perplexity

Problem Solving by Searching: Comprehensive Examination Answers

1. Criteria for Evaluating Search Strategies

Four principal criteria evaluate search strategies:
Completeness: Ensures the algorithm finds a solution if one exists (e.g., BFS is complete in finite graphs)[1][2].
Optimality: Guarantees the lowest-cost path (e.g., UCS for non-negative edge costs)[3][2].

Time Complexity: Number of nodes generated, often expressed as

$$O(b^d)$$

for branching factor

$$b$$

and depth

$$d$$

[1].

Space Complexity: Memory required, critical for deep searches (e.g., DFS uses

$$O(bm)$$

, where

$$m$$

is maximum depth)[2].

2. Problem Definition and Components

A **problem** in AI is defined by:

1. **Initial State**: Starting configuration (e.g., scrambled 8-puzzle).
2. **Actions**: Operators

$$A(s)$$

applicable in state

$$s$$

(e.g., moving tiles).
3. **Transition Model**: Result of applying action

$$a$$

in state

$$s$$

(e.g., new tile positions)[4] [5].
4. **Goal Test**: Determines if a state is terminal (e.g., correct puzzle arrangement).
5. **Path Cost**: Sum of action costs (e.g., fuel consumption in route planning)[5].

3. Real-World Problem Formulation

Real-world problems involve complex, unstructured scenarios (e.g., urban traffic management). Formulation steps:

1. **Abstraction**: Ignore irrelevant details (e.g., vehicle color in traffic routing).
2. **State Identification**: Define critical variables (e.g., traffic light timings).
3. **Constraint Definition**: Specify legal actions (e.g., speed limits).
   Example: VLSI chip design reduces to component placement and routing subproblems[5].

4. State-Space Search Technique

State-space search systematically explores all possible states using:

- States
  $$S$$
  : Distinct problem configurations
- Actions
  $$A$$
  : Transition operators between states
  - Goal Test: Terminal condition checker
  - Path Cost: Accumulated action costs
  Algorithm: Represented as
$$\langle S, A, T, G, C \rangle$$
, where
$$T$$
is transition model and
$$C$$
is cost function[1].

5. Uniform-Cost Search (UCS) Analysis

Definition: Expands least-cost nodes first using priority queues[3].
Merits:

- Optimal for non-negative edge costs

- Effective in weighted graphs
  Demerits:
- High memory (

$$O(b^{C^*/\epsilon})$$

, where

$$C^*$$

is optimal cost)
- Inefficient for uniform costs compared to BFS

6. Blind Search Definition

Blind (Uninformed) Search explores without domain knowledge:

- No heuristic guidance
- Examples: BFS, DFS, UCS
- Guarantees completeness but often inefficient[2].

7. Uninformed vs Informed Search

Criterion	Uninformed	Informed
Heuristic Use	No	Yes (e.g., Manhattan distance)
Time Complexity	Higher ($O(b^d)$	
)	Lower ($O(b^{d/2})$	
)		
Optimality	Conditionally guaranteed	With admissible heuristics

Example: A* vs BFS in maze solving[2].

8. Hill-Climbing Search and Drawbacks

Algorithm: Local search moving to higher-value neighbors[6].
Drawbacks:

- **Local Maxima**: Stops at suboptimal peaks (e.g., gradient ascent in non-convex functions).
- **Plateaus**: No improvement direction (e.g., flat error surfaces in ML).
- **Ridges**: Oscillates between sideways moves.

9. Hill Climbing as Greedy Search

Greedy Nature: Always selects immediate best neighbor.
Problems:

1. **Local Maxima Trap**: Example: Maximizing

$$f(x) = -x^2$$

starting at

$$x = 1$$

.
2. **Plateau Navigation**: Requires randomness (e.g., simulated annealing).

10. Admissible Heuristic

A heuristic

$$h(n)$$

is **admissible** if it never overestimates true cost to goal (

$$h(n) \leq h^*(n)$$

). Essential for A* optimality[2].

11. A* Algorithm and Example

Algorithm:

```
function A*(start):
    open = PriorityQueue(start)
    while not open.empty():
        node = open.pop()
        if node == goal: return path
        for neighbor in expand(node):
            f = g(node) + h(neighbor)
            if f < existing_cost(neighbor):
                update open with neighbor
```

Example: 8-puzzle with Manhattan distance heuristic reduces node expansions by 70%[1].

12. A* for Minimal Cost Path

Combines UCS (exact

$$g(n)$$

) and greedy search (heuristic

$$h(n)$$

). Prioritizes nodes with minimal

$$f(n) = g(n) + h(n)$$

, ensuring optimal path discovery[2].

13. A* Benefits Over UCS and Greedy

- **Optimality**: Achieves UCS's optimality with heuristic speed.
- **Efficiency**: Expands fewer nodes than UCS (

$$h(n)$$

guidance).
- **Completeness**: Guaranteed if heuristic is admissible[2].

14. Heuristic Search and A* Optimality

Heuristic Search uses domain knowledge (e.g.,

$$h(n)$$

) to guide exploration.
Optimality Proof:

1. Assume suboptimal goal

$$G_2$$

is generated before optimal

$$G_1$$

.
2. Let

$$n$$

be unexpanded node on optimal path to

$$G_1$$

.
3.

$$f(n) = g(n) + h(n) \leq g(G_1)$$

(admissibility).

4. Thus,

$$f(n) \leq f(G_1) < f(G_2)$$

, so

$$G_2$$

wouldn't be selected first. Contradiction[2].

15. Heuristic Functions in CSPs

Heuristic: Guides variable/value selection in constraint satisfaction.

- **Minimum Remaining Values (MRV)**: Chooses variable with fewest legal values.
- **Least Constraining Value (LCV)**: Maximizes future flexibility.
  Example: Map coloring prioritizes regions with most adjacent conflicts[5].

16. Depth-First Search (DFS)

Algorithm:

```
procedure DFS(node):
    if node is goal: return path
    mark visited
    for neighbor in node.children:
        if not visited:
            result = DFS(neighbor)
            if result: return result
    return null
```

Example: Maze solving using backtracking (e.g., left-hand rule)[2].

17. Best-First Search Evaluation

1. **Completeness**: No (may ignore promising paths).
2. **Optimality**: No (depends on heuristic quality).
3. **Time**:

   $$O(b^m)$$

   (worst-case).
4. **Space**:

   $$O(b^m)$$

   (stores entire frontier)[2].

18. DFS vs BFS Examples

DFS: Explores depth-first using stacks. *Example*: Solving n-queens via backtracking.
BFS: Level-order traversal using queues. *Example*: Shortest path in unweighted graphs[2].

19. BFS vs DFS Differences

Aspect	BFS	DFS
Data Structure	Queue	Stack
Optimality	Yes (unweighted)	No
Space Complexity	$O(b^d)$	
	$O(bm)$	

20. Advantages of BFS and DFS

BFS:

- Guarantees shortest path
- Complete in finite spaces
  DFS:
- Low memory (linear in depth)
- Faster for deep solutions

21. DFS vs Depth-Limited Search (DLS)

DLS imposes a depth cutoff

$$l$$

:

- Prevents infinite loops (e.g.,

  $$l = 10$$

  for game trees).
- Incomplete if solution depth >

  $$l$$

  [2].

22. Informed Search vs DFS

Informed Search uses heuristics (e.g., A∗), while DFS is uninformed. *Difference*: DFS blindly explores depth; informed methods prioritize promising nodes[2].

23. Iterative Deepening Search (IDS)

Combines BFS completeness with DFS memory efficiency:

1. Perform DFS with depth limit
$$l = 0, 1, \dots$$
2. Repeats search incrementally.
   Example: Chess AI evaluates moves to increasing depths[2].

24. IDS vs DFS Computational Cost

IDS Time:
$$O(b^d)$$
(repeats levels).
DFS Space:
$$O(bd)$$
vs IDS
$$O(d)$$
.
Trade-off: IDS sacrifices time for BFS-like completeness[2].

25. IDS vs Depth-Limited Search (DLS)

IDS: Gradually increases depth limit.
DLS: Fixed cutoff.
Key Difference: IDS is complete; DLS requires prior depth knowledge[2].

26. Genetic Algorithm (GA)

Stochastic optimization inspired by evolution:

1. **Population**: Candidate solutions.
2. **Selection**: Fitness-based reproduction.
3. **Crossover**: Combine parent traits.
4. **Mutation**: Introduce diversity.
   Application: Neural network hyperparameter tuning[1].

27. GA Flowchart

```
[Start] → Initialize Population → Evaluate Fitness → [Selection → Crossover → Mutatio
```

28. GA Operators

1. **Selection**: Tournament selection chooses top candidates.
2. **Crossover**: Single-point crossover merges parent chromosomes.
3. **Mutation**: Bit-flip introduces randomness[1].

29. Bidirectional Search

Searches from start and goal simultaneously:

- **Avoiding Repeats in DFS**: Track visited nodes in both directions.
- *Example*: Social network connection finding[2].

30. Constraint Satisfaction Problem (CSP)

Definition:

$$\langle X, D, C \rangle$$

where variables

$$X$$

have domains

$$D$$

under constraints

$$C$$

.
Example: Sudoku with cell variables, digit domains, and row/column/box constraints[5].

31. Contingency vs Exploration Problems

Contingency: Uncertain outcomes (e.g., poker with hidden cards).
Exploration: Unknown state space (e.g., robot mapping)[2].

32. Problem Types

- **Single-State**: Fully observable (e.g., 8-puzzle).
- **Multi-State**: Partial observability (e.g., poker).
- **Contingency**: Requires action-response pairs.

- **Exploration**: Active information gathering.

33. Bidirectional Search Strategy

Strategy: Concurrent forward/backward searches meeting midway.
Example: Route planning from both origin and destination cities[2].

⁂

1. https://www.scaler.com/topics/artificial-intelligence-tutorial/state-space-search-in-artificial-intelligence/
2. https://www.upgrad.com/blog/difference-between-informed-and-uninformed-search/
3. https://www.appliedaicourse.com/blog/uniform-cost-search/
4. https://aalimec.ac.in/wp-content/uploads/Material/cse/2/AI&ML NOTES watermark.pdf
5. https://www.appliedaicourse.com/blog/problem-formulation-in-ai/
6. https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/hill-climbing-algorithm-in-ai