AI LAB 2025

AN INTRODUCTION TO ARTIFICIAL INTELLIGENCE

SHAY BUSHINSKY, SPRING 2025

LOCAL SEARCH

Local Search

- (RN ch. 4)

IN THIS LECTURE

1. Introduction to local search
2. Hill climbing algorithms
   - Deterministic methods
   - Probabilistic methods
3. Beam Search
   - NLP NLU case study
4. Genetic Algorithms

LOCAL SEARCH ALGORITHMS ITERATIVE IMPROVEMENT SEARCH

PROBABILISTIC HILL CLIMBING METHODS

1. STOCHASTIC
2. FIRST CHOICE
3. RANDOM RESTART

1. STOCHASTIC HILL CLIMBING

- Randomly selects the **candidate** next state from a group of states that increase the objective function
- Tune selection probability p **in proportion to the improvement** of the objective function
- Converges slower than steepest ascent but finds solutions that the latter doesn't find

2. FIRST CHOICE HILL CLIMBING

- Generates new states in a random manner until a new one is better than the current
- This strategy is effective when a state has many successors

3. RANDOM RESTART HILL CLIMBING

- Perform the local hill climbing algorithm until local optima is found
- Then perform a random restart

- This algorithm is **complete** as it is bounded to find the global optima after a fixed number of iterations

BEAM SEARCH

- Going Parallel…

BEAM SEARCH

- Beam search is a heuristic search algorithm that explores a graph by expanding the most promising nodes.
- It's often used in contexts where the search space is very large and keeping every leaf node in memory (as in breadth-first or depth-first search) is not feasible.

BEAM SEARCH ALGORITHM OUTLINE

- Generalized Hill Climbing
- Keeps k states in memory instead of 1
- Starts by generating k random states
- Iteration: generate all descendants:
  - Choose the best k, and
  - Repeat until local optima found
- * like BFS, but expands only k states at each level using an objective function

DETAILING THE BEAM SEARCH ALGORITHM

1. **Initialize**:

   Start with only the root node of the search tree in the initial beam set (or queue).

2. **Set Beam Width (k)**:

   Define the beam width, *k*, which determines the maximum number of nodes to keep in the search frontier at any given time. This width is a fixed parameter of the algorithm.

3. **Expand Nodes**:

   From the current beam set, generate all successor nodes for each node in the set.

4. **Apply Heuristic**:

   Evaluate the successor nodes using a heuristic function that estimates how close each node is to the goal.

5. **Select Best Nodes**:

   1. Sort all successor nodes according to their heuristic values.
   2. Select the *k* best nodes according to the heuristic function (best-first approach).

DETAILING THE BEAM SEARCH ALGORITHM (CONT.)

6. **Update Beam Set**:

   The beam set for the next iteration of the algorithm is updated to include **only** these *k* nodes.

7. **Repeat**:

   Repeat the expansion, heuristic evaluation, and beam set update process.

8. **Goal Check**:

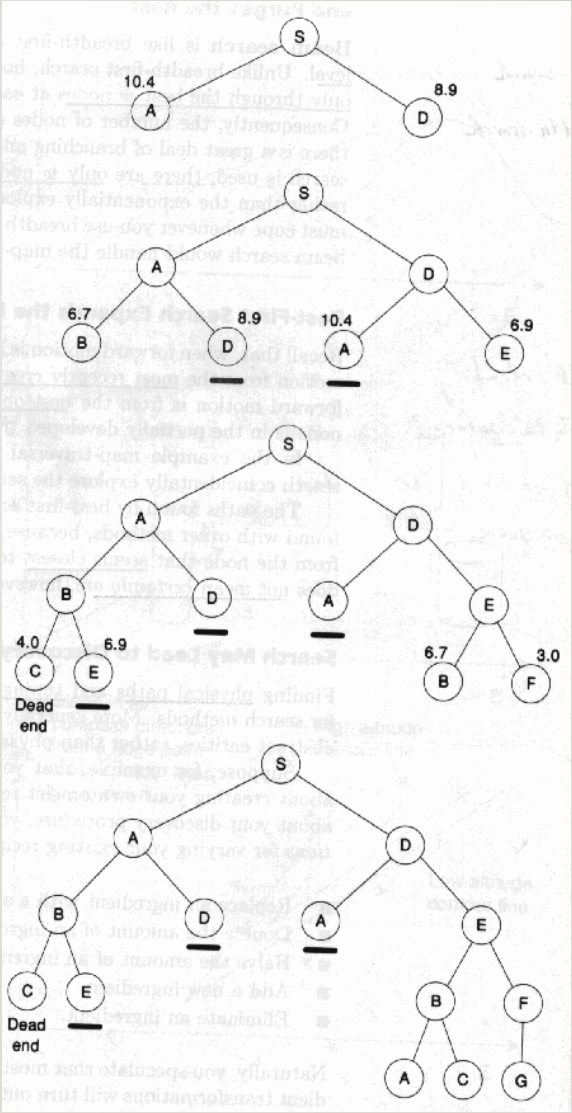
   1. At each iteration, check if any of the nodes in the beam set is the goal.
   2. If the goal is found, return success and the solution path.
   3. If no nodes are left or a predefined condition is met (like reaching a maximum depth), the algorithm ends with failure.

9. **Termination**:

   The algorithm terminates when either the goal is found, or there are no nodes left to explore (the beam is empty), or a predetermined condition is met (like reaching a maximum number of iterations).

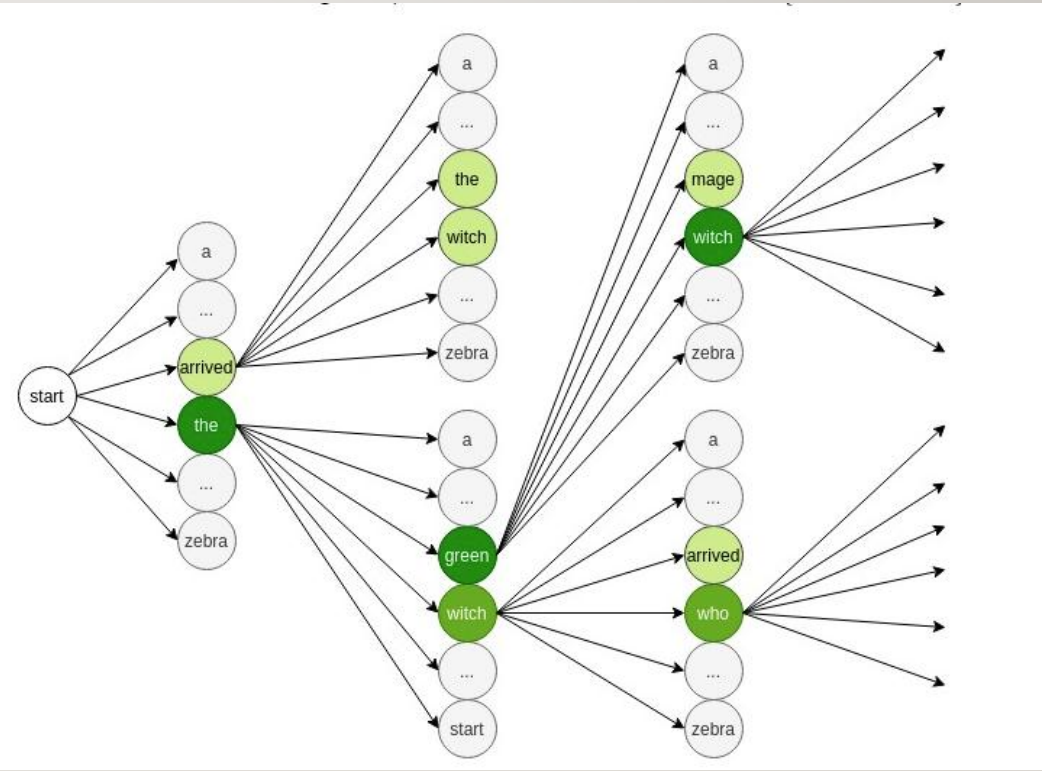
BEAM SEARCH EXAMPLE (K=2)

- Keep track of K states rather than just one
- – Modified breadth-first, contour created dynamically
- • Start with K randomly generated states
- • Stop if any goal state
- • Multiple Successors get generated for each of the k states
- • Choose top K successor states in next cycle
- • Example with **k=2** - strives to find goal G where numbers denote **distance to goal**



CONDITIONAL PROBABILITY IN LLMS AND BEAM SEARCH

- **Start Node**: Model begins with no prior context; probabilities are based on how likely a word is to start a sentence.
- **Sequential Decisions**: Each choice of word is based on the conditional probability of that word given the preceding sequence.
- **Beam Selection**: Beam search selects the top paths (beams) with the highest probabilities at each step, which depend on the sequence generated so far.
- **Cumulative Calculation**: The likelihood of a sequence is the product of conditional probabilities at each step.
- **Contextual Predictions**: The model's training on large datasets informs the conditional probabilities, ensuring contextually relevant word choices.
- **Autoregressive Nature**: Each new word is chosen based on the entire existing sequence, reflecting the model's autoregressive behavior.



For instance, the probability of the sequence "**the green witch**" is the product of the probabilities P("the" | start), P("green" | "the"), and P("witch" | "the green").

BEAM SEARCH SAMPLING

1. **Initialization**:
    1. Begin with a start token (often represented as <s> or similar) that signals the beginning of a sequence in the target language.
2. **First Word Selection**:
    1. Calculate the probability of each possible first word in the target vocabulary given the start token.
    2. Choose the top β (beam width, here β=2) most probable words as the initial branches of the search. For example, \{\text{arrived, the}\}.
3. **Probability Expansion**:
    1. For each word in the initial beam, calculate the probability of every possible subsequent word in the vocabulary.
    2. The potential continuations might look like \{\text{**arrived the**, **arrived witch**, **the green**, **the witch**}\} and so on.

BEAM SEARCH SAMPLING (CONT.)

4. **Beam Pruning**:

   From these expansions, again select the top *β* sequences based on their cumulative probabilities. For example, \{\text{the green, the witch}\}.

5. **Iterative Expansion**:

   Continue the process, iteratively expanding each sequence in the beam with subsequent words and pruning to maintain only the top *β* sequences at each step.

6. **End of Sequence**:

   Stop expanding once the end of a sentence is reached (usually a token like </s>), or a maximum sequence length is hit.

7. **Final Selection**:

   1. The final output sequence is the one that has the highest cumulative probability after the end token is reached or the process is terminated due to length.

BEAM SEARCH ANALYSIS

- If there's more than one goal node, it always catches the one nearest the node in generations
- This is of value if the tree is unbalanced, but wasteful if the goal nodes are at similar levels
- This is an **incomplete** search:

1. There is a danger that a goal finding route will be removed from Q before it can be explored
2. This may lead to not finding any goals at all

BEAM SEARCH COST

- At each level there are only k nodes stored
- This avoids the exponential explosion problem of breadth first search

- Time & Space complexity are quite efficient:
- **Time Efficiency** is O(bd)
- Worst case when it plunges not finding the goal

- **Space Efficiency** is also of order k: O(k)

PARALLELIZING

- If implemented in parallel, requires information passing between threads

- For instance, if a particular state largely yields better successors, it effectively signals other processes to converge and extend the search from that state.

OVERCOMING GREED

- Beam Search is susceptible to local optima
- Solution: add nondeterminism to encourage **diversity**

STOCHASTIC BEAM SEARCH

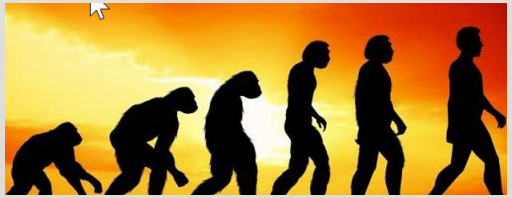
- Like beam search, but probabilistically choose the k nodes at the next generation
- The probability that a neighbor is chosen is in proportion to its heuristic value

Evolutionary perspective:

Like asexual reproduction: each node mutates, and the fittest ones survive

GENETIC ALGORITHMS

EVOLUTIONARY INSPIRED ALGORITHMS



OUTLINE

- Introduction to Genetic Algorithms
- Applications
- Configuration and Improvement
- Advanced Techniques
- Optimization Problems
- Algorithm Mechanics
- Execution: Example Implementation
- Conversion - Population Metrics: Genetic Drift, Genetic Diversity, Selection Pressure

GENETIC ALGORITHMS (GA)

- A genetic algorithms (GA) are based on a search metaheuristic
- They are inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA)
- Self Organizing Algorithms

WHEN TO CONSIDER GA?

- Problem elements are somewhat unpredictable
- There are more rules / different behaviors than data

CLASSIC EXAMPLES

1. Scheduling (manufacturing, transportation, project management)
2. Intrusion Detection Systems (adapt to changing attack vectors)
3. Network configuration optimization (load, topology, resource availability, response time)

STOCHASTIC BEAM SEARCH

- Like beam search, but probabilistically choose the *k* nodes at the next generation
- The probability that a neighbor is chosen is proportional to its heuristic value
- Like asexual reproduction: each node gives its **mutations,** and the fittest ones survive

EXTENDING BEAM SEARCH

- Like stochastic beam search, but pairs of nodes are combined to create the offspring(s)

DISCRETE OPTIMIZATION PROBLEMS

- A set of parameters – genomes or chromosomes
- A function combining them to a single value – the fitness function
- A set of constraints on the parameters – incorporated in the fitness function
- Goal: find the parameters that maximize / minimize the fitness function subject to the constraints

GA KEY QUESTIONS

- What is the **fitness function**?
- How is an individual **represented**?
- Which individuals **survive**?
- How are individuals **selected**?
- How do individuals **reproduce**?

BIOLOGY- LIFE

- Depends on proteins – composed of 20 amino acids essential for various cellular functions
- DNA Encoding: encodes instructions to produce specific proteins
- Ribosome: translates to assemble the amino acids in the correct sequence
- Carriers: each cell nucleolus consists of 23 chromosomes that form the genome of the individual: **genotype**
- Each chromosome encodes four amino acids {A, C, G, T}

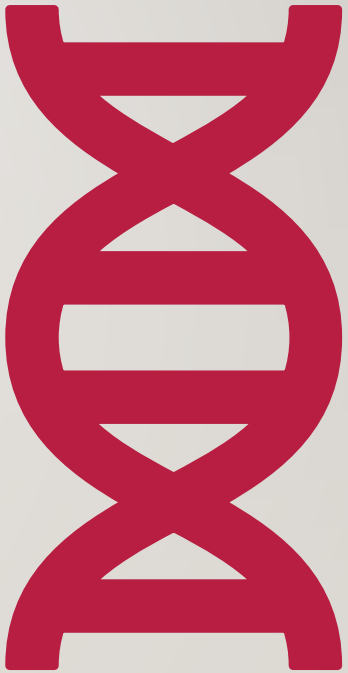
TYPES OF REPRESENTATIONS

- **Fixed length string**
  - Bit string (over a binary alphabet)
    - mapping of the SAT problem
    - Graph coloring – 2 bits represent a color
  - Over a small alphabet
    - A string over {C,G,A,T}
- * **Vector of real numbers**
  - Optimize f(x) – find a solution x=0.B1B2B3…Bn e.g., maximize f(x)=sin(2Pix^3)sin(25x) 0<x<1
- **Rule Base**
  - OHE Rules
- **Permutations**
  - N queens
  - TSM cities
- **Lists**
- **Trees**
  - Lisp/Scheme programs

TERMINOLOGY

Given a representation “ABC”:

- **Alleles:** "A", "B", and "C" are the **alleles**, which are the possible values that can occupy a gene location.
- **Gene Locations:** The **positions** of "A", "B", and "C" within the string (e.g., position 1, 2, and 3) are the **gene locations** (or loci).
- **Chromosome:** The **entire string "ABC"** represents the **chromosome**, which encodes a full candidate solution.



CASE STUDY: DIOPHANTINE EQUATIONS

- Solving a Diophantine (only integer solutions) equation: $a+2b+3c+4d=30$, where a,b,c,d are positive integers

- Demonstrate a good advantage of GA which is **finding many different solutions**

GA MODELING EXAMPLE

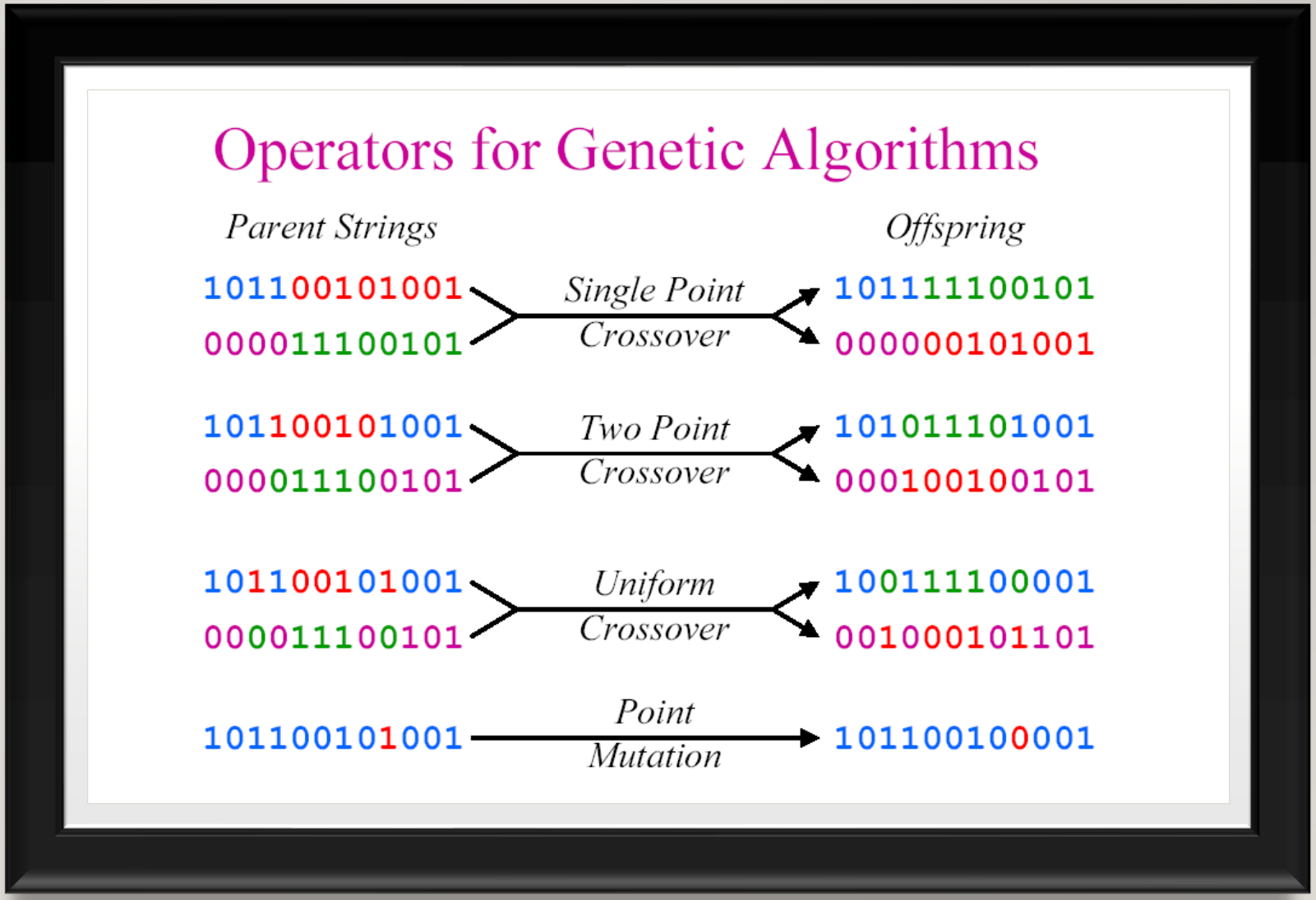
- Task: Find the maximum of the parabolic function: $31p\text{-}p^2$ $0< p <31$
- Genome: p's range can be represented by 5 bits: 0-31
- The function gets its maximum (240) at 15 and 16 coded 10000 and 01111

A RANDOM INITIAL POPULATION

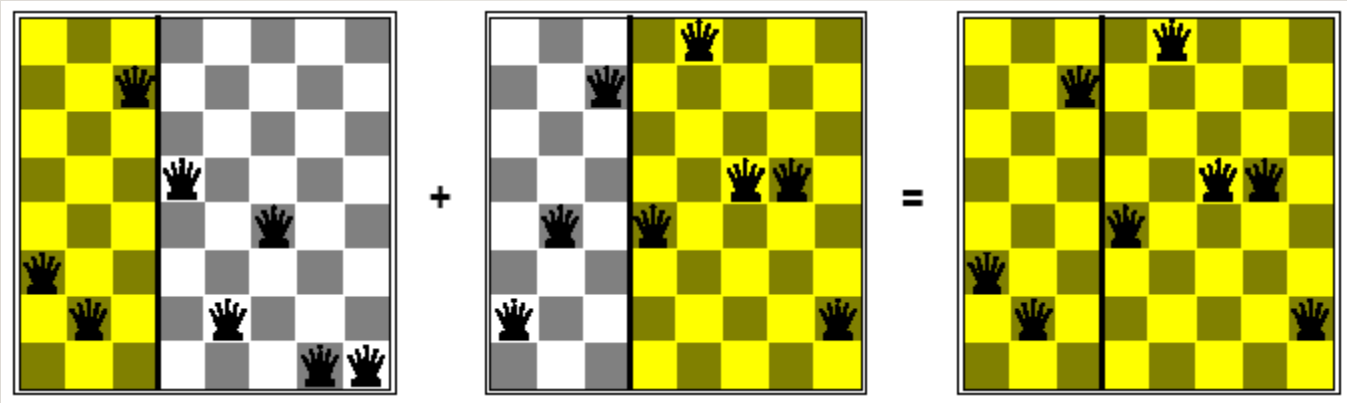
- The average population fitness is 128.5 which is close to the goal but will be improved in future generations

Genome	P	fitness
10110	22	198
00011	3	84
00010	2	58
11001	25	150

REPRODUCTION OPERATORS



SINGLE POINT CROSSOVER



LIST REPRESENTATION

- Lists data structure for genes:
  - **Mutations**: add/delete a node in the list
  - **Recombination**: exchange a node sequence
  - Single point/uniform crossover

RULE-BASE REPRESENTATION
MUSHROOM FEATURE ENCODING

Position	Attribute 1	Attribute 2	Classifier
1	Brown	Round cap	Edible
2	Red	No cap	Not-Edible
3	Blue		
4	Yellow		

Representing Solutions as Binary Strings:

Each feature is represented either in binary code or in **One Hot Encoding**:

- If the data set is Blue, Round-Cap and Edible, the translated string will be: "0010, 10, 10".
- If the data set is Brown, No-Cap and Not-edible, the translated string will be: "1000, 01, 01".

THE GENETIC ALGORITHM INPUT

1. The fitness function
2. A threshold defining an acceptable level of fitness for terminating the algorithm
3. The size of the population
4. Generation parameters
   - the fraction of the population that reproduces
   - the mutation rate

POPULATION SIZE (P)

- Indicates the number of chromosomes in the first generation

  This number is constant throughout the run of the program

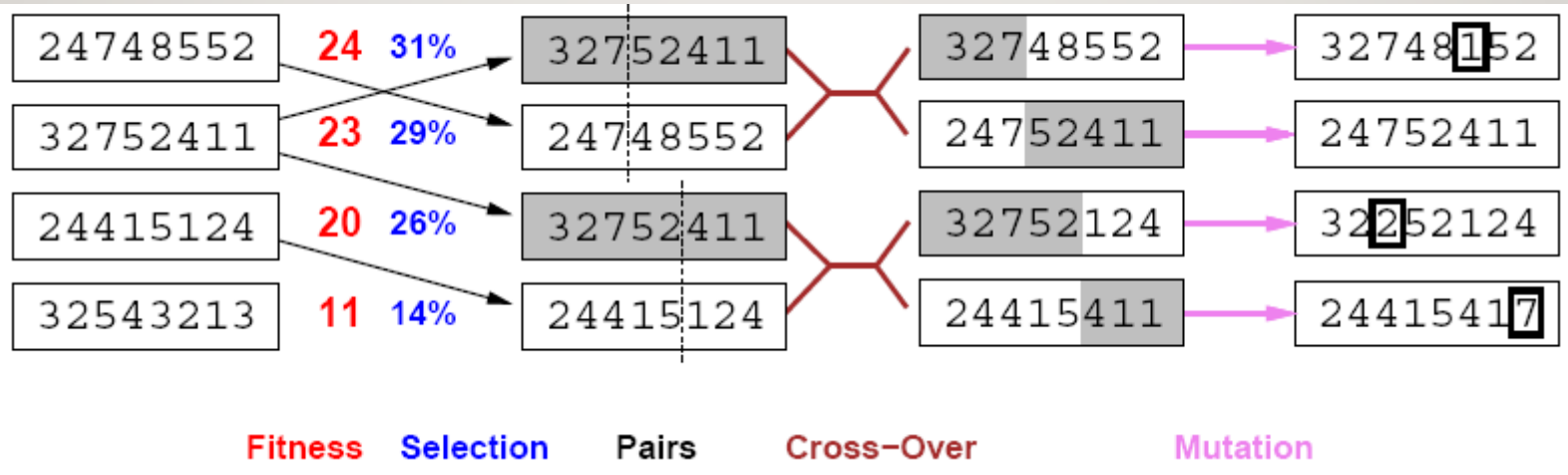
- A small population size value will run **faster** in real time but will take more generations to properly evolve (**converge**)

CROSSOVER RATE (R)

- Crossover rate: the percentage specifying the fraction of selected pairs that undergo crossover

  In most applications, this rate is very high, over 2/3rds or less than 1/3rd move on to the next generation…
  - A 100% crossover rate, means that crossover always takes place

THE GA FLOW: GENERATING SUCCESSORS FROM PAIRS OF STATES



THE GA ALGORITHM

- For each generation:
  - Randomly choose pairs of nodes where the fittest individuals are more likely to be chosen
  - For each pair, perform a cross-over: form two offspring each taking different parts of their parents
  - Mutate some values
  - Report best node found

THE GENETIC ALGORITHM

Genetic Algorithm (GA)

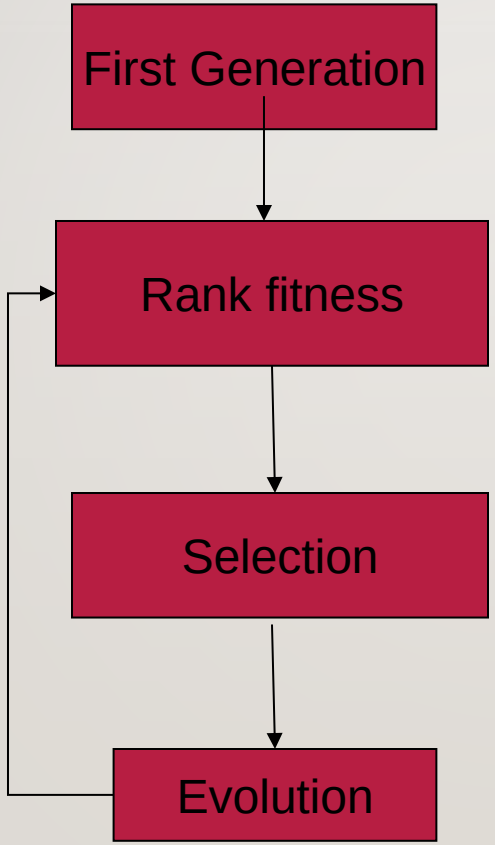
Inputs:

- $Fitness, Fitness\_threshold, p, r, m, e$
  (Fitness function, termination threshold, population size, crossover rate, mutation rate, elitism count)

Algorithm:

1. **Initialize** population P with p random solutions.
2. **Evaluate** fitness $Fitness(h)$ for each $h \in P$.
3. **While** $\max_h Fitness(h) < Fitness\_threshold$:
   1. **Select** $(1 - r)p$ individuals probabilistically based on fitness.
   2. **Crossover**: Form $(r \cdot p)/2$ pairs and generate offspring.
   3. **Mutate** each offspring with probability m.
   4. **Elitism**: Preserve the top e individuals.
   5. **Update** population P with offspring and elite individuals.

THE EVOLUTIONARY FLOW



GA TERMINATION CRITERIA

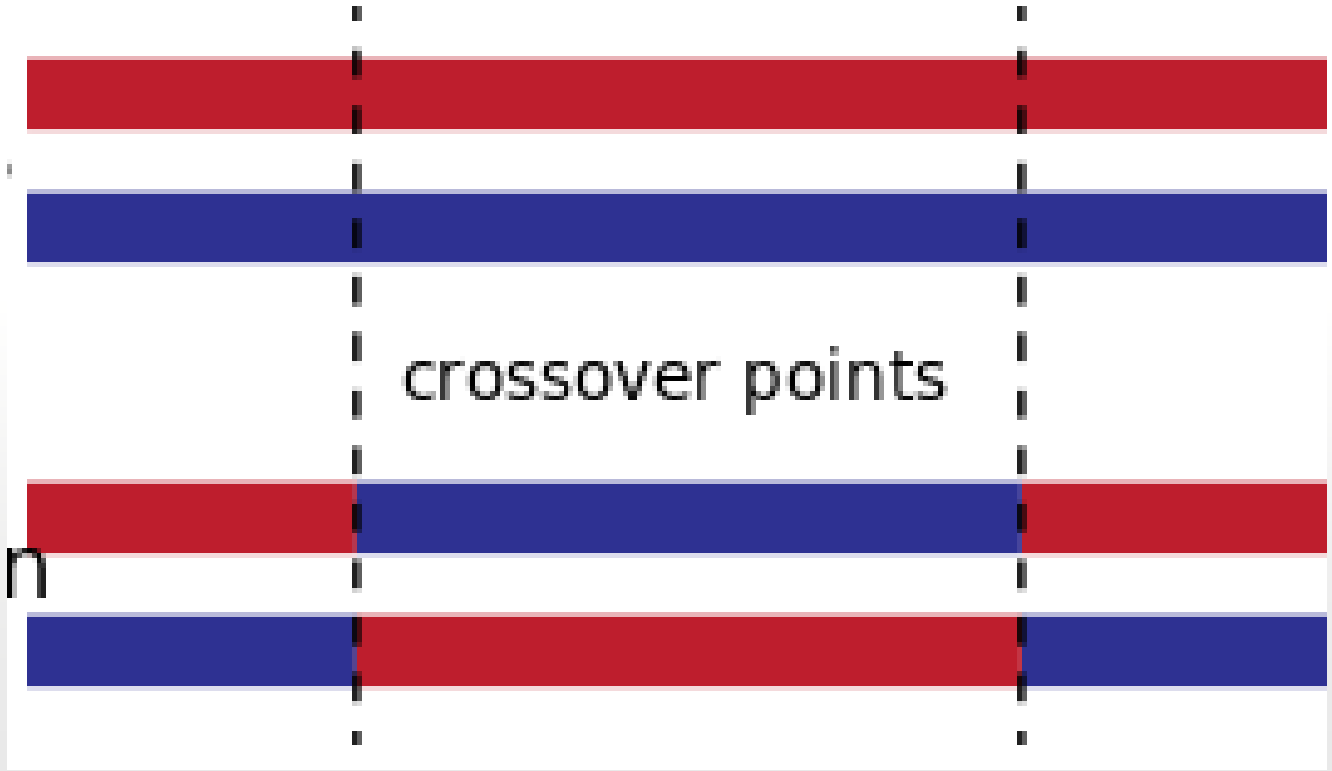
I. A solution is found that satisfies minimum criteria
II. Fixed number of generations reached
III. Allocated budget (computation time/money) reached
IV. The highest-ranking solution's fitness has reached a plateau
V. Manual inspection
VI. Combinations of the above

GA REPRODUCTION OPTIONS

- Single offspring
- Two children
- Multi

EXTENDING TWO POINT CROSSOVER

FOR TWO CHILDREN TO BE PRODUCED

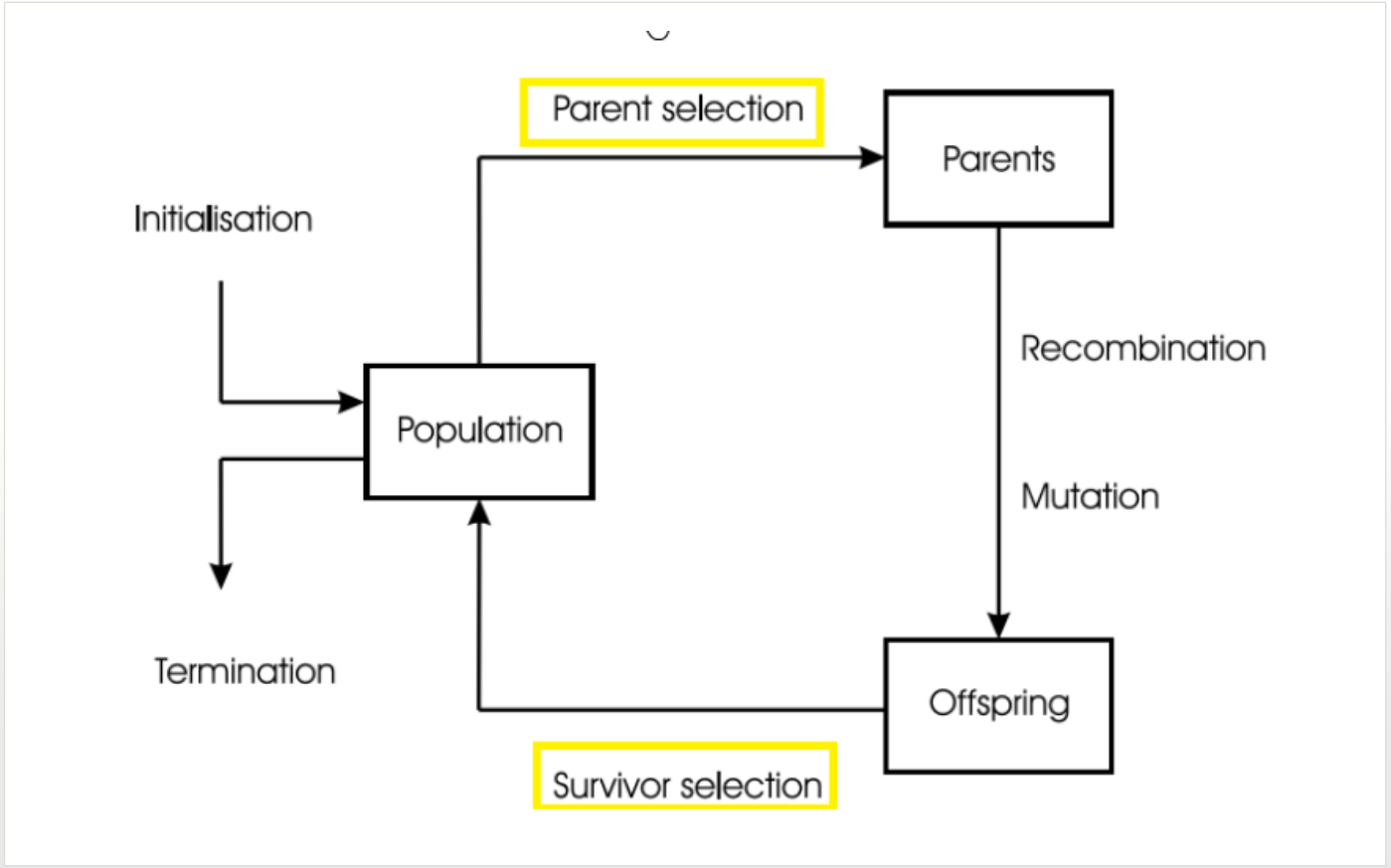


EXTENDING UNIFORM CROSSOVER

ALGORITHM EXAMPLE

CPP CODED HELLO WORLD!

SELECTION IN GA



SELECTION

- Selection occurs in two places:

1. Selection from current generation to take part in mating (**parent selection**)

2. Selection from parents + offspring to go into next generation (see **survivor selection** section)

SELECTION IN GA

- High quality individuals get a higher chance to become parents than those with low quality
- Low quality individuals are still given small positive chance
- Otherwise, the whole search could become too greedy and stuck in **local optima**

SELECTION PROCESS

- Two competing factors that need to be balanced in the selection process

1. **Selection pressure**
2. **Genetic diversity**

SELECTION PRESSURE IN NATURE

- External agents which affect an organism's ability to survive in their environment
  - Provides them with “unfair advantage”

- Example: Predation, Competition and Disease

SELECTION PRESSURE DEFINITION

- The tendency to select only the best members of the current generation to propagate to the next

- Required to direct the GA to an optimum

MEASURING SELECTION PRESSURE

- **Parent** Selection

- The ratio between fit_{top} / <fit>
  - the probability that the most fit member is selected to the probability that an average member is selected

- Also referred to as the **Exploitation Factor**

GENETIC DIVERSITY

- The maintenance of a diverse solution population
- Required to ensure that the solution space is adequately searched
- Especially in the earlier stages of the optimization process
- Also referred to as the **Exploration Factor**

MEASURING GENETIC DIVERSITY

- How different is each chromosome from the rest of the population
- Option 1:

$$diversity_i = \sum distance(i, j)$$

- Option 2:

$$diversity_i = \sum H_j$$

THE DISTANCE / SIMILARITY FUNCTION

- The distance function depends on the representation:
- For **binary** strings: **Hamming distance** (number of differing bits).
- For **real-valued** vectors: **Euclidean distance.**
- For **permutations** (e.g., TSP solutions): **Swap distance or Order-based distance**.

SHANNON ENTROPY AS A MEASURE OF GENETIC DIVERSITY

- Shannon entropy quantifies the uncertainty or randomness within a system.
  - Measures the diversity in allele frequencies within a population.
  - $H = -\sum$Pi log Pi
    - Where pi is the frequency of the ith allele.

  - Higher entropy indicates greater genetic diversity.
  - Lower entropy suggests reduced diversity or dominance of certain alleles.

SHANNON ENTROPY IN THE CONTEXT OF GA

Why logPi?

The log measures the **information content** of choosing allele i.

- Rare alleles (low Pi) carry **more information** when they appear.
- Common alleles (high Pi) carry **less information** — they're predictable.

For example:

- If Pi=1 (i.e., always the same allele), then logPi=0 so **no information/uncertainty**.
- If Pi=0.5, then logPi=−1, higher information/uncertainty.

Thus, in other words if the genes alleles are as rare as possible it means high H ensuring high diversity

BLENDING GENETIC DIVERSITY WITH FITNESS

- Diversity computation is expensive:
  - it is common to blend it with fitness computation (where k is problem specific)

$$score(i) = fitness(i) + k \cdot diversity(i)$$

BLENDING DIVERSITY WITH FITNESS

- score(i)=fitness(i)+k·diversity(i)
  - Instead of selecting individuals **only by fitness**, diversity is incorporated.
- k is a weight factor (tuned based on problem-specific needs):
  - **If k is too high** → The GA prioritizes diversity over fitness (could slow convergence).
  - **If k is too low** → The GA prioritizes fitness, risking premature convergence.
- This method ensures that even **suboptimal solutions with high diversity** have a chance to survive.

SHANNON ENTROPY EXAMPLE

If a gene has three possible values (A, B, C) and their probabilities in the population are:

- $P(A)=0.5$ $P(B) = 0.3$ $P(C)=0.2$

Then the Shannon entropy is:

- $H=-(0.5\log 0.5+0.3\log 0.3+0.2\log 0.2)$
- A higher entropy would indicate a well-diversified population, while lower entropy suggests the population is converging.

BALANCING => CONVERGENCE

- A <u>proper balance</u> between the selective pressure and genetic diversity must be maintained
- Guide the GA to converge in a reasonable time to a global optimum

- Too much pressure ➔ local optima convergence
- Too much diversity ➔ too slow convergence

ALPHA-GENES, ALPHA-CHROMOSOMES, AND DOMINANT MEMBERS IN GAS

- Alpha-Genes: Alleles that become dominant at specific gene positions, reducing diversity.
- Alpha-Chromosomes: Highly fit individuals that influence the population, possibly causing stagnation.
- Dominant Members: Individuals with very high fitness, leading to repeated selection and reduced exploration.
- **Risk**: Early convergence to local optima, loss of genetic diversity.
- **Solutions**: Adaptive mutation, diversity-based selection, tournament selection, or periodic restarts.
- **Goal:** Balance exploitation of good solutions with exploration for better diversity.

EXPECTED # OF COPIES OF AN INDIVIDUAL I

- E(Ni) = size * fit(i) / <fit>
- where:
  - size – population size
  - fitness(i) – fitness of individual i
  - <fit> - is the average population fitness

- Under high selection pressure => more copies of fitted individual are generated

THE GENETIC DRIFT EFFECT

- How variable is the evolving population
- Finding the time for the system to reach an **absorption state** where all population members are identical
- A function of:
  - The population size
  - The stochastic nature of the selection operator that may force convergence to a single member

FITNESS VARIANCE

- The <u>change in mean fitness</u> at each generation is a function of the population fitness variance

- At each generation, this **variance is reduced** due to the two factors:
  - selection pressure and the genetic drift

HOW DOES VARIANCE CHANGE

- The change in the population fitness variance due to selection, genetic drift, is dependent only on the **variance of the number of times** any individual population member is selected **- V[n]**

- If we select each population member once and only once, then V[n] = 0

VARIANCE OF INDIVIDUAL SELECTION

- $V = npq(n-1)$
- where:
  - V = variance of the number of times any individual is selected
  - n = total population size
  - p = probability of an individual being selected in a single draw (i.e., the proportion of the population that has the trait being sampled)
  - q = probability of an individual not being selected in a single draw (i.e., $1 - p$)

STRATEGIES TO MANAGE VARIANCE

- **Diverse Gene Pool**: Utilize mutations and crossover to introduce and maintain genetic diversity.
- **Balanced Selection Pressure**: Moderate the intensity of selection to prevent excessive convergence.
- **Adaptive Algorithms**: Dynamically adjust parameters like mutation rates based on population state to sustain fitness variance.

IMPROVING SOLUTION QUALITY

- In traditional GA, the fitness function is a fixed evaluation criteria used to determine the quality of each candidate solutions
- In **self-adaptive fitness**, the fitness function is allowed to adapt, and change based on the performance of individuals in the population

- **The fitness of an individual is allowed to change during the evolution process**

ADAPTIVE FITNESS

- To control genetic drift
- Design a (self) adapting fitness function to landscape

- Example 1: linear transformation – scaling
- Example 2: using RL

COMBATING GENETIC DRIFT

- Prevents Premature Convergence: Ensures new solutions continue to be explored.
- Encourages Diversity: Reduces over-reliance on early high-fitness individuals.
- Mimics Real-World Adaptation: Like changing traffic patterns in delivery optimization

CLASS IMPLEMENTATION

```
class AdaptiveGeneticAlgorithm(GeneticAlgorithm):
    def adaptive_fitness(self, individual, generation):
        # Dynamic fitness function that changes based on feedback
        base_fitness = self.evaluate_fitness(individual)
        penalty = (generation / self.generations) * 10  # Example penalty that increases over
        return base_fitness - penalty  # Fitness adapts dynamically
```

ADAPTIVE FITNESS

- Fitness criteria change during evolution
  - Adjusts based on individual performance
  - Responds to changing environments/problems
  - Uses feedback mechanisms to modify fitness criteria
- Adapts to dynamic problems
- Improves convergence and solution quality
- E.g. Manufacturing optimization adapts to prioritize successful configurations

ADAPTIVE FITNESS EXAMPLE

- **Optimizing Delivery Routes**:
- **Observation**: Monitor routes for shortest time, fuel efficiency, and customer satisfaction.
- **Metrics**: Track average delivery time, fuel consumption, and feedback scores.
- **Feedback Loop**: Use these metrics to adjust the fitness function, emphasizing routes that improve these metrics.
- **Adjustment**: Fitness function evolves to favor route plans that consistently yield better overall performance.

EXAMPLE: DELIVERY ROUTE OPTIMIZATION

- **Objective**: minimize total travel time
- **Traditional GA**: (total distance traveled + fuel cost)^-1
- **Problem:** This method **assumes static conditions**, not accounting for real-time traffic changes.
- **Dynamic Fitness Function**: adjust weights based on observed traffic patterns:
  - Initially, fitness = 1 / (total distance + fuel cost).
  - Midway, if traffic increases, fitness = 1 / (total distance + fuel cost + traffic penalty).
- => If **traffic conditions worsen**, the fitness function **adds a penalty** to routes that go through high-traffic areas.

EXAMPLE USAGE

```python
def adaptive_fitness(route, traffic_conditions):
    base_time = sum(route)
    traffic_penalty = traffic_conditions.get_penalty(route)
    return 1 / (base_time + traffic_penalty)

ga = AdaptiveGeneticAlgorithm(
    population_size=20, mutation_rate=0.05, crossover_rate=0.7, generations=100
)

best_route = ga.evolve()
print("Best Adaptive Route:", best_route)
```