
Name: ANISH SUJIT PATIL
Class: TY IT A
Roll No: 331011
Batch: A1
Title: *Comparative Study of DFS and Monte Carlo Tree Search (MCTS)*

Aim:

To compare the performance and computational characteristics of Depth-First Search (DFS) and Monte Carlo Tree Search (MCTS), and understand their respective algorithmic structures, practical applications, and suitability for solving different types of problems in both deterministic and uncertain environments.

Theory:

Depth-First Search (DFS):

DFS is one of the fundamental algorithms in computer science, used to explore nodes and edges of a graph. It explores a branch of the tree or graph as deeply as possible before backtracking. This method is especially effective for exhaustive search in puzzles, scheduling, and hierarchical structures.

- **Traversal Order:** Depth-wise; goes down one path until it can't, then backtracks.
- **Nature:** Deterministic and uninformed (doesn't use goal information).
- **Memory Use:** Requires minimal memory compared to other search algorithms like BFS.
- **Limitations:** DFS may fail to find the shortest path and may enter infinite loops if cycles exist and aren't handled properly.

Monte Carlo Tree Search (MCTS):

MCTS is a powerful decision-making algorithm that became widely known after its successful application in game-playing AIs such as AlphaGo. It builds a search tree using random simulations and updates the results using statistical feedback to make better decisions over time.

- **Traversal Strategy:** Alternates between exploring new nodes and exploiting known good paths.

- **Nature:** Probabilistic, adaptive, and heuristic-driven.
- **Components:** Includes four steps — Selection, Expansion, Simulation (Rollout), and Backpropagation.
- **Strengths:** Effective in situations where the search space is too large to explore exhaustively, or where the environment is uncertain.

Example & Algorithm:

DFS Algorithm (Python):

```
def DFS(node, goal, visited):
    if node == goal:
        return True
    visited.add(node)
    for neighbor in node.neighbors:
        if neighbor not in visited:
            if DFS(neighbor, goal, visited):
                return True
    return False
```

This implementation uses recursion to go deep into the node tree or graph. The visited set is essential to prevent cycles and ensure termination.

MCTS Algorithm (Pseudocode):

```
function MCTS(root):
    for iteration in range(N):
        node = Selection(root)
        child = Expansion(node)
        result = Simulation(child)
        Backpropagation(child, result)
    return best_child(root)
```

- **Selection:** Traverse tree based on UCT (Upper Confidence Bound).
- **Expansion:** Add child nodes for unexplored actions.
- **Simulation:** Perform a random playout.
- **Backpropagation:** Update parent nodes with simulation results.

Complexity Analysis:

Algorithm	Time Complexity	Space Complexity	Notes
DFS	O(V + E)	O(V)	Efficient for structured graphs and trees

MCTS	O(N × D)	O(N)	N = simulations, D = average depth of tree

---

Applications:

DFS Applications:

- **Maze Solving and Puzzle Games:** Explores paths deeply to find a valid solution.
- **Topological Sorting:** Scheduling tasks in order using DFS-based sort.
- **Pathfinding:** Basic algorithm to explore all nodes and paths in trees.
- **Cycle Detection:** Used in graph theory to find loops.

MCTS Applications:

- **Game AI:** Used in complex games like Go, Chess, and general board games.
- **Decision-Making in Robotics:** Helps simulate multiple outcomes.
- **Planning under Uncertainty:** Useful in environments like poker or simulations.
- **Real-Time Strategy (RTS) Games:** Provides fast decision-making in large trees.

Comparative Table:

Criteria	DFS	MCTS
Search Strategy	Depth-first traversal	Random simulation + statistical updates
Determinism	Deterministic	Probabilistic
Memory Usage	Low (linear in depth)	Moderate (grows with simulations)
Optimality	Not guaranteed	Improves with iterations; approximates optimal
Environment Suitability	Deterministic, structured	Complex, dynamic, and uncertain
Implementation	Simple, recursive	Complex (requires tuning and simulation control)
Scalability	Limited in large graphs	Scales well with computational power

---

Conclusion:

DFS and MCTS each have unique strengths and are best suited to different types of problems:

- **DFS** is ideal for problems with a fixed structure and clear depth, especially when any valid solution is sufficient. Its simplicity and low memory usage make it an effective algorithm for many classical graph-based problems.
- **MCTS** shines in situations involving uncertainty, large search spaces, or environments where exploration and adaptation are needed over time. It balances exploration and exploitation using statistical results to make intelligent decisions, especially in the field of game AI.

Both algorithms have become essential tools in the domain of Artificial Intelligence, Game Theory, and Computer Science research. Choosing the right algorithm depends on the nature of the problem and constraints such as computation time and environmental predictability.
