Local and Global Search

Parts adapted from:

- Chapter 4 of AI 2E by David Poole and Alan Macworth;
- AI a modern approach by Stuart Russel and Peter Norvig

Optimisation problems

Optimisation problem:

- A set of variables and their domains
- An objective function

Find an assignment that optimises (maximise / minimises) the value of the objective function.

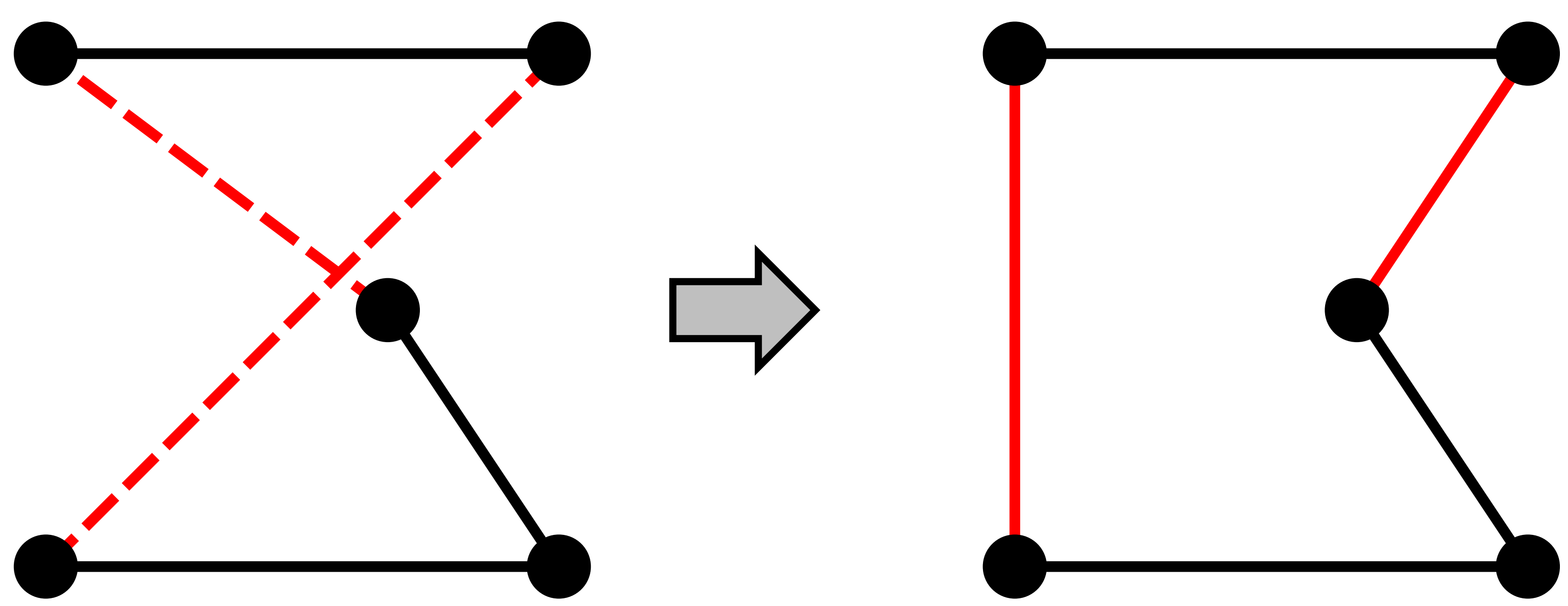
A **constrained optimisation problem**, in addition to the above, has a set of constraints that determine what assignments are allowed. In a constrained optimisation problem, the goal is to find an assignment that satisfies the constraints and optimises the objective function.

Local search

- Optimisation usually involves searching.
- Like CSP, the path is irrelevant; only the solution matters.
- In local search we use algorithms that *iteratively improve* a state.
- We keep a single current state, and in each iteration we try to improve it by moving to one of its neighbours.
- This takes constant space.
- The goal is to find an optimal state.
- Most local search algorithms are greedy.
- Two common local search algorithms are hill climbing (greedy ascent for maximisation) and greedy descent (for minimisation).

Example: Traveling Salesperson Problem

- Start with any complete tour, and perform pairwise exchanges
- Variants of this approach get very close to optimal solution very quickly with large numbers of cities.



Local search for CSPs

- A constrained satisfaction problem (CSP) can be reduced to an optimisation problem.

- Aim is to find an assignment with zero unsatisfied constraints.

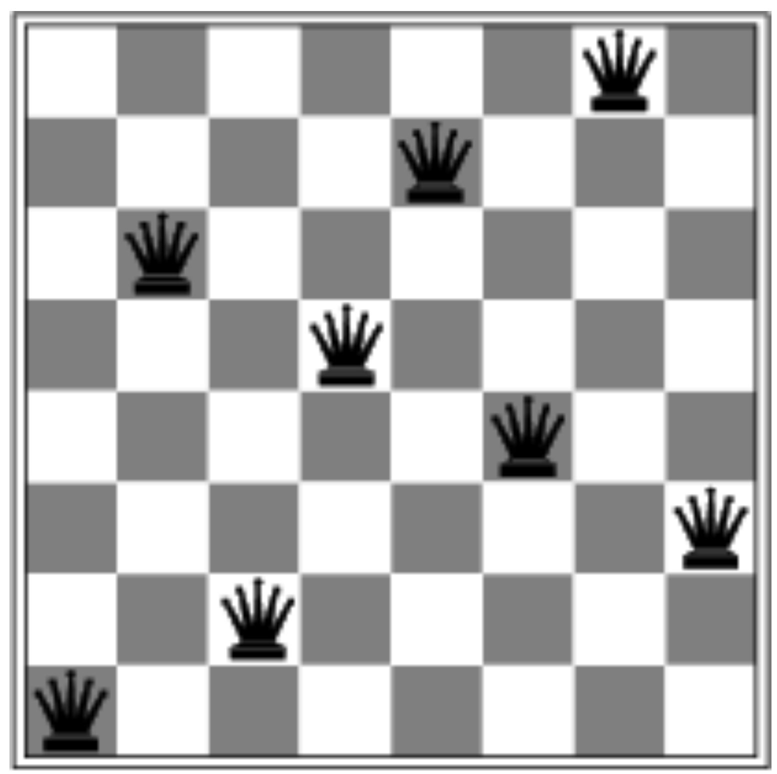
- Given an assignment of a value to each variable, a **conflict** is an unsatisfied constraint.

- The goal is an assignment with zero conflicts.

- Heuristic function to be minimised: the number of conflicts.

Local search: *n*-queens problem

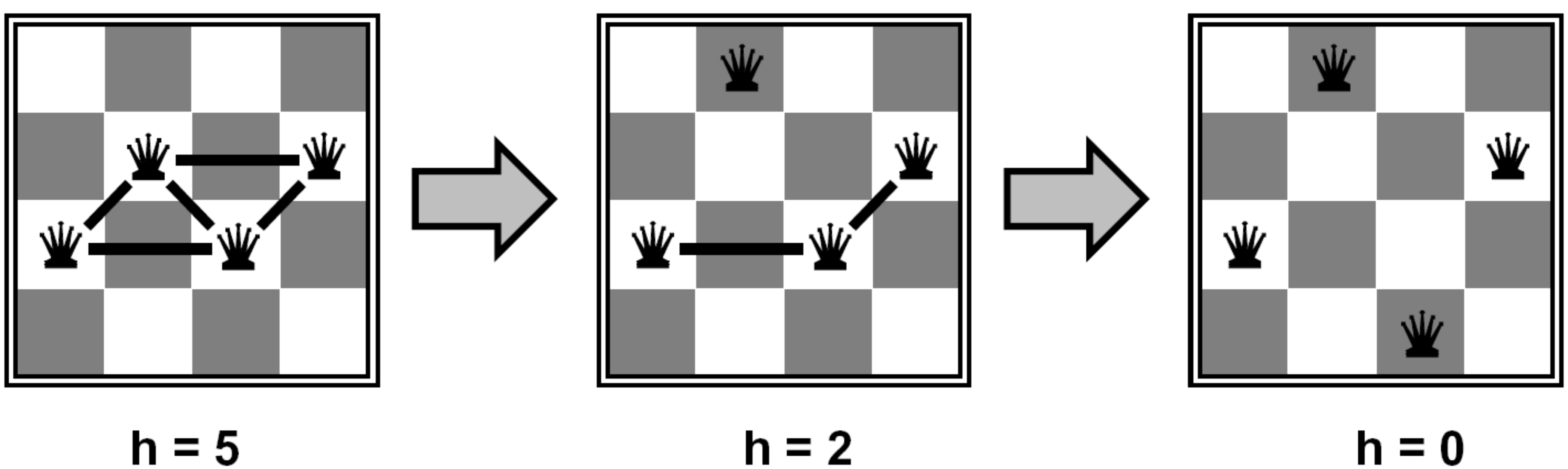
- Aim: Put n queens on an $n \times n$ board with no two queens attacking each other.
- The objective (heuristic) function to minimise: number of conflicts.



$h = 1$

Example: 4-Queens

- States: 4 queens in 4 columns (44 = 256 states)
- Obtaining neighbours: move queen in column
- Objective function to minimise: h(n) = number of pairs of queens that are attacking each other (number of conflicts)



Example: neighbours

- Objective function (conflict count): number of pairs of queens that are attacking each other.

- Number of conflicts in the current state: 17

Some Variants of Greedy Descent

- Find the variable-value pair that minimises the number of conflicts at every step.

- Select a variable that participates in the most number of conflicts. Select a value that minimises the number of conflicts.

- Select a variable that appears in any conflict. Select a value that minimises the number of conflicts.

Local Search Issues

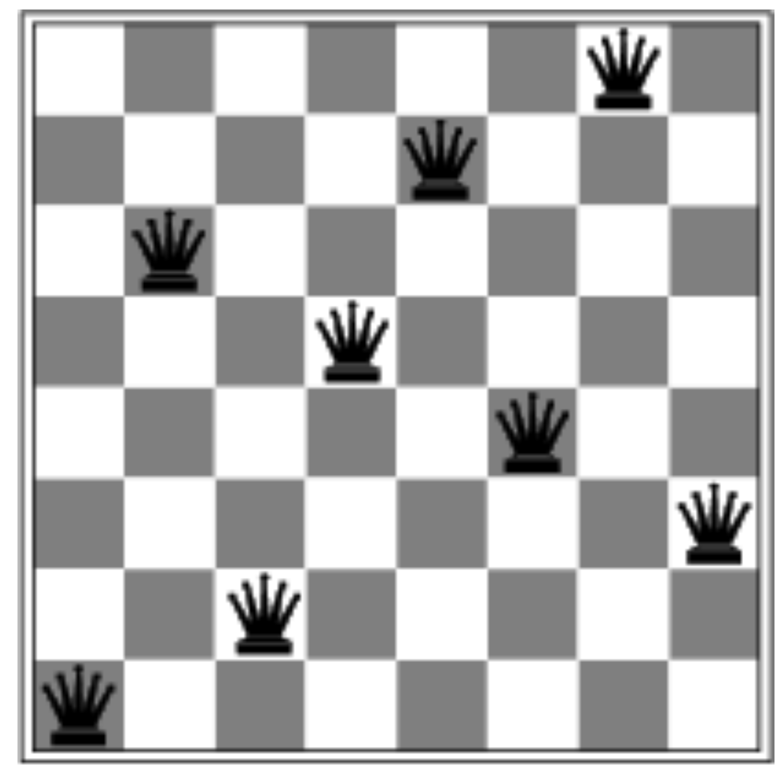
- Local search can get stuck in local optima or flat ares of the landscape of the objective function.

- Randomised greedy descent can help sometimes:

  - **random step**: move to a random neighbour.

  - **random restart**: reassign random values to all variables.

  - these make the search global.



a local minimum with a conflict count of 1.

Parallel search

- A total assignment is called an **individual**.
- Idea: maintain a population of *k* individuals instead of one.
- At every stage, update each individual in the population. Whenever an individual is a solution, it can be reported.
- Like *k* restarts, but uses *k* times the minimum number of steps.
- A basic form of global search.

Simulated Annealing

- Pick a variable at random and a new value at random.
- If it is an improvement, adopt it.
- If it isn't an improvement, adopt it probabilistically depending on a temperature parameter, T.
  - With current assignment n and proposed assignment n' we move to n' with probability $e^{(h(n)-h(n'))/T}$
- Temperature can be reduced.

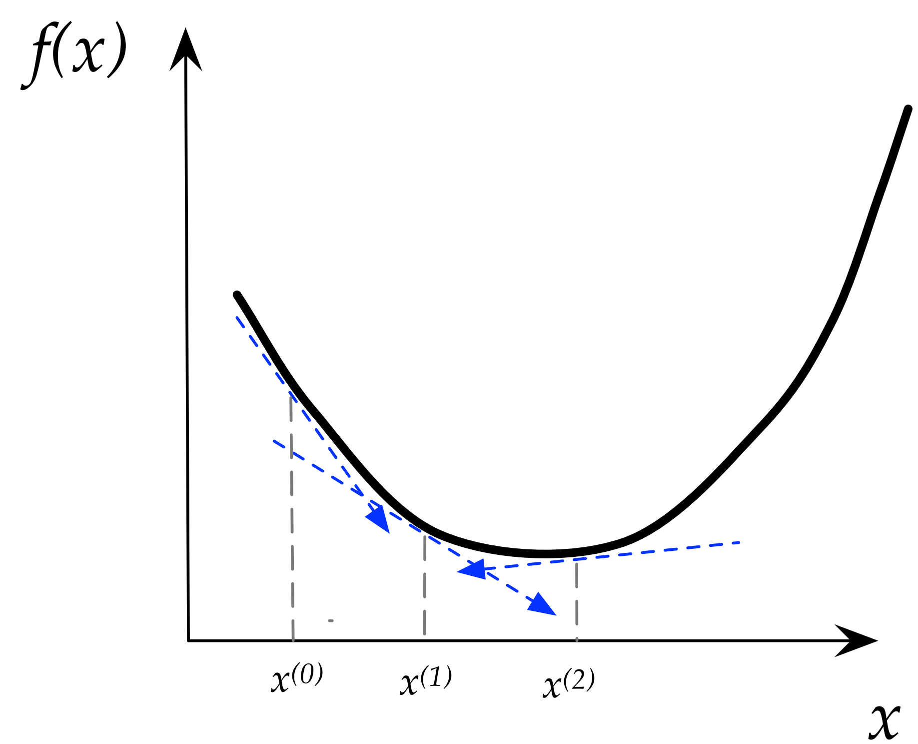
Probability of accepting a change:

Temperature	1-worse	2-worse	3-worse
10	0.91	0.81	0.74
1	0.37	0.14	0.05
0.25	0.02	0.0003	0.000005
0.1	0.00005	0	0

Gradient Descent

- A widely-used local search algorithm in numeric optimisation (e.g. in machine learning)
- Used when the variables are numeric and continues.
- The objective function must be differentiable (mostly).

1: Guess $\mathbf{x}^{(0)}$, set $k \leftarrow 0$
2: **while** $||\nabla f(\mathbf{x}^{(k)})|| \geq \epsilon$ **do**
3: $\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$
4: $\quad k \leftarrow k + 1$
5: **end while**
6: **return** $\mathbf{x}^{(k)}$



Evolutionary Algorithms

References:

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing, Springer

K. A. De Jong, Evolutionary Computation, MIT Press

J. C. Spall
Introduction to Stochastic Search and Optimization, John Wiley and Sons

Genetic (Evolutionary) Algorithms

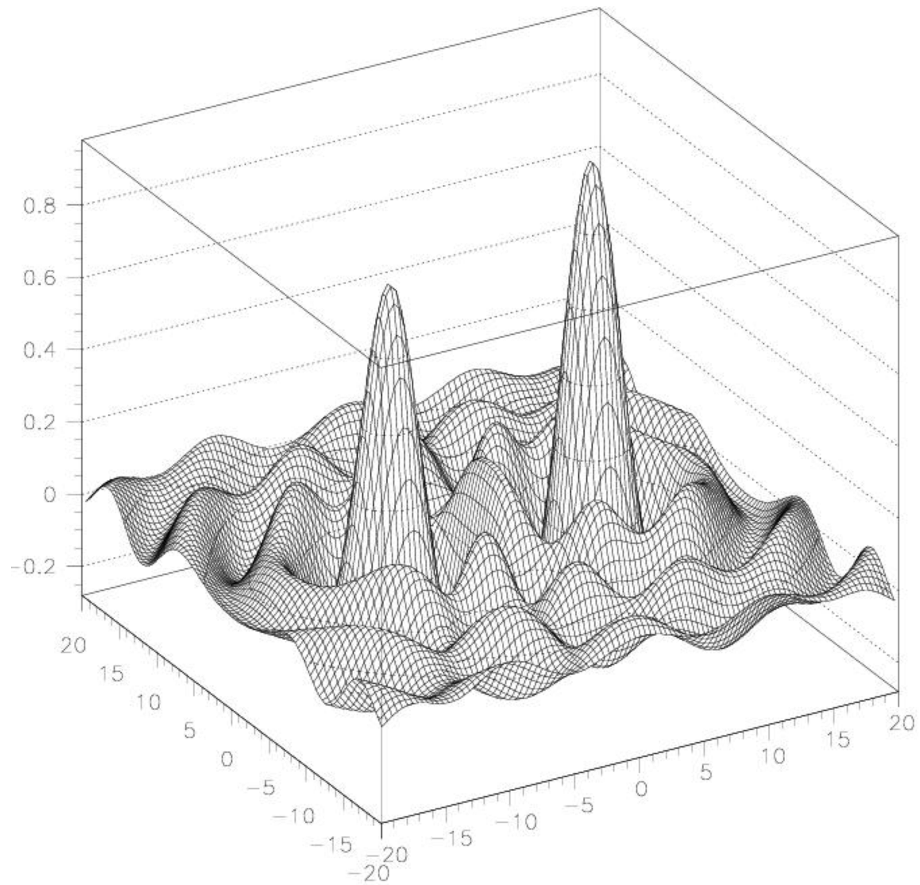
- Inspired by natural selection
- A form of global search
- Requires:
  - Representation
  - Evaluation function
  - Selection of parents
  - Reproduction operators
  - Initialisation procedure
  - Parameter settings

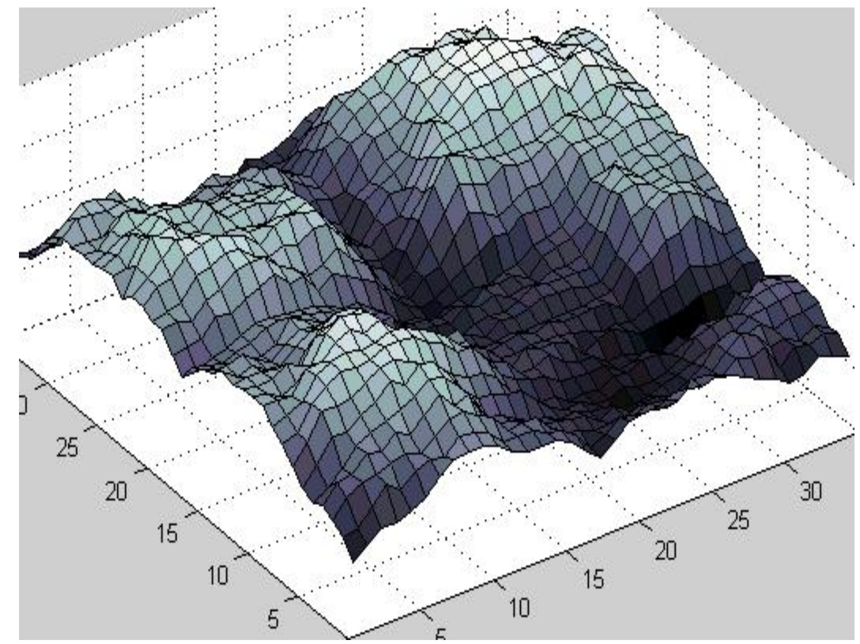
Evaluation: Fitness Function

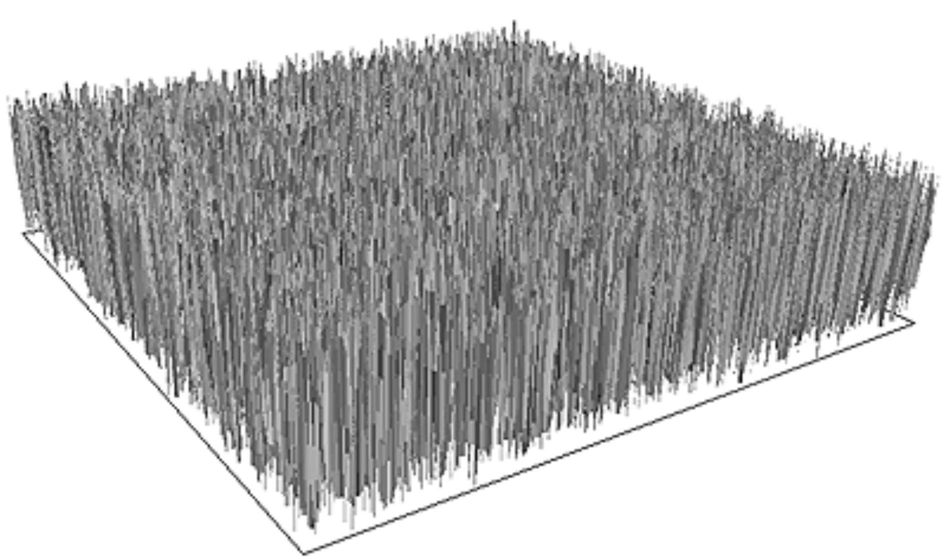
Purpose:

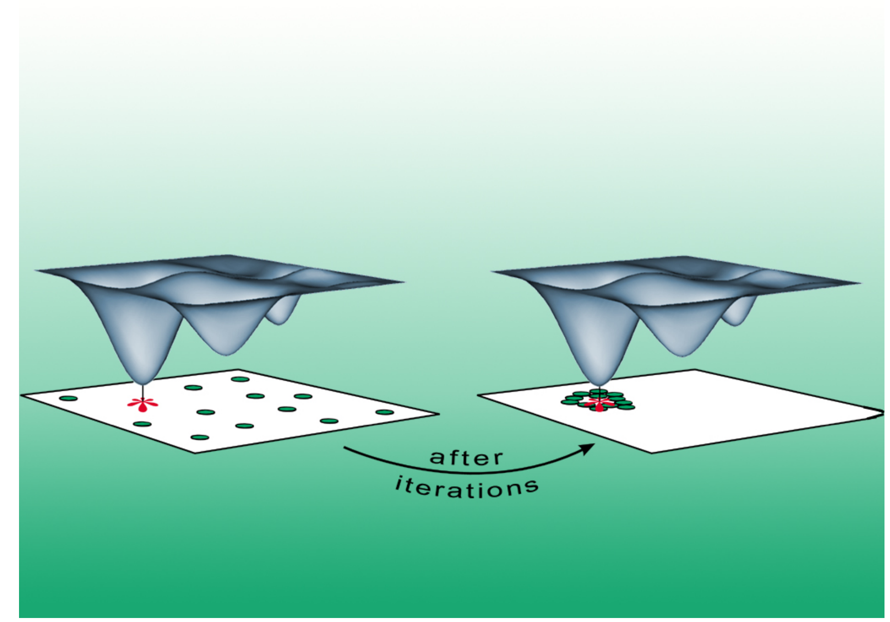
- Parent selection
- Measure for convergence
- For Steady state: Selection of individuals to die
- Should reflect the value of the chromosome in some “real” way
- It is a critical part of any EA / GA

Fitness landscapes

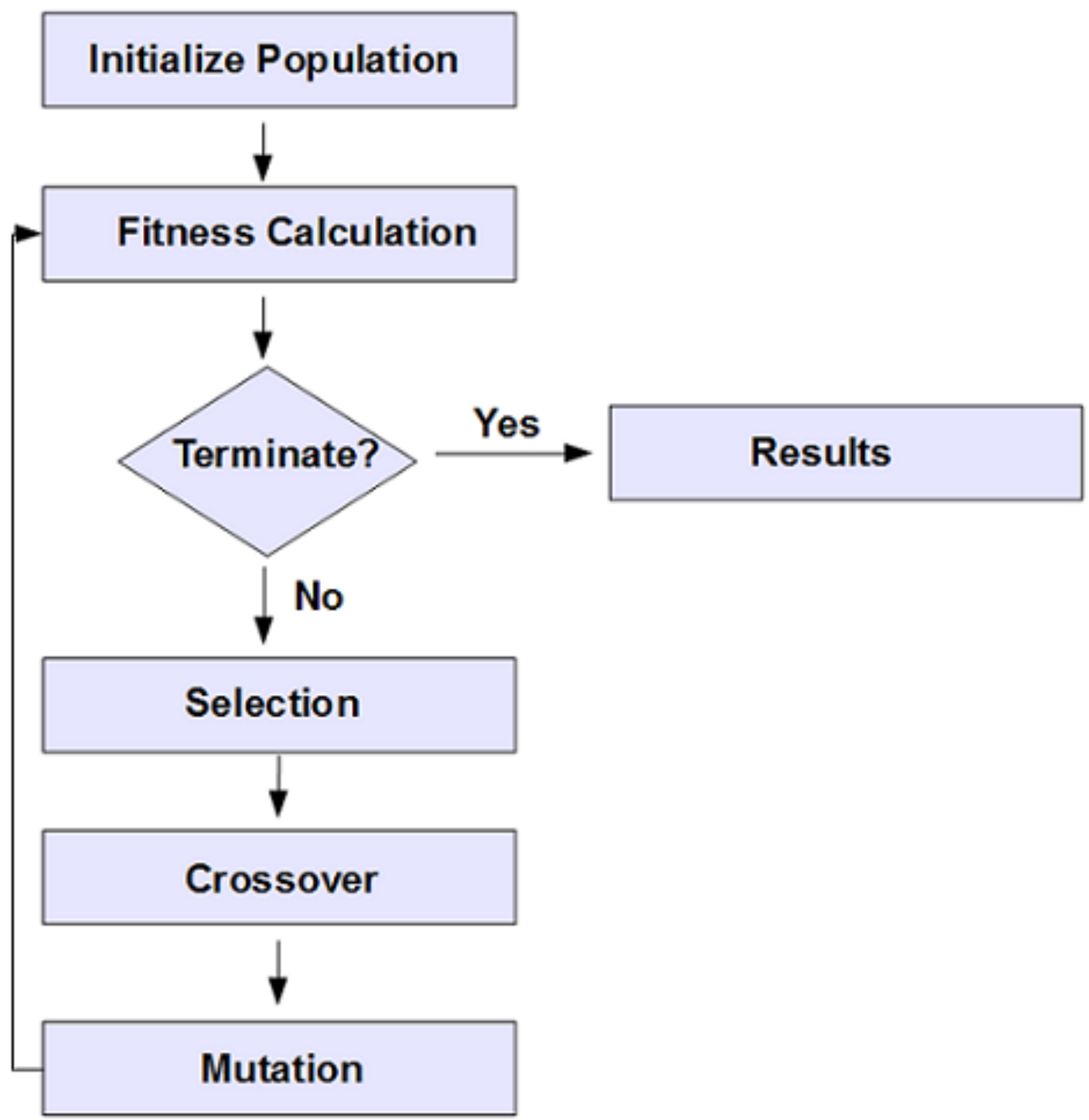








GA: Flowchart



Selection

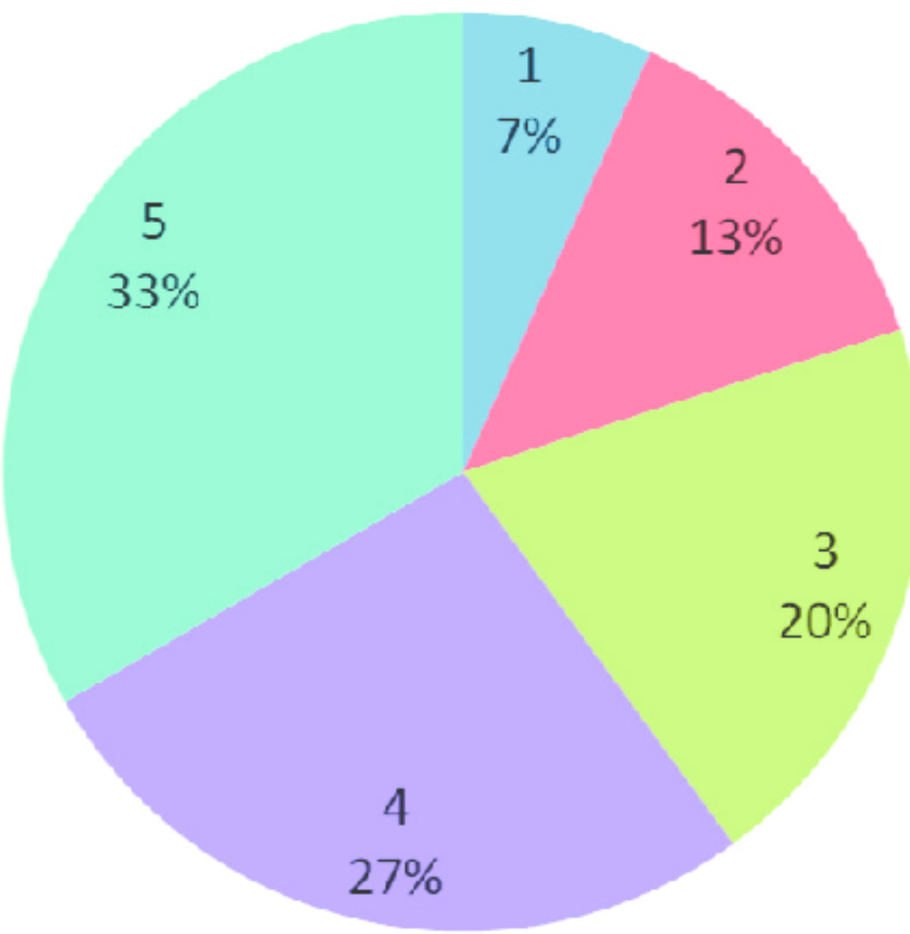
Main idea: better individuals should have higher chance of surviving and breeding.

Types:

- Roulette wheel selection
- Tournament selection
- … any mechanism that somehow overall achieves the main idea.

Roulette Wheel Selection

- Chances proportional to fitness
- Assign to each individual a part of the roulette wheel
- Spin the wheel *n* times to select *n* individuals



Individual	Fitness
1	1.0
2	2.0
3	3.0
4	4.0
5	5.0

Roulette Wheel Selection: Example

- Sum the fitness of all individuals, call it T
- Generate a random number N between 1 and T
- Return individual whose fitness added to the running total is equal to or larger than N
- Chance to be selected is exactly proportional to fitness
- Individual: 1, 2, 3, 4, 5, 6
- Fitness: 8, 2, 17, 7, 4, 11
- Running total: 8, 10, 27, 34, 38, 49
- N: 23
- Selected: 3

Selection: Tournaments

- *n* individuals are randomly chosen; the fittest one is selected as a parent.
- *n* is the “size” of the tournament.
- By changing the size, selection pressure can be adjusted.

Elitism

- Widely used in population models
- Always keep at least one copy of the fittest solution so far
- Results in non-decreasing (maximum) fitness over generation

Reproduction Operators

Crossover

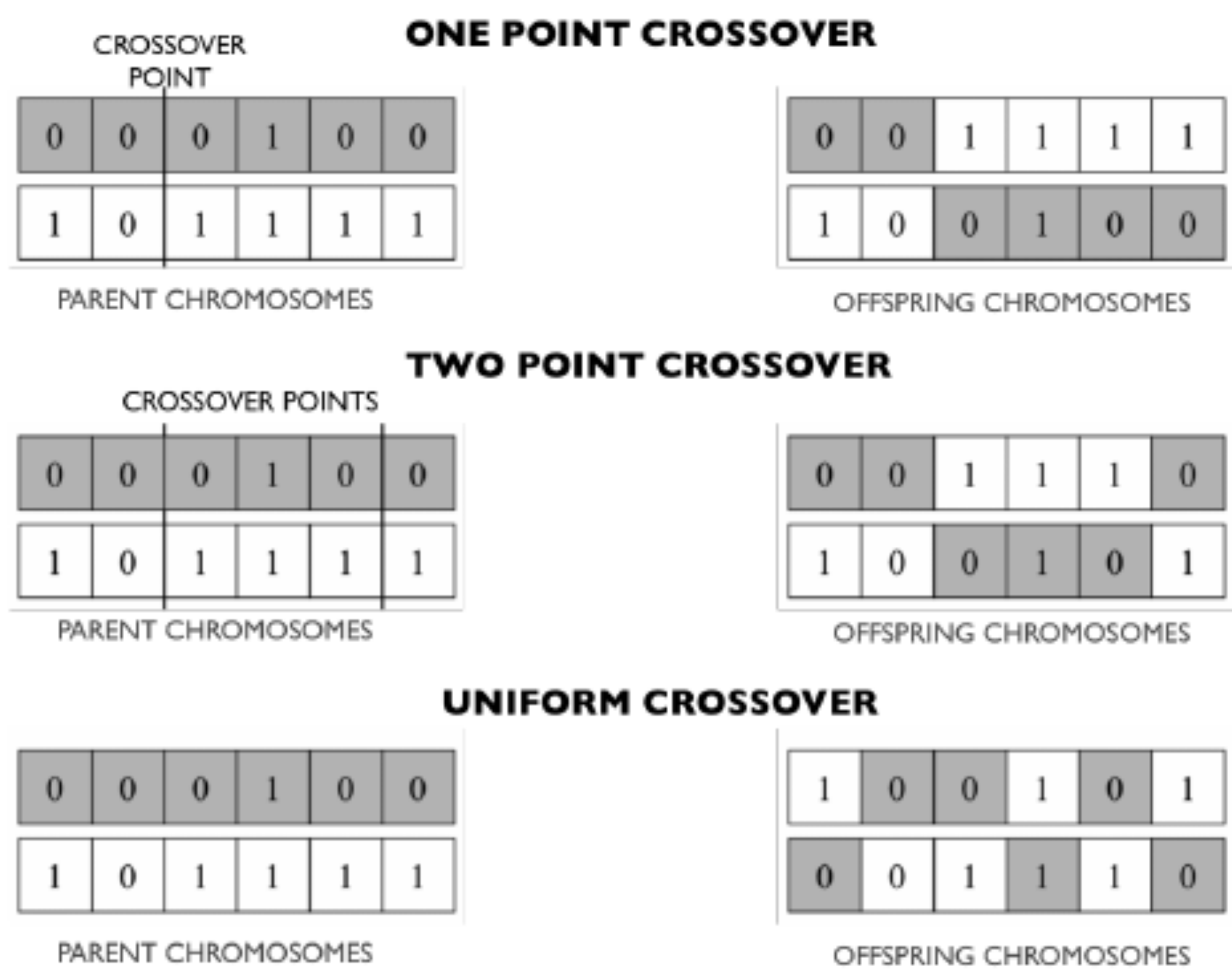
- Two parents produce two offspring
- There is a chance that the chromosomes of the two parents are copied unmodified as offspring
- There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring
- Typically the chance of crossover is between 0.6 and 1.0
- Types: 1-point, 2-point, Uniform, …

Mutation

- There is a chance that a gene of a child is changed randomly
- Typically the chance of mutation is low (e.g. 0.001)

Crossover

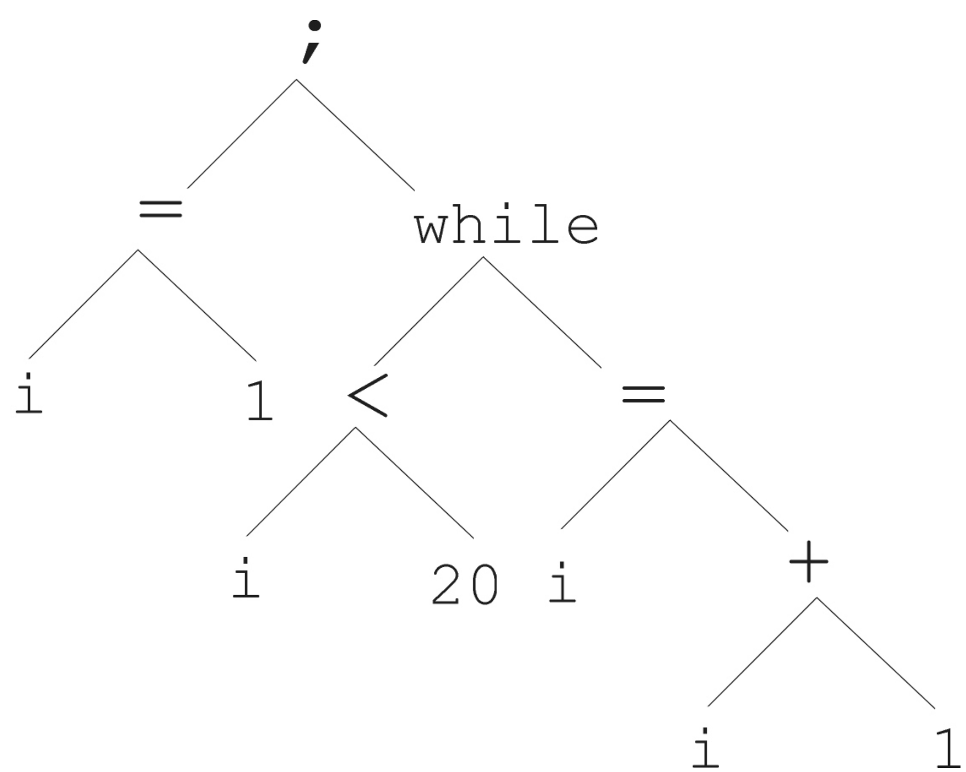
- Generate 1, 2, or a number of random crossover points.
- Split the parents at these points.
- Create offsprings by exchanging alternate segments.

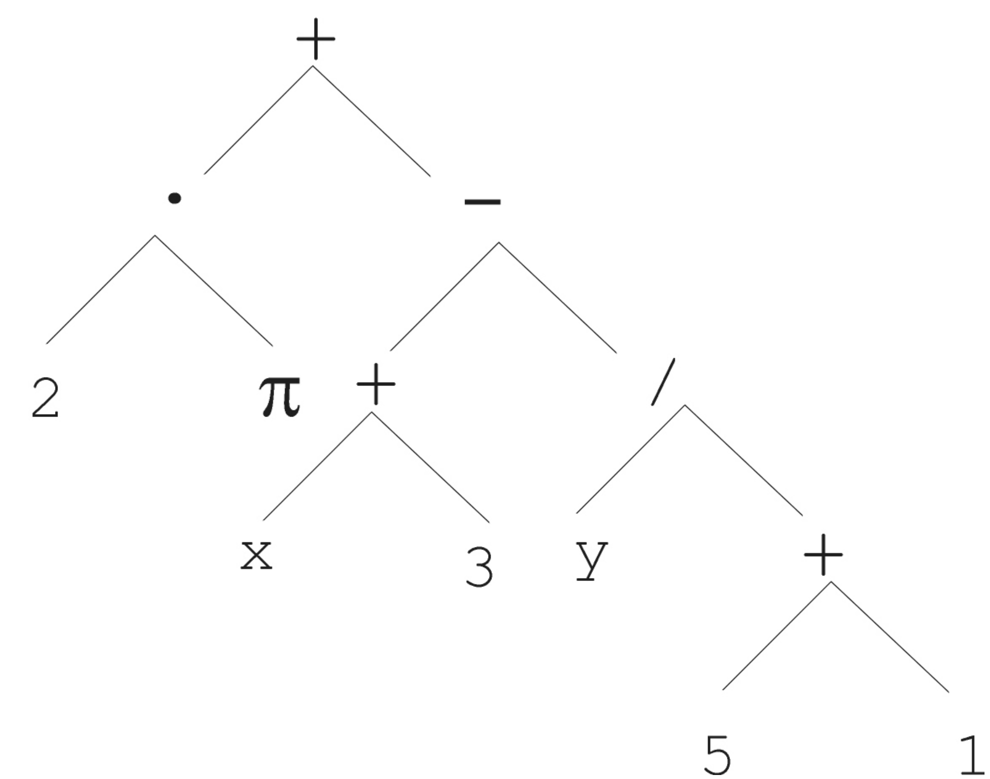


Crossover or Mutation?

- Purpose of crossover: combining somewhat good candidates in the hope of producing better children
- Purpose of mutation: bring diversity (new ideas!)
- Decade long debate: which one is better/necessary?
- A rather wide agreement:
    - it depends on the problem, but
    - in general, it is good to have both.
    - mutation-only-EA is possible, crossover-only-EA would not work.

Tree representation



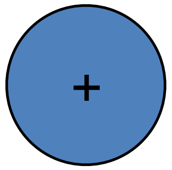



```
i =1;
while (i < 20)
{
        i = i +1
}
```

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

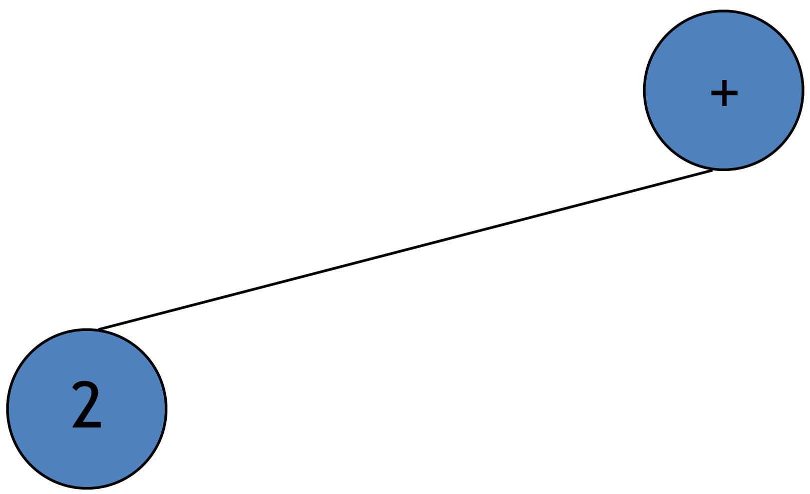
Randomly Generating Programs

(+ …)



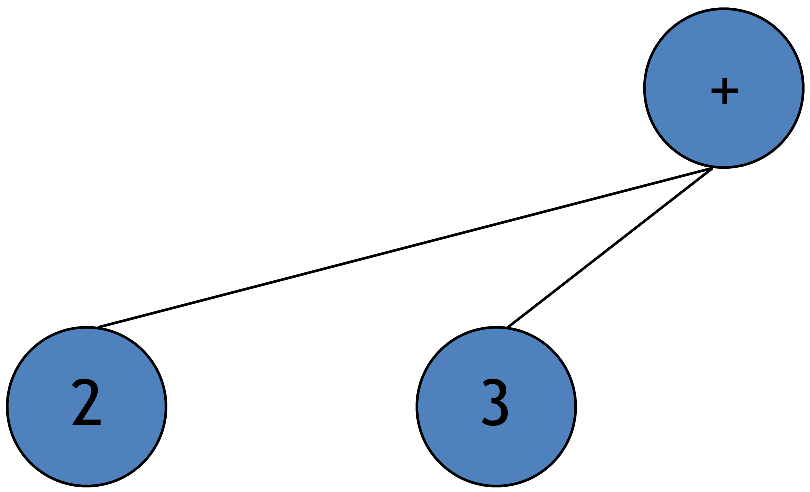
Randomly Generating Programs

(+ 2 …)



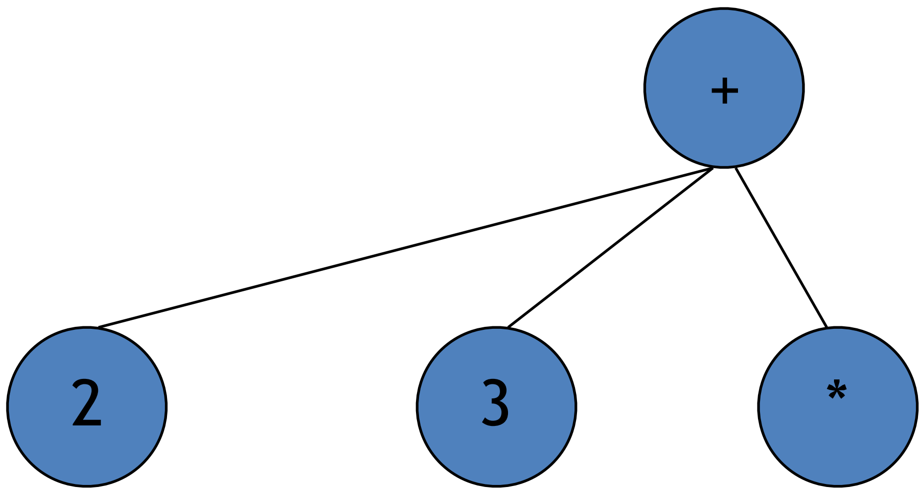
Randomly Generating Programs

(+ 2 3 …)



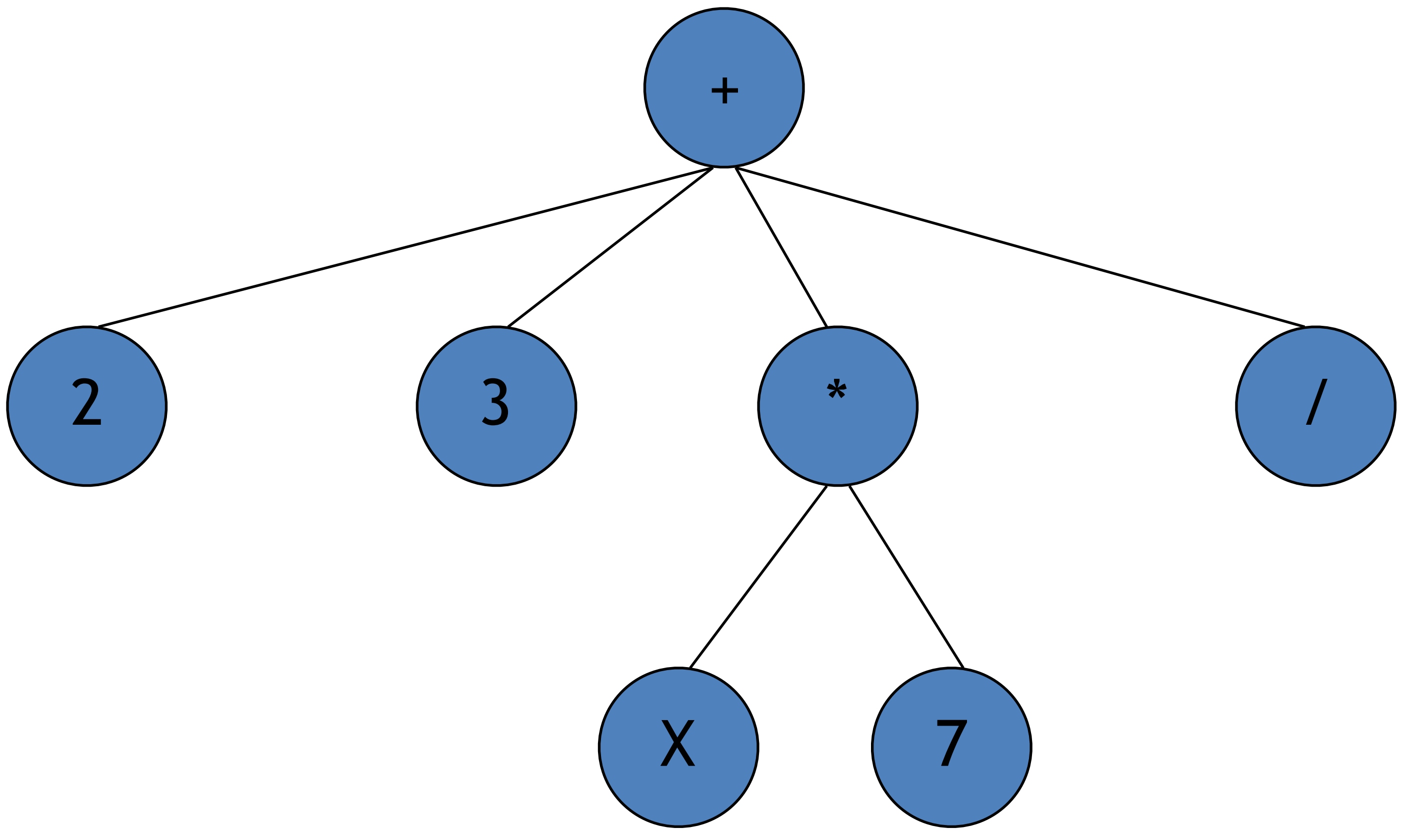
Randomly Generating Programs

(+ 2 3 (* …) …)



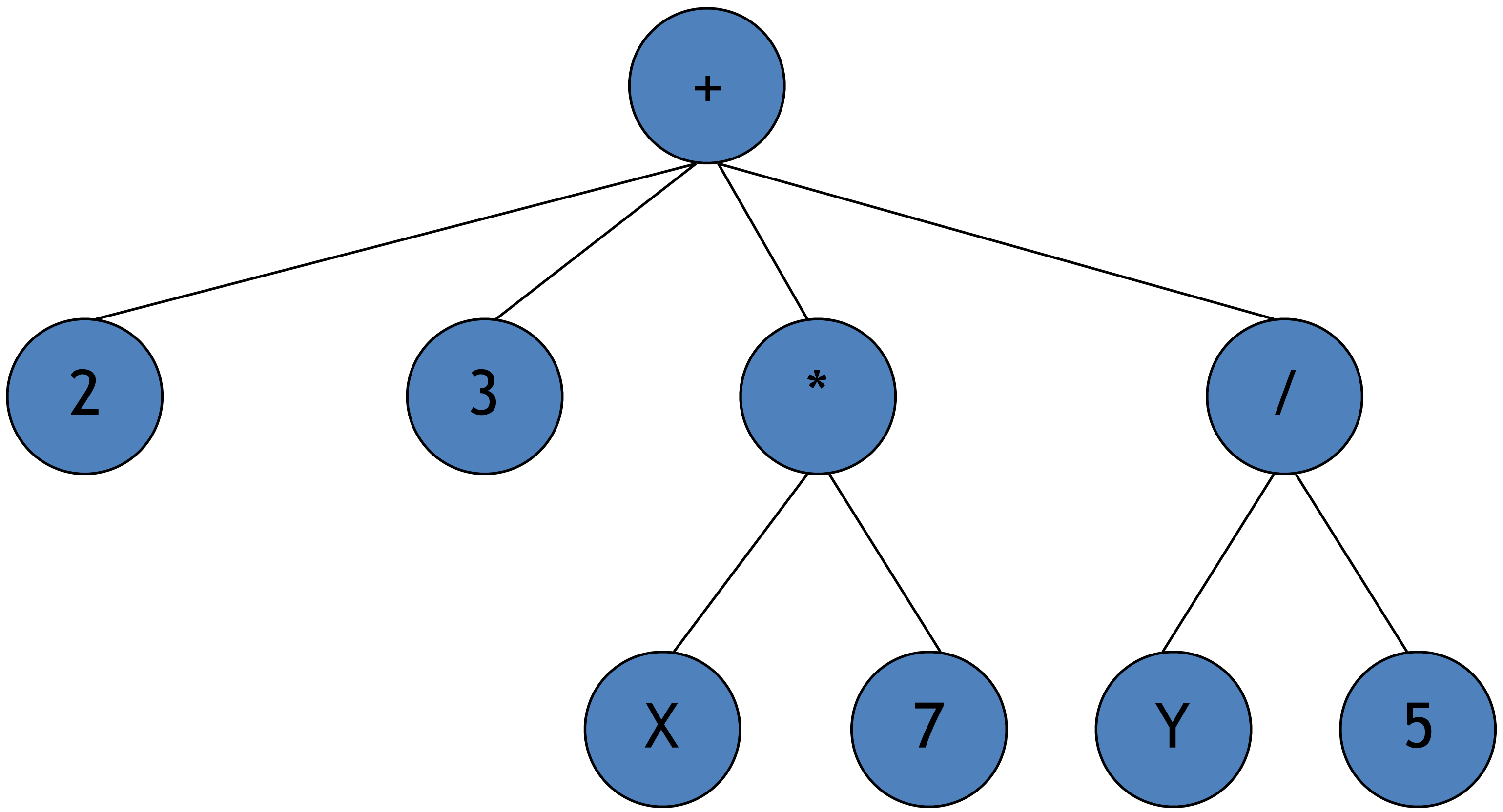
Randomly Generating Programs

(+ 2 3 (* X 7) (/ …))



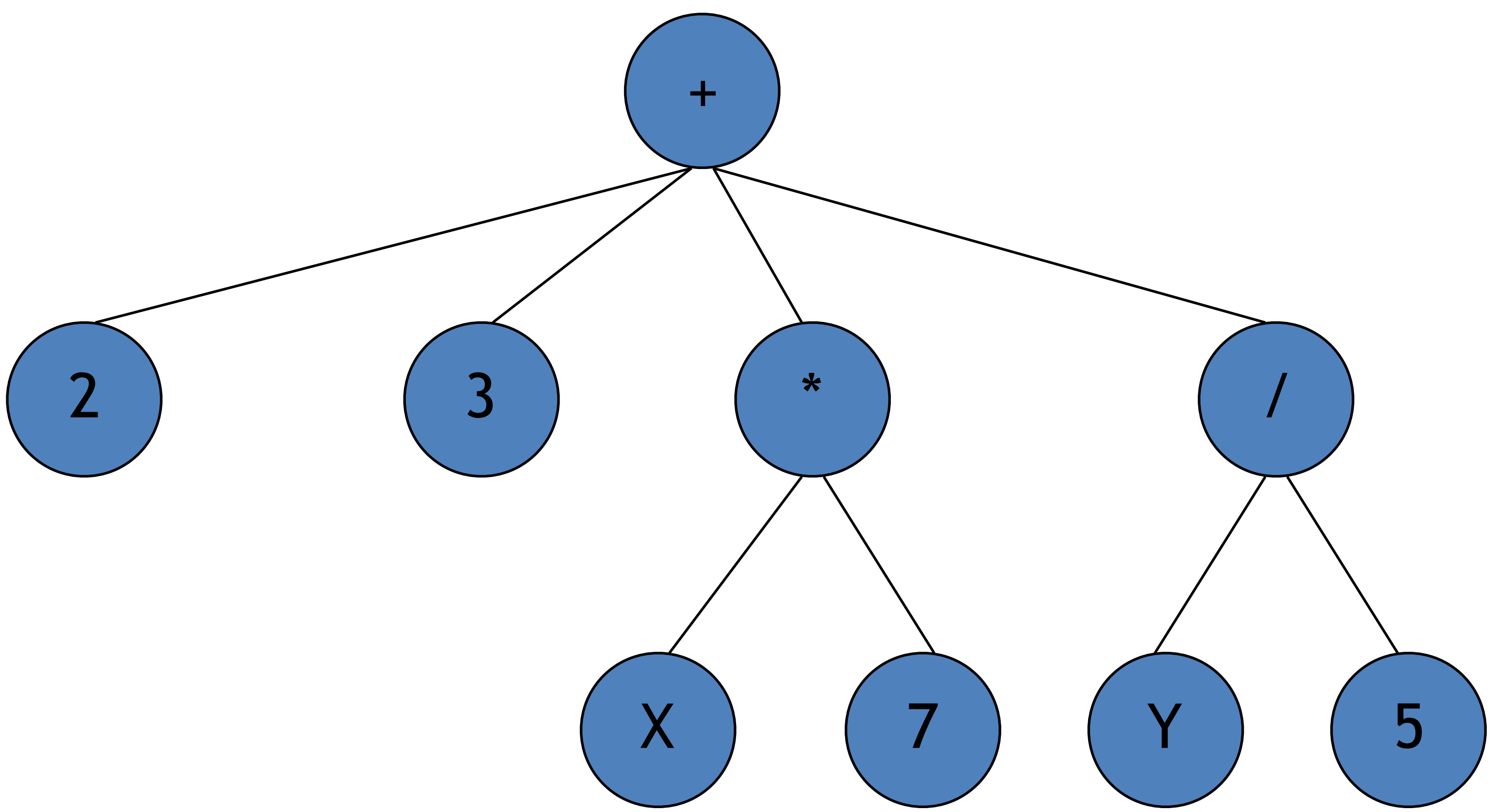
Randomly Generating Programs

(+ 2 3 (* X 7) (/ Y 5))



Mutation

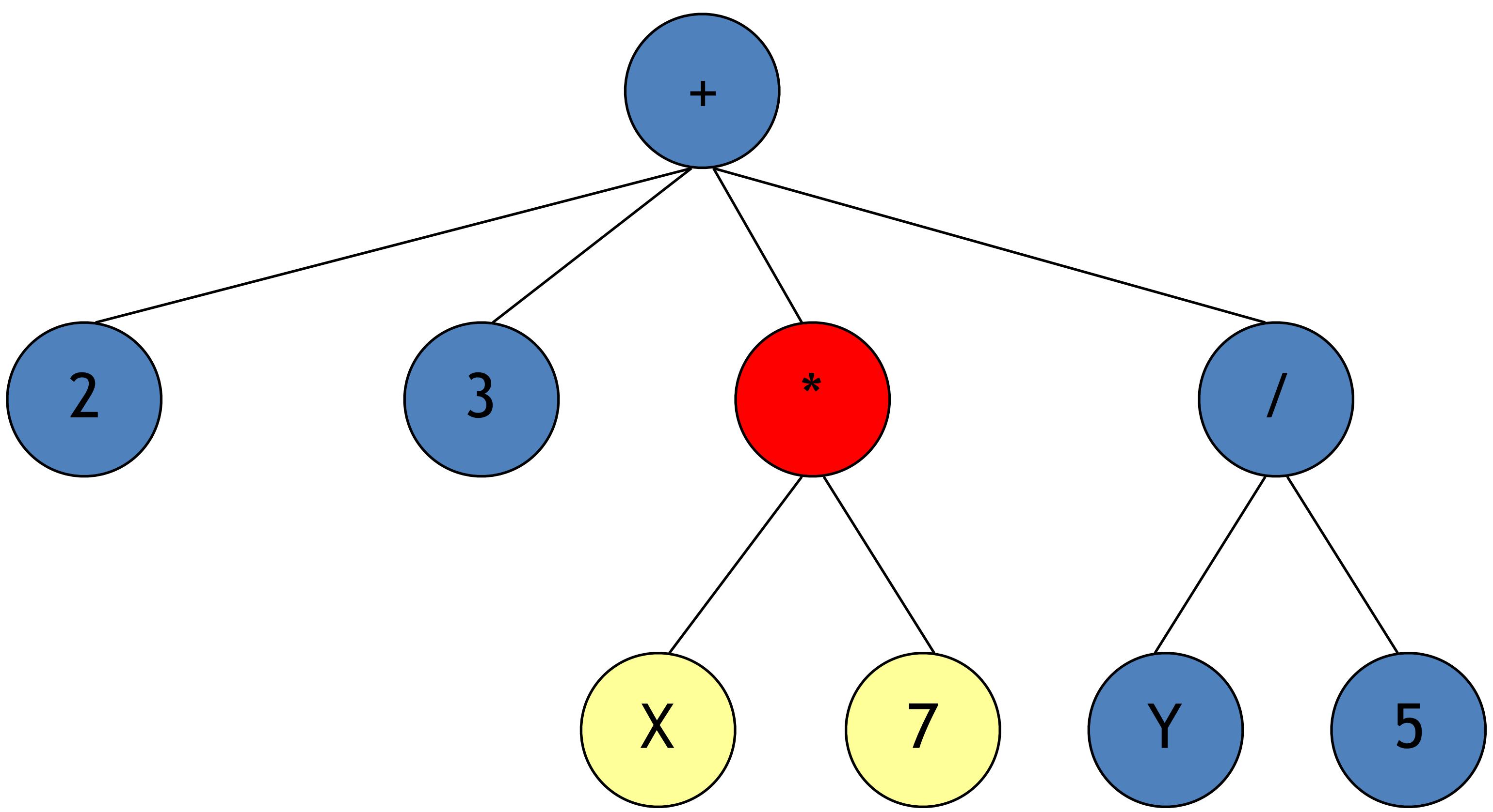
(+ 2 3 (* X 7) (/ Y 5))



Mutation

First pick a random node

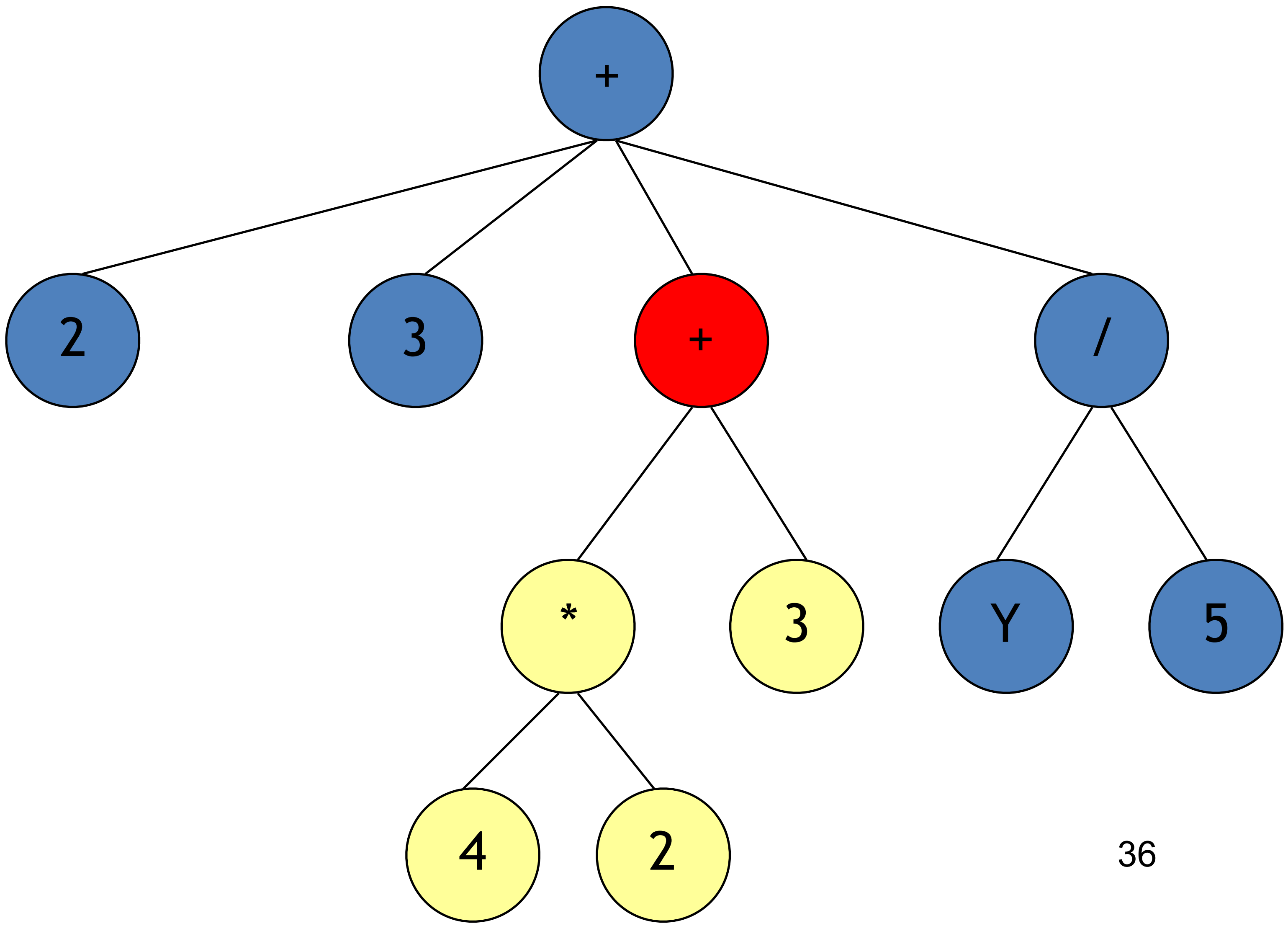
(+ 2 3 (* X 7) (/ Y 5))



Mutation

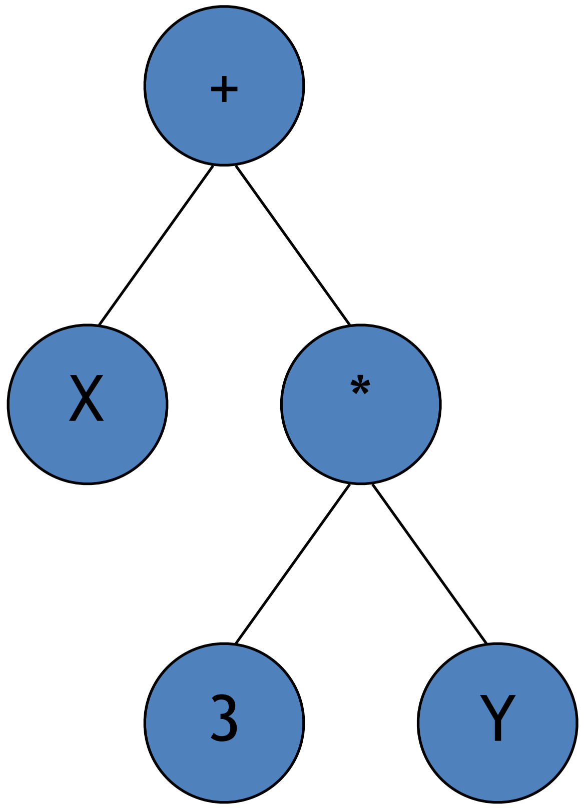
(+ 2 3 (+ (* 4 2) 3) (/ Y 5))

Delete the node and its children, and replace with a randomly generated program

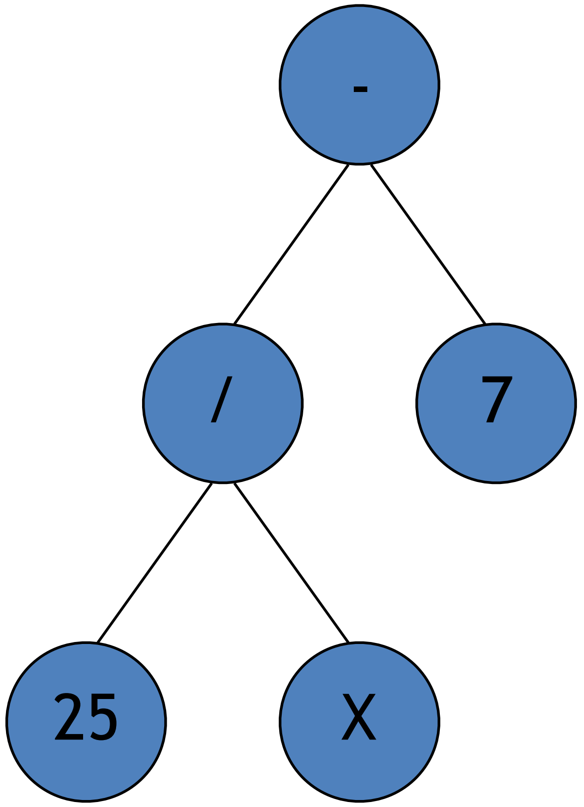


Crossover

(+ X (* 3 Y))

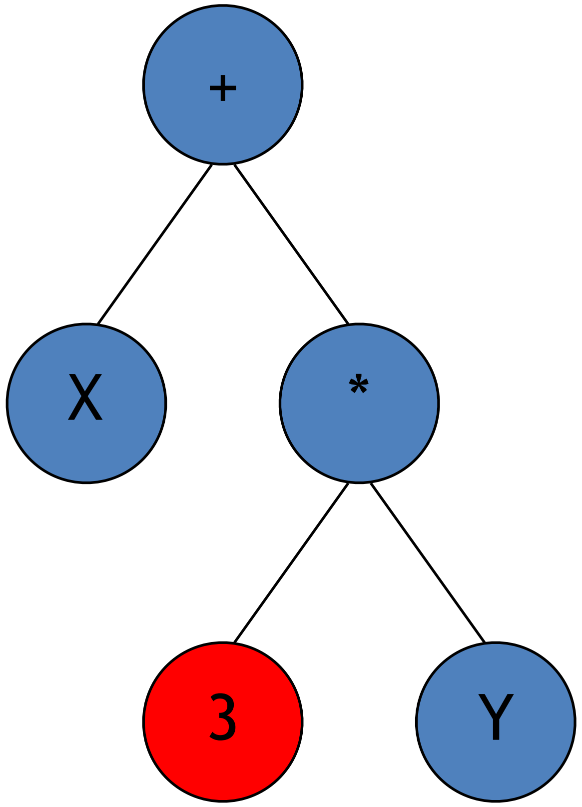


(- (/ 25 X) 7)



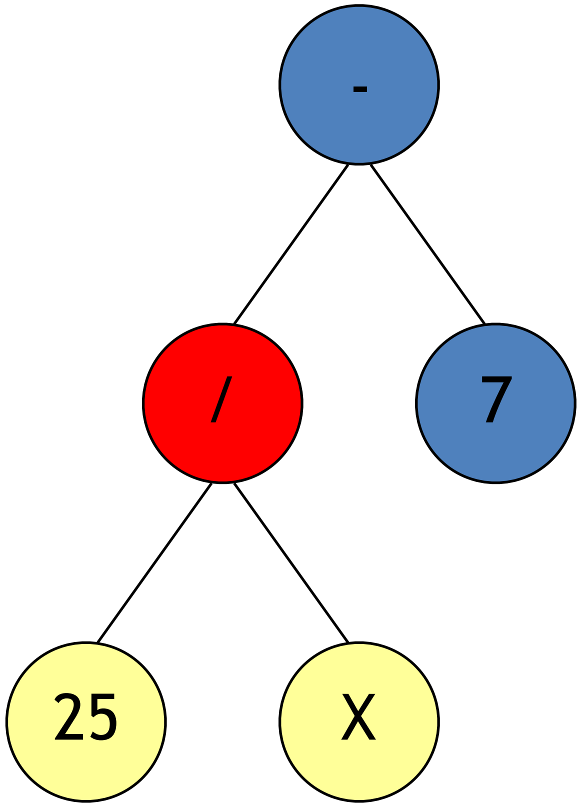
Crossover

(+ X (* 3 Y))



Pick a random node in each program

(- (/ 25 X) 7)



Crossover

(+ X (* (/ 25 X) Y))

(- 3 7)

Swap the two nodes