ZENITH STUDY HUB / AIFR

UNIT II | REVIEW v3

Search and Metaheuristic Optimization

PCC301-ROA / SPPU TE Robotics & Automation / 2024 Pattern

What is included

Pages 2-28	Original 27-page reviewed v2 guide, preserved with one visual label repair.
Pages 29-34	Six-page independent solved-question supplement, unchanged.
Status	Integrated review candidate. Not a final academic approval or website release.

Revision log and safeguards

- Original text, mathematics and vector diagrams otherwise retained.
- Original page 12 Figure 5 overlapping label repaired in this copy.
- Question-bank solutions added for Q2, Q4, Q6, Q8, Q10 and Q12.
- Nonzero goal heuristic in Q10 and SA temperature notation documented.
- Hill Climbing retained as supplemental question-bank content.
- No existing Drive source file was overwritten.

Study edition, 22 November 2026 | Source lineage: Drive reviewed v2 + independent QA supplement

ZENITH STUDY HUB

Artificial Intelligence for Robotics

UNIT II | SEARCH & OPTIMIZATION

BFS, DFS, Best-First, A*, Genetic Algorithm, Simulated Annealing, Tabu Search and Ant Colony Optimization

Syllabus-first learning guide with solved numericals

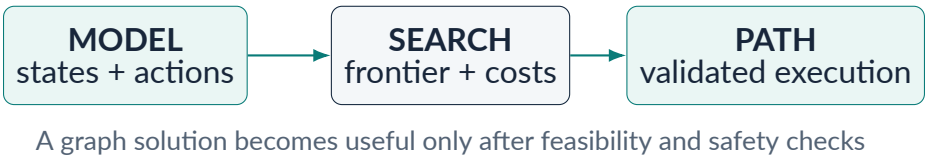


Figure 1. Original robotic problem-solving pipeline; graph planning is separate from real-time motor control.

Exact official Unit II syllabus coverage
State-space search: uninformed BFS and DFS; informed Best-First Search and A*; heuristic functions; problem solving in robotics.
Metaheuristics: Genetic Algorithm, Simulated Annealing, Tabu Search and Ant Colony Optimization, each with numerical examples.
Applications: path planning, routing and scheduling.

Course: PCC301-ROA | SPPU TE Robotics & Automation | 2024 Pattern

Prepared for learning review: Om Deshmukh | 21 November 2026

Learning route and exam strategy

Syllabus topic	Learning task and evidence
Problem formulation	Define states, edges, costs, start, goal, constraints and neighbour order.
BFS and DFS	Trace queue/stack updates and reconstruct paths on the same graph.
Best-First and A*	Calculate g, h, f, expand nodes and explain optimality assumptions.
Heuristics	Check admissibility, consistency and units on a weighted graph.
Genetic Algorithm	Roulette probabilities, crossover, mutation and one generation.
Simulated Annealing	Calculate worse-move acceptance and a cooling step.
Tabu Search	Use tabu restrictions and aspiration to choose an admissible move.
Ant Colony Optimization	Calculate transition probability and pheromone update.
Robotics applications	Compare methods for motion planning, routing and job scheduling.

Critical modelling rule. Numerical examples state their graph, move order and algorithm policy. A different neighbour order may change DFS's exact expansion trace without making the algorithm wrong. A heuristic's numerical values are illustrative unless an admissibility bound is justified.

Study in three passes
Understand: read the simple-language definitions and diagrams.
Calculate: copy the tables and reproduce each queue, priority, probability and update.
Apply: identify how your chosen plan becomes a physically valid robot trajectory and how a safety system handles unexpected hazards.

1 State-Space Search and Problem Formulation

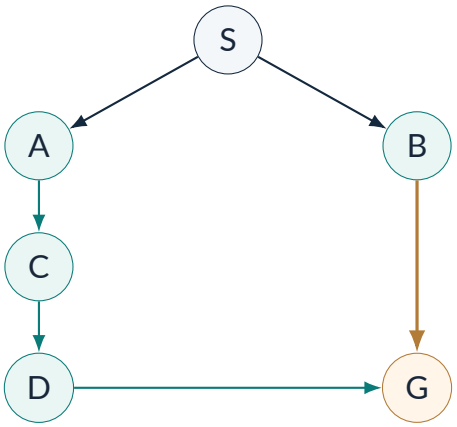
In simple language
A search algorithm explores possible future states to find an action sequence reaching a goal. It does not automatically know what states are safe, what a step costs, or whether a robot can execute the resulting motion. The engineer supplies that model.

1.1 Five ingredients of a robotic search problem

1. **State** s: the variables needed to predict what actions can do; for grid navigation, often a cell (x, y), sometimes with heading θ.
2. **Start** s_0: current estimated state or a named depot.
3. **Actions** $A(s)$: permitted moves such as north, south, east and west, excluding walls.
4. **Transition** $T(s, a)$: the next modelled state; physical uncertainty may require probabilistic modelling.
5. **Goal and cost**: a condition $G(s)$ plus edge cost $c(s, a, s')$ for length, time, energy or risk.

1.2 Search tree versus graph: why repeated states matter

A search *tree* records action sequences; two different sequences may reach the same state. A search *graph* represents distinct states and their links. In cyclic environments, searching without a visited/reached policy may revisit the same state indefinitely. A search node typically stores state, parent node, action, path cost g and depth.



All arrows cost one step; successor names are processed alphabetically

Figure 2. Original unweighted example graph used to compare BFS and DFS. The shortest start-to-goal path is S–B–G (two edges).

1.3 What a frontier stores

The **frontier** contains generated but not yet expanded candidates. An **expanded** node has had its successors examined. A **reached** set records states previously discovered; whether duplicate costlier paths must be updated depends on the algorithm. A parent pointer allows reconstruction of the final path after the goal is reached.

Check your understanding
A planner returns a five-cell route. Does that prove the robot can drive it? No: the route may violate wheel turning radius, speed, stopping-distance or dynamic-obstacle constraints. Geometric path planning and motion control are related but distinct engineering problems.

2 Breadth-First Search (BFS)

In simple language
BFS explores all nodes at depth 0, then depth 1, then depth 2, and so on. It uses a FIFO queue. With uniform positive step cost, the first goal found at the shallowest depth also has minimum path cost.

2.1 The queue principle



Figure 3. FIFO processing expands the oldest waiting state first. Queue order must be recorded in worked traces.

2.2 Step-by-step BFS trace on Figure 2

Use graph search: mark a state as discovered when enqueued, and test the goal when dequeued. Process successors alphabetically.

Step	Expanded	Queue after adding unseen successors (front first)
0	None	[S]
1	S	[A,B]
2	A	[B,C]
3	B	[C,G]
4	C	[G,D]
5	G	Goal dequeued: stop; parent chain is S–B–G.

WORKED EXAMPLE: Why BFS returns two steps, not four

After expanding S, B is only one action from the start and generates G. Although A has a deeper continuation A–C–D–G, BFS adds candidates in increasing depth order. It therefore dequeues G at depth 2 before D at depth 3. **Result:** S–B–G, total cost $2 \times 1 = 2$.

2.3 Pseudocode and implementation caution

Listing 1: BFS graph-search pseudocode: goal test on dequeue

```
queue = FIFO([start]); reached = {start}; parent[start]=None
while queue not empty:
    s = queue.pop_front()
    if goal(s): return path_from_parents(parent,s)
    for nxt in sorted(allowed_successors(s)):
        if nxt not in reached:
            reached.add(nxt); parent[nxt]=s
            queue.push_back(nxt)
return failure
```

BFS is complete for finite branching and reachable finite-depth goals, provided its implementation handles repeated states. It optimizes *number of edges*, not general weighted path cost. A two-edge route of costs $(7, 8)$ can be worse than a three-edge route of costs $(2, 2, 2)$.

3 Depth-First Search (DFS)

In simple language
DFS follows a path deeply before trying other alternatives. It uses a LIFO stack (or recursion). It can save frontier memory in tree-search settings but can miss a short goal while exploring a long branch, and it is not optimal.

3.1 Why DFS can choose a longer route

For the same graph in Figure 2, push successors in *reverse* alphabetical order so the alphabetically first successor is popped next. Use graph search with a reached set.

Step	Popped	Stack after push (top at left)
0	None	[S]
1	S	[A,B]
2	A	[C,B]
3	C	[D,B]
4	D	[G,B]
5	G	Goal popped: S–A–C–D–G.

WORKED EXAMPLE: DFS length comparison

DFS returns S–A–C–D–G, four steps, whereas BFS returns S–B–G, two steps. Both reach the goal, but the depth-first answer is not the shortest in this graph. Reversing successor order can change the exact DFS answer.

3.2 Complexity and memory

For illustrative *tree search* with branching factor b, maximum depth m and shallowest solution depth d, common upper bounds are BFS time/space $O(b^{d+1})$ (or $O(b^d)$ under alternate depth-counting conventions), DFS time $O(b^m)$ and DFS frontier space $O(bm)$. **Assumptions matter:** graph-search reached sets store additional memory; infinite-depth DFS may not find a goal on a different branch.

Exam trap. DFS is neither generally optimal nor complete in infinite/cyclic state spaces without appropriate safeguards. BFS's cheapest-solution property requires equal step costs; for nonnegative weighted edges, compare uniform-cost search or A*.

4 Informed Search: Best-First and A*

In simple language
Informed search uses a heuristic $h(n)$, an estimate of the remaining cost to a goal. Greedy Best-First chooses the node estimated closest to the goal. A* also accounts for the known path cost already spent, usually producing

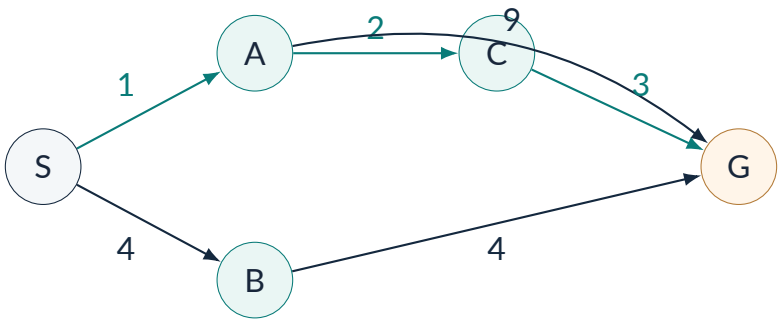
more reliable least-cost routes when its assumptions hold.

4.1 The three quantities you must distinguish

$$g(n) = \text{cost accumulated from start to } n,$$
$$h(n) = \text{estimated remaining cost to goal},$$
$$\boxed{f(n)} = g(n) + h(n).$$

Greedy Best-First Search: expand the frontier node with the lowest $h(n)$; it may find a solution quickly but can be suboptimal. **A*:** expand the frontier node with the lowest $f(n) = g(n) + h(n)$. **Uniform-cost search:** a useful comparison, expand lowest $g(n)$; it does not use a heuristic. The word "best" in Best-First refers to the chosen *evaluation function*, not guaranteed optimality.

4.2 Worked weighted graph (all edge costs positive)



Heuristic estimates: $h(S) = 5,\ h(A) = 4,\ h(B) = 3,\ h(C) = 2,\ h(G) = 0$

Figure 4. Original directed, weighted planning graph. S–A–C–G has cost 6; S–B–G has cost 8; S–A–G has cost 10.

4.3 Greedy Best-First trace

At S, candidates A and B have $h(A) = 4$ and $h(B) = 3$. Greedy chooses B. Expanding B reveals G with $h(G) = 0$, so Greedy next chooses G and returns **S–B–G with total cost** $4 + 4 = 8$. It overlooks the cheaper S–A–C–G route in this example.

4.4 Complete A* worked trace

Use graph search with a priority queue, update a state's stored cost if a cheaper path is found, and break ties alphabetically. Test the goal when it is *removed* as the minimum-priority candidate.

Node	$g(n)$	$h(n)$	$f(n) = g + h$
S	0	5	5
A (from S)	1	4	5
B (from S)	4	3	7
C (from A)	3	2	5
G (from C)	6	0	6

WORKED EXAMPLE: A* priority queue in successive expansions

Initially: S has $f = 5$. **Expand S:** A($g = 1, f = 5$), B($g = 4, f = 7$). Choose A. **Expand A:** C($g = 3, f = 5$), direct G($g = 10, f = 10$). Choose C. **Expand C:** update G to $g = 6, f = 6$, better than the earlier $g = 10$. B remains at $f = 7$. **Next choose G:** return S–A–C–G with total cost $1 + 2 + 3 = 6$.

Do not stop when a goal is merely generated. With A*, the appropriate goal test is usually when the minimum-f frontier node is removed, along with the required heuristic and search-implementation conditions. Early stopping when any goal is first seen can be suboptimal.

4.5 Important counterexample: fewer moves is not always less cost

On the same weighted graph, there is a route S–A–G with only two edges and cost $1 + 9 = 10$. There is also S–B–G with two edges and cost $4 + 4 = 8$. The route S–A–C–G has three edges but cost $1 + 2 + 3 = 6$. Under alphabetical successor order, BFS can return the first two-edge goal route at cost 10; it does not compare the accumulated edge weights. A* returns the three-edge cost-6 route under the stated heuristic. This is why an engineer must state whether a robot route is optimized for *number of moves*, *distance*, *time* or *energy*.

Weighted-edge rule. BFS's shortest-path guarantee concerns the fewest edges (or equal-cost edges). For unequal nonnegative edge costs, use uniform-cost search or A* with a justified heuristic; do not label the BFS route "minimum cost."

4.6 Heuristic functions and what makes them safe to use

A heuristic is **admissible** if it never overestimates the true minimum remaining path cost: $0 \leq h(n) \leq h^*(n)$ and $h(G) = 0$. A heuristic is **consistent** if for every transition $n \to n'$:

$$\boxed{h(n) \leq c(n, n') + h(n')} \tag{1}$$

Consistency implies admissibility when goal heuristics are zero and goals are reachable. It supports A* graph-search optimality with nonnegative edge costs under the usual reached/closed-set policy. A* tree search has a weaker admissibility requirement; reopening closed states when improved costs are found can be necessary for inconsistent heuristics.

WORKED EXAMPLE: Check every edge for consistency

For Figure 4, $h(S) = 5$ and edge S–A costs 1, so $5 \leq 1 + h(A) = 5$: pass. Edge A–C: $h(A) = 4 \leq 2 + h(C) = 4$: pass. C–G: $2 \leq 3 + 0 = 3$: pass. B–G: $3 \leq 4 + 0 = 4$: pass. S–B: $5 \leq 4 + 3 = 7$: pass. A–G: $4 \leq 9 + 0 = 9$: pass. Thus the illustrated heuristic is consistent.

4.7 Practical heuristic examples for mobile robots

For a four-neighbour unit-cost grid, Manhattan distance $h = |x - x_g| + |y - y_g|$ is admissible when moves cost at least one unit and no teleportation is allowed. Euclidean distance $h = \sqrt{(x - x_g)^2 + (y - y_g)^2}$ is a lower bound for distance-like edge cost if each move's cost is at least its geometric length. Do not blindly use cell-distance as a lower bound on *time* if some moves have unequal maximum speeds or special shortcuts. A heuristic and g must use compatible units.

4.8 Search-comparison table

Method	Priority	When it makes sense
BFS	Smallest depth	Uniform step cost; path with fewest moves.
DFS	Deepest frontier	Explore a branch with small stack frontier; not shortest.
Greedy Best-First	Smallest h	Prioritize estimated closeness; no guaranteed cheapest result.
Uniform-cost	Smallest g	Nonnegative path costs; compare without heuristic.
A*	Smallest $g + h$	Least-cost route with appropriate heuristic and graph policy.

5 Metaheuristics: Why Optimization is Different

In simple language
An optimization method searches for a good configuration: route order, robot placement, machine schedule or controller parameter values. In difficult combinatorial problems, a metaheuristic may quickly discover useful solutions without proving that they are globally optimal.

5.1 Objective, feasibility and algorithm budget

For a route R, one illustrative objective is $J(R) = L(R) + \lambda P(R)$, where L is route length and P is a penalty for congestion. The weight λ converts the penalty into length-compatible units. Some conditions, such as driving through a human-only zone, should be *hard constraints*, not just tiny penalties. An optimizer needs a maximum number of iterations or evaluations; repeated runs and baseline comparisons are essential because randomized results vary.

5.2 Local optimum versus global optimum

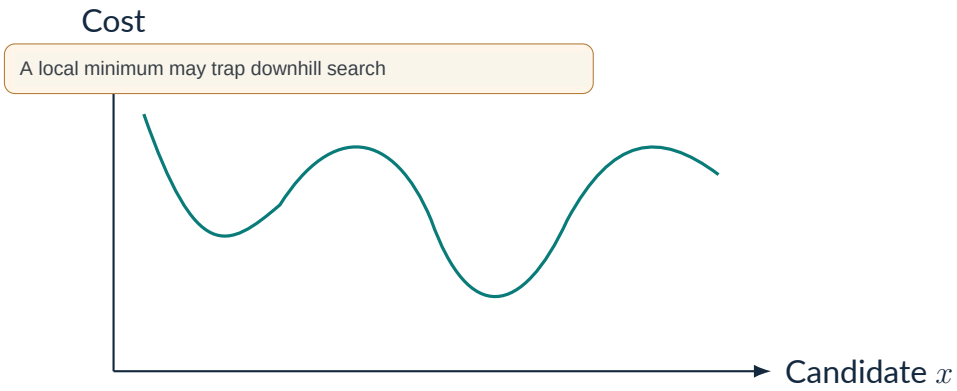


Figure 5. Conceptual optimization landscape, not data for a specific robot. Search heuristics may escape local minima at a computational cost.

Check your understanding
A shortest geometric route might intersect a forbidden safety zone. Why must an optimizer's representation reject such routes instead of assigning

a tiny soft penalty? Because the low cost of unsafe options does not make them legally or physically permissible.

6 Genetic Algorithm (GA) With Numerical

In simple language
A genetic algorithm maintains a collection (population) of candidate solutions. It assigns each a quality score, selects candidates to reproduce, combines parts of their representations and occasionally changes them. Repeating this creates new generations that may improve the best observed solution.

6.1 Representation determines validity

A *chromosome* stores a proposed answer. For a binary example, a chromosome might be `1011`. For a four-job schedule it might be a permutation $(3, 1, 4, 2)$. Choosing a representation is not a small detail: ordinary one-point crossover of permutations can produce duplicate or missing jobs and therefore requires a valid permutation-preserving operator or repair rule. For robot paths, mutation must likewise respect connectivity and collision constraints.

6.2 GA step-by-step flow

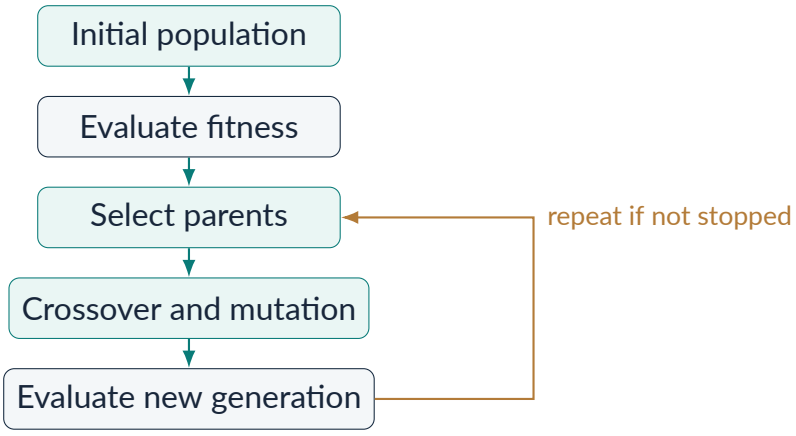


Figure 6. Original conceptual GA loop. A practical implementation must define replacement, elitism, valid offspring and stopping rules.

6.3 Fitness and roulette-wheel selection

Fitness-proportional selection uses $p_i = F_i / \sum_j F_j$ when every fitness is nonnegative and the sum is positive. Higher F receives higher selection probability. For a minimization cost J, do not use raw J as fitness or you may reward the worst candidate; first define a sensible transformation such as $F = 1/(1 + J)$ for nonnegative costs.

Candidate	Fitness	Probability	Cumulative interval
A	4	$4/20 = 0.20$	$[0, 0.20)$
B	6	$6/20 = 0.30$	$[0.20, 0.50)$
C	8	$8/20 = 0.40$	$[0.50, 0.90)$
D	2	$2/20 = 0.10$	$[0.90, 1)$

WORKED EXAMPLE: Select one parent

The total fitness is $4 + 6 + 8 + 2 = 20$. Draw a uniformly distributed random number r in $[0, 1)$. If $r = 0.65$, it falls in C's cumulative interval $[0.50, 0.90)$, so **C is selected**. Probabilities sum to $0.20 + 0.30 + 0.40 + 0.10 = 1.00$. A different random draw can select another parent.

6.4 One-point crossover and bit-flip mutation

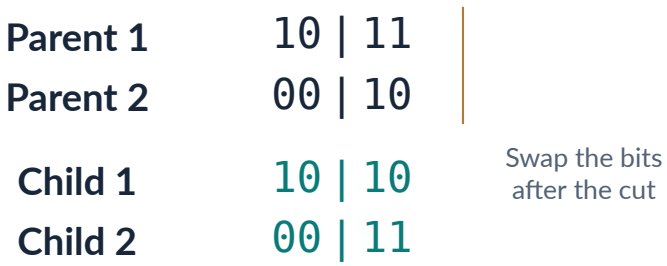


Figure 7. One-point crossover after the second bit gives children 1010 and 0011. For a bit-flip mutation, 1010 becomes 1000 when its third bit is flipped.

WORKED EXAMPLE: Parent selection and variation: an illustrative GA iteration

Choose population A–D from the fitness table. Select parents via two roulette draws, e.g. $r_1 = 0.65$ selects C and $r_2 = 0.35$ selects B. For a separate illustrative two-parent bitstring demonstration, crossover parents `1011` and `0010` after bit 2, giving children `1010` and `0011`. Flip bit 3 of `1010` to produce `1000`. **Important:** the fitness-table letters and the bitstring demonstration are separate examples; do not assume the selected C and B have these bit patterns without explicitly assigning them.

6.5 Why replacement and elitism matter: a complete mini-generation

For an *independent* maximization example, define binary-chromosome fitness as the **number of ones** and start with two parents: 1011 (fitness 3) and 0010 (fitness 1). Cut after bit 2: children are 1010 and 0011 (both fitness 2). If the third bit of 1010 mutates from 1 to 0, it becomes 1000 (fitness 1). **Without elitism**, replacing both parents with 1000 and 0011 lowers the best fitness from 3 to 2. **With one-elite preservation**, keep 1011 and select an additional offspring according to the declared replacement policy; the best remains at least 3. Thus crossover, mutation and survivor selection must be analysed separately; evolution does not guarantee improvement every generation.

Do not mix example assumptions. The earlier four-candidate roulette table uses externally supplied fitness values $(4, 6, 8, 2)$. The two-bitstring-parent example above uses the *count of ones* as its own fitness rule. Their numerical results are deliberately not interchangeable.

6.6 Common GA mistakes

- Confusing minimizing cost with maximizing fitness.
- Forgetting that roulette selection samples probabilistically: higher fitness does not guarantee selection.
- Using ordinary binary crossover directly for a permutation route representation.
- Omitting elitism or a stop rule and assuming every generation improves the

current best.

- Failing to validate collision-free, capacity-limited and otherwise feasible offspring.

7 Simulated Annealing (SA) With Numerical

In simple language
Simulated annealing moves between neighbouring candidate solutions. It usually accepts improvements, and sometimes accepts a worse candidate. Accepting a worse move can escape a local minimum. Its willingness to do so decreases as the temperature parameter cools.

7.1 Acceptance rule for a minimization problem

Define $\Delta J = J_{new} - J_{current}$. If $\Delta J \leq 0$, accept. If $\Delta J > 0$, accept with the illustrative Metropolis probability

$$\boxed{P(\text{accept worse move}) = \exp(-\Delta J/T)}, \qquad T > 0. \tag{2}$$

The scale of T must match the scale of the objective difference ΔJ, so the ratio is dimensionless. Random draw $r \in [0, 1)$ decides acceptance: accept when $r < P$.

WORKED EXAMPLE: A worse solution accepted at a high temperature

A robot-route candidate has cost 12. A proposed neighbour costs 15, so $\Delta J = 15 - 12 = 3$. At temperature $T = 5$:

$$P = \exp(-3/5) = \exp(-0.6) \approx 0.5488. \tag{3}$$

For a random draw $r = 0.43$, $0.43 < 0.5488$, so accept this worse move. If $r = 0.75$ instead, reject it. After one geometric cooling step with multiplier $\alpha = 0.8$, $T_{next} = 0.8(5) = 4$. A proposal of the same cost increase at $T = 4$ would be accepted with probability $e^{-3/4} \approx 0.4724$, less than before.

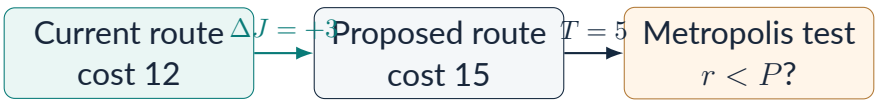


Figure 8. SA decision flow: a cost increase is not always rejected; probability reduces as temperature falls.

7.2 Cooling schedules and limitations

A geometric cooling schedule is $T_{k+1} = \alpha T_k$, with $0 < \alpha < 1$. Choose initial temperature, move generator, cooling speed, iteration budget and stopping rule. Too-fast cooling can trap a poor solution; too-slow cooling wastes evaluations. No finite practical run guarantees a global optimum simply because the method was inspired by annealing.

Check your understanding
A candidate improves cost from 15 to 13. Must you calculate $e^{-\Delta/T}$ to decide acceptance? No. With the stated minimization rule, every non-worse move is accepted directly.

8 Tabu Search With Numerical

In simple language
Tabu Search is a local-search method that temporarily forbids recently used moves or their reversals. This short-term memory discourages oscillation between the same candidates. An aspiration rule may override tabu status when a candidate is sufficiently valuable.

8.1 Tabu tenure, aspiration and diversification

Neighbourhood: candidate solutions one allowed edit away, such as swapping two jobs. **Tabu list:** a record of recent move attributes, often their reversals. **Tenure:** how many iterations an attribute stays forbidden. **Aspiration:** a defined reason to permit an otherwise tabu move, commonly if it improves the best-known solution. **Diversification:** periodically directing search toward less-explored regions.

WORKED EXAMPLE: Select the best admissible neighbour

Current schedule cost is 20 (smaller is better). The best global cost encountered earlier is 16. The current neighbours are A(cost 18), B(cost 17) and C(cost 19). Move to B is on the tabu list. With the specific aspiration policy *"override tabu only if the candidate cost is strictly better than 16"*, B's cost 17 fails aspiration. Candidates A and C are permitted, so choose A with cost 18. Record the prescribed tabu attribute for the accepted move for the chosen tenure.

Neighbour	Cost	Status	Decision
A	18	Allowed	Choose, best admissible.
B	17	Tabu; does not beat 16	Not allowed under stated aspiration policy.
C	19	Allowed	Allowed but worse than A.

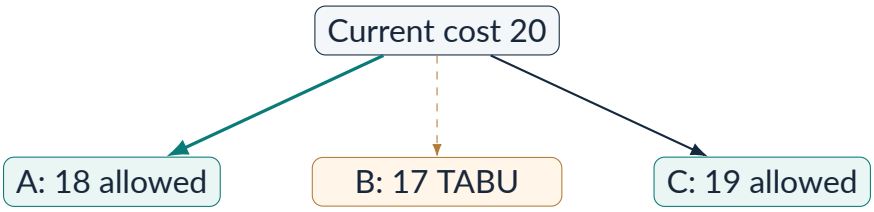


Figure 9. Original Tabu Search decision diagram. The lowest-cost neighbour is not necessarily admissible.

Tabu is not a permanent ban. The move attributes expire after their tenure, and aspiration criteria differ among implementations. A solver should explicitly document whether it stores full solutions, move attributes or inverse moves.

9 Ant Colony Optimization (ACO) With Numerical

In simple language
Ant Colony Optimization maintains artificial pheromone values on candidate graph edges. An artificial ant selects a next move using both pheromone and a local heuristic such as inverse distance. Better routes may deposit more pheromone, while evaporation reduces old influence.

9.1 Transition probability

For ant k at node i, let N_i^k be its feasible next neighbours. A common ACO rule is

$$\boxed{p_{ij}^{(k)} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in N_i^k} \tau_{il}^{\alpha}\eta_{il}^{\beta}}}, \qquad j \in N_i^k. \tag{4}$$

Here τ_{ij} is pheromone strength, η_{ij} is heuristic desirability, and $\alpha, \beta \geq 0$ are weights. Often $\eta_{ij} = 1/d_{ij}$ for positive edge distance d_{ij}. The probabilities over the currently feasible neighbours sum to one.

WORKED EXAMPLE: Probability of two possible robot routes

From junction S, an ant may choose edge 1 with $\tau_1 = 2$ and $d_1 = 2$, or edge 2 with $\tau_2 = 1$ and $d_2 = 4$. Let $\alpha = 1$, $\beta = 2$ and $\eta = 1/d$.

$$
\begin{aligned}
W_1 &= 2(1/2)^2 = 0.5000, \\
W_2 &= 1(1/4)^2 = 0.0625, \\
P_1 &= \frac{0.5000}{0.5625} = \frac{8}{9} \approx 0.8889, \\
P_2 &= \frac{0.0625}{0.5625} = \frac{1}{9} \approx 0.1111.
\end{aligned}
$$

Interpretation: route 1 is more likely but route 2 still has a nonzero sampling probability. Unlike Greedy Best-First, this is a stochastic next-edge choice rather than a guarantee that route 1 will be picked.

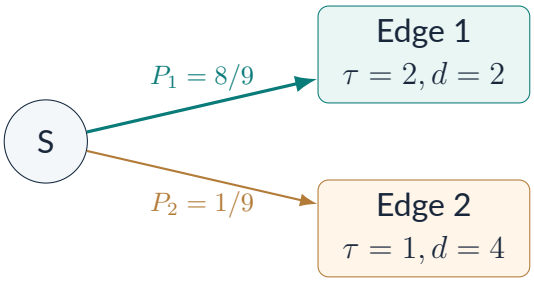


Figure 10. Original two-edge ACO choice diagram. Probabilities are computed from pheromone and inverse-distance desirability.

9.2 Pheromone update and evaporation

A simple illustrative update for an edge used by an ant is

$$
\boxed{\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \Delta\tau_{ij}}, \qquad \Delta\tau_{ij} = Q/L \tag{5}
$$

where $0 \leq \rho \leq 1$ is the evaporation rate, Q a defined deposit constant, and $L > 0$ the complete route cost. In a multi-ant iteration, deposits can be summed across ants that used the edge; untouched edges may only evaporate. This is one illustrative ant-system variant, not a claim that all ACO implementations use an identical update.

WORKED EXAMPLE: Calculate one pheromone-update step

Edge 1 starts with $\tau_1 = 2$. Let $\rho = 0.20$ and one ant using edge 1 finish a complete route with $L = 8$ and deposit constant $Q = 4$. Then

$$\tau_1^{new} = (1 - 0.20)(2) + 4/8 = 1.60 + 0.50 = \boxed{2.10}.$$

If edge 2 is not used by any ant in the iteration and has $\tau_2 = 1$, it only evaporates: $\tau_2^{new} = 0.8(1) = \boxed{0.8}$.

9.3 Why ACO needs repeated evaluation

With many route combinations, pheromone can over-concentrate on a mediocre path. Engineers tune pheromone limits, exploration rate, evaporation, ant count, feasibility handling and termination. ACO does not automatically respect driving direction, battery limits or dynamic collision risks; those must be built into admissible moves and planning constraints.

10 Applications in Robot Path Planning, Routing and Scheduling

In simple language
The same algorithm can be useful for very different robotics tasks only if its mathematical representation matches the real constraints. Choosing an AI method begins with a clear objective, a model of allowed actions, and a way to check a solution before execution.

10.1 Path planning: where may a robot travel?

Grid planning discretizes space into free cells and obstacles. BFS is effective for minimum-move paths when all moves have equal cost. A* typically finds a minimum-model-cost route with fewer expansions when a useful heuristic is available, although worst-case search remains expensive. **Geometric motion planning** may additionally require heading, turning radius, collision geometry, stopping distance and velocity limits. A shortest grid route is not a motor command.

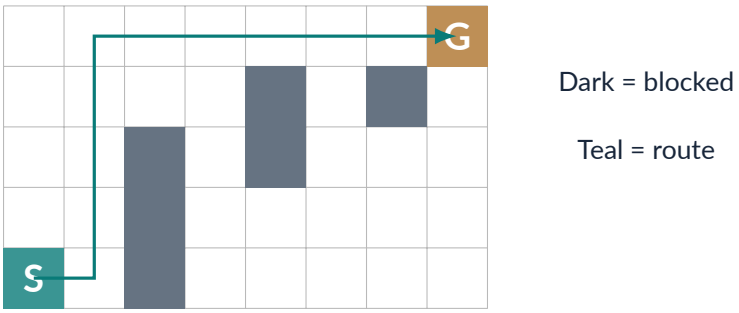


Figure 11. Conceptual free-cell grid and obstacle-avoiding route. The illustrated route is an example, not an algorithmic optimality certificate or a verified executable vehicle trajectory.

10.2 Routing: in what order should deliveries occur?

A mobile robot may have to visit n shelves. A search state could be $(current_node, visited_set, battery_level)$. The objective might be total travel distance, on-time deliveries or weighted energy, with hard constraints for one-way aisles, battery reserve and no-go zones. GA or ACO may find useful visit orders when exact enumeration is too expensive. The route order should then be checked against a feasible path planner on each segment.

WORKED EXAMPLE: A small order-of-visits comparison

Suppose the depot is D and shelves are A and B, with direction-dependent travel costs (for example, one-way aisles) $d(D, A) = 3$, $d(A, B) = 2$, $d(B, D) = 5$ and $d(D, B) = 4$, $d(B, A) = 2$, $d(A, D) = 3$. Going D–A–B–D costs $3+2+5 = 10$ distance units. D–B–A–D costs $4+2+3 = 9$ units. The second order is cheaper *under the given distance table*; time-dependent traffic or one-way paths could change this result.

10.3 Scheduling: which job runs on which machine?

A robot-assisted manufacturing cell may assign jobs to machines subject to durations, predecessors, tool changes, robot transfer time and machine downtime. Its objective could minimize *makespan* (completion time of the last job), lateness or energy. A schedule representation must enforce the rule that a machine cannot process two exclusive jobs simultaneously.

WORKED EXAMPLE: Two machines and two jobs

Assume jobs A and B can run concurrently on separate machines. A takes 4 min on M1 or 6 min on M2; B takes 5 min on M1 or 3 min on M2. Assignment A to M1 and B to M2 gives makespan $\max(4, 3) = 4$ min. Swapping jobs gives makespan $\max(6, 5) = 6$ min. If both jobs require the same gripper or shared transport robot at once, additional resource constraints would change the feasible schedule.

10.4 A safe integration architecture

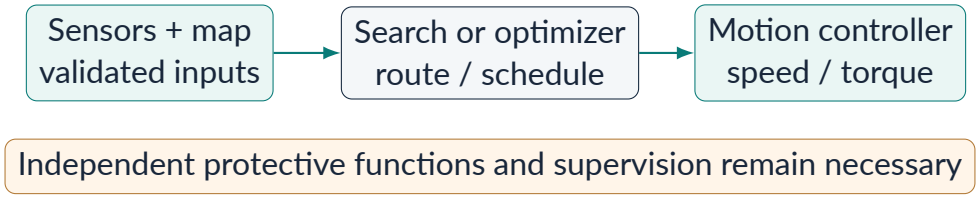


Figure 12. A search or optimization planner feeds a conventional controller; its output does not replace risk-assessed protective functions.

10.5 Choosing a method and its limitations

Method	Typical useful context and key caution
BFS	Minimum move count on a uniformly weighted finite grid; memory grows rapidly.
DFS	Quick exploration, reachability checks; not shortest and may waste time in deep branches.
Greedy Best-First	Fast heuristic-guided first answer; may return an expensive route.
A*	Least-cost graph route given compatible costs and justified heuristic; large maps may still overwhelm memory.
Genetic Algorithm	Route order, schedule or parameter combinations; solution validity and randomness require testing.
Simulated Annealing	Local improvement with occasional worse-move acceptance; cooling schedule affects outcomes.
Tabu Search	Swap-based scheduling and routing; must document tabu tenure and aspiration criteria.
Ant Colony Optimization	Graph routing with stigmergic exploration; parameter sensitivity and premature convergence.

Engineering boundary. This guide uses conceptual planning examples. It does not verify an industrial safety function, collision-avoidance system, robotic application, or actual production schedule. Machine-specific controllers must satisfy applicable limits, protective standards and independent validation.

11 Exam Workshop, Numerical Practice and Common Errors

11.1 Exam-ready answer structure

For a five- or six-mark question, write the definition, identify the queue/stack-/priority or objective function, draw a small labelled diagram, show one com-

plete numerical step with units, and finish with the method's key strength and limitation. On numerical questions, write down the tie-break policy, edge costs, starting population or parameter settings before calculating.

11.2 Ten practice problems (original, syllabus-derived)

1. Formulate an autonomous robot's warehouse route problem: state, goal, actions, transition and cost.
2. Trace BFS on Figure 2 with goal test at dequeue and write each queue.
3. Trace DFS on Figure 2 with reverse alphabetical pushes and compare path lengths.
4. Distinguish Greedy Best-First and A* on the weighted graph in Figure 4; calculate g, h, f.
5. Check consistency for the heuristic on all six directed edges of Figure 4.
6. Calculate roulette probabilities for fitness values $4, 6, 8, 2$ and select for $r = 0.65$.
7. Perform one-point crossover of 1011 and 0010 after bit 2, then flip bit 3 of the first child.
8. At $T = 5$, calculate SA acceptance for cost 12 to 15, then cool by factor 0.8.
9. With current cost 20, neighbours A(18), B(17, tabu), C(19) and global best 16, select the next Tabu Search move.
10. Compute ACO edge probabilities from $\tau = (2, 1)$, $d = (2, 4)$, $\alpha = 1$, $\beta = 2$, then update pheromone using $\rho = 0.20, Q = 4, L = 8$.

11.3 Worked answer key

Q.	Concise verifiable result
1	State (x, y) or (x, y, θ); start known pose; goal target; obstacle-free moves; cost chosen with units.
2	Queue: [S], [A,B], [B,C], [C,G], [G,D]; goal S–B–G, 2 edges.
3	Stack: [S], [A,B], [C,B], [D,B], [G,B]; goal S–A–C–D–G, 4 edges.
4	Greedy: S–B–G cost 8; A*: S–A–C–G cost 6.
5	All six Figure 4 edges satisfy $h(n) \leq c(n, n') + h(n')$.
6	$p = (0.20, 0.30, 0.40, 0.10)$; $r = .65$ selects C.
7	Children 1010 and 0011; first child after bit-3 flip: 1000.
8	$P = e^{-0.6} \approx 0.5488$; next $T = 4$.
9	B is tabu and fails aspiration; choose A with cost 18.
10	$P_1 = 8/9$, $P_2 = 1/9$; edge 1 new pheromone 2.10, unused edge 2 0.80.

11.4 Eight pitfalls to avoid in written exams

1. Saying BFS is optimal for arbitrary positive weighted edges. It minimizes depth when costs are uniform.
2. Giving DFS one universal node expansion order without specifying successor ordering.
3. Confusing Greedy h with A* $g + h$ or reporting h as cost already traveled.
4. Claiming an A* solution is always optimal regardless of heuristic/graph policy.
5. Choosing a GA chromosome representation that creates impossible routes after crossover.
6. Accepting every SA worsening move or forgetting the random-draw condition.
7. Ignoring tabu restrictions when selecting the lowest-cost candidate.
8. Forgetting to normalize ACO transition weights or evaporate pheromone on unused edges.

One-minute summary
BFS = FIFO shortest in *steps*. DFS = LIFO deepest first. Greedy = smallest h. A* = smallest $g + h$, with heuristic and implementation conditions. GA = population selection and variation. SA = sometimes accepts worse solutions. Tabu = temporary memory blocks cycling. ACO = pheromone and heuristic-weighted probability. A valid robot solution still needs collision, dynamics and safety checks.

Sources, provenance and limitations

Syllabus authority: Savitribai Phule Pune University, TE Robotics and Automation, 2024 Pattern, PCC301-ROA, *Artificial Intelligence for Robotics*, Unit II, *Search and Metaheuristic Optimization Techniques*, supplied *TE Syllabus.pdf*, printed p. 14. Exact syllabus topic coverage above is syllabus-derived.

Technical sources:

1. UC Berkeley CS188, *Uninformed Search*, BFS/DFS and uniform-cost conditions. https://inst.eecs.berkeley.edu/~cs188/textbook/search/uninformed.html
2. UC Berkeley CS188, *Informed Search*, g, h, f, A* and heuristics. https://inst.eecs.berkeley.edu/~cs188/textbook/search/informed.html
3. Russell and Norvig, *Artificial Intelligence: A Modern Approach*, official sample chapter *Solving Problems by Searching*. https://aima.eecs.berkeley.edu/4th-ed/pdfs/newchap03.pdf
4. Cornell University Computational Optimization Open Textbook, *Genetic Algorithm*. https://optimization.cbe.cornell.edu/index.php?title=Genetic_algorithm
5. Delft University of Technology MUDE, *Genetic Algorithm*, roulette selection and crossover example. https://mude.citg.tudelft.nl/book/2025/optimization/genetic_algorithm.html
6. Cornell University Computational Optimization Open Textbook, *Simulated Annealing*. https://optimization.cbe.cornell.edu/index.php?title=Simulated_annealing
7. M. Dorigo et al., *Ant Colony Optimization*, Wiley Encyclopedia of Operations Research and Management Science (2011). https://iridia.ulb.ac.be/~mdorigo/Published_papers/2011/DorMonOliStu2011eorms.pdf
8. Tabu Search decision example is an original pedagogical illustration of tabu tenure and aspiration; consult standard tabu-search literature for implementation variants and formal algorithm definitions.

Figure provenance: Diagrams, graph topologies, arbitrary costs, fitness tables, schedule data and numbers are original instructional examples. They are not measured factory data, copied textbook figures, or validated industrial hardware diagrams. No external photographs are embedded because the licensed file bytes could not be retrieved reliably into this build environment.

Review boundary: This document is a syllabus-mapped learning candidate, not an instructor-certified textbook, a past-paper archive, evidence of hardware testing, a guarantee of globally optimal metaheuristic results or a released website file. Mathematical calculations are reproducible under their stated assumptions.

AIFR UNIT II Independent Numerical QA Supplement

PCC301-ROA | SPPU TE Robotics & Automation | 2024 Pattern

Scope: Six exact numerical problems from the supplied AIFR Unit-Wise Question Bank; additional warnings on heuristic assumptions, simulated-annealing notation and required v3 edits. Results were recalculated independently. This supplement does not certify the existing 27-page v2 PDF or replace its page-by-page visual review.

Check	Result
Q2 BFS visiting order	A, B, C, D, E, F, H, G
Q4 ACO: first move B to E	1/3 = 33.33%
Q6 A*: A to G	A-B-D-F-G; cost 7
Q8 SA worse-move acceptance	exp(-15/50) = 0.740818
Q10 A*: A to J	A-F-G-I-J; cost 10
Q12 SA worse-move acceptance	exp(-10/55) = 0.833753

Verification and interpretation

The A* answers were checked against an independent uniform-cost shortest-path calculation on the exact supplied edges. The agreement verifies these particular path costs; it is not a proof that arbitrary inconsistent or nonzero-goal heuristics guarantee optimality.

Critical warning: The Question Bank gives h(J) = 3 even though J is the goal in Q10. This violates the usual h(goal) = 0 admissibility condition. For a corrected classroom problem set h(J) = 0 and rerun the priority queue; for reproducing the original question, use its printed values and disclose the assumption.

Documents reviewed: official TE Syllabus.pdf (PCC301-ROA Unit II), AIFR_Unit_Wise_Question_Bank.pdf, and Zenith_AI_Unit_02_Search_Metaheuristics_Reviewed_v2.pdf. Exact Drive links appear on the final page.

Q2 | Breadth-First Search from A

Undirected connections: A-B, A-C, A-D, B-E, C-F, C-H, D-G. Each vertex is marked discovered when enqueued; the queue is FIFO; visit neighbours alphabetically.

Step	Dequeue	Queue after expansion	Visited order
0	-	A	-
1	A	B, C, D	A
2	B	C, D, E	A, B
3	C	D, E, F, H	A, B, C
4	D	E, F, H, G	A, B, C, D
5	E	F, H, G	A, B, C, D, E
6	F	H, G	A, B, C, D, E, F
7	H	G	A, B, C, D, E, F, H
8	G	empty	A, B, C, D, E, F, H, G

BFS spanning-tree edges: A-B, A-C, A-D, B-E, C-F, C-H, D-G.

Important: the answer above is the traversal order, not merely one goal path. A graph with a specified goal would additionally require identifying when the search stops.

Q4 | Ant Colony Optimization, first move B to E

At B, feasible unvisited cities are A, C, D, E. The distances are 12, 15, 20, 10. The question specifies equal pheromone strength 1, heuristic 1/d, and exponents alpha = beta = 1.

Weight for each candidate is 1/d. Sum = 1/12 + 1/15 + 1/20 + 1/10 = 3/10 = 0.3.

P(B to E) = (1/10)/(3/10) = 1/3 = 33.33%.

No second-move probabilities or later pheromone updates are needed: the question asks for the first move only.

Q6 | A* on directed graph A to G

Directed edge costs: A-B 1, A-C 4, B-D 3, B-C 2, C-E 5, D-F 2, D-G 4, E-G 3, F-G 1. Heuristics: h(A,B,C,D,E,F,G) = (5,6,4,3,3,1,0).

Define g = path cost so far, h = stated goal estimate, and f = g + h. Use lowest f, break ties alphabetically, update a state if a cheaper path is found, and test the goal when popped.

Selected	g	h	f	Key event
A	0	5	5	Generate B (f=7), C (f=8)
B	1	6	7	Improve C to g=3 (f=7); add D (f=7)
C	3	4	7	Add E at g=8 (f=11)
D	4	3	7	Add F at g=6 (f=7), G at g=8 (f=8)
F	6	1	7	Improve G to g=7, f=7
G	7	0	7	Goal popped; stop

Expansion order: A, B, C, D, F, G.

Path: A-B-D-F-G. Cost = 1 + 3 + 2 + 1 = 7.

Independent check: uniform-cost search also returns 7 for the supplied graph. No negative-cost edges appear.

Q8 | Simulated Annealing

For minimization, old cost = 100 and proposed cost = 115. Therefore Delta = 115 - 100 = 15. With the question's numerical temperature parameter T = 50:

P(accept worse candidate) = exp(-Delta/T) = exp(-15/50) = exp(-0.3) = 0.740818 = 74.08%.

This is a probability, not a statement that the move is certainly accepted. A random r drawn uniformly on [0,1) accepts the move only if r is less than 0.740818.

Q10 | A* on undirected graph A to J

Use all undirected edges and weights from the question bank: A-F 3; A-B 6; B-D 2; B-C 3; C-D 1; C-E 5; D-E 8; E-I 5; E-J 5; I-J 3; I-H 2; I-G 3; F-G 1; F-H 7.

Heuristics: A 10, B 8, C 5, D 7, E 3, F 6, G 5, H 3, I 1, J 3. All stated values are used in the trace below, even though h(J)=3 is not admissible. Tie-breaking: alphabetical.

Selected	g	h	f	Key event
A	0	10	10	Generate F (f=9), B (f=14)
F	3	6	9	Generate G (g=4, f=9), H (g=10, f=13)
G	4	5	9	Generate I (g=7, f=8)
I	7	1	8	Improve H to g=9 (f=12); add J g=10 (f=13); E g=12 (f=15)
H	9	3	12	No better frontier routes
J	10	3	13	Goal popped; stop

Expansion order: A, F, G, I, H, J.

Path: A-F-G-I-J. Cost = 3 + 1 + 3 + 3 = 10.

Independent uniform-cost verification also gives shortest cost 10. Because h(J) = 3 > 0 = true cost-to-go at J, the textbook A* admissibility guarantee cannot be invoked using this supplied heuristic, even though this numerical answer is optimal.

Q12 | Simulated Annealing

Old cost = 20, proposed cost = 30, so Delta = 10. With the stated numerical algorithm parameter T = 55:

P(accept worse candidate) = exp(-10/55) = exp(-2/11) = 0.833753 = 83.38%.

The question bank uses degrees Celsius as a temperature label. In these optimization exercises, the exponent must be dimensionless: the algorithmic T is scaled to the objective-cost difference. Do not silently convert it from Celsius to kelvin; that would change the defined optimization problem.

Hill Climbing | Required supplementary teaching coverage

Question 13 asks for Hill Climbing even though it is not explicitly named in the official Unit II topic list. It is useful as supplemental local-search background but must not displace the four named syllabus metaheuristics.

Term	Exam-ready explanation
Hill climbing	Repeatedly move to an improving neighbour under an explicitly specified evaluation function. Stop when no improving neighbour exists.
Local maximum/minimum	A candidate better than nearby candidates but not necessarily globally best.
Plateau	Neighbouring states with equal evaluation scores, giving no obvious improving direction.
Ridge	Improvement direction may require combined moves that single-coordinate neighbour operators cannot make.
Remedies	Random restarts, sideways moves with bounds, stochastic selection, larger neighbourhoods; SA can sometimes accept worse moves.

Required changes for Unit II v3

1. Add the six exact solved numericals and their stated algorithm conventions. 2. Clarify Q10 nonzero-goal heuristic. 3. Explain numerical SA temperature instead of implying a physical Celsius conversion. 4. Add short supplemental Hill Climbing section. 5. Explicitly distinguish question-bank examples from the reviewed guide's original examples. 6. Render and inspect all original PDF pages and figures before release.

Review status: arithmetic PASS for the six results in this supplement; broader academic QA and visual PDF approval remain OPEN.

Source traceability and publication gate

Source A | Official course syllabus

SPPU TE Robotics & Automation, 2024 Pattern, PCC301-ROA, Unit II (Search and Metaheuristic Optimization Techniques). Source Drive ID: 1filXvrqbGbqBjZ-QXVcqEdUXWVMcktck.

Source B | AIFR unit-wise question bank

Questions 2, 4, 6, 8, 10, 12 and 13. Source Drive ID: 1QGsRzNIH7ivQgr9BtyhqkaW-qKd4D3Ou. Questions were transcribed from the Drive PDF text returned in this review.

Source C | Existing reviewed guide

Zenith_AI_Unit_02_Search_Metaheuristics_Reviewed_v2.pdf. Source Drive ID: 1Cc532Z4kyMHWee4VTItmGhn2Y_gnmYbm. This supplement is a separate review artifact and does not alter the existing guide.

Review scope and limits

The numerical answers were verified using independently implemented queue/priority-queue and shortest-path calculations. PDF page rendering and visual inspection of the original Unit II guide, plus all-figure technical review, remain mandatory before any publication-ready v3 claim. No website or Drive revision was replaced.

Release decision

Gate	Status
Official Unit II topic inventory vs guide text	Preliminary text-level coverage seen; full mapping open
Six worked numerical answers in this supplement	PASS (independent arithmetic)
Actual question-bank Q10 heuristic warning	DOCUMENTED
Full 27-page original PDF visual inspection	OPEN
Diagram and source-asset accuracy	OPEN
Final academically approved replacement	NOT APPROVED