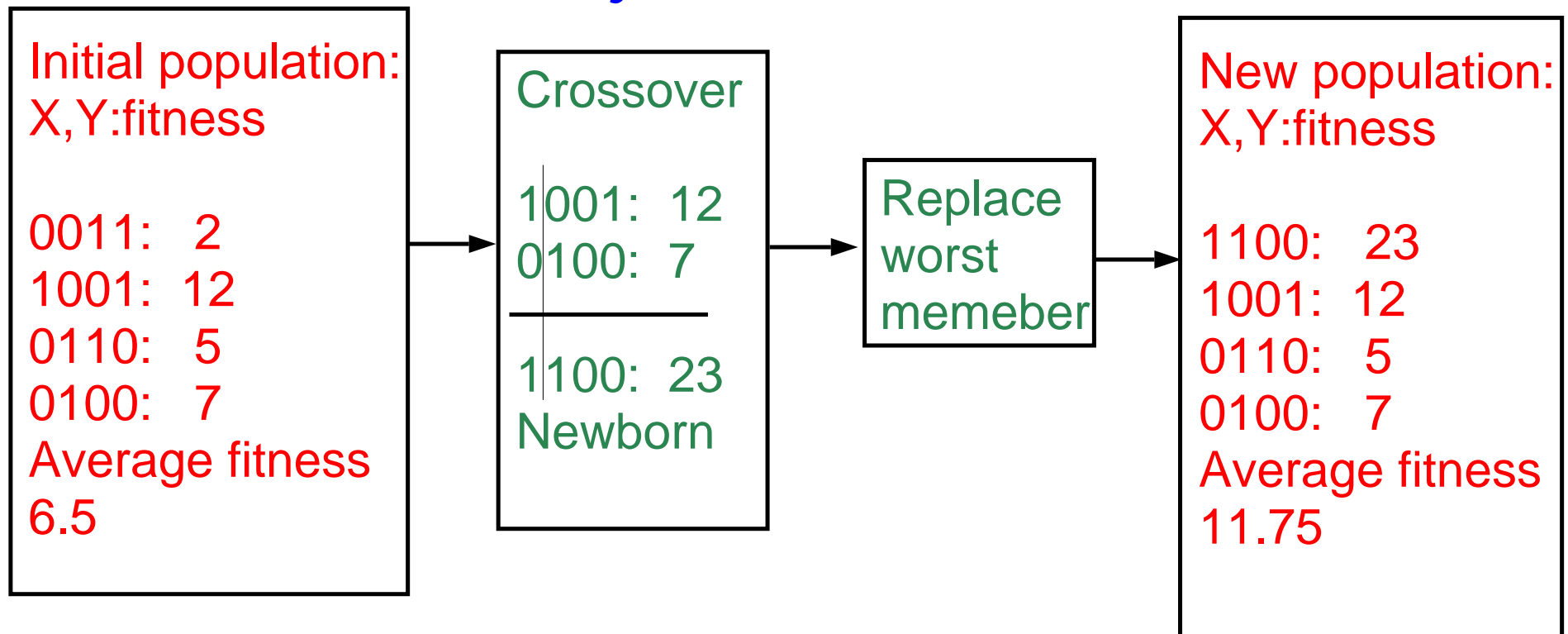
Example: numerical optimization

- maximize $2X^2 - y + 5$ where $X:[0,3], Y:[0,3]$



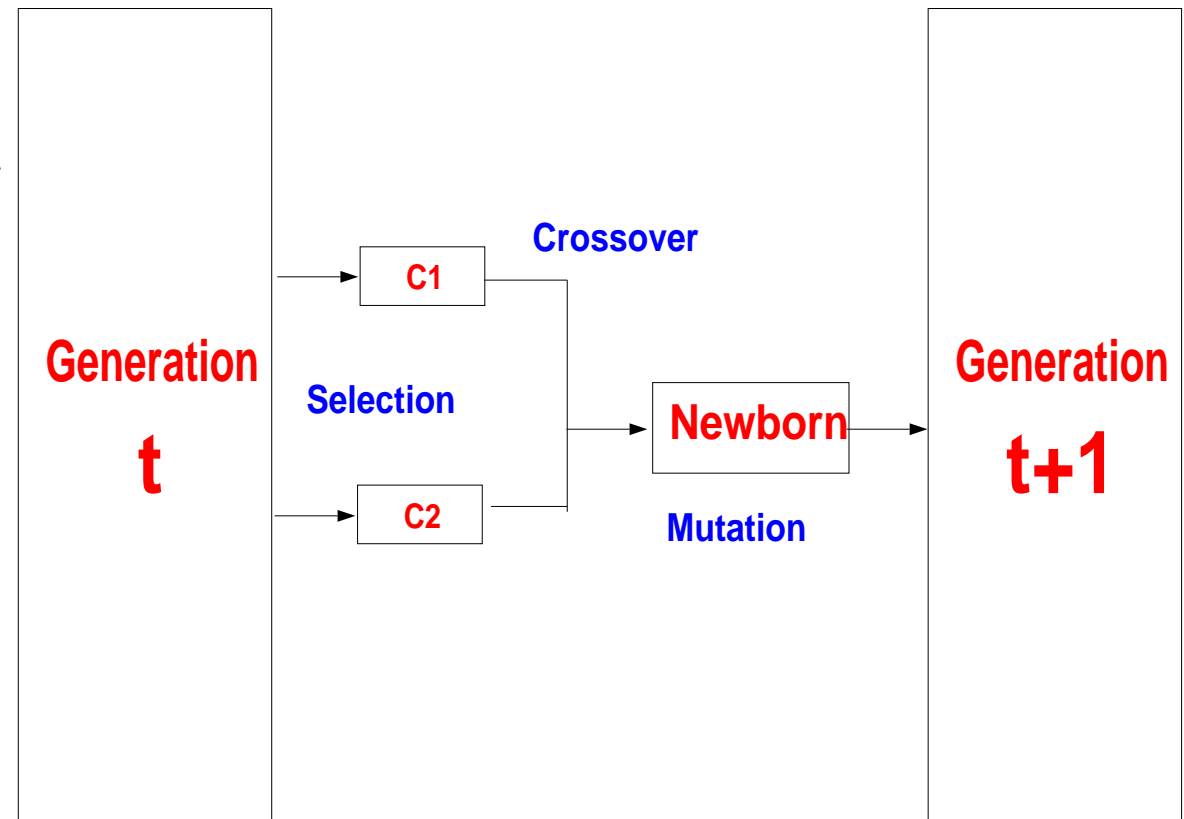
Example with binary representation

- maximize $2X^2 - y + 5$ where $X:[0,3], Y:[0,3]$



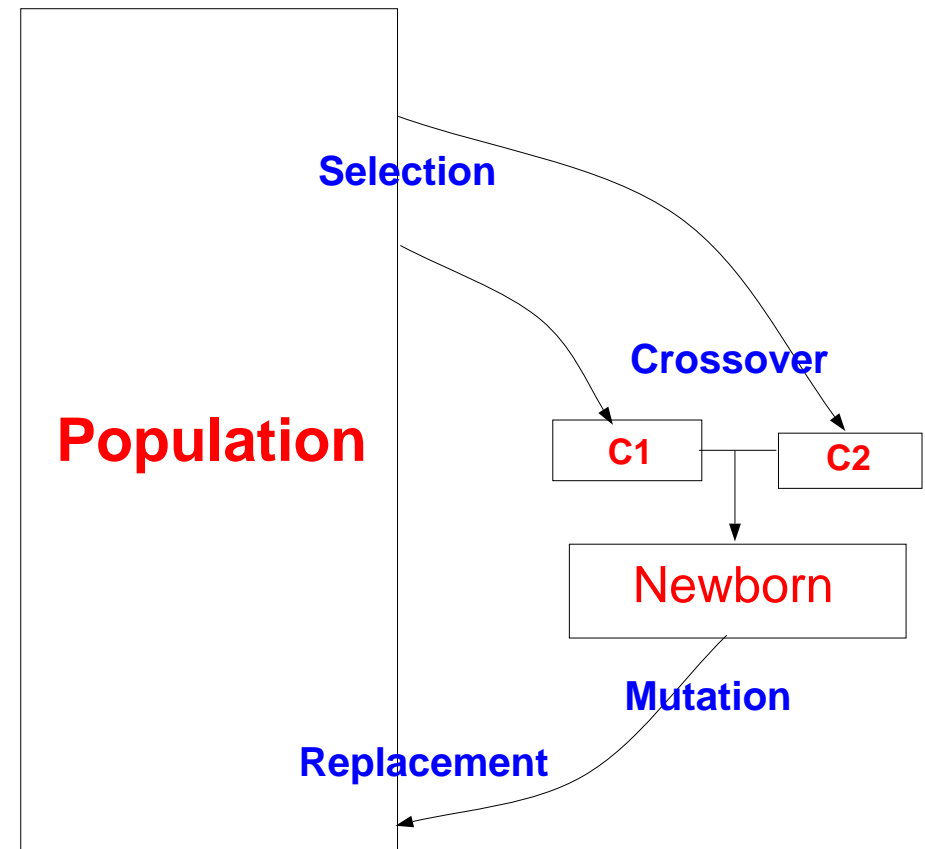
Elements of a generational genetic algorithm

- Representation
- Fitness function
- Initialization strategy
- Selection strategy
- Crossover operators
- Mutation operators



Elements of a steady state genetic algorithm

- Representation
- Fitness function
- Initialization strategy
- Selection strategy
- Crossover operators
- Mutation operators
- Replacement strategy



Selection strategies

- **Proportional selection (roulette wheel)**
 - selection probability of individual = $\text{fitness} / \text{sum of fitnesses}$
- **Rank based selection**
 - Example: decreasing arithmetic/geometric series
 - better when fitness range is very large

Crossover Operators

- **Point crossover (classical)**
 - Parent1=x1,x2,x3,x4,x5,x6
 - Parent2=y1,y2,y3,y4,y5,y6
 - Child =x1,x2,x3,x4,y5,y6
- **Random crossover**
 - Parent1=x1,x2,x3,x4,x5,x6
 - Parent2=y1,y2,y3,y4,y5,y6
 - Child =x1,x2,y3,x4,y5,y6
- **Arithmetic crossover**
 - Parent1=x1,x2,x3
 - Parent2=y1,y2,y3
 - Child = $(x1+y1)/2, (x2+y2)/2, (x3+y3)/2$

Mutation Operators

- change one or more components
- Let Child= $x_1, x_2, P, x_3, x_4 \dots$
- Gaussian mutation:
 - $P \leftarrow P \pm \Delta p$
 - Δp : (small) random normal value
- Uniform mutation:
 - $P \leftarrow P_{new}$
 - p_{new} : random uniform value
- boundary mutation:
 - $P \leftarrow P_{min} \text{ OR } P_{max}$
- Binary mutation=bit flip

Advantages of Genetic-Algorithm based optimization

- Finds global optima
- Can handle discrete, continuous and mixed variable spaces
- Easy to use (short programs)
- Robust (less sensitive to noise, ill conditions)

Disadvantages of Genetic-Algorithm based optimization

- Relatively slower than other methods (not suitable for easy problems)
- Theory lags behind applications

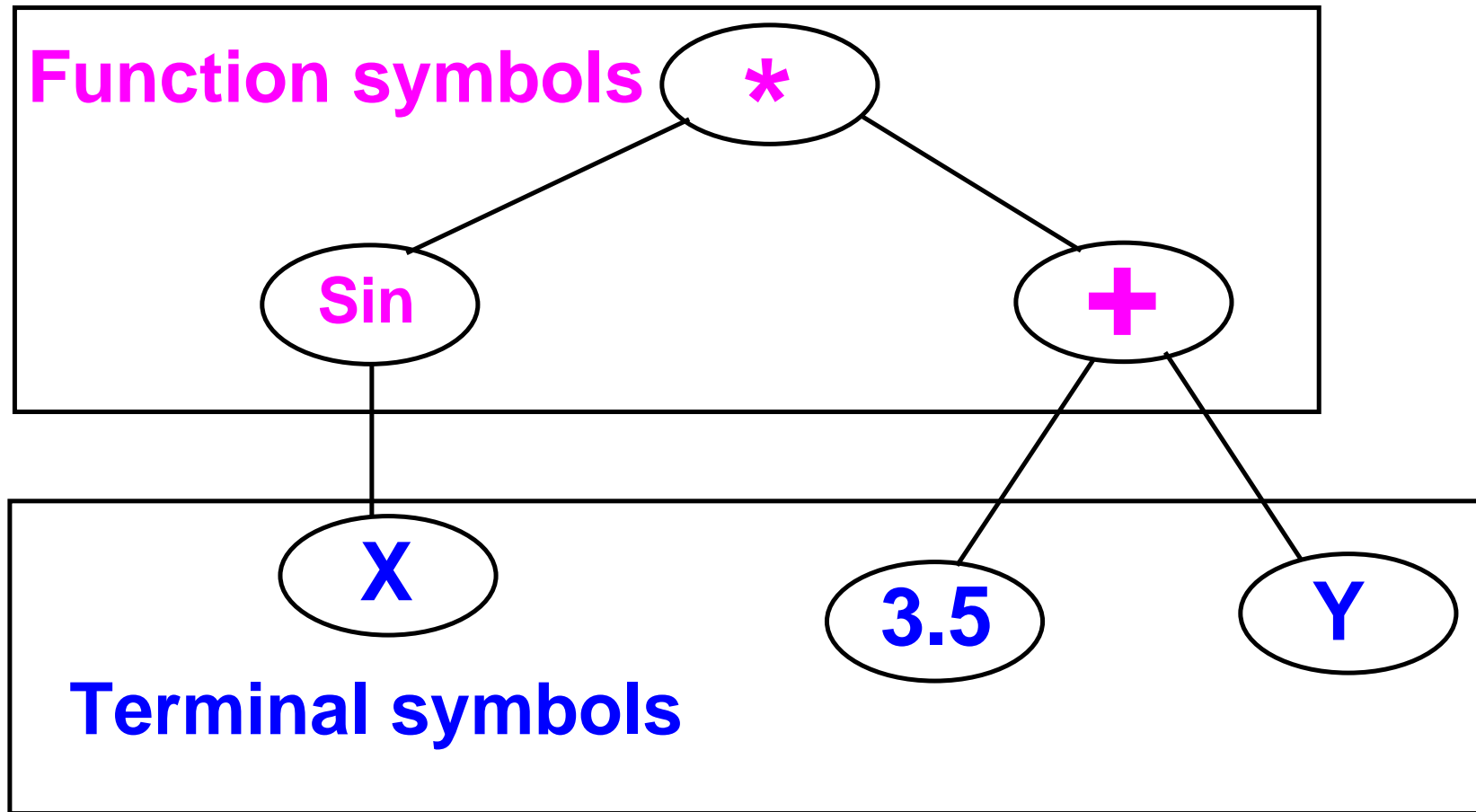
Genetic Programming (GP)

- Introduced (officially) by John Koza in his book (genetic programming, 1992)
- Early attempts date back to the 50s (evolving populations of binary object codes)
- Idea is to evolve computer programs
- Declarative programming languages usually used (Lisp)
- Programs are represented as trees

GP individuals

- A population of trees representing programs
- The programs are composed of elements from the FUNCTION SET and the TERMINAL SET
- These sets are usually fixed sets of symbols
- The function set forms "non-leaf" nodes. (e.g. +,-,*,sin,cos)
- The terminal set forms leaf nodes. (e.g. x,3.7, random())

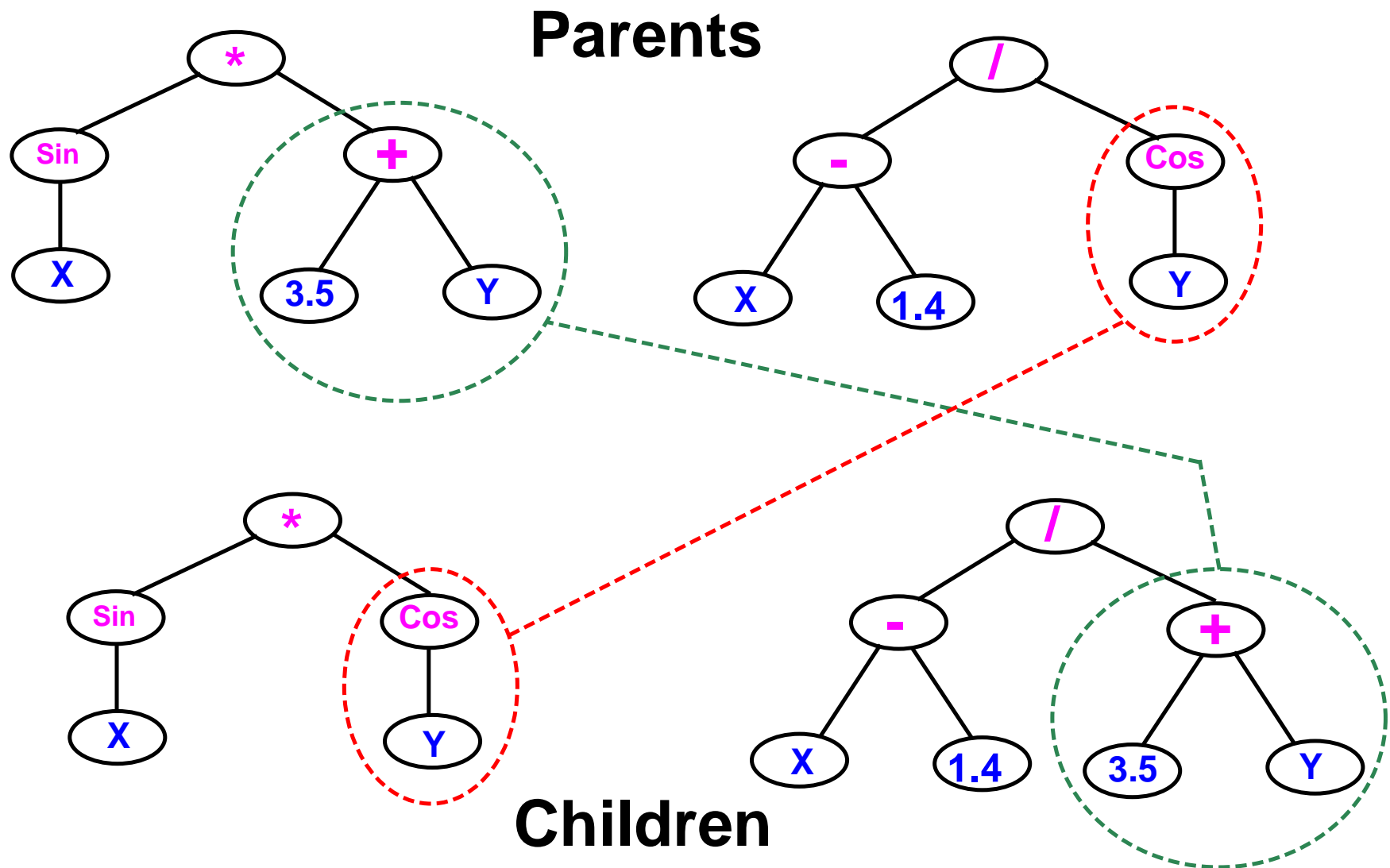
Example: GP individual



GP operation

- Fitness is usually based on I/O traces
- Crossover is implemented by randomly swapping subtrees between individuals
- GP usually does not extensively rely on mutation (random nodes or subtrees)
- GPs are usually generational with a generation gap
- GP usually uses huge populations (1M individuals)

Example: GP crossover



Advantages of GP over GAs

- More flexible representation
- Greater application spectrum
- If tractable, evolving a way to do things is more useful than evolving the things.
- Example: evolving a learning rule for neural networks (Amr Radi, GP98) vs. evolving the weights of a particular NN.

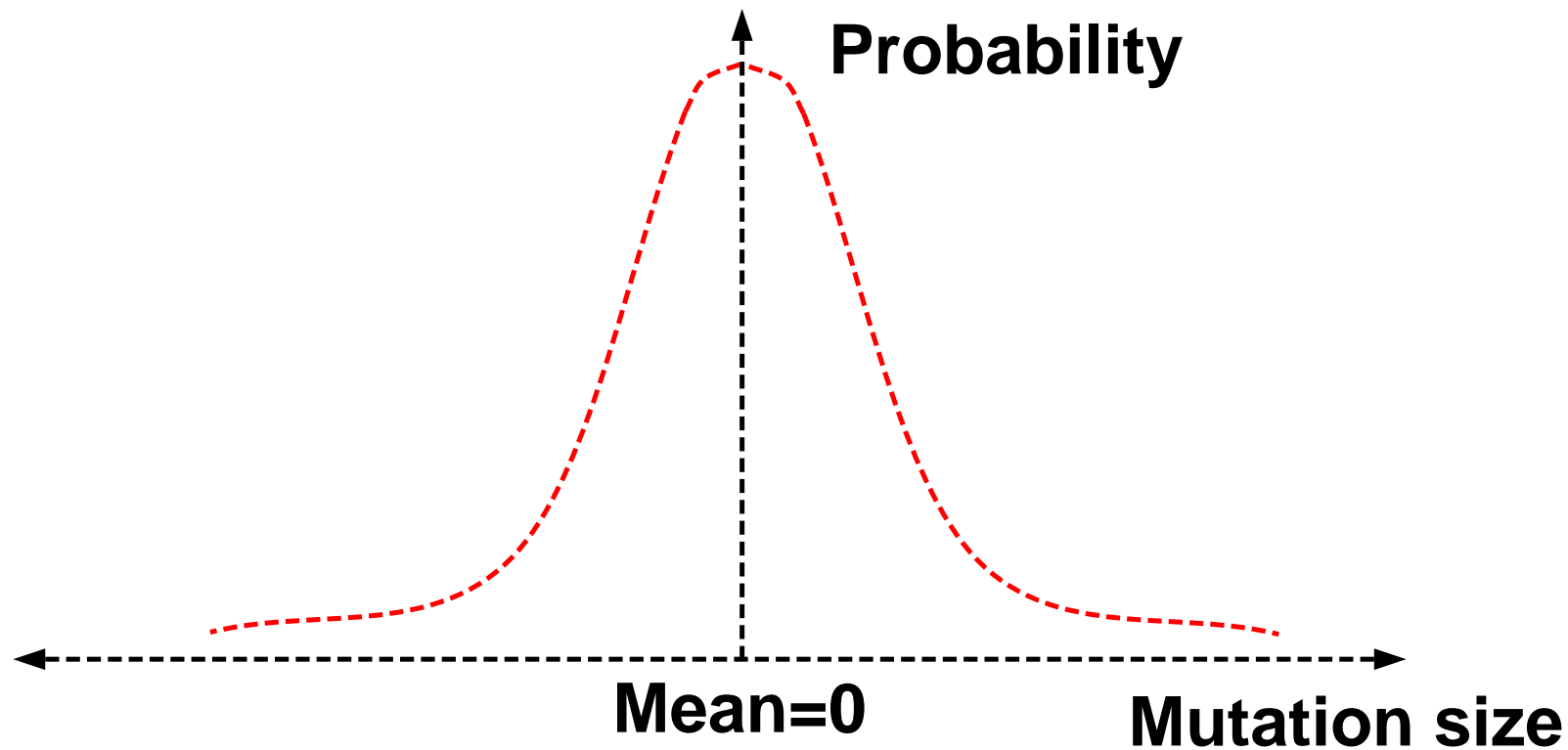
Disadvantages of Genetic Programming

- Extremely slow
- Very poor handling of numbers
- Very large populations needed
- Criticism from classical GA community: no schema theory or any theory!

Evolution Strategies (ES)

- Were invented to solve numerical optimization problems
- Originated in Europe in the 1960s
- **Initially: two-membered or (1+1) ES:**
 - one PARENT generates one OFFSPRING per GENERATION
 - by applying normally distributed (Gaussian) mutations
 - until offspring is better and replaces parent
 - This simple structure allowed theoretical results to be obtained (speed of convergence, mutation size)
- **Later: enhanced to a ($\mu+1$) strategy which incorporated crossover**

Normal (Gaussian) mutation



Modern evolution strategies

- Schwefel introduced the multimembered ESs now denoted by $(\mu+\lambda)$ and (μ,λ)
- (μ,λ) ES: The parent generation is disjoint from the child generation
- $(\mu+\lambda)$ ES: Some of the parents may be selected to "propagate" to the child generation

ES individuals

- **Real valued vectors consisting of two parts:**
 - Object variable: just like real-valued GA individual
 - Strategy variable: a set of standard deviations for the Gaussian mutation
- **This structure allows for "Self-adaptation" of the mutation size**
 - Excellent feature for dynamically changing fitness landscape

Machine learning and evolutionary computation

- In machine learning we seek a good hypothesis
- The hypothesis may be a rule, a neural network, a program ... etc.
- GAs and other EC methods can evolve rules, NNs, programs ...etc.
- Classifier systems (CFS) are the most explicit GA based machine learning tool.

Elements of a classifier system

- **Rule and message system**
 - if <condition> then <action>
- **Apportionment of credit system**
 - Based on a set of training examples
 - Credit (fitness) given to rules that match the example
 - Example: Bucket brigade (auctions for examples, winner takes all, existence taxes)
- **Genetic algorithm**
 - evolves a population of rules or a population of entire rule systems

The Michigan approach: population of rules

- Evolves a population of rules, the final population is used as the rule and message system
- Diversity maintenance among rules is hard
- If done well converges faster
- Need to specify how to use the rules to classify
 - what if multiple rules match example?
 - exact matching only or inexact matching allowed?

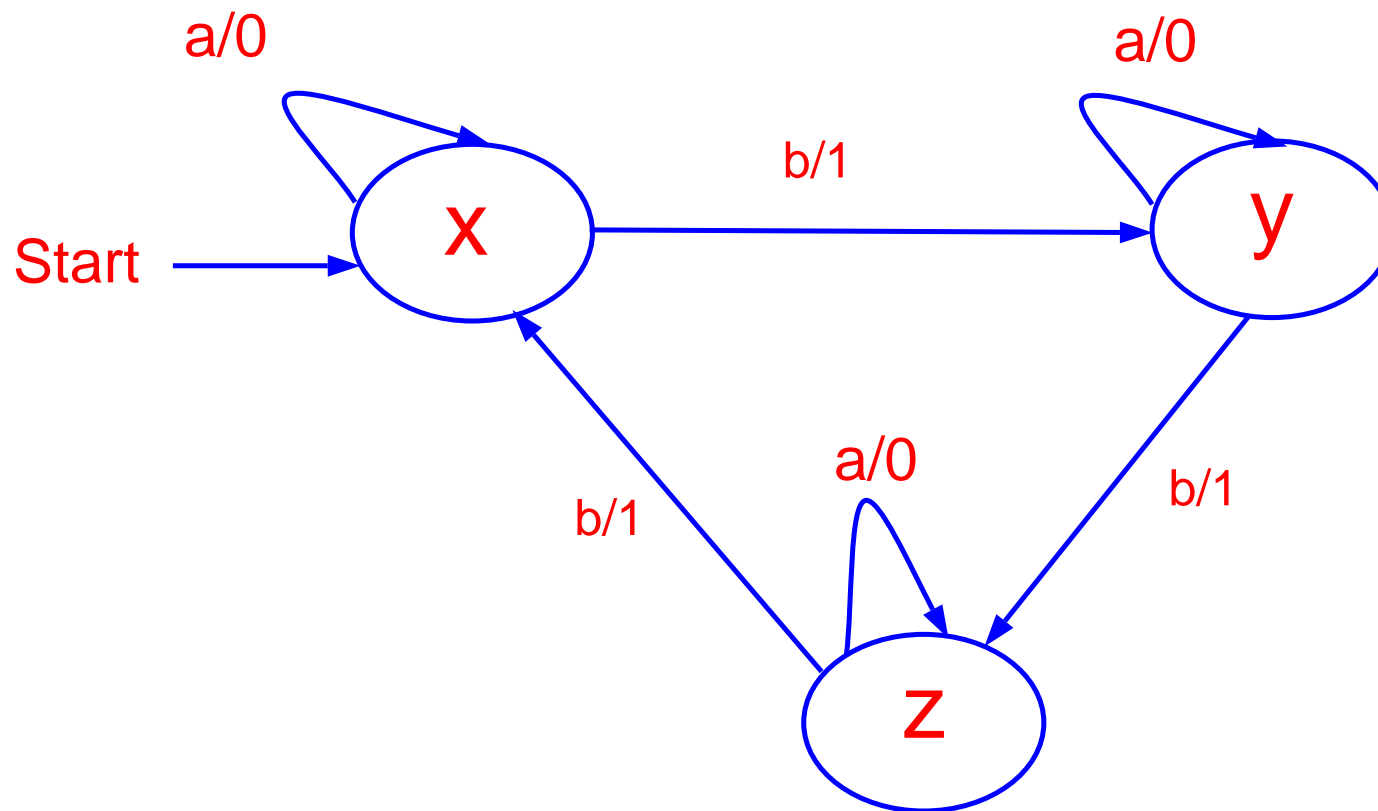
The Pitt approach

- Each individual is a complete set of rules or complete solution
- Avoids the hard credit assignment problem
- Slow because of complexity of space

Evolution programming (EP)

- Evolves finite state machines (or similar structures)
- Relies on mutation
- Fitness based on training sequence(s)
- Uses rank based selection
- Good for sequence problems (DNA) and prediction in time series

EP individual



EP mutation operators

- Add a state (with random transitions)
- Delete a state (reassign state transitions)
- Change an output symbol
- Change a state transition
- Change the start state

The bigger picture

- All evolutionary computation models are getting closer to each other
- The choice of method is important for success
- EC provides a very flexible architecture
 - easy to combine with other paradigms
 - easy to inject domain knowledge

EC journals

- Evolutionary Computation
- IEEE transactions on evolutionary computation
- Genetic programming
- other: AIEDAM, AIENG ...

EC conferences

- Genetic and evolutionary computation conference (GECCO)
- Congress on evolutionary computation (CEC)
- evolutionary programming (EP)
- Parallel problem solving from nature (PPSN)
- other: AI in design, IJCAI, AAAI ...