

Assignment 1

Q 1. Compare the time and space complexity of BFS and DFS.

We assume a (possibly infinite) implicit search tree with uniform branching factor b and solution depth d .

1. Breadth-First Search (BFS)

- Expands nodes level by level.
- Time complexity: visits every node up to level d , so

$$T_{\text{BFS}} = \sum_{i=0}^d b^i = O(b^d).$$

- Space complexity: stores the entire frontier of the last level, of size b^d , so

$$S_{\text{BFS}} = O(b^d).$$

2. Depth-First Search (DFS)

- Follows one branch down to depth d before backtracking.
- Time complexity: in the worst case explores all nodes to depth $d \Rightarrow$

$$T_{\text{DFS}} = O(b^d).$$

- Space complexity: only the current path plus unexplored siblings on the call stack $\Rightarrow O(d)$ (or $O(bd)$ if you count all siblings), but crucially linear in depth, not exponential:

$$S_{\text{DFS}} = O(d).$$

Summary table:

Algorithm	Time	Space
BFS	$O(b^d)$	$O(b^d)$
DFS	$O(b^d)$	$O(d)$

Q 2. N-Queens for $N = 5$ as a state-space search and solution by DFID.

1. Formulation

- **State:** a partial placement of queens in the first k rows, $0 \leq k \leq 5$, with no two attacking.
- **Start:** $k = 0$ (empty board).
- **Successors:** place a queen in row $k + 1$ in any column that does not conflict with existing queens.
- **Goal:** $k = 5$ (all 5 queens placed legally).

2. Depth-First Iterative Deepening (DFID)

DFID runs depth-limited DFS repeatedly with increasing limits $L = 0, 1, 2, \dots$ until a solution is found.

- **Limit** $L = 0$: only depth-0 (no placements) \Rightarrow fail.
- **Limit** $L = 1$: place one queen in row 1 in any of 5 columns \Rightarrow partial states; none reach full depth \Rightarrow fail.
- ...
- **Limit** $L = 5$: DFS searches the full depth-5 tree of legal placements. The first complete depth-5 node is a valid 5-queens solution.

Complexity:

- Time: $O(b^d)$ where $b \leq N$ and $d = N$.
- Space: $O(d) = O(N)$.

DFID thus combines BFS's completeness (will find the 5-queen solution if one exists) with DFS's $O(N)$ space.

Q 3. Apply DBDFS with bound = 5 on the given graph. Show OPEN/CLOSED each iteration.

We number neighbors of each node in alphabetical order when pushing.

Depth-Bounded DFS (DBDFS) runs a single depth-limited DFS to depth 5, tracking OPEN (stack) and CLOSED (visited in this run). We stop upon discovering node G.

– **Iteration 1:** bound = 0

OPEN = [S: depth 0]

CLOSED = []

- Pop S (depth 0 = bound) \Rightarrow no expansion.

\Rightarrow No solution found.

– **Iteration 2:** bound = 1

OPEN = [S: 0], CLOSED = []

- Pop S ($0 < 1$) \Rightarrow expand to A,B,C,D (each now depth 1).

OPEN = [A:1, B:1, C:1, D:1]; CLOSED = [S]

- Pop D (1=bound) \Rightarrow cannot expand.
- Pop C, B, A similarly; none expand; no goal G reached.

– **Iteration 3:** bound = 2

OPEN = [S:0], CLOSED = []

- Pop S \Rightarrow expand \rightarrow OPEN=[A:1, B:1, C:1, D:1], CLOSED=[S].
- Pop D (1<2) \Rightarrow expand D's children in alphabetical order (ignore those in CLOSED):
Neighbors of D: {S,C,I}.
S,C \in CLOSED \rightarrow only I.
OPEN = [A:1, B:1, C:1, I:2]; CLOSED=[S,D].
- Pop I (depth 2=bound) \Rightarrow no expand.
- Pop C (1<2) \Rightarrow expand \rightarrow neighbors {S,B,G,H,D}. S,B,D closed \rightarrow add [G:2, H:2].
OPEN = [A:1, B:1, G:2, H:2, I:2]; CLOSED=[S,D,C].
- Next pop H or G (both depth 2). When we pop **G**, we see it is the goal.

At the moment **just before** popping *G*, the lists are:

OPEN = [A, B, G, H, I]

CLOSED = [S, D, C]

DBDFS with bound = 2 finds *G* on its third iteration.

Q 4. Hill Climbing vs Beam Search. Which is complete?

Aspect	Hill Climbing	Beam Search (<i>k</i> -beam)
Memory	1 current state	Up to <i>k</i> states ("beam")
Move selection	Choose single best neighbor	Expand all beam states, keep top <i>k</i>
Local optima	Easily trapped (no backtracking)	Less so if <i>k</i> large—but can still drop the path to goal if beam too narrow
Completeness	Not complete	Not guaranteed; completeness if <i>k</i> \rightarrow very large (\geq branching factor)

Q 5. Tabu Search expansions for the SAT instance

$$(\neg a \vee d) \wedge (c \vee b) \wedge (\neg c \vee d) \wedge (\neg d \vee \neg b) \wedge (a \vee \neg d)$$

Variables (*a, b, c, d*). Start $\mathbf{x}^0 = (0, 0, 0, 0)$. Tabu tenure = 2 (once we flip a variable, it cannot flip back for 2 iterations).

1. Evaluate initial x^0 :

Clause truths:

$$(\neg 0 \vee 0) = 1, \quad (0 \vee 0) = 0, \quad (\neg 0 \vee 0) = 1, \quad (\neg 0 \vee \neg 0) = 1, \quad (0 \vee \neg 0) =$$

Score = 4/5.

2. Tabu list = \emptyset . Generate neighbors by flipping each bit:

- Flip a : $(1, 0, 0, 0) \Rightarrow \text{score} = 3$
- Flip b : $(0, 1, 0, 0) \Rightarrow \text{score} = 5$
- Flip c : $(0, 0, 1, 0) \Rightarrow \text{score} = 4$
- Flip d : $(0, 0, 0, 1) \Rightarrow \text{score} = 3$

3. Best move = flip $b \rightarrow \mathbf{x}^1 = (0, 1, 0, 0)$ with score 5 (all clauses satisfied).

- Update tabu: forbid flipping b for next 2 iterations.
- Terminate: we found a fully satisfying assignment in one expansion.

Thus, **after one expansion**, tabu search reaches a solution. (If no perfect solution had appeared, we'd continue flipping the best non-tabu move, decrement tabu tenures each iteration, etc.)

Q 6. Genetic Algorithms for solving a 3-SAT instance

Example 3-SAT formula on variables x_1, x_2, x_3, x_4 :

$$F = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_2).$$

We solve via a Genetic Algorithm (GA) as follows.

1. Encoding (Chromosome Representation)

Represent each candidate as a bit-string $\langle x_1, x_2, x_3, x_4 \rangle$, where bit = 1 means true.

2. Initial Population

Generate N random 4-bit strings, e.g.

1011, 0110, 1100, 0001, ...

3. Fitness Function

For an assignment c , define

$$\text{fitness}(c) = \# \text{ of clauses in } F \text{ satisfied by } c, \quad \text{ranges } 0 \leq f \leq 3.$$

4. Selection

Use fitness-proportionate (roulette-wheel) or tournament selection to pick parents.

5. Crossover

Single-point: choose crossover point $k \in \{1, 2, 3\}$, swap tails.

Example: parents 1011 and 0110, cut at $k = 2$:

$$\underbrace{10}_{\text{head}} \mid \underbrace{11}_{\text{tail}}, \quad \underbrace{01}_{\text{head}} \mid \underbrace{10}_{\text{tail}} \longrightarrow \{ 10 \ 10 = 1010, \quad 01 \ 11 = 0111 \}.$$

6. Mutation

Flip each bit with small probability p_{mut} . E.g. flip one bit in 1010 \Rightarrow 1000.

7. Replacement

Form new population from offspring (and perhaps top parents).

8. Termination

Stop when a chromosome has fitness = 3 (all clauses satisfied), or after max generations.

Over successive generations, GA tends to improve average fitness until a fully satisfying assignment appears.

Q 7. Ant Colony Optimization (ACO) for the Traveling Salesman Problem

Given n cities, complete graph with edge distances d_{ij} . We describe ACO steps:

1. Initialize

Set pheromone levels $\tau_{ij} = \tau_0 > 0$ for all edges.

2. Construct Ant Tours

For each of m ants:

- Start at a random city i .
- While tour incomplete, choose next city j (not yet visited) with probability

$$P_{i \rightarrow j} = \frac{[\tau_{ij}]^\alpha \left[\frac{1}{d_{ij}} \right]^\beta}{\sum_{k \notin \text{visited}} [\tau_{ik}]^\alpha \left[\frac{1}{d_{ik}} \right]^\beta}.$$

- This builds a complete permutation (tour).

3. Pheromone Update

- **Evaporation:** $\tau_{ij} \leftarrow (1 - \rho) \tau_{ij}$, with $0 < \rho \leq 1$.
- **Deposit:** for each ant k that used edge (i, j) in its tour of length L_k :

$$\tau_{ij} \hat{+} = \frac{Q}{L_k},$$

where Q is a constant.

4. Iteration

Repeat construction and update until convergence or max iterations.

As pheromone concentrates on shorter tours, ants increasingly follow better edges, yielding near-optimal TSP tours.

Q 8. A* search on the given graph (start A , goal J)

We label each node n with $g(n)$ = cheapest known cost from A to n , $h(n)$ =heuristic, $f(n) = g(n) + h(n)$. We maintain OPEN as a min-heap by f , CLOSED as expanded.

Given edge costs and heuristics (red next to nodes):

Step-by-step:

1. Initialize:

OPEN = $\{A\}$ with $g(A) = 0$, $h(A) = 10$, $f(A) = 10$.

CLOSED = \emptyset .

2. Pop A (lowest f). Expand to neighbors B and F :

• B : $g = 0 + 6 = 6$, $h = 8 \implies f = 14$.

• F : $g = 0 + 3 = 3$, $h = 6 \implies f = 9$.

OPEN = $\{F(9), B(14)\}$. CLOSED = $\{A\}$.

3. Pop F ($f = 9$). Expand to G and H (skip A):

• G : $g = 3 + 1 = 4$, $h = 5 \implies f = 9$.

• H : $g = 3 + 7 = 10$, $h = 3 \implies f = 13$.

OPEN = $\{G(9), B(14), H(13)\}$. CLOSED = $\{A, F\}$.

4. Pop G (tie at $f = 9$, lowest g). Expand to C and I :

• C : $g = 4 + 3 = 7$, $h = 7 \implies f = 14$.

• I : $g = 4 + 1 = 5$, $h = 1 \implies f = 6$.

OPEN = $\{I(6), B(14), H(13), C(14)\}$. CLOSED = $\{A, F, G\}$.

5. Pop I ($f = 6$). Expand to H, E, J (skip G):

• H : new $g = 5 + 2 = 7$, $h = 3 \implies f = 10$ (improves over 13).

• E : $g = 5 + 5 = 10$, $h = 3 \implies f = 13$.

• J : $g = 5 + 3 = 8$, $h = 0 \implies f = 8$.

OPEN = $\{J(8), H(10), B(14), C(14), E(13)\}$. CLOSED = $\{A, F, G, I\}$.

6. Pop J ($f = 8$), it is the goal.

Resulting optimal path (via back-pointers):

$A \xrightarrow{3} F \xrightarrow{1} G \xrightarrow{1} I \xrightarrow{3} J$,

total cost $0 + 3 + 1 + 1 + 3 = 8$.

Q 9. Conditions for A* optimality

A* is guaranteed to find an optimal solution if its heuristic $h(n)$ satisfies:

1. Admissibility:

$$\forall n, \quad h(n) \leq h^*(n),$$

where $h^*(n)$ = true cost of the cheapest path from n to a goal.

2. Consistency (Monotonicity):

$$\forall (n \rightarrow n'), \quad h(n) \leq c(n, n') + h(n').$$

Consistency implies admissibility and ensures that the f -value along any path is non-decreasing, so once a node is expanded, its best path cost is final.

Q 10. Best-First Search vs Recursive Best-First Search (RBFS)

Feature	Best-First Search	Recursive Best-First Search (RBFS)
OPEN list	Global priority queue (stores all frontier)	Implicit via recursion (no global)
Memory	Potentially exponential in worst case	Linear in search depth $O(d)$
Node selection	Expand node with best f or h globally	Expand best child recursively, backtrack with threshold
Reopening/Backtracking	Can re-open nodes if heuristic inconsistent	Maintains “alternative cost” limit
Completeness/Optimality	Complete & optimal if A* conditions hold	Also optimal (with admissible heuristic) but uses less memory

RBFS simulates A*'s best-first order using depth-first recursion plus a limit on the best alternative f -value, trading time (re-expansions) for linear memory.