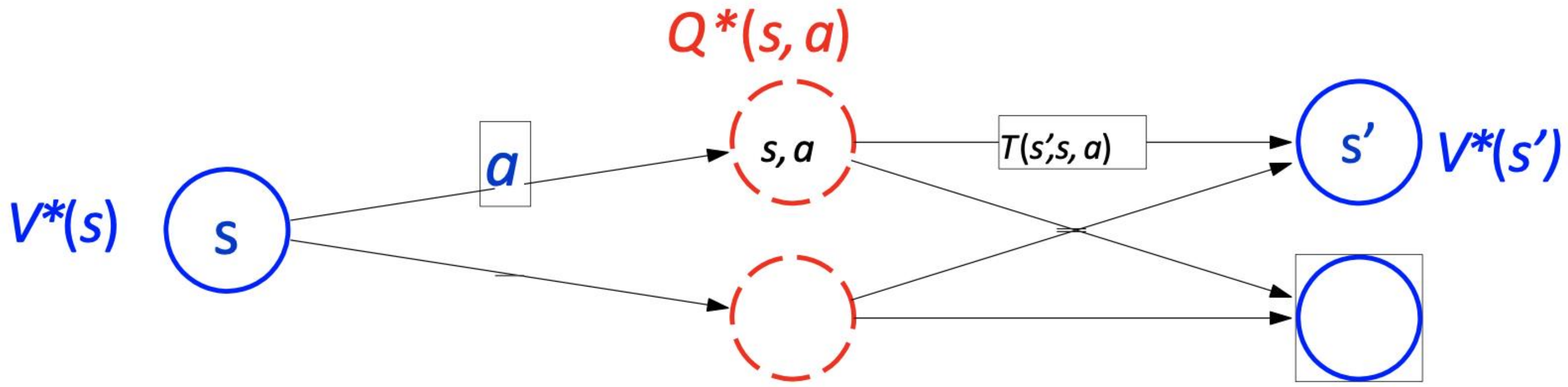
AIFA: Stochastic Planning MDP

07/04/2025

Koustav Rudra

Optimal Values and Q-values

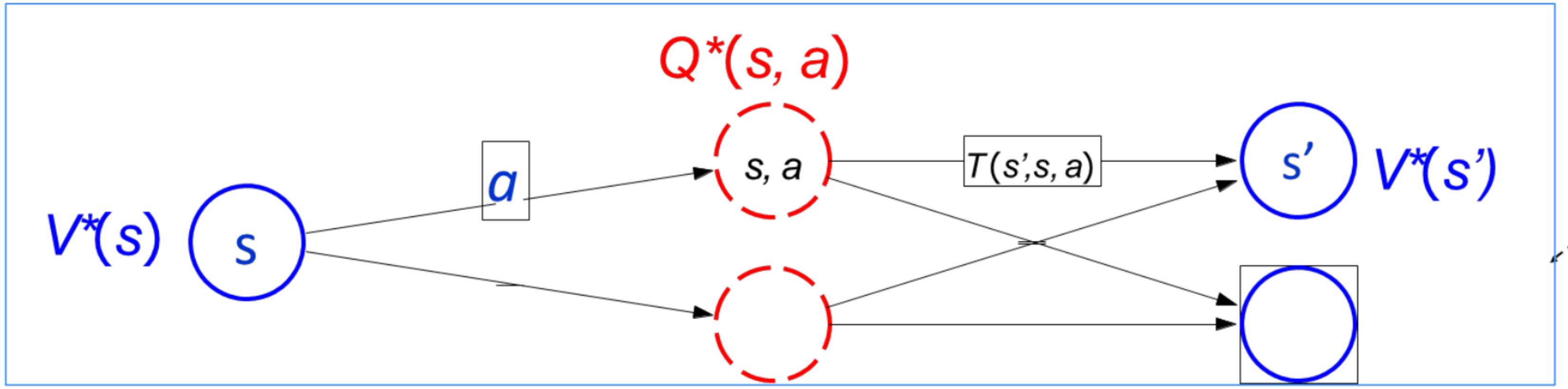


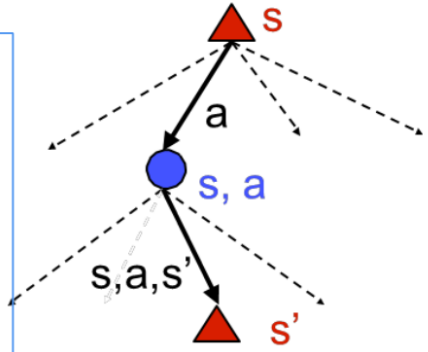
$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a Q^*(s,a)$$

The Bellman Equations

- Definition of “optimal utility” leads to a simple one-step lookahead relationship amongst optimal utility values:





$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

Value Iteration

1. For each state *s*, initialize $V(s) := 0$.
2. **for** until convergence **do**
3. For every state, update

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Complexity of each iteration: $O(S^2A)$

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example: Value Iteration

V_2

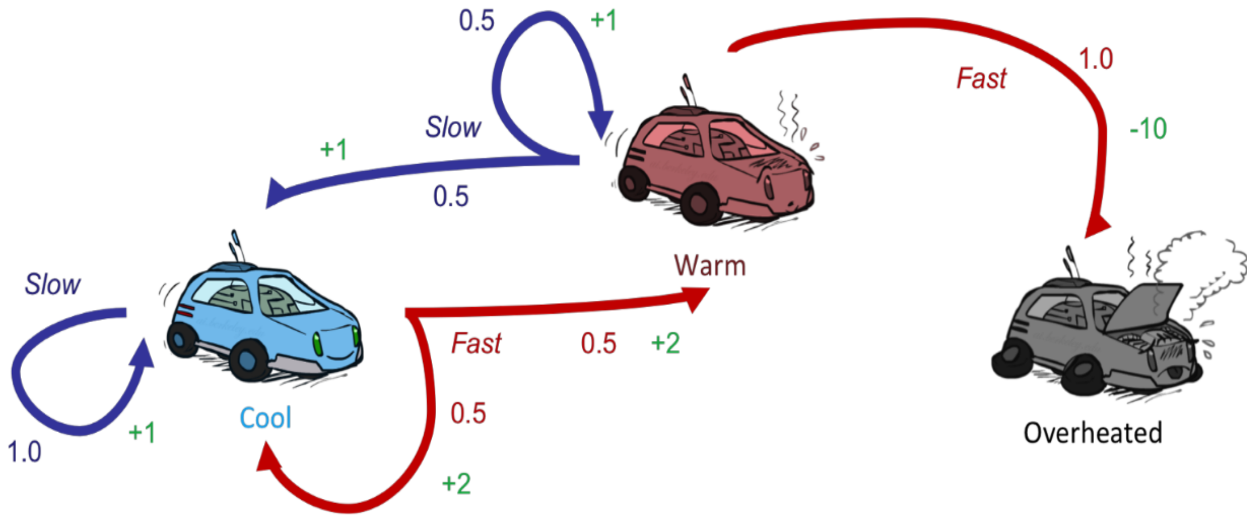
V_1

S: 1

F: .5*2+.5*2=2

V_0

0 0 0

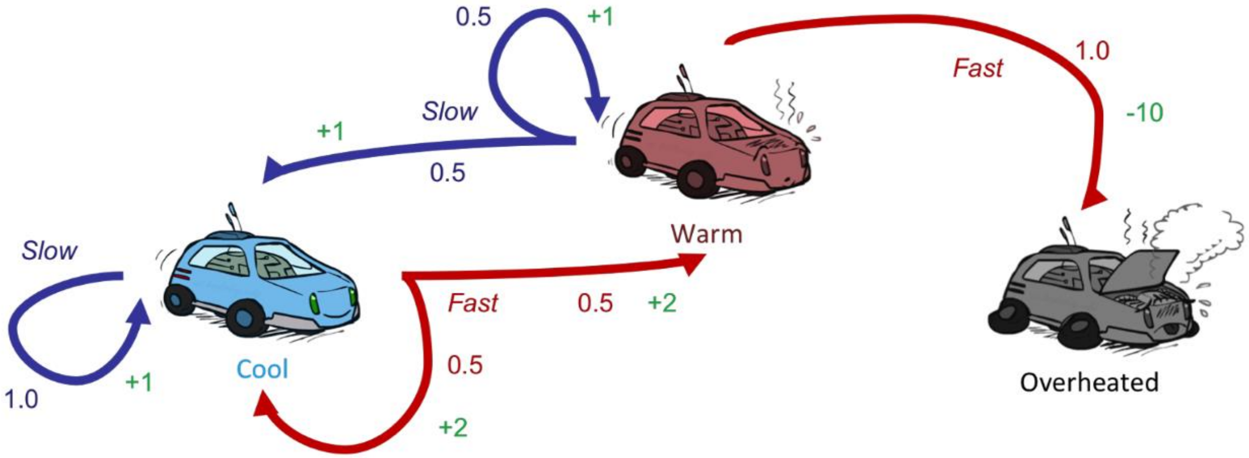


Assume no discount!

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Example: Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

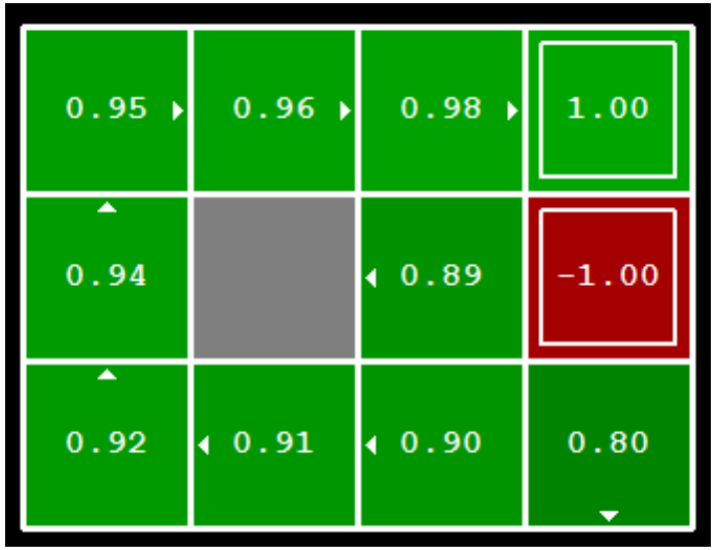
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)

- How should we act?

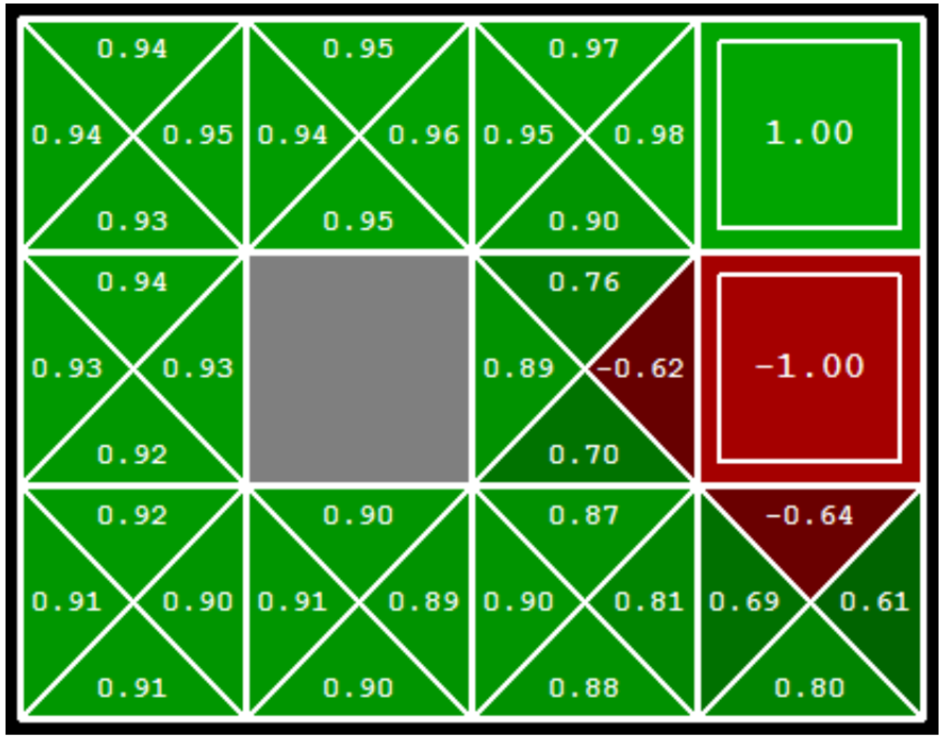
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$



- This is called policy extraction, since it gets the policy implied by the values

Computing actions from Q-values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Important lesson: actions are easier to select from q-values than values!

AIFA: Stochastic Planning MDP [Policy Iteration]

07/04/2025

Koustav Rudra

Policy Iteration

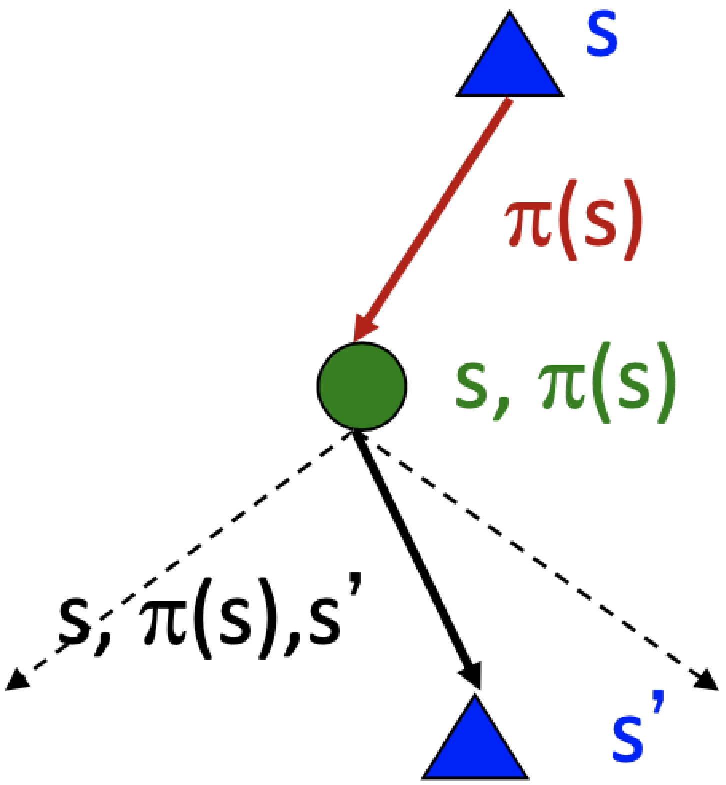
- Alternative approach for optimal values:
  - Step 1: Policy Evaluation:
    - calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy Improvement:
    - update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
    - Repeat steps until policy converges

Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

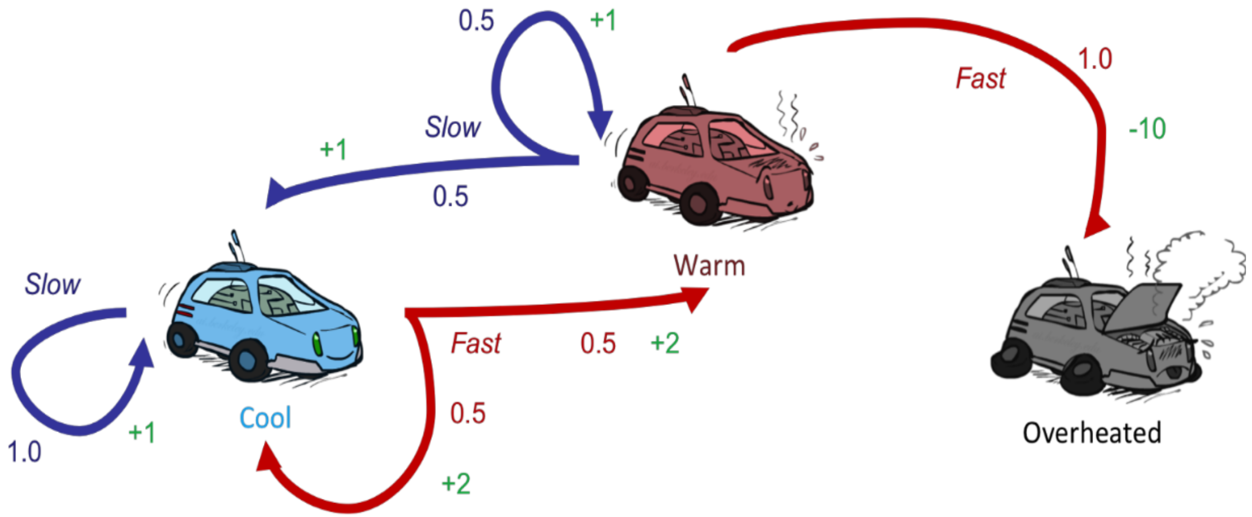
- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s') \right]$$

Example: Policy Evaluation

Policy: slow

V_2			
V_1	1	1	0
V_0	0	0	0



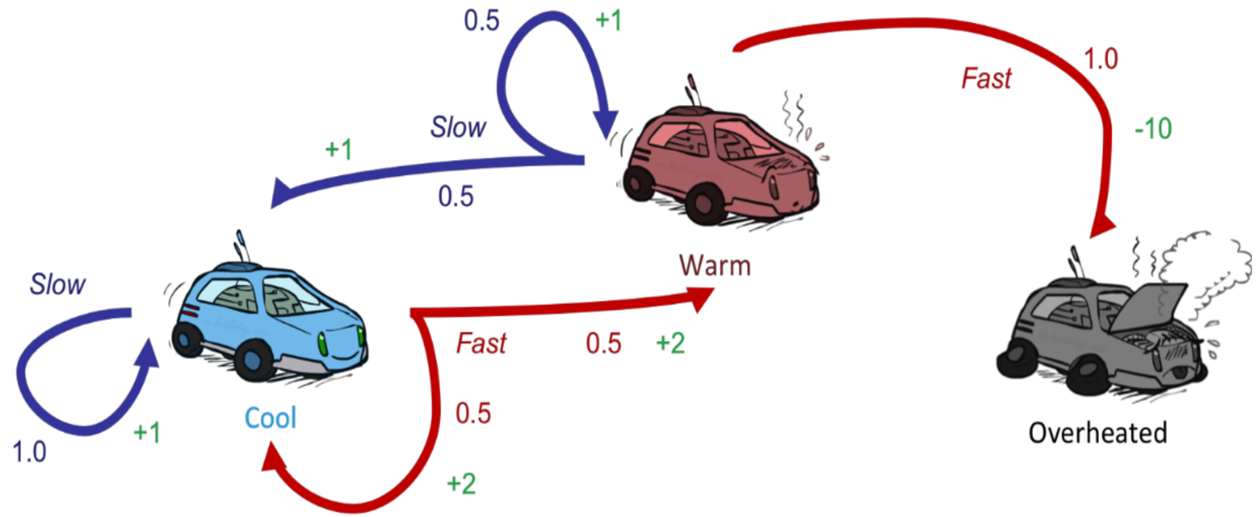
Assume no discount!

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Example: Policy Improvement

Policy: slow

V_2			
V_1	1	1	0
V_0	0	0	0



Assume no discount!

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it

- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)

- Both are dynamic programs for solving MDPs

AIFA: Genetic Algorithm

07/04/2025

Koustav Rudra

Genetic Algorithms

- Intelligent search techniques
- Maintaining a population of candidate solutions for a given problem
- Search the solution space by applying variation operators

Nature	Genetic Algorithm
Environment	Optimization Algorithm
Individuals	Feasible Solution
Individual’s Adaptation	Solution Quality
Population Species	Set of Feasible solutions
Selection, Recombination, and Reproduction	Variation Operators

What is Genetic Algorithm?

- Genetic Algorithms refer to a family of computational models inspired by Darwin's theory of biological evolution – **Survival of the Fittest**
  - Tall trees grow near mountains
  - Animals with far in the wintry region

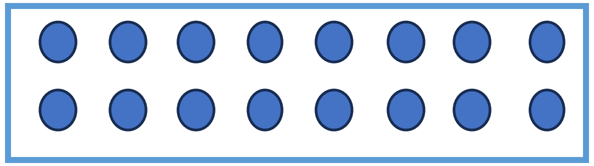
- How nature does this selection?
- What is the essential process of nature?

- The idea is one of Natural Selection organizing principle for optimizing individuals and populations of individuals

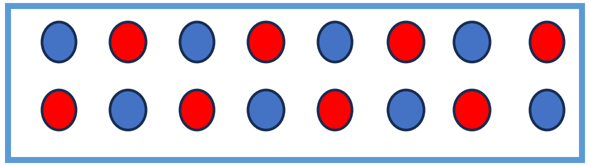
- GAs mimic Natural Selection to optimize more successfully

- Problems are solved by an evolutionary process resulting in a best (fittest) solution (survivor)

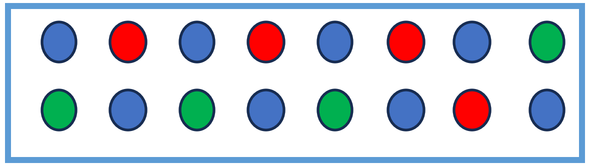
What is Genetic Algorithm?



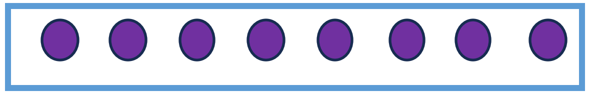
Points in the search space



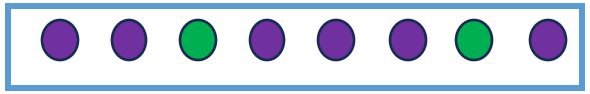
Randomly select initial population to start with



Select four points to generate next generation based on fitness function



Generate population for second round using crossover and mutation



Generate population for second round using crossover and mutation
6 from new generation and 2 from previous one

Genetic Algorithm vs Search Techniques

- Inspired by "Natural evolution", GAs involve direct manipulation of the coding achieved by the crossover and mutation operators

- GAs begin their search from many points, not from a single point, contain population of feasible solutions to the problem

- GAs do not need auxiliary information like gradients at points
    - They search via sampling

- GAs search by stochastic operators, not by deterministic rules

- They use random choice to guide highly exploitative search

Genetic Algorithm Process

- Encode potential solutions in terms of chromosome-like data structure

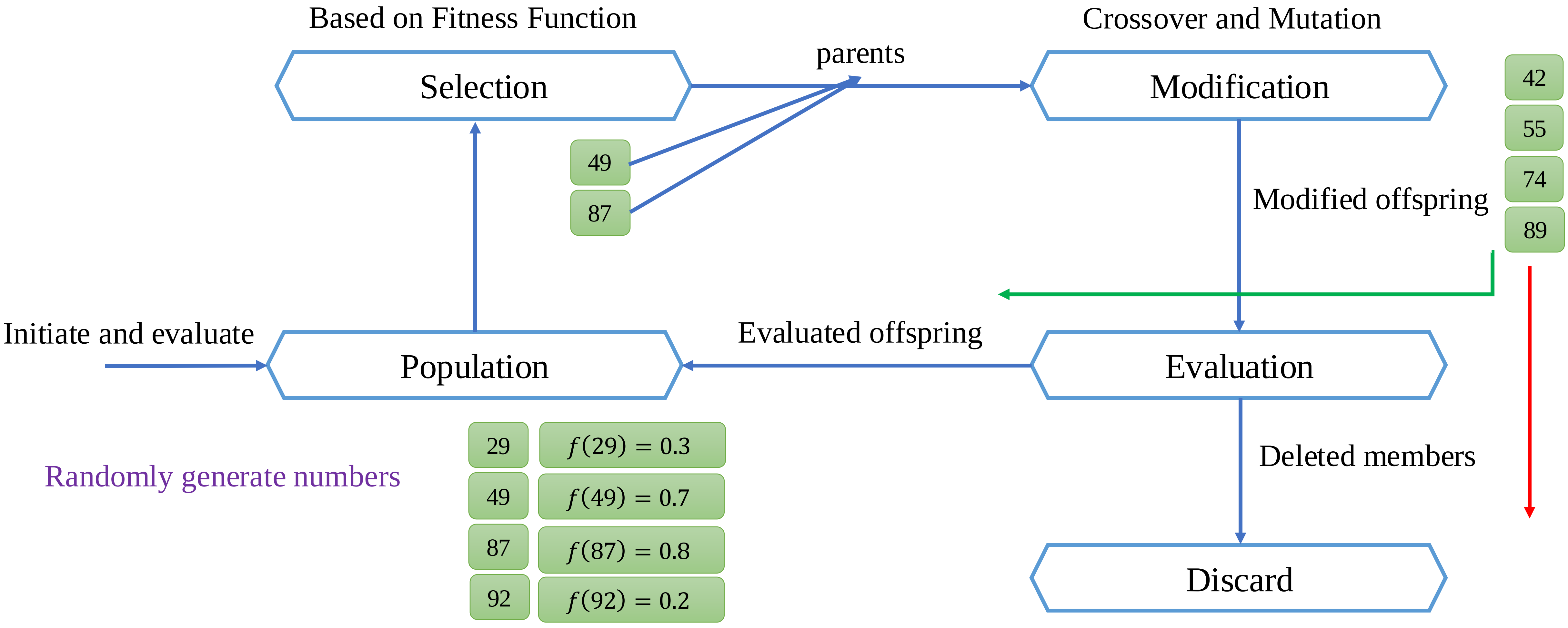
- Select parents on the basis of the fitness of the solutions to produce offspring for next generation, who contain the characteristics of both parents

- Employ recombination operators (selection, crossover, and mutation) repeatedly to preserve the good portions of the strings

- Good portions of the strings usually lead to an optimal or near-optimal solution
    - The method is applied over a desired number of generations

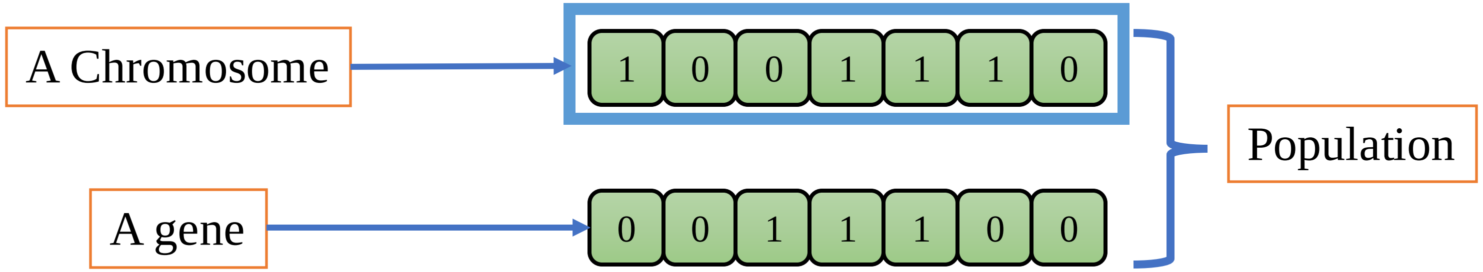
- If well designed, population will converge faster

GA: Evolutionary Cycle



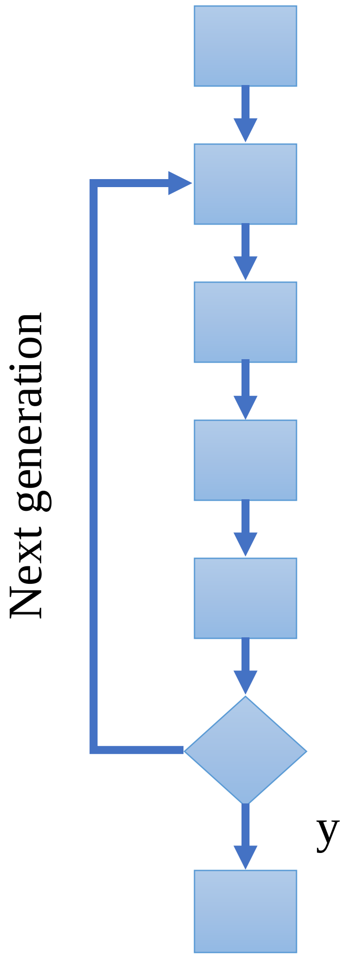
Find a number between 1 to 100 that fits functional value f(x)

Genetic Algorithms



- Binary encoding uses 0’s and 1’s in a chromosome
- Each bit corresponds to a gene
- The values for a given gene are alleles
- A set of chromosomes forms population

GA Over Generations



Initialize Population (initialization)

Select individual for mating (Selection)

Mate individuals and produce children (crossover)

Mutate children (Mutation)

Insert children into population (insertion)

Are stopping criteria satisfied?

Finish

Chromosome Encoding

- Binary Encoding
- Real Encoding
- Permutation Encoding
- Value Encoding
- Tree Encoding

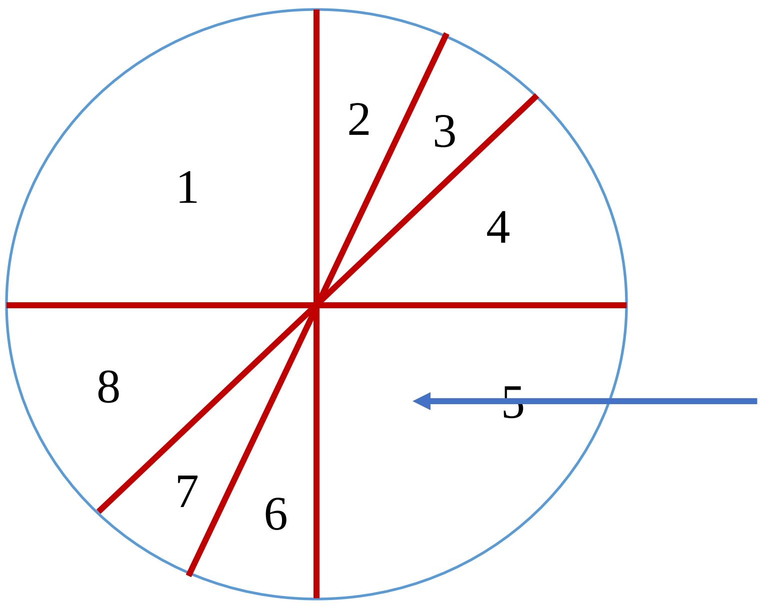
- Which one to use?
- When?

Selection Schemes

- Roulette wheel selection without scaling
- Roulette wheel selection with scaling
- Stochastic tournament selection with a tournament size of two
- Remainder stochastic sampling without replacement
- Remainder stochastic sampling with replacement
- Elitism

- Which one to use?
- When?

- Balance between population diversity and selection pressure



Crossover and Mutation Examples

- Single point crossover/One crossover point
  - Parent1: 101|100
  - Parent2: 110|111
  - Child1: 101|111
  - Child2: 110|100
- Two point crossover/ two crossover points
  - Parent1: 10|11|00
  - Parent2: 11|01|11
  - Child1: 10|01|00
  - Child2: 11|11|11
- Mutation (bit inversion)
  - 100100
  - 110100

GA: Parameters

- Population Size:
  - Problem specific
  - A good population size is about 20-30
  - The best population size depends on the size of encoding string (chromosomes)
  - More the encoded sizes, more should be the population size of

- Crossover Probability:
  - Should be high generally, about 80%-95%

- Mutation Rate:
  - Should be very low
  - Best rates seem to be 0.5%-1%

- Crossover and Mutation Type:
  - Operators depend on chosen embedding

GA Algorithms: Benefits

- Easy to understand and modular in structure – separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Solution is obtained all the time – solution quality improves with additional knowledge gained
- Inherently parallel; easily distributed
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications

When to Use Genetic Algorithm

- Alternate solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements
  - Near-optimal solution will suffice
  - Adequate computational power is available
  - The problem does converge to an optimal solution

GA Application Types

Domain	**Application Types**
Control	Gas pipeline, pole balancing, missile evasion, pursuit
Design	Semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	Manufacturing, facility scheduling, resource allocation
Robotics	Trajectory planning
Machine learning	Designing neural networks, improving classification algorithms, classifier systems
Signal processing	Filter design
Game playing	Poker, checkers, prisoners’ dilemma
Combinatorial optimization	Set covering, travelling salesman, routing, bin packing, graph colouring and routing

Thank You