ECE457A Notes

July 21, 2019

1 Introduction

1.1 Intelligence

- **Intelligence**: The ability to acquire and apply knowledge and skills.
- **Artificial Intelligence**: The science of creating intelligent machines, including intelligent computer programs.

1.2 Rational Thinking & Rational Behavior

- **Rational System**: A logical system that optimizes a given set of criteria.
- **Rational Thinking**: A logical system that achieves goals via logical inferencing.
- **Rational Behavior**: A logical system that perceives its environment and acts to achieve goals according to some set of beliefs.

1.3 Agents

- **Agent**: Senses its environment and acts on collected information.
- **Rational Agent**: An agent that acts in a way that is expected to maximize performance on the basis of perceptual history and built-in knowledge.

1.4 Types of Agents

- **Simple Reflex Agents**: Follow a lookup-table approach; needs fully observable environment.
- **Model-Based Reflex Agents**: Add state information to handle partially observable environments.
- **Goal-Based Agents**: Add concept of goals to help choose actions.
- **Utility-Based Agents**: Add utility to decide "good" or "bad" when face with conflicting goals.
- **Learning Agents**: Add ability to learn from experience to improve performance.

1.5 Environments

- **Fully vs. Partially Observable**:
  - *Fully Observable*: Sensors can detect all aspects relevant to the choice of an action.
  - *Partially Observable*: Missing Information or Inaccurate Sensors.

- **Deterministic vs. Stochastic**:
  - *Deterministic*: Environments that are only influenced by their current state and the next action executed by the agent.
  - *Stochastic*: Randomness/Noise.
- **Episodic vs. Sequential**:
  - *Episodic*: The choice of an action in each episode does not depend on previous episodes.
  - *Sequential*: An agent is required to "think ahead".
- **Static vs. Dynamic**: N/A.
- **Discrete vs. Continuous**: N/A.
- **Single Agent vs. Multi Agent**: N/A.

1.6 Cooperative and Adaptive Algorithms

- **Cooperative**: Solve Joint Problems.
- **Adaptive**: Change Behavior While Running.

2 Search Problem Formulation

2.1 Characteristics of Search Problems

- Large, Non-Polynomial Search Space Size
- Large, Non-Polynomial Constraints Size

2.2 Well-Structured vs. Ill-Structured Problems

- **Well-Structured Problems**: Problems for which the existing state and desired state are clearly identified, and the methods to reach the desired state are fairly obvious.
- **Ill-Structured Problems**: Situation in which its existing state and the desired state are unclear and, hence, methods of reaching the desired state cannot be found.

  1. Start & Improve Guess
  2. Search Alternatives
  3. Forward Search from Problem to Answer
  4. Backward Search from Goal to Problem Situation

2.3 Optimization Problems

- **Optimization Problem**: Finding the best solutions from a set of solutions subject to a set of constraints.

2.4 Types of Optimization Algorithms

- **Exact Algorithms**:
  - Find Optimal Solution
  - High Computational Cost
- **Approximate Algorithms**:

  - Find Near-Optimal Solution
  - Low Computational Cost

2.5 Approximate Algorithms

- **Heuristics**: A solution strategy or rules by trial and error to produce acceptable (optimal or sub-optimal) solutions to complex problems in a reasonably practical time.
- **Constructive Methods**: A solution is constructed by iteratively introducing a new component.
- **Local Search Methods**: An initial solution is improved by iteratively applying actions.

2.6 Goal and Problem Formulation

- *Requirements for Search: Goal Formulation + Problem Formulation*
- *Closed World Assumption: All necessary information about a problem domain is available in each percept so that each state is a complete description of the world.*

2.7 Problem Formulation Template

- **State Space**: Complete/Partial Configuration of Problem
  - *Required*: Each State = **UNIQUE**
- **Initial State**: Beginning Search State
- **Goal State**: Ending Search State
- **Action Set**: Set of Possible State Transitions
- **Cost**: Comparison Function between Solutions

2.8 Extra Terminologies

- **State**: Any Possible Agent/Problem Configuration
- **Transition Model**: Action Description

3 Graph Search Algorithms

3.1 Search Tree Terminology

- **Node**: Search Problem State
- **Edge**: Search Problem Action
- **Fringe**: Frontier/Leaves of Search Tree
- **Branching Factor (b)**: The maximum number of child nodes extending from a parent node.
- **Maximum Depth (m)**: The number of edges in the shortest path from the root node to the furthest leaf node.
- **Optimal Goal Depth (d)**: the number of edges in the shortest path from the root node to an optimal goal node.

3.2 Properties of Search Algorithms

- **Completeness**: Guarantee Find A Goal Node
- **Optimality**: Guarantee Find Best Goal Node
- **Time Complexity**: # of Nodes Generated
- **Space Complexity**: # of Nodes Stored

3.3 Generic Search

- *Fringe = Queue-Like Data Structure*

1. Choose Node
2. Test Node
3. Expand Node

3.4 Local Search Strategies

- **Uninformed Strategies**: No knowledge of the direction of goal nodes.
  - Breadth-First
  - Depth-First
  - Depth-Limited
  - Uniform-Cost
  - Depth-First Iterative Deepening
  - Bidirectional
- **Informed Strategies**: Domain knowledge of the direction of goal nodes.
  - Hill Climbing
  - Best-First
  - Greedy Search
  - Beam Search
  - A
  - A*

4 Uninformed Search Strategies

4.1 Breadth-First Search

- Expand the shallowest unexpanded nodes, storing the fringe to be expanded in a FIFO queue.

4.2 Properties of Breadth-First Search

- *Completeness*:
  - Yes - If b is Finite
- *Optimality*:
  - Yes - If Cost = Depth

- *Time Complexity*: $O(b^{d+1}) \approx O(b^d)$
- *Space Complexity*: $O(b^{d+1}) \approx O(b^d)$

4.3 Uniform Cost Search

- Expand the lowest cost unexpanded node, storing the fringe to be expanded in a minimum priority queue.
- **Required**: *No Zero/Negative-Cost Edges*

4.4 Properties of Uniform Cost Search

- Let C^* be the path cost to the goal.
- Let ϵ be the minimum cost of all other actions.
- *Completeness*:
  - Yes - If b is Finite
- *Optimality*: Yes
- *Time Complexity*: $O(b^{\frac{C^*}{\epsilon}+1}) \approx O(b^d)$
- *Space Complexity*: $O(b^{\frac{C^*}{\epsilon}+1}) \approx O(b^d)$

4.5 Depth-First Search

- Expand the deepest unexpanded nodes, storing the fringe to be expanded in a LIFO stack.

4.6 Properties of Depth-First Search

- *Completeness*:
  - No - If Search Space with Infinite-Depth/Loops
- *Optimality*: No
- *Time Complexity*: $O(b^m)$
- *Space Complexity*: $O(bm)$

4.7 Depth-Limited Search

- Execute DFS with a maximum search depth as a restriction.
  - Prevents Infinite-Depth Problem
  - Prevents Loops Problem

4.8 Properties of Depth-Limited Search

- Let l be the maximum search depth.
- *Completeness*:
  - Yes - If Solution's Depth $d \leq l$
- *Optimality*: No
- *Time Complexity*: $O(b^l)$
- *Space Complexity*: $O(bl)$

4.9 Iterative Deepening Search

- Iteratively, execute DLS with an increasing maximum search depth l until a solution is found.

4.10 Properties of Iterative Deepening Search

- *Completeness*:
  - Yes - If b is Finite
- *Optimality*:
  - Yes - If Cost = Depth
- *Time Complexity*: $O(b^d)$
- *Space Complexity*: $O(bd)$

4.11 Breadth-First vs. Depth-First Strategies

4.12 Breadth-First Strategies

- High Memory Requirement
- Never Stuck on Infinite Depths
- Find Shortest Path to Goal

4.13 Depth-First Strategies

- Low Memory Requirement
- Stuck on Infinite Depths
- Find Any Path to Goal

4.14 Avoiding Repeated States

- *Increasing Computational Costs*:

1. Do not return to the state your just came from.
2. Do not create paths with cycles in them.
3. Do not generate any state that was ever created before.

5 Informed Search Strategies

5.1 Overview

- Apply domain knowledge in a problem to search the "most promising" branches first.
- Potentially, find solutions faster or cheaper than uninformed search algorithms.

5.2 Heuristics

- A **heuristic function** $h(n)$ can be used to estimate the "goodness" of node n.
  - $\forall n, h(n) \geq 0$
  - $h(n) = 0 \implies n$ is a goal node.

  - $h(n) = \infty \implies n$ is a dead end that does not lead to a goal.

- **Admissible/Optimistic**: If a heuristic function never overestimates the cost of reaching the goal.

5.3 Strong vs. Weak Methods

- **Strong Methods**: *Specific Approach to Some Problems*
- **Weak Methods**: *General Approach to Many Problems*

5.4 Examples of Weak Methods

- **Mean-End Analysis**: A strategy where a representation is formed for the current and goal state, and actions are analyzed that shrink the difference between the two.
- **Space Splitting**: A strategy where possible solutions to a problem are listed, and then classes of these solutions are ruled out to shrink the search space.
- **Subgoaling**: A strategy where a large problem is split into independent smaller ones.

5.5 Best-First Search/Greedy Search

- *~Uniform Cost Search with Priority Queue*
  - Minimize $f(n) \mapsto h(n)$
  - Greedy If $f(n) = h(n)$

5.6 Properties of Best-First Search/Greedy Search

- *Completeness*:
  - No - Stuck in Loops
- *Optimality*:
  - No
- *Time Complexity*: $O(b^m)$
- *Space Complexity*: $O(b^m)$

5.7 Beam Search

- *~Breadth-First Search + Reduced Memory Requirements*
  - Expands Best β (*Beam Width*) Nodes Per Level

5.8 Properties of Beam Search

- *Admissible*:
  - No
- *Completeness*:
  - No

- *Optimality*:
  - No
- *Time Complexity*: $O(\beta b)$
- *Space Complexity*: $O(\beta b)$

5.9 A Search/A* Search

- *A Search: Best-First Search with* $f(n) = g(n) + h(n)$
  - $g(n)$ is the cost from the start to n.
  - $h(n)$ is the cost from n to the goal.
- *A* Search: Constraint* $h(n) \leq h^*(n)$
  - $h^*(n)$ is the actual minimal path cost from n to the goal.

5.10 Properties of A Search

- *Admissible*:
  - No
- *Completeness*:
  - No - If $h(n) \to \infty$
- *Optimality*:
  - No

5.11 Properties of A* Search

- *Admissible*:
  - Yes - If $h(n) \leq h^*(n)$
- *Completeness*:
  - Yes - If b is finite and only fixed positive costs.
- *Optimality*:
  - Yes

5.12 Hill Climbing Search

- *Improvement of Depth-First Search*
- *~Beam Search with* $\beta = 1$
- *~Greedy Search with No Backtracking*
- *Not Complete at Local Minima, Plateaus, Ridges*

1. Start with an arbitrary solution.
2. Attempt to improve the solution by changing a single element at a time.
3. Sort the successors of a node according to their heuristic values, and then adding them to the list to be expanded.
4. Make changes until no further improvements can be found.

5.13 Rule of Hill Climbing Search

- If there is a successor s for node n such that:
  - $h(s) < h(n)$ and
  - $h(s) \leq h(t)$ for all successors t of n.
- True $\implies$ Advance from n to s.
- False $\implies$ Halt at n.

5.14 Heuristics

- **Perfect Heuristic**: If $h(n) = h^*(n)$, then only nodes on the optimal solution are expanded.
- **Null Heuristic**: If $h(n) = 0$, then A* behaves like uniform cost search.
- **Better Heuristic**: If $h_1(n) < h_2(n) < h^*(n)$, then h_2 is a better heuristic than h_1.

6 Game Playing as Search

6.1 Overview

- Games involve playing against an opponent, where search problems involve finding a good move, waiting for an opponent's response, and then repeating.
- Time is typically limited in each search.

6.2 Problem Formulation of Games

- **Initial State**: *Initial Position + Whose Move It Is*
- **Operators**: *Legal Player Moves*
- **Goal (Terminal Test)**: *Is Game Over?*
- **Utility (Payoff)**: *Measures Outcome/Desirability*

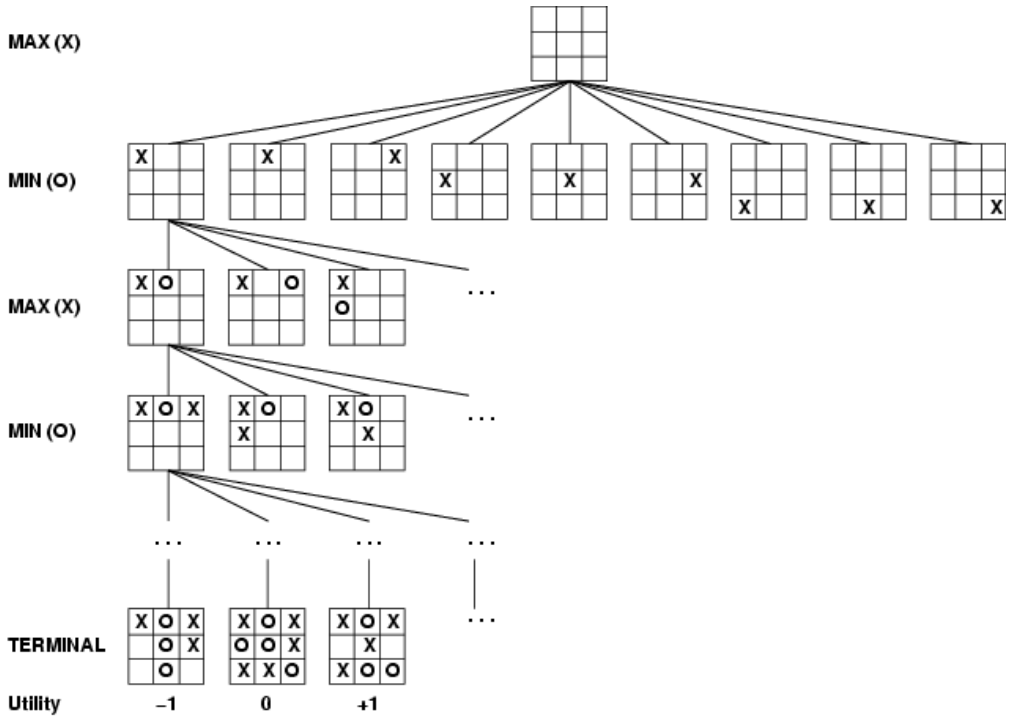
6.3 Types of Games

- **Perfect Information**: Each player has complete information on the opponent's state and available choices.
- **Imperfect Information**: Each player does not have complete information on the opponent's state and available choices.

6.4 Max Min Strategy

- With perfect information and two players, a game tree can be expanded to describe all possible moves of the player and the opponent in the game.
- **Zero Sum Games**: *Player Win* $\implies$ *Opponent Loss*
- **Minimax Principle**: *Minimize the maximum losses that occur.*

6.5 Minimax Algorithm



Example of Minimax Algorithm

- **Important Note**: *Bottom-Up*

1. Generate the game tree labeling each level with alternating MAX$(player)$ and MIN$(opponent)$ labels.
2. Apply the utility function to each terminal state (leaf) to get its minimax value.
3. Extrapolate these minimax values to determine the utility of the nodes on level higher in the search tree.
    - For a MAX$(player)$ level, select the maximum minimax value of its successors.
    - For a MIN$(opponent)$ level, select the minimum minimax value of its successors.
4. From the root node, select the move which leads to the highest minimax value.

6.6 Limited Depth

- For complicated games, a limited depth of the game tree should be explored.
- An **evaluation function** $f(n)$ is used to measure the "goodness" of a game state.

6.7 Properties of Minimax Algorithm

- *Completeness*:
    - Yes - If game tree is finite
- *Optimality*:

  - Yes - If opponent is optimal

- *Time Complexity*: $O(b^d)$
- *Space Complexity*: $O(bd)$

6.8 α-β Pruning

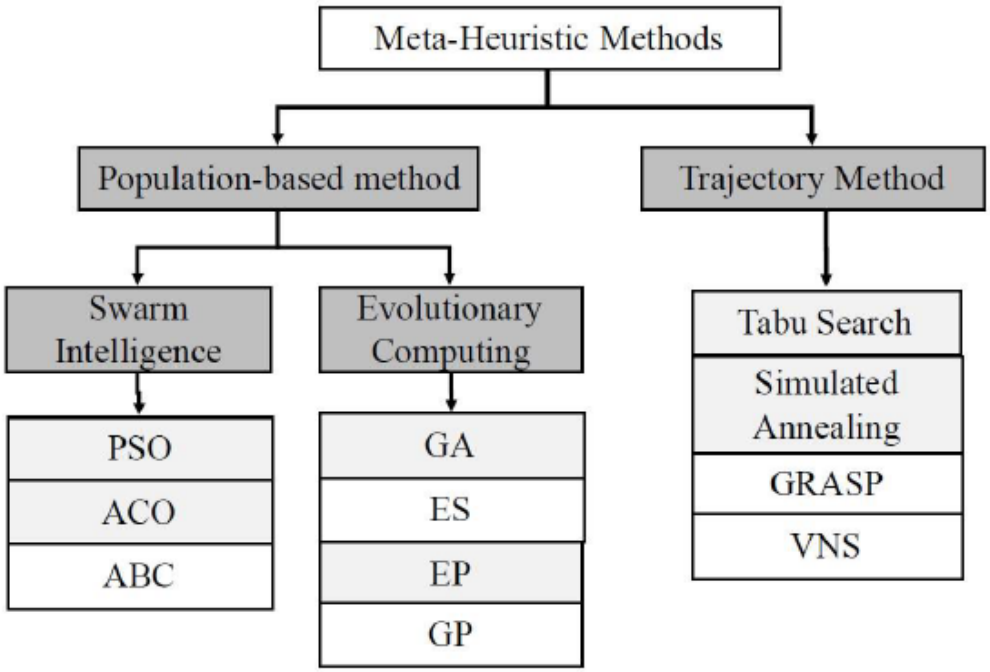
- *Branch and Bound: Reduce # of Generated/Evaluated Nodes*
  - *Avoid Processing Subtrees $\neq$ Affecting Result*
- **Alpha (α)**: The best value for MAX seen so far.
  - Used in MIN nodes
  - Assigned in MAX nodes
  - Never Decreases
- **Beta (β)**: The best value for MIN seen so far.
  - Used in MAX nodes
  - Assigned in MIN nodes
  - Never Increases
- **Alpha Cutoff** (*Lower Bound*): When the value of a minimum position is less than or equal to the alpha-value of its parent, stop generating further successors.
- **Beta Cutoff** (*Upper Bound*): When the value of a maximum position is greater than the beta-value of its parent, stop generating further successors.

6.9 α-β Minimax Algorithm Revisions

1. Search discontinued below any MIN with $\beta \leq \alpha$ of one of its ancestors.
   - Set final value of the node to be this β value.
2. Search discontinued below any MAX with $\alpha \geq \beta$ of one of its ancestors.
   - Set final value of the node to be this α value.

7 Metaheuristics

7.1 Overview



Overview of Metaheuristic Methods

- **Metaheuristics**: High-level heuristics designed to select other heuristics to solve a problem by exploring and exploiting its search space.
  - *Approximate Solutions*
  - *Nondeterministic Solutions*

7.2 Properties

- Mechanisms to avoid getting trapped in confined areas of the search space.
- Not problem-specific; may use domain-specific knowledge from heuristics controlled by upper-level strategy.
- Search history to guide the search.
- Hybrid search models where the search identifies neighborhoods where a goal may lie, and then the search is intensified in that area.

7.3 Population-Based Methods

- Population-based methods are metaheuristic approaches that apply multiple agents to a search space and can handle multiple simultaneous solutions.

7.4 Trajectory Methods

- Trajectory methods are metaheuristic variants of local search that apply memory structure to avoid getting stuck at local minima, and implement an explorative strategy that tries to avoid revisiting nodes.

8 Simulated Annealing

8.1 Physical Annealing Analogy

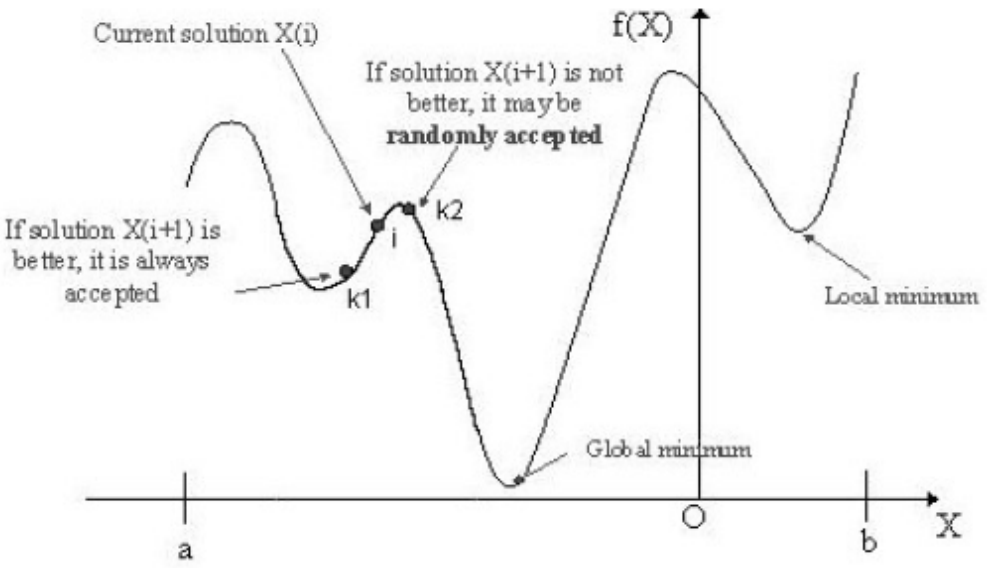
- Physical annealing involves heating a substance (e.g. a metal) and then letting it cool to increase its ductility and reduce hardness.
- The goal is to make the molecules in a cooled substances arrange themselves in a low-energy structure, and the properties of this structure are influenced by the temperatures reached and the rate of cooling.
- A sequence of cooling times and temperatures is referred to as an annealing or cooling schedule.

8.2 Simulated Annealing Algorithm

- Let $s = s_0$ be a current solution initialized to s_0.
- Let $t = t_0$ be a current temperature initialized to t_0.
- Let α be a temperature reduction function.

1. Repeat,
    1. Repeat,
        1. Select a solution s_i from the neighborhood $N(s)$.
        2. Calculate the change in cost ΔC.
        3. If $\Delta C < 0$, then accept the new solution: $s = s_i$.
        4. Else, generate a random number $x \in (0, 1)$.
        5. If $x < \exp(\frac{-\Delta C}{t})$, then accept the new solution: $s = s_i$.
    2. Until maximum number of iteration for t.
    3. Decrease t using α.

8.3 Strategy



Simulated Annealing Strategy

- Simulated annealing always accepts better solutions.
- Simulated annealing randomly accepts worse solutions.
- At higher temperatures, explore parameter space.
- At lower temperatures, restrict exploration.

$$\text{Low Temperature} \wedge \text{High Change in Cost} \implies \text{Low Acceptance Probability}$$

8.4 Annealing Schedule

- **Annealing Schedule**: *Adjusts Temperature*
  - *Initial Temperature*
  - *Final Temperature*
  - *Temperature Decrement Rule*
  - *Temperature Iterations*

8.5 Initial Temperature

- The initial temperature should be high enough to allow exploration to any part of the search space.
- If the initial temperature is too hot, simulated annealing would behave too randomly.
- The maximum change of a cost function should be considered when setting the initial temperature.
- **General Rule**: Set the initial temperature to accept around 60% of worse solutions.

8.6 Final Temperature

- The final temperature should be quite low but not neccessarily have to reach zero.
- A search using simulated annealing can be stopped once no better moves are being found and no worse moves are being accepted.

8.7 Temperature Decrement Rule

- **Linear**: $t = t - \alpha$
- **Geometric**: $t = t \times alpha$
- **Slow Decrease**: $t = \frac{t}{1+\beta t}$

8.8 Temperature Iterations

- Enough iterations should be allowed at every temperature for the system to be stable at that temperature.
- If the search space is very large, a large number of iterations may be required.
- If the slow decrease rule is used, one iteration per temperature should be used.

8.9 Convergence

- Simulated annealing is guaranteed to eventually converge to a solution at a constant temperature, assuming some sequence of moves leads to the goal state.
- When temperature is not constant, convergence can still be guaranteed but only under conditions that result in very slow temperature reduction and an exponential increase in the number of iterations at each temperature.

8.10 Advantages and Disadvantages

8.11 Advantages

- Easy
- Widely Applicable

8.12 Disadvantages

- Time Complexity
- Many Tunable Parameters

8.13 Adaptation

- **Adaptation** refers to adapting the critical parameters of the simulated annealing algorithm.

8.14 Initial Temperature

- Finding the right temperature is very problem-specific, and different search algorithms can be applied in finding this temperature.

8.15 Cooling Schedule

- Some cooling schedules require that only the cooling rate α is specified, and the remainder of parameters are automatically determined using a linear random combination of previously accepted states and parameters to estimate new steps and parameters.

8.16 Probability of Acceptance

- Some attempted adaptations concerning the probability of acceptance include using a lookup table for the relevant calculations (to decrease computation time) or using a different, non-exponential probability formula.

8.17 Cost Function

- Cost functions that return similar values for many different states tend to not lead to effective search.
- As an alternative, a cost function can have a penalty term associated with certain types of states, and weighting of these penalty terms can vary dynamically.

8.18 Cooperation

- Cooperative simulated annealing involves multiple concurrent runs of a simulated annealing search algorithm on a search space.
- Potential solutions are produced by somehow combining the value of a run with the value of a random previous run of the algorithm.
- This exchange of information from other solutions is known as cooperative transition, and is a concept borrowed from genetic algorithms.

9 Tabu Search

9.1 Overview

- **Tabu Search**: A general trajectory-based metaheuristic strategy for controlling inner heuristics.
  - *Combination of Local Search Strategy + Search Experience Model* $\implies$ *Escape Local Minima + Explorative Strategy*

9.2 Local Search Strategy

1. Start with an initial feasible solution.
2. Repeat,
    1. Generate a neighboring solution by applying a series of local modifications.
    2. If the new solution is better, then replace the current solution.

9.3 Local Search Strategy Challenges

1. It can be costly to consider all possible local modifications.
2. It can get stuck in local optimum.

9.4 Basic Ideas

- Penalize moves that take a solution back to a previously visited (tabu) state.
- Accepts non-improving solutions in order to escape from local optima and eventually find a better solution.

9.5 Use of Memory

- **Short-Term Memory** (*Recency of Occurrence*): Prevent the search from revisiting recently visited solutions.
  - **Tabu List**: A short-term memory structure that stores recent moves applied to the current solutions or their attributes.
  - **Tabu Tenure**: The number of iterations T for which a certain move or its attributes are kept in the list.
  - Complete solutions are rarely used because of the space requirement.
- **Long-Term Memory** (*Frequency of Occurrence*): Prevent the search from revisiting frequently visited solutions.

9.6 Neighborhood

- When selecting a new state, consider neighbors that are not on the tabu list.

$$N(s) - T(s)$$

- The neighborhood structure $N(s)$ can be reduced and modified based on history and knowledge.

9.7 Termination Conditions

- *Sample Conditions*:
  - No feasible solution in the neighborhood of current solution.
  - Reached the maximum number of iterations allowed.
  - The number of iterations since the last improvement is larger than a specified number.
  - Evidence shows that an optimum solution has been obtained.

9.8 Candidates and Aspiration

- A **candidate list** stores the potential solutions in a neighborhood to be examined.
  - Isolate regions of a neighborhood with desirable features, aiming to find a solution more efficiently.
  - At times, it can be desirable to include a move in a candidate list even if it is tabu in order to prevent **stagnation**.
  - **Aspiration Criteria**: *Approaches for Canceling Tabus*

9.9 Tabu Search Algorithm

- Let $s = s_0$ be a current solution initialized to s_0.
- Let $N(s)$ be the neighborhood of the current solution.
- Let $T(s)$ be the tabu list of the current solution.
- Let $A(s)$ be the aspiration list of the current solution.

1. Repeat,
    1. Select the best solution s' from $N^*(s) = N(s) - T(s) + A(s)$.
    2. Memorize s' if it is improves the best known solution.
    3. Let $s = s'$.
    4. Update $T(s)$ and $A(s)$
2. Until termination criteria are met.

9.10 Selecting Tabu Restrictions

- *Sample Restrictions*:
    - Not picking a move that involves the same exchange of positions of a tabu move.
    - Not picking a move that results in positions that previously appeared in a tabu move.

9.11 Selecting Tabu Tenure

- *Sample Strategies*:
    - Statically assigning T to be a constant: $\sqrt{n}$ where n is the problem size.
    - Dynamically letting T vary between a T_{min} and a T_{max}.
        * *Advantages: Better Limits Cycles*
        * *Disadvantages: May Contain Cycles Longer Than Tabu Tenure*

9.12 Selecting Aspiration Criteria

- *Sample Strategies*:
    - By default, where a tabu move becomes admissible if it yields a better solution than any found so far.
    - By objective, where a tabu move becomes admissible if it yield a solution better than an aspiration value.
    - By search direction, where a tabu move becomes admissible if the direction of the search remains constant.

9.13 Intensification

- **Intensification**: The process of exploiting a small portion of the search space (e.g. penalizing solutions far from the current solution).
- *General Idea*: Locally optimize a best known solution while trying to preserve the general components of that solution (based on short-term memory).

9.14 Diversification

- **Diversification**: The process of forcing the search into unexplored areas (e.g. penalizing solutions close to the current solution).
- *General Problem*: Tabu search can miss some good solutions in unexplored search space areas.

9.15 Diversification Strategies

- **Restart Diversification**: Where components rarely appearing in solutions are forced into new solutions.
- **Continuous Diversification**: Where the evaluation of possible moves is biased by a term related to component frequency.

9.16 Adaptation

- **Adaptation** refers to a series of techniques for varying the tabu tenure.
  - If the tabu tenure is too small, then cycles in the search are likely.
  - If the tabu tenure is too big, then many moves could be prevented at each iteration.

9.17 Adaptation Techniques

- Randomly select a new tenure from a pre-computed range every predetermined number of iterations.
- Set the tenure to one if a best-so-far solution is found, decreasing it in an improving phase, and increasing it in a worsening phase.

9.18 Cooperation

- Cooperative tabu search involves multiple concurrent runs of a tabu search algorithm on a search space.
- **Synchronous Communication**: Search agents exchange information every fixed number of iterations.
- **Asynchronous Communication**: Search agents relay their best-so-far results to a central memory.
- **Forced Diversification**: A search agent can replace its own best-so-far solution with an incoming solution.
- **Conditional Import**: A seach agent can replace its own best-so-far solution with an incoming solution only if the incoming solution is better.

9.19 Observations on Tabu Search Cooperation

- Increasing the number of search agents improves the solution up to a certain point.
- Increasing the number of synchronization messages increases the computation time due to message passing overhead.
- Conditional imports are almost always preferable to forced diversification.

10 Swarm Intelligence

10.1 Overview

- **Swarm Intelligence**: The collective behavior of decentralized, self-organized systems.

10.2 Interactions

- **Swarm**: A group of agents that communicate with each other by acting on their local environment.
- Complex problem-solving behavior may emerge not as a result of any particular individual, but rather as a result of their interactions.
- Interactions between individuals may be direct (physical contact) or indirect (via local change to the environment, **stigmergy**).

10.3 Properties

- **Flexibility**: System performance is adaptive to internal or external changes.
- **Robustness**: System can perform even if some individuals fail.
- **Decentralization**: Control is distributed among individuals rather than allocated to some master.
- **Self-Organization**: Global behaviors emerge as a result of local interactions.

10.4 Models of Behavior

- **Swarm**: Group with little parallel alignment.
- **Torus**: Group in which individuals rotate around an empty core in one direction.
- **Dynamic Parallel Group**: Group where individuals are polarized and move as a coherent group, but can still move throughout the group such that the group density and form fluctuates.
- **Highly Parallel Group**: Group similar to dynamical parallel group but with minimal fluctuations.

10.5 Swarm Intelligence Problem Solving

- **Proximity**: The swarm should be able to carry out simple space and time computations.
- **Quality**: The swarm should be able to respond to quality factors in its environment.
- **Diverse Response**: The swarm should not commit to excessively narrow channels of exploration.
- **Stability**: The swarm should not rapidly alter its behavior in response to all environmental changes.
- **Adaptability**: The swarm should be willing to change its behavior when it is worth the computational price.

11 Ant Colony Optimization

11.1 Background

- **Ant Colony Optimization**: A search technique based on the swarm intelligence of ants.

- Ants interact with one another through stigmergy, meaning that individual ants can make changes to their environment to be picked up by other ants.
- Ants do this by forming trails with substances known as pheromone. Ants tend to follow paths with higher pheromone concentrations. Pheromone also evaporates over time, so more recent pheromone deposits have a greater influence than older ones.
- Ants can use this mechanism to find the shortest path to a destination. If multiple paths are available, ants will pick their initial paths randomly. Ants that take the shorter path will return faster, so their trail will have more unevaporated pheromone. This makes other ants more likely to pick this trail, and eventually, they converge on this shortest path.

11.2 ACO Algorithm

- Let $G = (N, E)$ be a graph where N is a set of nodes and E is a set of edges.
- Let $d_{i,j}$ be the length of each edge (i, j).
- Let $\tau_{i,j}$ be the amount of pheromones of each edge (i, j).

11.3 Initialization

- Initialize all edges with a small amount of pheromones.
- Initialize the source node with a group of m ants.

11.4 Transition Rule

- At each node i, with adjacent nodes N_i, an ant can move to an adjacent node j with the following probability.

$$\frac{\frac{\tau_{i,j}^{\alpha}}{d_{i,j}^{\beta}}}{\sum_{n \in N_i} \frac{\tau_{i,n}^{\alpha}}{d_{i,n}^{\beta}}}$$

- α and β balance the local and the global search abilities respectively.

11.5 Pheromone Evaporation and Update

- In each step, the amount of pheromone on the trail is evaporated (i.e. $\tau = \tau \cdot (1 - p)$).
- In each step, the amount of pheromone on the trail is increased if ants choose the trail (i.e. $\tau = \tau + \Delta\tau$).
  - **Ant Density Model**: Where a constant value is added.
  - **Ant Quality Model**: Where a constant is divided by the edge length.
  - **Online Delayed Model**: Where an ant first builds a solution, then traces its path backwards and adds pheromone based on the solution quality.

11.6 Termination Criteria

- *Sample Criteria*:
  - *Maximum Number of Iterations Reached*
  - *Good Enough Solution Reached*
  - *Stagnation Occurs*

11.7 Tunable Parameters

- *Number of Ants*
- *Maximum Number of Iterations*
- *Initial Pheromone*
- *Pheromone Delay Parameters (i.e. p)*

11.8 Ant Colony System Algorithm

- The **ant colony system** (ACS) algorithm features the following extensions on ACO:
  - The transition rules sometimes just choose the best path (i.e. acts greedily) instead of applying probabilistic selection.
  - Pheromone update is only based on the best solution (i.e. highest increase in τ either globally or in iteration).

11.9 Max-Min Ant System Algorithm

- The **max-min ant system** algorithm is an extension that restricts pheromone values within a range.
  - The max and min pheromone values can be adjusted to favor exploration over exploitation during early phases of the search.

11.10 Adaptation

11.11 ACSGA-TSP

- Have a genetic algorithm running on top of ACS to attempt to optimize its parameter values.
- Each ant has certain parameters (e.g. p, β) encoded into a chromosome.
- New generations are formed by crossing over the best-performing ants.

11.12 Near Parameter Free ACS

- Apply an ant approach to optimize ant parameters.

11.13 Cooperation

- Heterogeneous cooperation approaches involve ants in different colonies having different behavior.
  - This can be used, for instance, to optimize for different criteria of a solution.
- Homoegeneous cooperation approaches involve ants in different colonies having similar behavior.
  - Exchange of information can take place in a similar way to that of genetic algorithms.
  - Homogeneous cooperation implementations can be course-grained, where each process holds a single ant, or fine-grained, where each process holds a colony.
  - An effective version of homogeneous cooperation is a circular exchange of locally best solutions.

11.14 Advantages and Disadvantages

11.15 Advantages

- Memory of entire colony retained.
- Poor solutions rarely converged on due to many combinations of path selection.
- Effective handling of dynamic environments.

11.16 Disadvantages

- Theoretical analysis is limited.
- Many parameters to tune.
- Convergence may take long.

12 Particle Swarm Optimization

12.1 Background

- **Goal**: *Simulate Collective Behavior*
  - Individuals have no knowledge of the global behavior of the group.
  - Individuals have the ability to move together based on social interaction between neighbors.
- **Separation Behavior**: Each agent tries to move away from its nearby mates if they are too close.
- **Alignment Behavior**: Each agent steers towards the average heading of its nearby mates.
- **Cohesion Behavior**: Each agent tries to go towards the average position of its nearby mates.
- **Roost**: An attractor for agents.
  - Represented using the agent's previous best position and the the neighborhood's best position.
  - By adjusting the positions of the swarm proportion to the distance from the best positions, they converge to the goal.

12.2 Overview

- **Particle Swarm Optimization**: A stochastic optimization approach that manipulates a number of candidate solutions at once.
- **Particle**: An individual solution.
- **Swarm**: A whole population of solutions.

12.3 PSO Algorithm

- Let x_i be a particle's current position.
- Let v_i be a particle's current velocity.
- Let $pbest_i$ be a particle's best position.
- Let $Nbest$ be a particle's neighborhood's best position.
  - If the neighborhood is the whole swarm, the best position is called the *global best*: $gbest_i$ or p_g.

- If the neighborhood is restricted to few particles, the best position is called the *local best*: $lbest_i$ or p_l.

12.4 Motion Equations

$$v_{t+1}^{id} = w * v_t^{id} + c_1 r_1^{id}(pbest_t^{id} - x_t^{id}) + c_2 r_2^{id}(Nbest_t^{id} - x_t^{id})$$
$$x_{t+1}^{id} = x_t^{id} + v_{t+1}^{id}$$

- w is the inertia weight.
- c_1 and c_2 are the acceleration coefficients.
- r_1 and r_2 are randomly generated numbers in $[0, 1]$.
  - Generated for each dimension and not for each particle.
- t is the iteration number.
- i and d are the particle number and the dimension.

12.4.1 Interpretation of Motion Equations

- **Inertia**: $w * v_t^{id}$
  - A particle cannot suddenly change its direction of movement.
- **Cognitive Component**: $c_1 r_1^{id}(pbest_t^{id} - x_t^{id})$
  - c_1 makes a particle trust its own experience.
- **Social Component**: $c_2 r_2^{id}(Nbest_t^{id} - x_t^{id})$
  - c_2 makes a particle trust the swarm experience.

12.5 Synchronous Update

1. Initialize the swarm.
2. While the termination criteria is not met,
   1. For each particle,
      1. Update the particle's velocity.
      2. Update the particle's position.
      3. Update the particle's personal best.
   2. Update the $Nbest$.

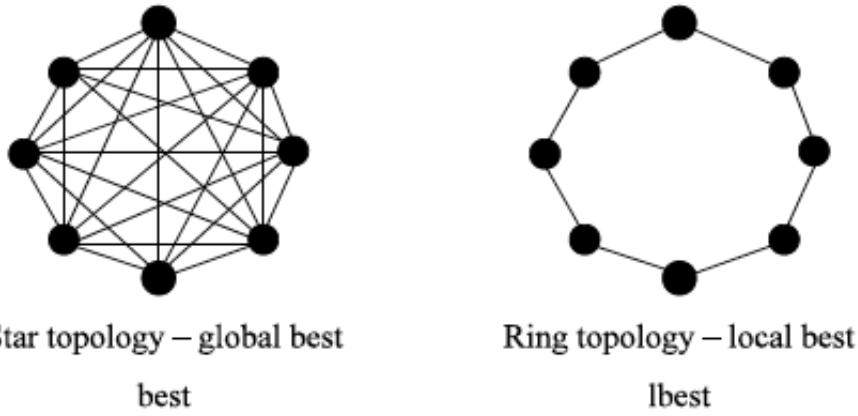
12.6 Asynchronous Update

1. Initialize the swarm.
2. While the termination criteria is not met,
   1. For each particle,
      1. Update the particle's velocity.
      2. Update the particle's position.
      3. Update the particle's personal best.
      4. Update the $Nbest$.

12.7 Termination Criteria

- *Sample Conditions*:
  - A max number of iterations has been reached.
  - A max number of function evaluations have been reached.
  - An acceptable solution has been found.
  - No improvement over a number of iterations.

12.8 Neighborhoods



Neighbourhoods

12.9 Global Best Model

- Each particle is influenced by all the other particles.
- The fastest propagation of information.
- Easily stuck in local minima.

12.10 Local Best Model

- Each particle is influenced only by particles in its own neighbourhood.
- The slowest propagation of information.
- Does not get easily stuck in local minima.

12.11 Initialization

- Initialize $x_i = \text{random}([\alpha, \beta])$.
- Initialize $v_i = 0$ or $v_i = \epsilon$.
- Initialize $pbest_i = x_i$.
- Initialize $c_1 = c_2$.
  - If $c_1 = 0$, social-only model; particles are all attracted to $Nbest$.
  - If $c_2 = 0$, cognition-only model; particles are independent hill climbers.
  - Small c promotes smooth trajectories.
  - Large c promotes abrupt trajectories.

- Initialize w to balance exploration and exploitation.
  - Small w promotes exploitation.
  - Large w promotes exploration.

12.12 Convergence

- Studies of PSO proved what suitable parameters guarantee convergence of a deterministic PSO algorithm.

12.13 Binary PSO

- Each position is either a one or a zero.
- Each velocity is the probability that one element will be a one or a zero.
  - The sigmoid function $sig(v_t^{id}) = \frac{1}{1+e^{-V_t^{id}}}$ ensures the velocities represent probabilities.
  - If $r < sig(v_{t+1}^{id})$, then $x_{t+1}^{id} = 1$, else $x_{t+1}^{id} = 0$.
    - Where r is a randomly generated number in $[0, 1]$.

12.14 Permutation PSO

- Each position is a permutation.
- Each velocity is the set of swaps to be performed on a particle.

12.15 Adding a Velocity to a Position

- Apply the sequence of swaps defined by the velocity to the position vector.

12.16 Subtracting Two Positions

- Subtracting two positions should produce a velocity.
- Produces the sequence of swaps that could transform one position to the other.

12.17 Multiplying a Velocity by a Constant

- Change the length of the velocity vector (number of swaps) accoding to the constant c:
  - If $c = 0$, the length is set to zero.
  - If $c < 1$, the velocity is truncated.
  - If $c > 1$, the velocity is augmented.

12.18 Permutation PSO (Alternative)

- **Space Transformation**: *Continuous Domain to Permutation*
- **Great Value Priority**: All the elements in the position vector and their indices were sorted in descending order.
  - The sorted indices are treated as a permutation.
  - If x_i had the highest value in the position vector, i comes first in the permutation vector.

12.19 Adaptation

- **Tribe**: A group of connected particles.
  - All tribes communicate to decide the global optimum amongst all the different solutions.
- **Neutral Particle**:: A particle whose *pbest* did not improve in the last iteration.
- **Good Particle**: A particle whose *pbest* improved in the last iteration.
- **Excellent Particle**: A particle whose *pbest* improved in the last two iterations.
- A tribe is marked as *good* depending on the value of G and T.

$$Tribe = \begin{cases} \text{Good} & \text{Uniform}(0,1) < \frac{G}{T} \\ \text{Bad} & \text{otherwise} \end{cases}$$

  - Where G is the number good particles in the tribe.
  - Where T is the number particles in the tribe.
- A good tribe deletes its worst particle to conserve the number of performed function evaluations.
- A bad tribe generates a new random particle, and all the new random particles form a new tribe.
- Each new random particle gets connected to the tribe that generated it through its best particle.

12.20 Cooperation

12.21 Concurrent PSO

- Two different swarms are updated in parallel, both using different algorithms.
- The swarms exchange their *gbest* values every pre-determined number of iterations.
- Both swarms track the better *gbest*.

12.22 Cooperative PSO

- Have different swarms optimizing different variables of the problem / different dimensions of the solution.
- The fitness of any particle is determined by its value and the value of the best particles in all the other swarms.
- **Important Note**: Assumes that the problem variables are independent.

12.23 Hybrid Cooperative PSO

- Have one swarm use the normal PSO algorithm.
- Have another swarm use the concurrent PSO algorithm.
- Each swarm is updated for one iteration only.
- When the PSO swarm gets updated, the *gbest* values are sent to the CPSO swarm.
  - The CPSO swarm uses the elements of the received *gbest* to update random particles of its sub-swarms.
- When the CPSO swarm gets updated, its sends its context vector to the PSO swarm.
  - The PSO swarm uses the received context vector to replace a randomly chosen particle.

13 Genetic Algorithms

13.1 Overview of Evolutionary Algorithms

- **Evolutionary Algorithms**: Population-based meta-heuristic methods whose behavior is inspired by biological evolution.
- A population of individuals compete for limited resources, with the "fitter" individuals being used as seeds to form future generations.
- Over time, the population rises in overall fitness due to the principles of natural selection.
- Evolutionary algorithms are stochastic, with the variation operators (*crossover* and *mutation*) propagating changes to new generations.

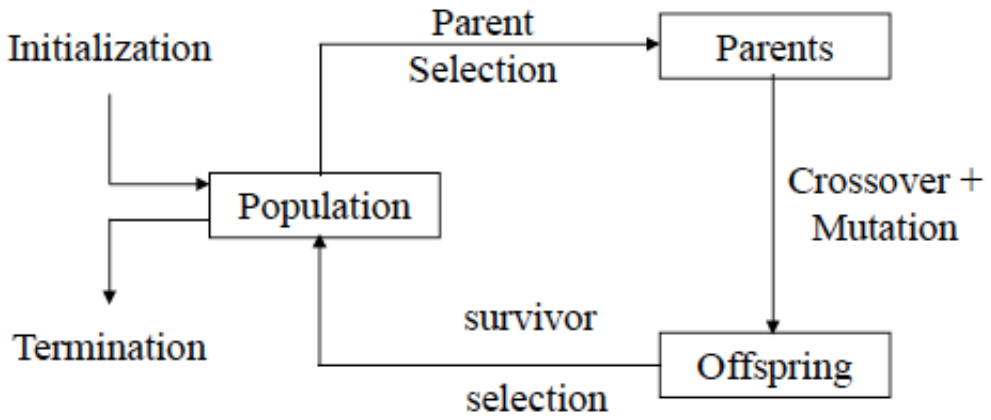
13.2 Active Information

- **Conservation of Information Theorems**: Any search algorithm performs on average as well as random search without replacement unless it takes advantage of problem-specific information about the search target or the search-space structure.
- Three measures of information that can increase the effectiveness of a search can be categorized as:
  - **Endogeneous Information**: Measures the difficulty of finding a target through random search.
  - **Exogeneous Information**: Measures the difficulty of finding a target once problem-specific information is applied.
  - **Active Information**: The difference between exogeneous and endogeneous informaton.
    - i.e., Measures the contribution of problem-specific information in solving a problem.

13.3 Overview of Genetic Algorithms

- **Genetic Algorithms**: A class of evolutionary algorithms that operate by maintaining a population of candidate solutions and iteratively applying a set of stochastic operators, namely *selection*, *reproduction*, and *mutation*.
- Inspired by Darwin's theory of natural selection, the population eventually moves towards fitter solutions.

13.4 Simple Genetic Algorithms



General Scheme of Simple Genetic Algorithms

- **Representation**: *Binary Strings*
- **Recombination**: 1-*Point, N-Point, or Uniform*
- **Mutation**: *Bitwise Bit-Flipping with Fixed Probability*
- **Parent Selection**: *Fitness-Proportionate*
- **Survivor Selection**: *All Children Replace Parents*
- **Speciality**: *Emphasis on Crossover*

13.5 Algorithm

1. Initialize the population with random candidates.
2. Evaluate all individuals.
3. Repeat,
    1. Select parents,
    2. Apply crossover.
    3. Mutate offspring.
    4. Replace current generation.
4. Until termination criteria is met.

13.6 Termination Criteria

- *Sample Criteria*:
    - A specified number of generations (or fitness evaluations).
    - A minimum threshold reached.
    - No improvement in the best individual for a specified number of generations.
    - Memory/time constraints.

13.7 SGA Representation

- **Chromosomes/Genotype**: Representation of candidate solutions as binary strings.

- *Individual Gene*: An individual binary digit.
- *Ordering of Genes*: The most important factor for the performance of a SGA.

- **Gray Coding**: Small differences in the underlying solution result in small changes in the binary representation.

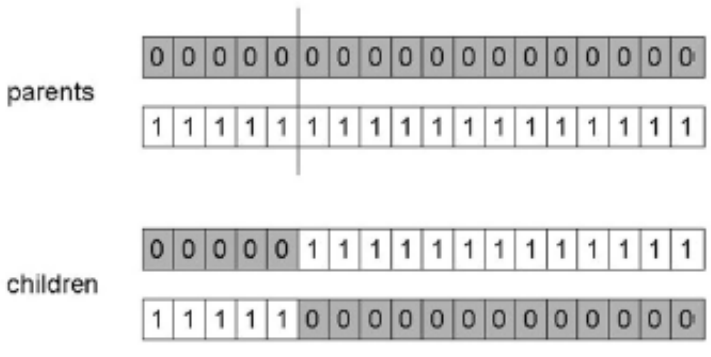
13.8 SGA Selection

- **Fitness Proportional Selection Algorithms**: Select parents with a probability proportional to their fitness.
  - *Advantage*: Probabilistically, N parents can be selected from a population of N solutions $\implies$ A balance between exploration and exploitation.
  - *Disadvantage 1*: No guarantee on the distribution of selected parents, since each selection is done independently.
  - *Disadvantage 2*: Premature convergence with one highly fit member dominating a population.
  - *Disadvantage 3*: Lack of selection pressure at the end of runs with similar fitness.
    - Consider *ranked selection* or *tournament selection*.
- **Roulette Wheel Technique**: Implement FPS by assigning each individual a part of the roulette wheel proportional to their probability, and "spin" the wheel N times to select N individuals.
  - i.e., Multinomial Sampling.

13.9 SGA Crossover

- **Crossover**: A recombination is applied between two parents with a probability of P_c, which is typically in the range $(0.6, 0.9)$.
- If no crossover occurs, the two parents are copied to two offsprings unmodified.

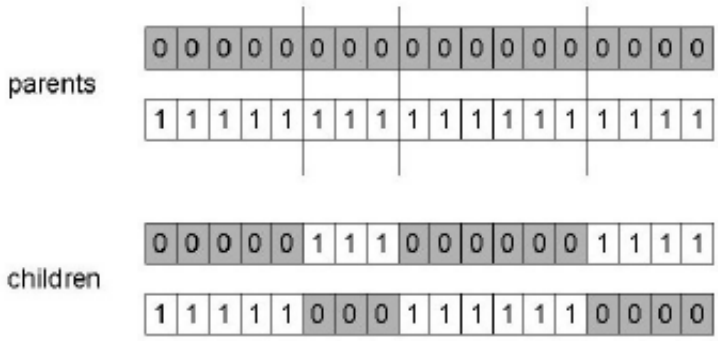
13.10 1-Point Crossover



1-Point Crossover Example

- A random point is chosen on the two parents, and the two children are formed by exchanging the tails this point partitions.

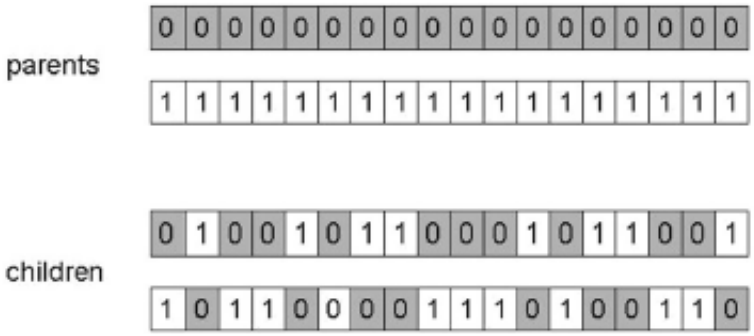
13.11 N-Point Crossover



N-Point Crossover Example

- A generalization of 1-point crossovers where N points are chosen on the two parents, and the two children are formed by combining alternating partitions between points.

13.12 Uniform Crossover



Uniform Crossover Example

- Each gene has an independent 0.5 chance of undergoing recombination, which makes inheritance independent of position.
- Prevents transmitting co-adapted genes.

13.13 SGA Mutation

- After recombination, each gene can be altered with a probability of P_m, typically between $(\frac{1}{\text{Population Size}}, \frac{1}{\text{Chromosome Length}})$.
- Crossovers tend to result in large changes in a population, while mutation results in small ones.
- Thus, crossovers are considered to be *explorative*, making a big jump to a possibly unexplored area between two parent solutions.
- Thus, mutations are *exploitative*, introducing small amounts of new information to further explore near an existing solution.

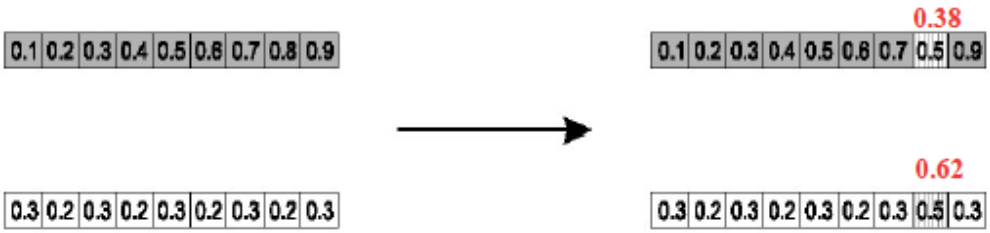
13.14 SGA Population Models

- **Generational Genetic Algorithm Model** (*GGA*): Individuals survive for exactly one generation before they are replaced by offspring.
- **Steady-State Genetic Algorithm Model** (*SSGA*): Part of a population is replaced by offspring.
- **Generational Gap**: The proportion of the population replaced between successive generations is known as a generational gap.
  - *GGA*: 1.
  - *SSGA*: $\frac{1}{\text{Population Size}}$.
- **Survivor Selection**: The process of selecting individuals from parents and offsprings to make up the next generation.
  - e.g., Age-Based $\Longrightarrow$ Delete Oldest.
  - e.g., Fitness-Based $\Longrightarrow$ Delete Worst.

13.15 Real-Valued Genetic Algorithms

13.16 Crossover

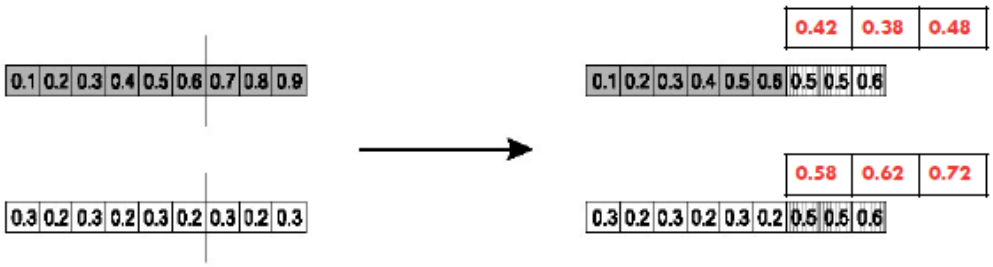
13.16.1 Single Arithmetic Crossover (Single Random)



Single Arithmetic Crossover Example

- For a single parent gene pair x and y, one child's gene becomes $\alpha x + (1 - \alpha)y$, and the reverse for the other child.

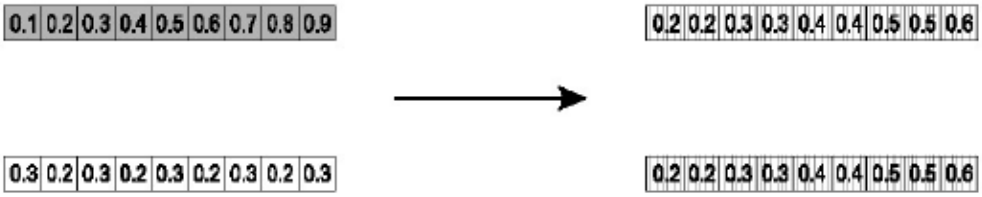
13.16.2 Simple Arithmetic Crossover (Random After k)



Simple Arithmetic Crossover Example

- For each parent gene pair x and y, after a certain gene pair k, one child's gene becomes $\alpha x + (1 - \alpha)y$, and the reverse for the other child.

13.16.3 Whole Arithmetic Crossover (All Random)



Whole Arithmetic Crossover Example

- For each parent gene pair x and y, one child's gene becomes $\alpha x + (1 - \alpha)y$, and the reverse for the other child.

13.17 Mutation

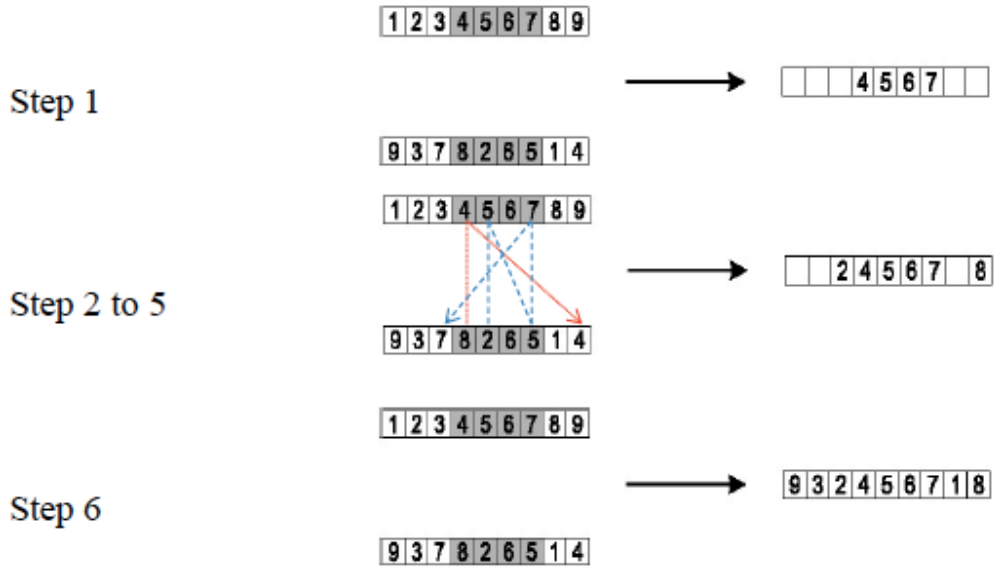
- Assign a uniform random value between some lower and upper bound to a gene.
- Some variants on this technique also adds some noise (e.g. from a Gaussian distribution) to this number.

13.18 Permutation Genetic Algorithms

- **Permutation Problems**: *Arranging Elements.*
  - *Adjacency of Elements.*
  - *Overall Order of Elements.*

13.19 Crossover

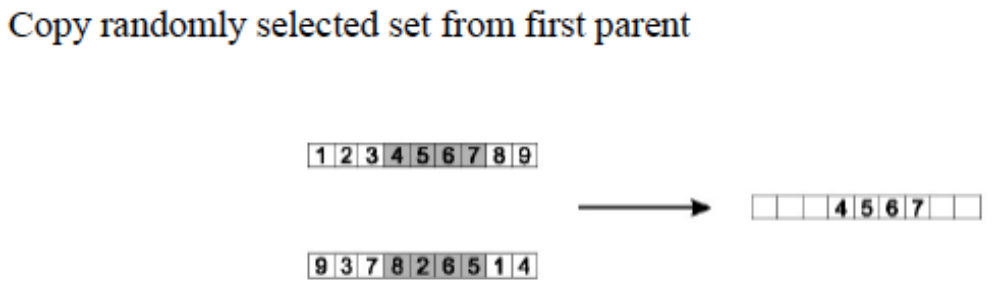
13.19.1 Partially Mapped Crossover (PMX)



Partially Mapped Crossover Example

1. Copy a random segment from parent P_1.
2. Starting from the first crossover point, look for elements in parent P_2 that have not been copied.
3. For each of these elements i, look in the offspring to see which element j has been copied in its place.
4. Place i in the position occupied by j in P_2. If the place occupied by j in P_2 has already been filled in by k, put i in the position occupied by k in P_2.
5. The rest of the offspring can be filled in from P_2.

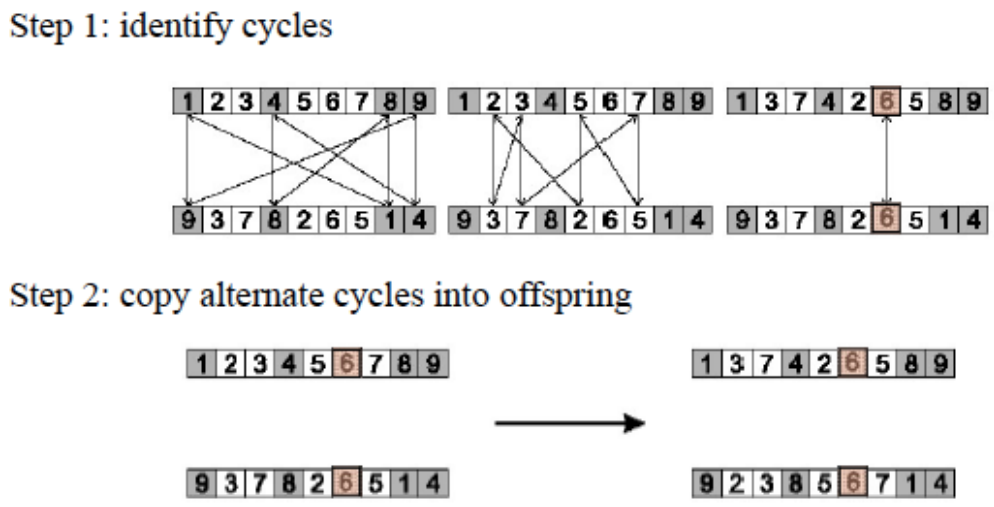
13.19.2 Order 1 Edge Crossover



Order 1 Edge Crossover Example

1. Choose arbitrary part from P_1.
2. Copy this part to the first child.
3. Starting from the right of the cut point of the copied part, copy the elements in the order of P_2 that are not yet in the child, wrapping around if needed.
4. Analogous for the second child, with parent roles reversed.

13.19.3 Cycle Edge Crossover



Cycle Edge Crossover Crossover Example

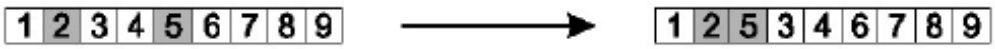
1. Form a cycle of genes from P_1 by the following,

1. Start with the first gene from P_1.
2. Go to the position in P_1 that has the value of the corresponding gene in P_2.
3. Add this gene to the cycle.
4. Repeat this cycle formation until the first gene of P_1 is reached.

2. Put the genes of the cycle in the positions from P_1.
3. Repeat steps above for the second parent.

13.20 Mutation

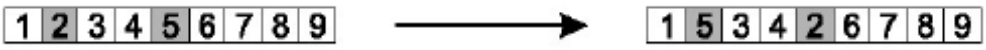
13.20.1 Insert Mutation



Insert Mutation Example

- Pick two elements, and move one to immediately follow the other, shifting any elements if necessary (i.e. similar to insertion sort).

13.20.2 Swap Mutation



Swap Mutation Example

- Pick two elements and swap their order.

13.20.3 Inversion Mutation

Inversion Mutation Example

- Pick two genes and reverse the subsequence between them.

13.20.4 Scramble Mutation

Scramble Mutation Example

- Pick two genes and find a random permutation of the genes between them.