AI LAB 2025

ARTIFICIAL INTELLIGENCE LAB
SHAY BUSHINSKY, SPRING 2025

LAB3 META HEURISTICS

LOCAL SEARCH SWARM INTELLIGENCE

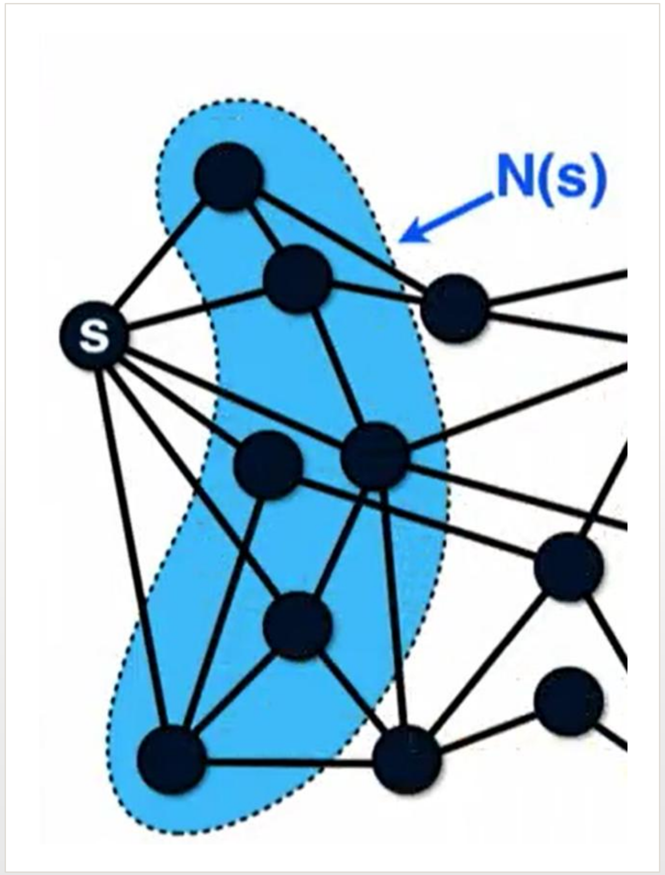
IN THIS LECTURE

- Neighbourhoods
- Multi Step Heuristics
- Meta Heuristics
- Iterated Local Search (Simulated Annealing + Tabu Search)
- ALNS

A LIST OF NP-C REFERENCED PROBLEMS

- Bin Packing
- 0-1 Knapsack
- MKP Multi Knapsack Problem
- Graph Coloring
- TSP
- TTP – Traveling Tournament Problem
- CVRP

DEFINITION: THE NEIGHBORHOOD OF A STATE



Neighborhood:
A set of solutions reachable from the current one via a small modification.

THE MULTI-KNAPSACK PROBLEM (MKP)

- Given multiple **knapsacks (suitcases)**, each with a weight limit.
  Assign items (each with a weight and value) to these knapsacks to **maximize total value** without exceeding any bag's capacity.
- Each item goes in **one suitcase only**
- Each suitcase has a **maximum capacity**
- The goal is to **maximize the total value**
- It is forbidden to **exceed any weight limit**

BIN PACKING VS. MKP

Feature	Bin Packing	Multiple Knapsack (MKP)
Number of bins/knapsacks	Unlimited (minimize used)	Fixed
Item characteristic	Size only	Size and value
Must-pack rule	Yes	No (you can skip items)
Objective	Minimize number of bins	Maximize total value packed

NEIGHBORHOOD EXAMPLES IN MKP

To explore neighboring solutions (i.e., small changes), you can:

- **Move Item**: Move an item from one knapsack to another (if capacity allows)
- **Swap Items**: Swap two items between different knapsacks
- **Drop & Add**: Remove an item and add a different one to the same or a different knapsack
- **Reassign Group**: Reassign a group of items collectively to redistribute load/value

FORMS USED ACROSS THIS LECTURE

i. **Metaheuristics**: Explore/improve solutions by moving within or across neighborhoods

ii. **Iterated Local Search (ILS)**: Local search on neighborhoods + perturbation to escape local optima

iii. **Simulated Annealing:** Probabilistically accepts worse neighbors to avoid getting stuck

iv. **Tabu Search:** Uses memory to avoid revisiting recent neighbors

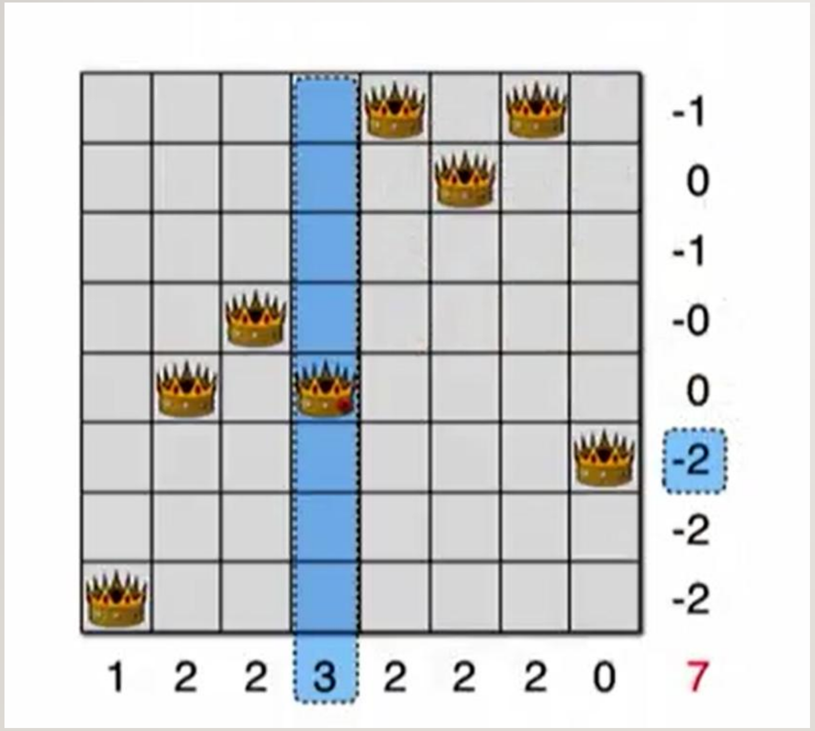
v. **ALNS (Adaptive Large Neighborhood Search):**

Dynamically selects from a pool of neighborhood operators (destroy/repair pairs)

MULTI-STAGE HEURISTICS

Purpose: avoid scanning the entire neighborhood

MULTI-STAGE HEURISTICS EXAMPLE 1



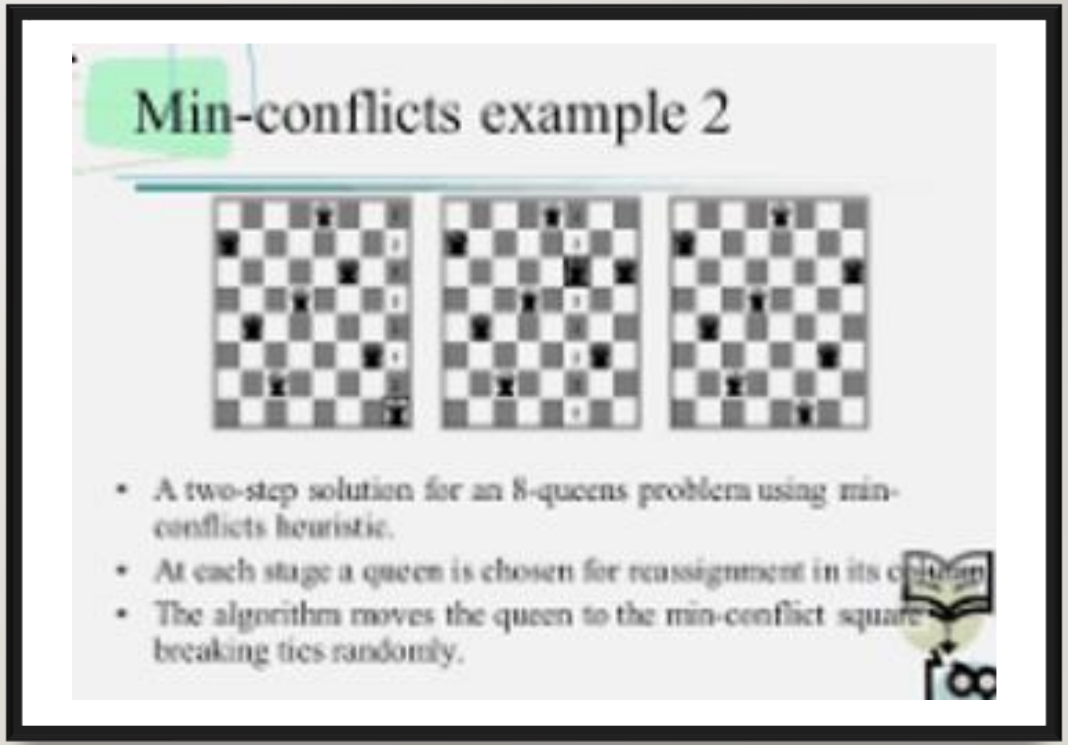
- **max/min. conflicts** N-queens:
- The Two Greedy Stages:

1. Select the <u>variable</u> with **most** violations
2. Select the <u>value</u> with **fewest** resulting violations

MULTI-STAGE HEURISTICS EXAMPLE 2

- **min. conflicts** N-queens:
- The Two Stages:

1. Select **a** variable <u>randomly</u>
2. Select value with fewest resulting violations



Min-conflicts example 2

- A two-step solution for an 8-queens problem using min-conflicts heuristic.
- At each stage a queen is chosen for reassignment in its column
- The algorithm moves the queen to the min-conflict square breaking ties randomly.

COMPARING N-QUEENS CANDIDATE NEIGHBORHOODS

- **The next legal move heuristic**: guaranties best move selection but has quadratic time complexity o(n^2)
- The Max-Min or Min-Conflict heuristics – approximate best move selection yet have linear time complexity o(n)
- Speed vs Perfect tradeoff

SUDOKU EXAMPLE

- **Stage 1: Select an empty cell**
- Choose an empty cell in the grid. Selection can be random or based on some heuristic (e.g., the cell with the fewest possible values).
- **Stage 2: Assign a value**
- Place a value in the selected cell that does not violate the Sudoku constraints (no repeating numbers in the same row, column, or sub-grid).

TSP EXAMPLE
USING GREEDY INSERTION

- **Stage 1: Select a starting city**
- Choose a starting city randomly or based on some heuristic (e.g., the city with the most connections).
- **Stage 2: Build the tour**
- Add the next city to the tour by selecting the city that, when inserted, results in the smallest increase in the total tour length. Continue this process until all cities are included in the tour.

META HEURISTICS

A HIGHER-LEVEL METHOD TO FIND OR GENERATE HEURISTICS FOR GOOD SOLUTIONS TO OPTIMIZATION PROBLEMS

WHAT IS META-HEURISTICS

- Higher-level frameworks for guiding lower-level heuristics (e.g., local search, crossover) across many problem instances.
- They adapt search operators or parameters to new optimization problems.
  - *Example*: A genetic algorithm that dynamically adjusts its mutation rate based on recent progress.

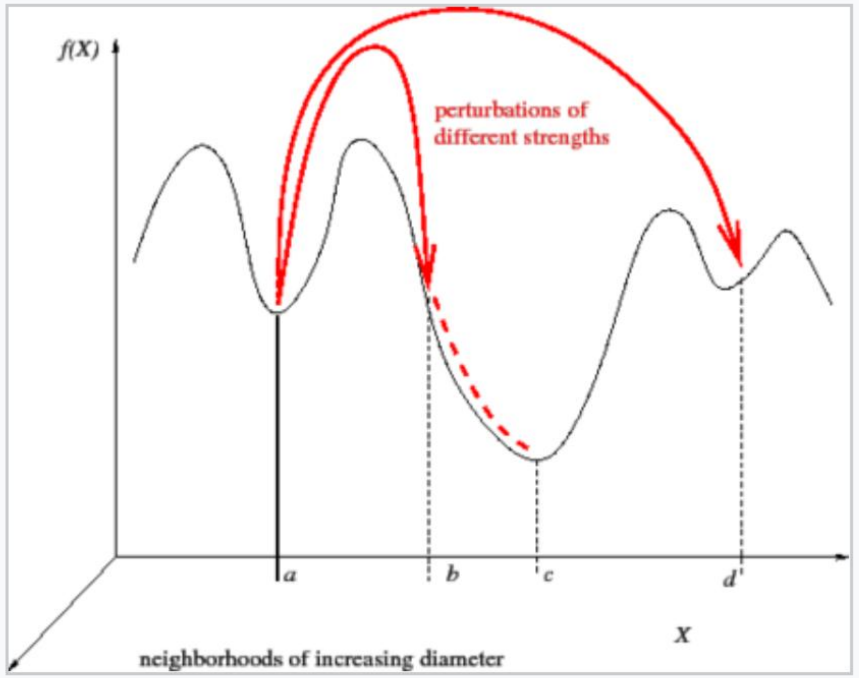
META-HEURISTICS OBJECTIVE

- Goal is to escape local optima
- Drive the search towards global optima
- No guarantee
- Memory / Learning (exploit non-local info)

ITERATED LOCAL SEARCH

META HEURISTIC ALGORITHMS

ILS - MOTIVATION


f(X)
perturbations of different strengths
a
b
c
d
X
neighborhoods of increasing diameter

Iterated local search *kicks* a solution out from a local optimum

ITERATED LOCAL SEARCH

- A modification of local search or hill_climbing methods for solving discrete optimization problems

- Fighting local optima - a (simple) modification consists of iterating calls to the local search routine
- Each time starting from a **different initial configuration**

ITERATED LOCAL SEARCH

- **Purpose:** Enhance solution quality for complex optimization by balancing exploration & exploitation
- **Core Mechanism:** Repeated local searches from diverse starting configurations
- **Error Strategy:** Begin with lenient acceptance, then tighten improvements as search progresses
- **Scalability:** Naturally parallelizable and integrates easily with hybrid meta-heuristics

THE ILS ABSTRACT INTERFACE

- F: the heuristic function f(s)
- N: the neighbourhood of state s
- L: the Legal state function
- S: select state function (from neighbourhood)

THE GENERAL META HEURISTIC ALGORITHM

```
Function IteratedLocalSearch(f, N, L ,S) {
    s := GenerateInitialSolution();
    s* := s;
    for k := 1 to MaxSearches do:
        s := LocalSearchStep(f,N,L,S,s);
        if (f(s) < f(s*) s* = s;
        s:= GenerateNewSolution(s);
    return s*;
}
```

- * Possibly change for MaxSearch to while termination condition not met

SIMULATED ANNEALING

META HEURISTIC EXAMPLE 1

THE METROPOLIS HEURISTIC

- Inspired by statistical physics
  - Accept a move if improves the objective function
  - If it doesn’t, accept it with some **probability**

- The probability depends on:
  - 1. How bad the move is
  - 2. How advanced the algorithm is in the search

PROBABILITY FUNCTION DESIGN

- Assume we are at state s and consider moving to state s’
- Assume Δf = f(s)-f(s’), heuristic degradation
- Then the probability of selecting the move:

- P(s,s’,T) = 1.0 if Δf > 0 i.e., s’ is an improvement
- $e^{\frac{-\Delta f}{T}}$ otherwise i.e. s’ is a degradation
- Assume searching for minima

ACCEPTANCE PROBABILITY ANALYSIS

- If the Δf is very big, then the denominator will result in a very small probability

- If T is very big, then the probability tends to 1.0 - accepting bad moves
- If T is very small, then the probability tends to 0.0 – reluctant to accept bad moves

THE ANNEALING SCHEDULE

- Start with a high T (temperature), effectively resulting in a **random walk**
- **Cooling**: As search progresses, decrease the T hence search becomes **more greedy**
- At T= 0: search becomes **hill climbing**

THE METROPOLIS STEP

- Function Metropolis-Step(t, N, s)
  - Select a **neighbor** n with probability 1/N
  - (N = number of neighbors)
    - If (f(n) > f(s)) return n
    - else with probability exp(-(f(n)-f(s)) return n
    - else return s

- * assuming increase in f is an improvement

SIMULATED ANNEALING AS ILS

```
• Function **SimulatedAnnealing**(f, N) {
    • s:= GenerateInitialSolution();
    • t1:= InitTemparature(s);
    • s* := s';
    • For k := 1 to maxSearches do
        • s := localsearchStep(f,N,L-All,MetropolisStep(tk,s));
        • if f(s) < f(s*) then s* = s;
        • tk+1 := UpdateTemperature(s, tk);
  return s*;
  }
```

TEMPERATURE COOLING

- Customary:
  - Initialize: T s.t probability will be 0.5
  - Update: $t_{k+1} = \alpha * t_k$
  - where $0.0 < \alpha <= 1.0$

PRACTICAL OPTIONS UNDER ILS

1. Start at multiple points – different initialization
2. Restart
3. Reheat - if cooled too fast
   a. Increase initial temperature
   b. Use a geometric annealing schedule instead of linear decrease

TABU SEARCH

META HEURISTIC - EXAMPLE 2

TABU SEARCH

- As in hill climbing, work with **one state**
- Iteratively, test every neighbor, and move to the best one, **even if that means going downhill**
- However, do not move downhill to states that have been visited recently (or frequently) - marked as "Tabu states" or "Tabu nodes"

- Variant: do not apply the last n operators – keep the operations that move from one state to another

TABU SEARCH

- Attempts to jump out of local optima and reach a global optima
- Issue: expensive to maintain all the visited nodes
- Solution: **“short-term” memory** – “Tabu-list”
- Policy Parameters:
  - How long to maintain the tabu states?
  - How to manage the tabu states memory?

A HOSTILE NEIGHBORHOOD

- State x determines a **neighborhood** N(x)
- N(x) is dynamic – updated during the search
- N(x) is maintained implicitly via a **tabu-list**: all neighbors recently visited become tabu
- They are remembered for **n moves** after, until dropped from the tabu list
- **n** is a parameter labeled the "**tabu tenure**"

THE TENURE (n) PARAMETER

- Can be very expensive to keep all nodes in memory
- Can keep a short list of recent nodes as the more tenure tabu nodes may not be in the neighbourhood anymore (no risk of oscillation)

- Can adjust n dynamically:
  - **Decrease** when objective function is degrading
  - **Increase** when is objective function is improving

TABU LIST "ABSTRACT STATES"

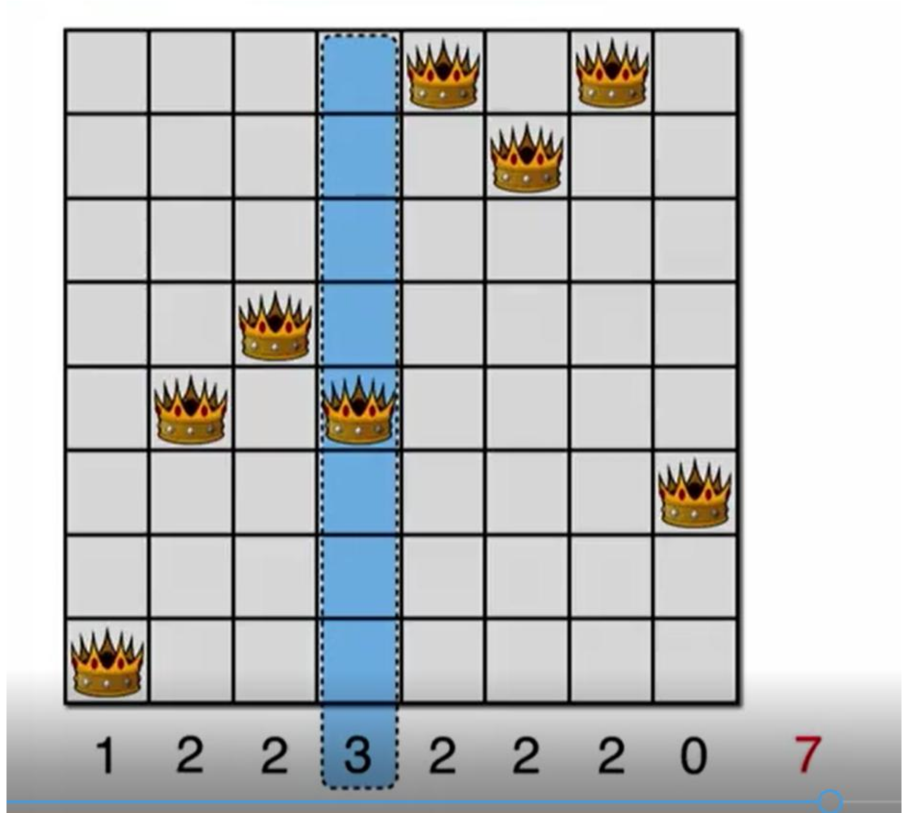
- Don't store the states (may be costly) and time inefficient

- Store Operators: for example, if you travelled from a to b don't store the nodes a,b but rather store the transition a-b to avoid repeating this route

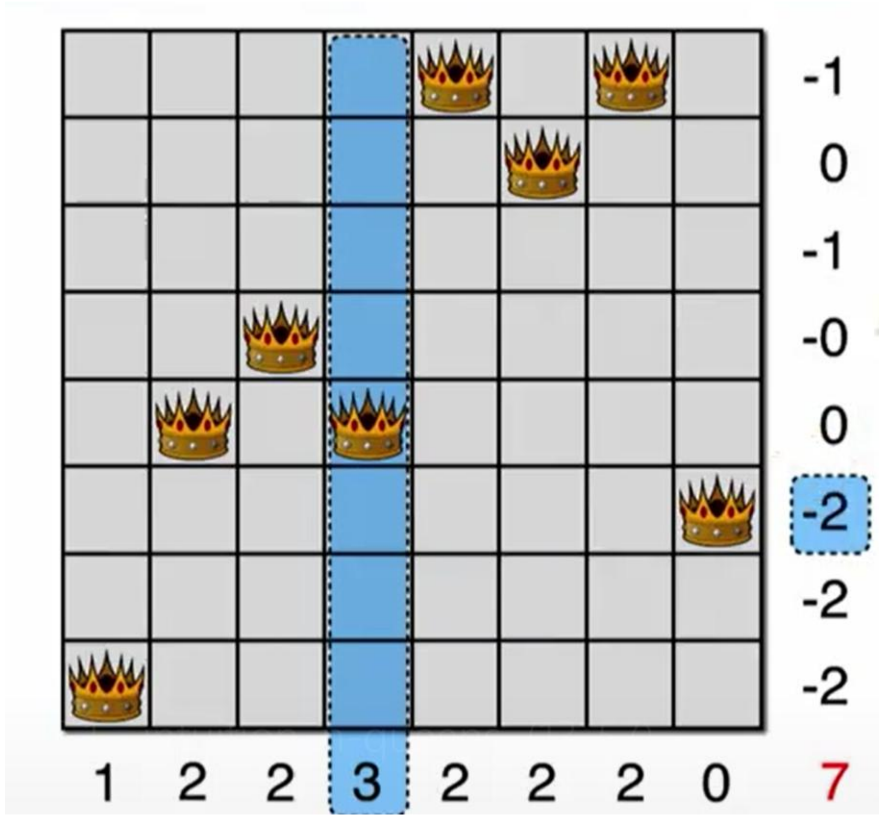
THE N-QUEENS TS EXAMPLE

- Use min-conflicts algorithm:
  - Select a queen
  - Find all possible alternative squares for her
  - Select the min conflicts square that is **not in the tabu** list (meaning the specific queen that was not on that square for a while…)
  - Mark that square as tabu for a number of iterations

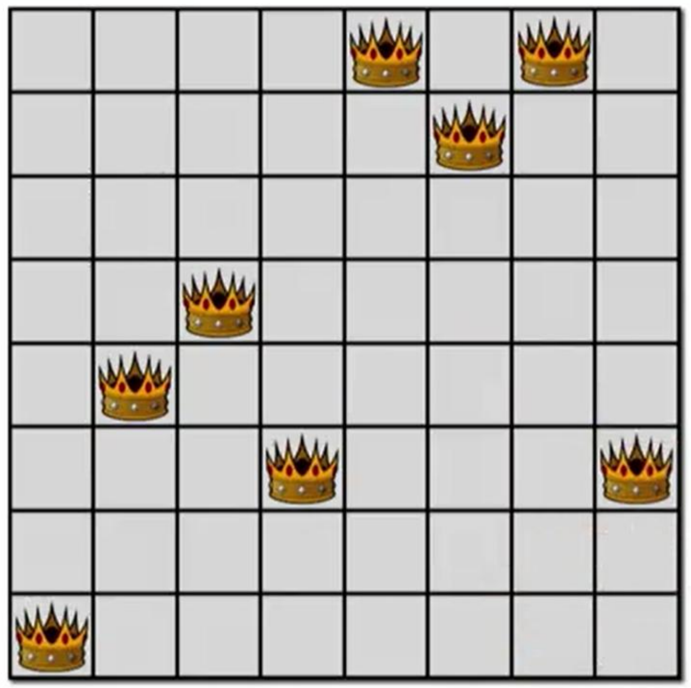
INTUITION FOR N-QUEENS



EVALUATION MOVES



IMPROVEMENT STEP



SUMMARY

Tabu list prevents "undoing" recent moves too soon.

Min-conflicts finds locally best options.

- This combo helps escape local minima traps while still optimizing effectively.

THE ILS TABU SEARCH INTERFACE

- N – The Neighborhood of States
- s - The current state
- t – The tabu list – states seen so far
- S – The select state function out of N
- L – The function returning Legal states
- satisfiable(s) – s doesn’t violate constraints

TABU SEARCH AS ILS

```
Function LocalSearch(f,N,L,S,s1)
    S* := s1
    T := <s1>
    For k := 1 to MaxTrials do
        If satisfiable(s) & f(Sk) < f(S*) then
            S* := Sk
        Sk+1 := S(L(N(Sk), t), t);  // select
        T := t :: Sk+1;  // concatenate
    return S*
```

N: the full neighborhood of the current state
S: a selection function from N
L: legal states filter
t: a tabu list tracking recent moves
satisfiable(s): used to check if a solution meets constraints

ABSTRACT TABU-SEARCH

- Select the best configuration that is not tabu, i.e., has not been visited before

- Function TabuSearch(f,N,s)
    - return LocalSearch(f,N,L-NotTabu, S-Best)
    - Where:
        - Function L-NotTabu(N,t)
            - return {n in N s.t. not n in t}

- Explore the neighborhood N of the current state s
- Filter out **tabu** states using L-NotTabu
- Use a selection rule S-Best (e.g., pick the best available state)
- Continue **iteratively** (looping through moves) until a stopping condition is met

TABU SEARCH VARIANTS

TECHNIQUES WITH TABU SEARCH

1. TABU PROBLEM FEATURES

- Rather than states, populate the Tabu list with attributes or problem features
- E.g., **expensive arcs** in the TSP problem
- Each tabu feature may deny several solutions (e.g., all tours containing such arcs)

2. ASPIRATION CRITERIA

- Sometimes the Tabu is **too strong :**
  - prevents the algorithm from re-exploring a solution that is by far better than the one in hand

- Solution: Use an "**aspiration criteria**":
  - **overriding** a tabu-feature if the corresponding tabu state improves the current best solution
  - e.g., ("simulated annealing")

"If a tabu move leads to a clearly better solution, go for it — break the tabu!"

EFFECIENT NODE DATA STRUCTURE

- Use a node hash as a tabu-list
- In the hash keep per tabu node:
  - the f value
  - the Δ of the f value
- Usage: if the current f value is better than the f value in the previous iteration – it is not tabu

TABU SEARCH
EXPLOITAION / EXPLORATION

- **Phenomena**: long walks that don't improve anything
- <u>Two solutions</u>:
  - 1. **Intensification**: keep high-quality solutions aside and after a fixed number of moves return to another good one and proceed
  - 2. **Diversification**: diversify the current state – Introduce randomness to the current state to escape local traps.
    • Example (N-Queens): randomly move some queens regardless of conflicts.
    • Example (Knapsack): flip the inclusion/exclusion of a few items.

TABU SEARCH EXPLOITAION / EXPLORATION (CONT.)

- 3. **Strategic Oscillation**

  Idea: Intentionally move between feasible and infeasible regions during the search.

  Purpose: Explore more of the solution space, escape local optima, and discover high-quality solutions.

  How:

    Define a control strategy (fixed or adaptive) to regulate time spent in each region.

    Occasionally accept infeasible solutions if they lead to promising directions.

    - Adjust the oscillation rate based on progress or stagnation.

  - Change the percentage of time spent in the feasible and infeasible regions

STRATEGIC OSCILLATION KNAPSACK EXAMPLE

The algorithm **occasionally allows solutions over 15 kg** (infeasible) — say, up to 17 kg temporarily.

Example Behavior:

1. Current solution: 14.5 kg, value = 120
2. Swap adds a great item → weight = 16.8 kg, value = 150
   → Accept **infeasible** move
3. A few steps later, drops another item → weight = 14.2 kg, value = 135
   → Returns to **feasible** region, now with a better solution path

ACO

DISCRETE ANT COLONY OPTIMIZATION

ALTERNATIVE METHODS TO CREATE ORDERED BEHAVIOR

1. Controlled top-down

2. **Emerging** via micro-interactions

לֵךְ-אֶל-נְמָלָה עָצֵל; רְאֵה דְרָכֶיהָ וַחֲכָם.

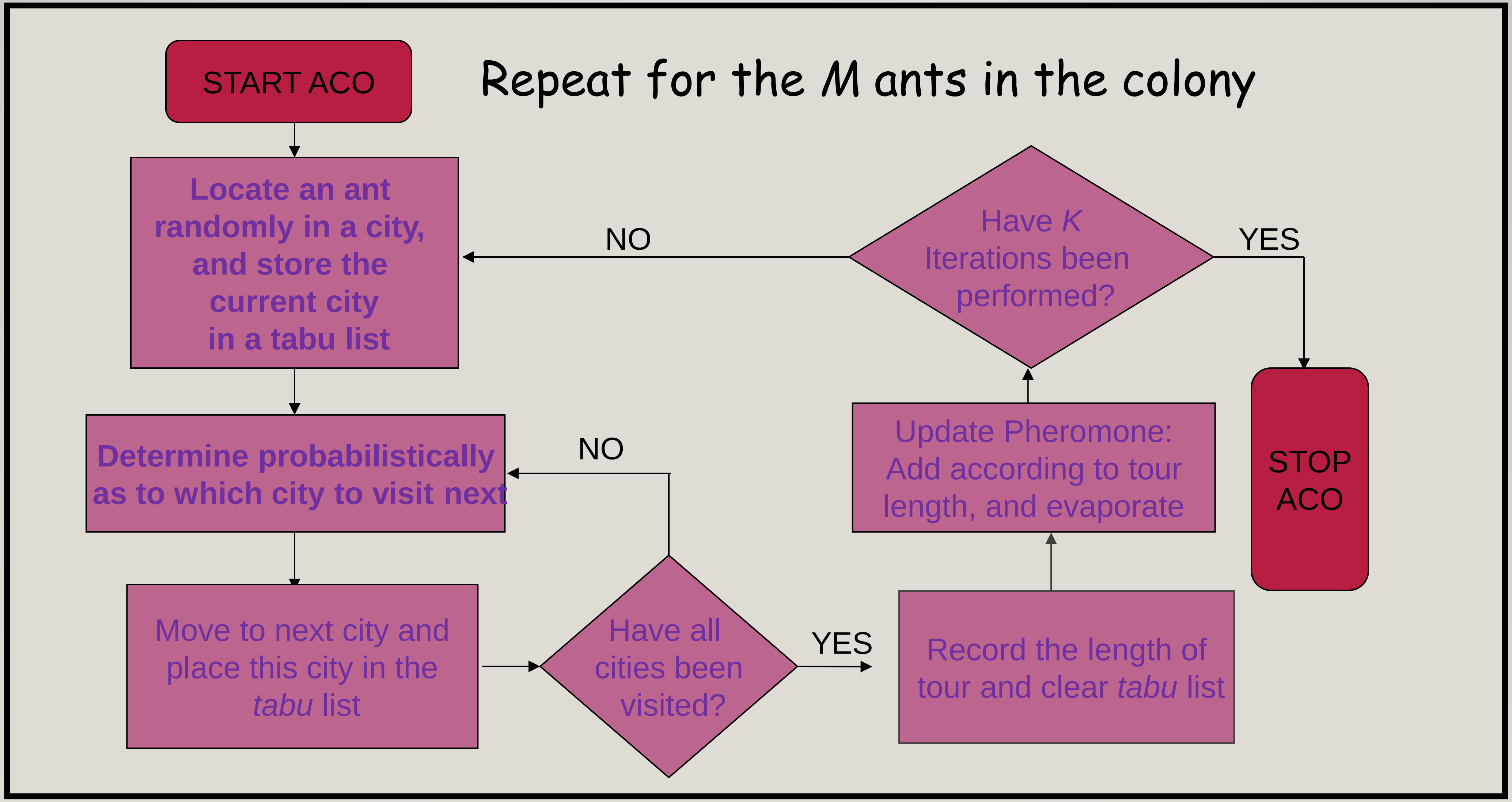
אֲשֶׁר אֵין-לָהּ קָצִין-- שֹׁטֵר וּמֹשֵׁל.

ACO CHARACTERISTICS

- Exploit a **positive feedback** mechanism (RL)
- Exploit a **global data structure** that changes dynamically as each ant transverses the route

- Has an element of **distributed computation** to it involving the population of ants
- Involves **probabilistic transitions** among states

ACO FOR TSP


START ACO
Repeat for the *M* ants in the colony
Locate an ant randomly in a city, and store the current city in a tabu list
Determine probabilistically as to which city to visit next
Move to next city and place this city in the *tabu* list
Have all cities been visited?
NO
YES
Record the length of tour and clear *tabu* list
Update Pheromone: Add according to tour length, and evaporate
Have *K* Iterations been performed?
NO
YES
STOP ACO

THE ACO PARAMETERS

- **Trail intensity** is given by value of τ_{ij} which indicates the intensity of the pheromone on the trail segment, (ij)
- **Trail visibility** is $\eta_{ij} = 1/d_{ij}$: a heuristic function of the desirability of adding edge (d_{ij} is distance from i to j)
- The **importance** of the intensity in the probabilistic transition is α
- The **importance** of the visibility of the trail segment is β

THE PHEROMONE PARAMETERS

- The trail persistence or **evaporation rate** is given as ρ

- **Q** is a constant and the amount of pheromone laid on a trail segment employed by an ant

- **Initial pheromone** is a small amount on all edges

PHEROMONE EVAPORATION: BALANCING EXPLORATION AND EXPLOITATION

- It's a parameter $\rho \in (0,1)$ that controls **how quickly pheromone trails decay** over time.
- In the pheromone update rule: $\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}$
- The term $(1-\rho)\cdot\tau ij$ **fades old pheromones**
- A higher ρ (closer to 1) → **faster evaporation** → more exploration
- A lower ρ → **slower evaporation** → more exploitation (ants strongly follow past paths)

THE PROBABILISTIC CITY SELECTION

- Each ant iteratively builds a tour by selecting the next city to visit, using this probability
- Determined by τ_{ij} - pheromone content in an edge (*ij*), η_{ij} - trail visibility, and α,β: their importance

Probability for ant *k* to select edge (*ij*)

$J_k(i)$ is the set of cities not visited by ant *k* before step *i*

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in J_k(i)} [\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} & \text{if } j \in J_k(i) \\ 0 & \text{if } j \notin J_k(i) \end{cases}$$

THE PHEROMONE UPDATING FOR EACH ANT

- $T_k(t)$ is the tour found by k-th ant, and its length is L_k. Pheromone increment to each edge of this tour is:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \dfrac{Q}{L_k} & where\ edge\ (i, j) \in T_k(t) \\ 0 & if\ edge\ (i, j) \notin T_k(t) \end{cases}$$

- Update pheromone using this increment, but also evaporate some pheromone: $\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t)$

VARIATIONS AND CHALLENGES

- Balancing Exploration and Exploitation
  - **Exploration**: Trying new directions, even when the heuristics and current pheromone level indicate they aren't good

➔Avoid being stuck in local optima

  - **Exploitation**: "Believing" the heuristics and pheromone trail and following it with high probability

➔ Find a good solution quickly

CONTROL VARIATIONS

- When to update?
  - All ants?
  - Best ant(s)?
    - In this colony?
    - Across all past colonies?
  - Remove pheromones from trails used by worst ants?
- How many ants?

Note: Some variations introduce global, centralized control, unlike "real" ants

LOCAL OPTIMA IN ACO

- Like any search process, ACO can also get stuck in local optima
- Options when change has stopped:
  - **Restart** – re-initialize all the pheromones
  - Increase exploration, decrease exploitation
  - Change the update scheme: reduce effect of best ants and let "not-so-best" ants change pheromones
  - Find parts of the solution that seem to be improvable, and:
    - Change pheromones for those parts, or …
    - Temporarily change value function to focus change directions

TECHNIQUES FOR INCREASING EXPLORATION IN ACO

1. **Pheromone Evaporation Rate**:
   - Increase evaporation rate to reduce influence of old paths.
2. **Pheromone Initialization**:
   - Start with low initial pheromone levels.
3. **Exploration-Exploitation Balance**:
   - Adjust parameters to favor exploration

   (e.g., lower alpha, higher beta).

TECHNIQUES FOR INCREASING EXPLORATION IN ACO

5. **Stochastic Decision Rules**:
   - Introduce randomness in path selection.
6. **Multiple Colonies**:
   - Use multiple independent or periodically communicating colonies.

- **Random Restarts**:
  - Periodically reset pheromone trails.

5. **Diversification Strategies**:
   - Implement pheromone smoothing to level path differences.

TECHNIQUES FOR INCREASING EXPLORATION IN ACO

9. **Ant Memory**:
   - Encourage ants to explore new paths by using memory.
10. **Dynamic Pheromone Updates**:
    - Change update rules based on search progress.
11. **Hybrid Approaches**:
    - Combine with other optimization techniques.

WHY “ACO AS ILS”?

- Though ACO is population-based, this version acts like **Iterated Local Search**:
    - Starts with random samples
    - Repeatedly constructs solutions with guidance (like a local neighborhood)
    - Uses the best-so-far idea and pheromone learning for intensification

EXAMPLE ENERGY FUNCTION OPTIMIZATION

Optimizing a **continuous function** (e.g., minimizing a cost or energy function) over some domain.

ACO is used as a population-based method to search this domain.

Inputs:

nPop: total population size

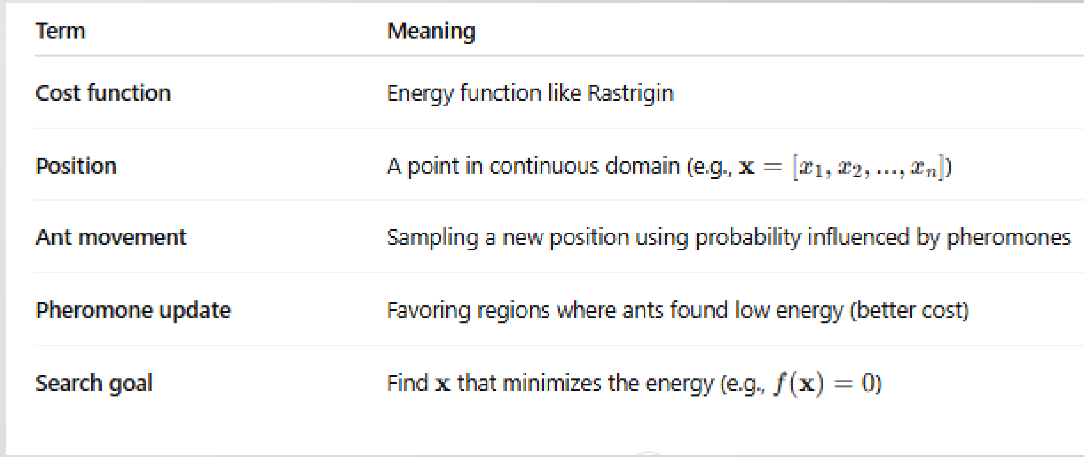
nVar: number of variables (problem dimensionality)

domainBounds: valid ranges for variables

costFunc: the objective function to minimize

hyperParams: includes number of ants (nAnts), ρ (pheromone influence), and Q (pheromone deposit factor)

- nEpochs: number of iterations the algorithm will run



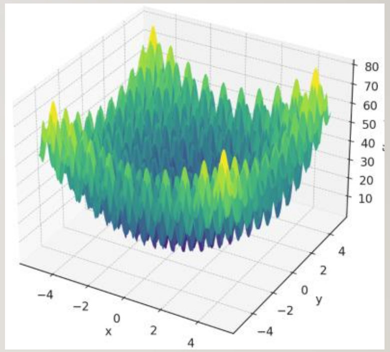
Term	Meaning
Cost function	Energy function like Rastrigin
Position	A point in continuous domain (e.g., $\mathbf{x} = [x_1, x_2, \ldots, x_n]$)
Ant movement	Sampling a new position using probability influenced by pheromones
Pheromone update	Favoring regions where ants found low energy (better cost)
Search goal	Find $\mathbf{x}$ that minimizes the energy (e.g., $f(\mathbf{x}) = 0$)

ACO AS ILS
ANT COLONY OPTIMIZATION OF A CONTINUOUS DOMAIN FUNCTION

```
Input: nPop, nVar, domainBounds, costFunc, hyperParams ← {nAnts, $\rho$, Q}, nEpochs
Output: bestSol, bestCostFuncValue
Procedure Main():
    Generate initial population of size nAnts
    Initialize the pheromone trail and parameters
    Evaluate the initial population according to the cost function, costFunc
    Find the best solution of the population
    While (i < nEpochs)
        Construct new ant population solution
        Update the pheromone
        Determine best global ant solution
    End While
    Return global best solution, bestSol
```

ANT POSITIONS AND THE OPTIMIZATION GOAL (RASTRIGIN FUNCTION)

- Each **position** is a point in x∈Rn
  → e.g., for n=2, an ant is at (x1,x2)
- The goal:
  Minimize the objective function over the space
  Example: **Rastrigin Function**



$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

- Global minimum at x=0 where f(x)=0

HOW ANTS MOVE IN CONTINUOUS ACO

- Ants **sample new positions** based on pheromone-influenced probabilities
- Often done using **Gaussian sampling** near good solutions

Example:

If a good solution is found at $(2.3, -1.5)$, ants in the next epoch may be sampled from:

$$x_1 \sim \mathcal{N}(2.3, \sigma^2), \quad x_2 \sim \mathcal{N}(-1.5, \sigma^2)$$

- This guides the search toward promising areas while still exploring nearby.

COMPARING ILS VS. SIMPLE ACO

ACO FOR N-EPOCHS IN ILS

- Periodic restarts
- Strategic adjustments
- Promotes diversity
- Enhances robust exploration

SINGLE ACO FOR N*K ITERATIONS

- Continuous pheromone updates
- Risk of premature convergence
- Lacks strategic interventions
- Limited ability to escape local optima

ALNS

ADAPTIVE LARGE NEIGHBORHOOD SEARCH

ALNS META HEURISTIC

- ALNS stands for Adaptive Large Neighborhood Search
- It is particularly effective for solving problems such as vehicle routing, scheduling, and various types of packing and routing problems

ALNS APPLICATIONS

1. **Crew Scheduling**: In airline and railway industries
2. **Job Shop Scheduling**: In manufacturing, scheduling a set of jobs on a set of machines to minimize the total time needed to complete all jobs, while respecting precedence constraints and machine availability
3. **Nurse Scheduling**: In healthcare, assigning nurses to shifts
4. **Container Terminal Scheduling**: loading, unloading, transportation, maximize throughput

NEIGHBORHOOD EXPLORATION

- The ALNS method is based on iteratively improving an initial solution by exploring its neighborhood (**Iterative Repair Method**)
- Deciding to accept or reject the new solution based on a predefined acceptance criterion
- The process is repeated until a stopping condition is met, such as:
  1. Reaching a time limit or a
  2. Certain number of iterations

THE ALNS FRAMEWORK

1. What is the Solution Representation?
2. What are the Neighborhood Structures?
3. Adaptive Search Strategy
4. Acceptance Criterion
5. Stopping Condition

STEP 1: SOLUTION REPRESENTATION

The problem is represented as a solution with a specific structure, which should allow the ALNS to manipulate and evaluate it

STEP 2: NEIGHBORHOOD STRUCTURES

These are a **set of operators** that define how a solution can be modified to generate a new solution

These operators can involve adding, removing, or swapping elements within the solution

STEP 3: ADAPTIVE SEARCH STRATEGY

This strategy is used to **select the neighborhood operator** to be applied in each iteration

The selection is based **on a probability distribution** that is updated during the search process:

Giving **more weight to operators that have been more successful** in improving the solution quality

STEP 4: ACCEPTANCE CRITERION

This criterion is used to decide whether the new solution generated should replace the current solution or not

A commonly used acceptance criterion is the **Simulated Annealing (SA) acceptance rule**, which accepts worse solutions with a probability that decreases over time

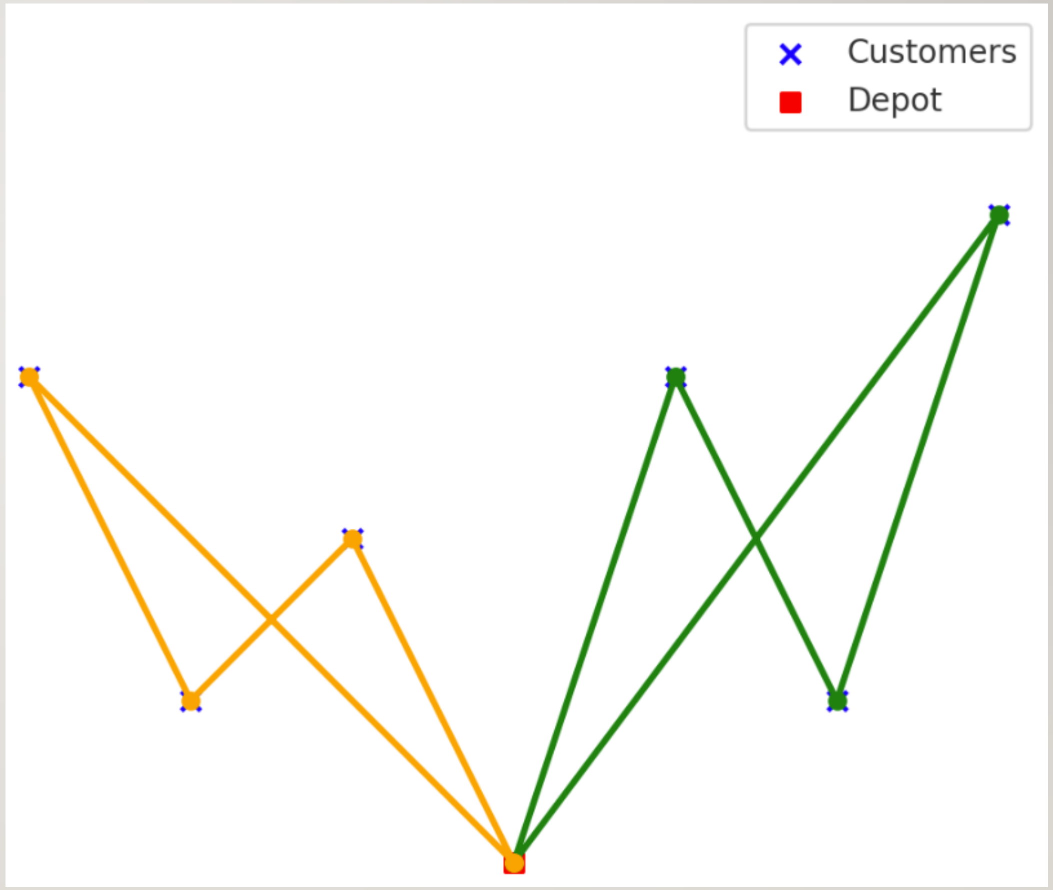
STEP 5: STOPPING CONDITION

The search process continues until a predefined stopping condition is met, such as a maximum number of iterations or a time limit or convergence criteria

ALNS CASE STUDY: VRP

THE VEHICLE ROUTING PROBLEM (VRP)

- **Goal:** Minimize total distance (or cost) to deliver goods to customers using a fleet of vehicles.
- **Given:**
  - A depot
  - A set of customers with known demands
  - A fleet of vehicles with capacity limits
- **Constraints:**
  - Each customer is visited exactly once
  - Total demand on any route ≤ vehicle capacity
  - All routes start and end at the depot
- **Objective:**
  - Minimize total travel distance, time, or cost



1.VRP SOLUTION REPRESENTATION

- A solution is represented as a sequence of customers visited by each vehicle

- For example:
  - for three customers (A, B, and C) and two vehicles (V1 and V2):
  - a possible solution could be: V1: [A, B], V2: [C]

2. NEIGHBORHOOD STRUCTURES

- We define **these operators** for this problem:
- Given: V1: [A, B], V2: [C]

1. **Relocate**: Move a customer from one route to another, e.g., V1: [A], V2: [C, B]
2. **Swap**: Swap two customers between two routes, e.g., V1: [A, C], V2: [B]
3. **Cross-Exchange**: Exchange **subsequences** of customers between two routes (while respecting capacity constraints)
4. **Two-Opt**: Reverse the order of customers within a route, e.g., V1: [B, A], V2: [C]

3. VRP ADAPTIVE SEARCH STRATEGY

In the beginning, all neighborhood operators have the same probability of being selected

As the algorithm progresses, the probabilities are updated based on the success of each operator in improving the solution

4. VRP ACCEPTANCE CRITERION

Example: use the Simulated Annealing (SA) acceptance rule (“Metropolis step”)

Initially, worse solutions have a higher probability of being accepted, but this probability decreases over time, allowing the algorithm to converge to an optimal solution

5. STOPPING CONDITION

- Stop the algorithm after a certain number of iterations or a time limit or a convergence criteria

SUMMARY: ALNS FOR STRUCTURED NEIGHBORHOODS

Use ALNS when:

- **For simple, fast, and effective adaptive strategy** for large combinatorial problems.
- When the **set of operators is fixed,** and the **reward signal is sparse**
  - (e.g., only solution improvement counts).
- For **low overhead** and **interpretable adaptation** (e.g., scoring/ranking operators).
- For solving problems like **Vehicle Routing**, **Scheduling**, or **Bin Packing**, where local search operators (destroy/repair) are well defined.

THE END