Search

Introduction

A search problem can be described as searching for a particular path along a graph.

Key concepts:

State

> A specific configuration of a problem (e.g. chess board while playing).

State space

> All possible states of a problem/game (e.g. all possible chess boards configurations).
> They are represented as directed graphs. A line in this graph would be a single change (action) in a state to produce another one. Multiple changes will be represented by any trajectory in the graph (several lines = chain of actions).

Action/Operators

> Generate a successor of a state. Given a state, there are a finite number of possible actions to generate a successor.

Initial State and Goal State

> In these types of problems we want to get from a initial state (e.g. the start of a chess game) to a goal state in which the problem is solved (e.g. one of the players wins the chess game)

Solution

> A trajectory in the graph of the state space. It begins at the initial state and ends in the goal state. This trajectory usually is compost of actions (unless it requires only one/none change to complete the problem).

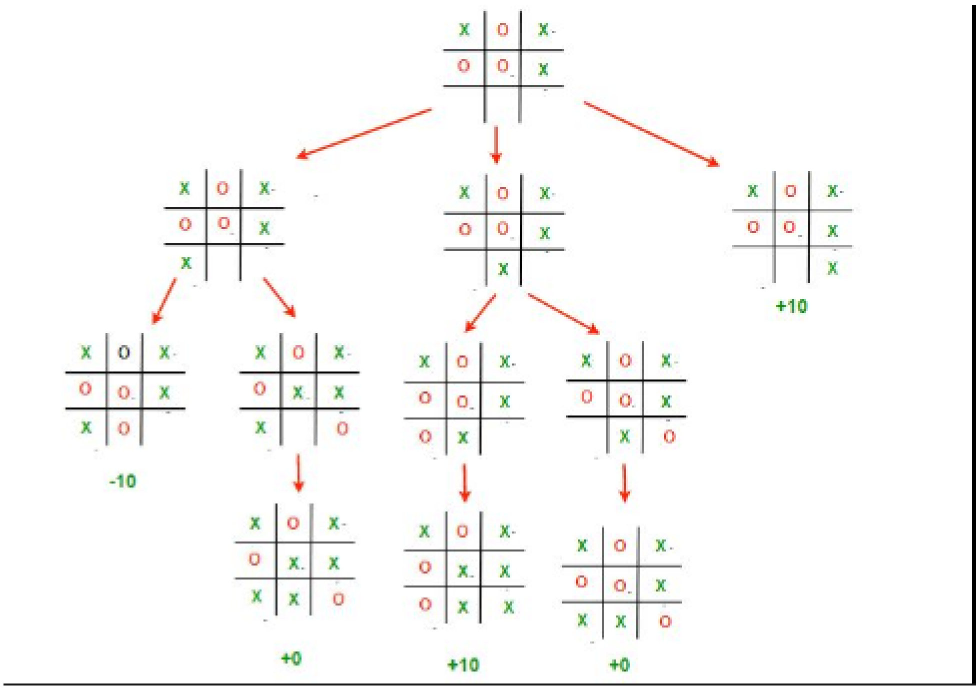
Problem Instance

> The possible game states given an initial and goal states (state-space + initial and goal)

Game Three

> A graph representing all possible game states of a game. The games represented are usually sequential and multiplayer (*por turnos*), for example chess, checkers (*damas*) or tic-tac-toe (*tres en raya*).

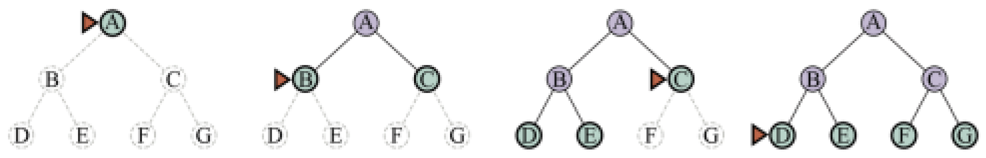
Game three in tic-tac-toe:



Uninformed/Blind Search

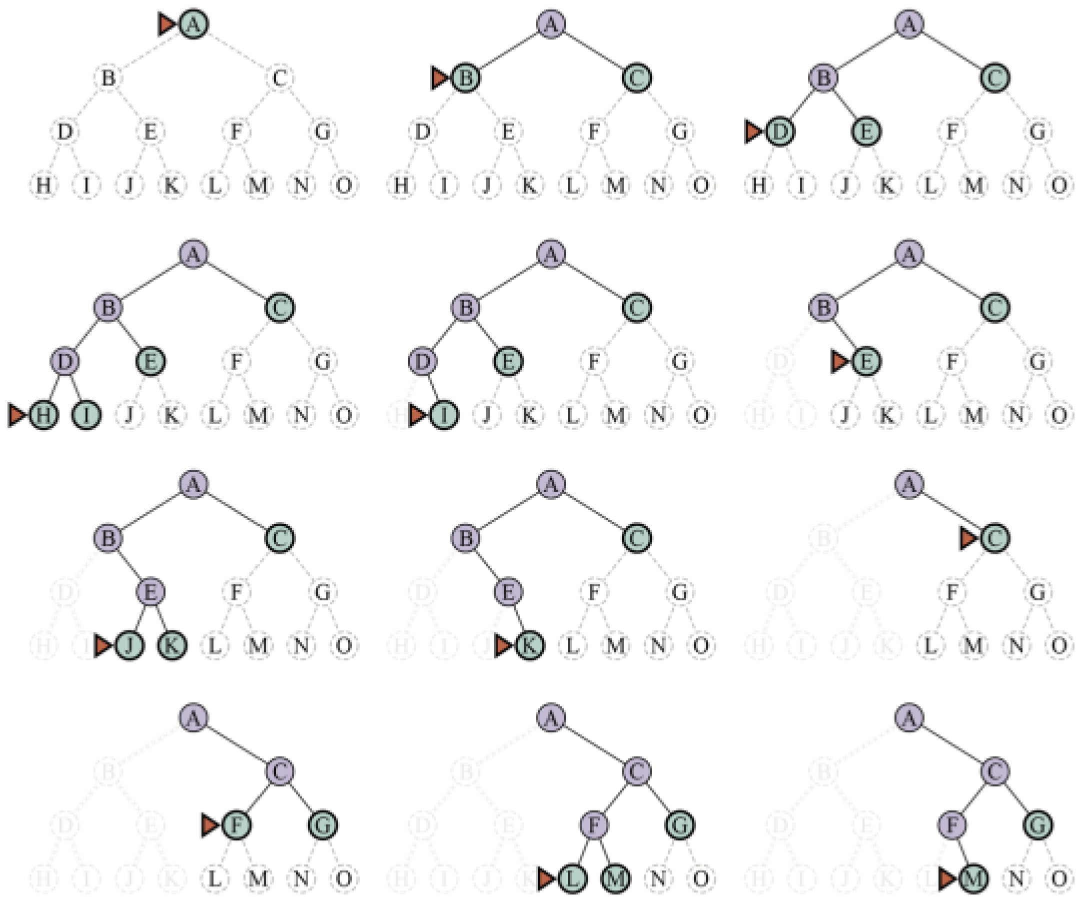
Access only to the problem definition.

Breadth-first search (BFS) #important

> Expand first the shallowest (unexpanded) node first
>
> 
>
>
> Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

Depth-first search (DFS) #important

> Expand first the deepest (unexpanded) node first. Can add a depth limit (Depth-limited search) for large space graphs.



A dozen steps (left to right, top to bottom) in the progress of a depth-first search on a binary tree from start state A to goal M. The frontier is in green, with a triangle marking the node to be expanded next. Previously expanded nodes are lavender, and potential future nodes have faint dashed lines. Expanded nodes with no descendants in the frontier (very faint lines) can be discarded.

Uniform-cost search

> Expand the node with lowest path cost.

Iterative deepening search

> Depth-first with deepening depth limits (it goes deeper over time)

Bidirectionial search

> Two searches, one from the initial state and other one from the goal state. If they meet, there is a complete path.

Informed Search

Use of a Heuristic function $h(n)$ gives hints about the location of the goal.
Heuristic Function

> $h(n) =$ estimated cost of the cheapest path form the state at node n to a goal state

Examples:

- Greedy best-first search: Expands with minimal $h(n)$
- A-star search: Expands with minimal $f(n) = g(n) + h(n)$
- Variants to A-star (Bidirectional A*, IDA*, SMA*, Weighted A*)

Offline Search

Offline algorithms are designed to process an entire dataset at once. The whole data related to the problem is given the algorithm from the beginning.

The search of the solution and the execution of said solution are completely separated stages of the problem-solving. Thus, there are two separate phases:

1. Solution computation (search)
2. Solution execution (action) (e.g. doing a move on a game)
   Being two separated phases, the solution execution does not affect the search (e.g. there is no trial and error), this does not happen in the real world.

In other words, they compute a complete solution before taking their first action.

Offline systematic search

The combinational explosion was the main cause of the winter of AI. The algorithms that search blinded these spaces are called weak. Only work really well in Toy Problems (small "invented" problems).

Systematic Search

> Can **eventually** reach any state that has a connection to the initial state.

Previous algorithms that we have seen that are in this category:

- A-Star and its variants
- Blind Search (BFS, DFS, + more)

This type of search is mostly done in controlled environments where for example the game state and the rules are known (e.g. chess game). For more complex environment (many real world problem), we have more algorithms. An important category of these is Local Search.

Local Search

These algorithms are often inspired by natural processes:

- Simulated annealing
- Genetic Algorithms

  > They are parallel: work in several solutions at a time.

- Ant Colony Optimization (ACO)

  > Pheromone trails, multi agents that search different paths

Used specially for optimization problems.
The main idea is that they do not keep track of previous visited states. They are also greedy, they look for immediate improvement, searching from a start state to neighboring states. This means that they are not systematic (they might never explore a portion of the search space where the solution actually resides). The categories of local and systematic search are mutually exclusive.

Key advantages:

- small memory usage (they don't need to keep track)
- find reasonable solutions in large/infinite state spaces

To guide them through the search they use a heuristic function (part of the Informed Search category). Thus, being heuristic algorithms, they do not guarantee that the path in the search it is optimal, because they work by minimizing a cost.

Hill climbing

The simplest local search method. Is and optimization algorithm. It starts with an arbitrary solution and tries to find a better one by making incremental changes.
Maximizes or minimizes a cost function $\mathcal{L}(\mathbf{x})$ (function that determines how close it is to the solution/how well the optimization is going).
A small change is made in a single parameter x_i (other algorithms would do a change to all parameters) and a criteria (there are multiple options) determinates whether the change improves $\mathcal{L}(\mathbf{x})$, if true, that change is accepted and made permanent.
If the change makes the optimization worse, then another one is tried, until a good one is found.
If no change made can improve the value of $\mathcal{L}(\mathbf{x})$, then the search is done and $\mathbf{x}$

is the "locally optimal" solution.
This solution can be a local optima but not the global, there could be a more optimal point. If we are trapped in this local optima we can escape:

- Accept steps into higher function values
- Random-start hill climbing

A change in a parameter usually is called a step.

Discrete Hill Climbing:

> If the algorithm searches a discrete then the parameters can be represented as vertex in graphs (there are no values between one possible value of x and another one). Note that the graph is the state space and a vertex is an state.

Continuous Hill Climbing:

> If the space is continuous then we can use a `step_size` parameter that determines how much faster or slower we change the parameters. That is, how bigger or smaller we make the parameter when we change it, the magnitude of the step size. Usually involve the use of a gradient (partial derivates).

Genetic Algorithms

Based on natural selection. The two main components are:

- Solutions in the state space that can be genetically generated
- Cost function to evaluate how good the candidates are in solving the problem
  These are parallel search algorithms, they combine exploration and exploitation.

To "evolve" these solution/candidates we use operators. They take a generation t (all the candidates) to the next generation $t+1$.

There are 3 operators that are used to do this (used in sequence):

1. Selection: select the best chromosomes (defined by a threshold)
2. Crossover/Recombination: stochastically (randomly) generate new solutions
3. Mutation: changing the bits in the solution (probabilistically) to have greater diversity in the solutions
   If this process is conducted many times, it is expected that we arrive at an optimal solution.

There are several variables that can be changed (mutation rate, mixing number, initial number of individuals, culling...). See 4.1.4 in the 4th edition.

Online Search

In online search the solution creation and its execution are interleaved. An agent interleaves computation and action: first it takes an action, then it observes the environment and computes the next action.

The **domains** of these algorithms usually have unknown terrain and are dynamic (see types of environments) (e.g. disaster rescue, navigation on Mars or The Moon).

The simplest online search algorithms are single agents, but these algorithms mainly are multi-agent. These are two different families of algorithms.

Find a complete solution and executed. Then if possible found other complete solution based on the current state. Iterate until a "very optimal" solution is found.

Single agent approaches

There are two main types:

- Incremental search
- Real-time search
  The difference is the available time to make the first move. If it needs to happen quickly then real-time search is used (e.g. competitive gaming).

Incremental Search

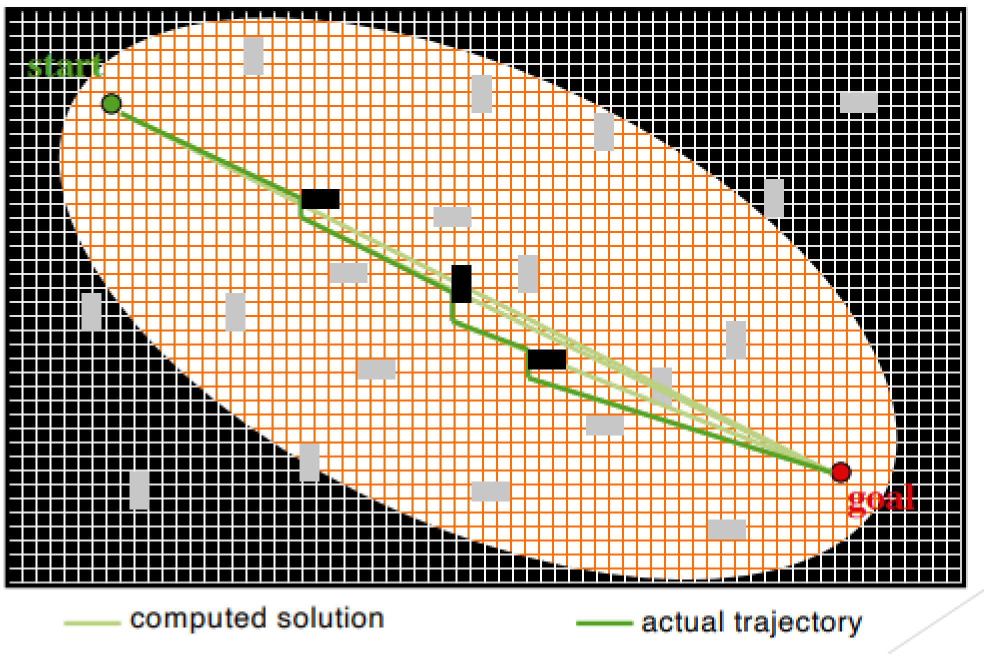
Method:

1. Search for a complete solution from the start state and execute the solution
2. If unfeasible, search for a new complete solution from the current state and execute the solution
3. Repeat (1) and (2) until finding a goal
   This is a try and repeat method that works specially well for unknown environments.

Example Incremental D-star

Path-finding in unknown terrain
Draw a straight line and if obstacles are in the way we change the route to go

around them.



Real Time Heuristic Search

If the problem must be handled in real time, the algorithm searches the local state around the current state.
Method:

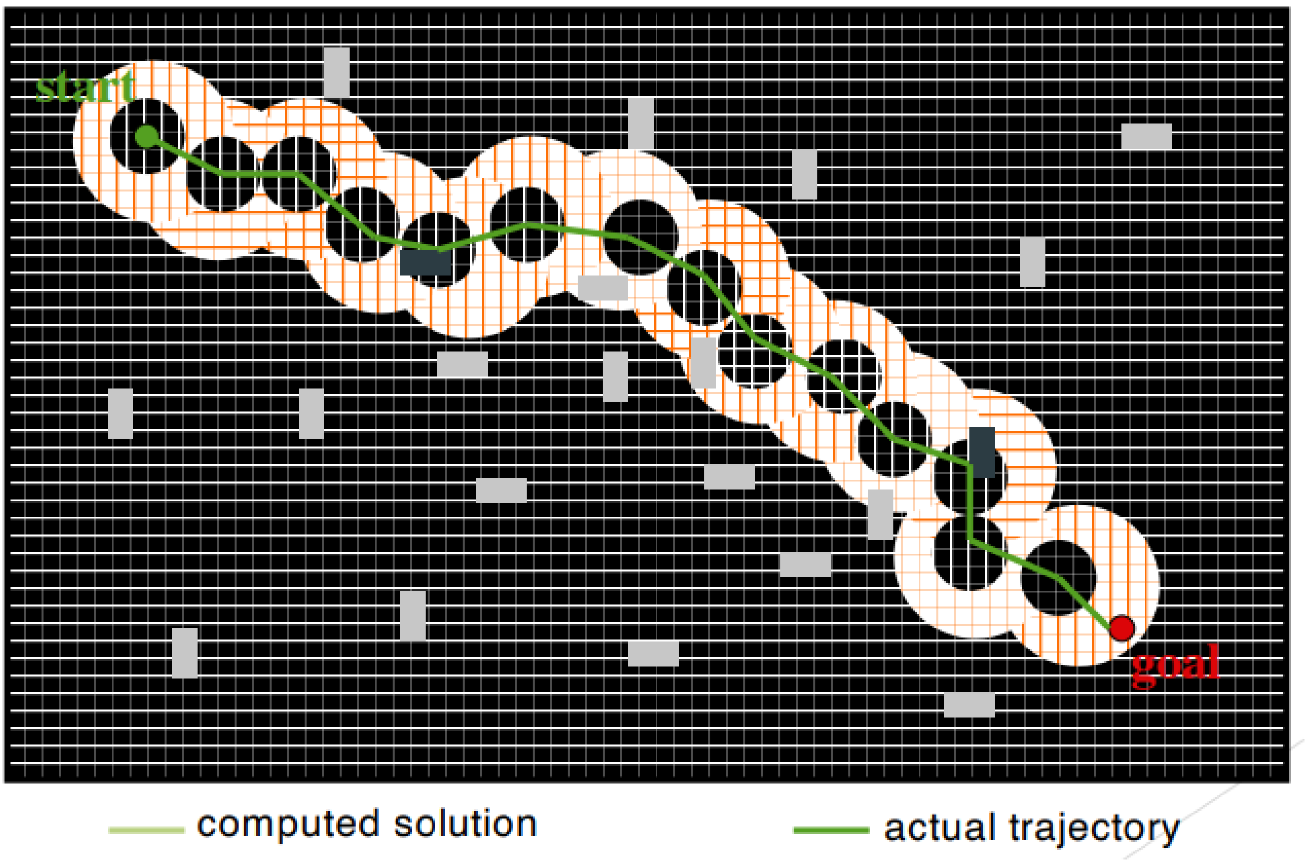
1. Looks for adjacent states (in local space) to find most optimal (i.e. the closest to solution)
2. Moves to the optimal neighboring state
3. Repeat (1) and (2) until convergence to optimality

The solution is not optimal, this approach converges to optimality after repeated iterations of the search and the execution, the path/solution is a valid one but it might not be the optimal solution.
The search and execution of the solution are interleaved (step 3).

Example Real-Time LRTA-star

Update the heuristic estimation of the state, based on the best available state in the local space:

1. Calculate cost of all adjacent states
2. Find the best one evaluating the minimal sum
3. Update the heuristic value of the current state
4. Move to the successor with the updated heuristic



See more at https://www.scitepress.org/papers/2013/44494/44494.pdf

Multi Agent Approaches

Two agents (zero sum game)

These algorithms form part of a category called Adversarial Search.

Assumptions:

- Two players
- Sequential game
- Perfect Information
    - known environment and rules
    - no random elements
- Zero sum game: if the "utility" for A is $x \Rightarrow$ for B is $-x$

The main idea behind these algorithms is the game tree, the representation of the game being played. The different possible moves for each player are represented in turns. If the game tree exponentially explored, then the tree is expanded to a specific level (depth). Can work for Tic-Tac-Toe, but not for Chess.

Minmax

Search and back propagate the outcome to update the algorithm.
When evaluating the next move for a player, the algorithm will minimize the opponent's maximum payoff, minimize their chances of winning. This is done by an evaluation function.

The nodes (game states) in the game tree can be classificated into 3 categories with different values from the perspective of a player (player A):

- value 1 (A wins)
- value -1 (B wins)
- value 0 (draw)
  Then the algorithm (playing for A) will maximize wins choosing the nodes with value 1 and minimize wins for B, not choosing the nodes with -1

An evaluation of this values is done to certain depth. The best move is done by picking a node terminal (a node at maximum depth) and back-propagating the values. The backpropagation is done to take into account the move that the other player will do (the most favorable move $= -1$).

We cannot plan to make a move that first requires the other player to make a bad move for them, we have to assume that they will do the best possible move for them to win the game.

More than two agents

Algorithms that we already have seen:

- Ant Colony Optimization (ACO)
- Genetic Algorithms
  These are **not** adversarial search algorithms.

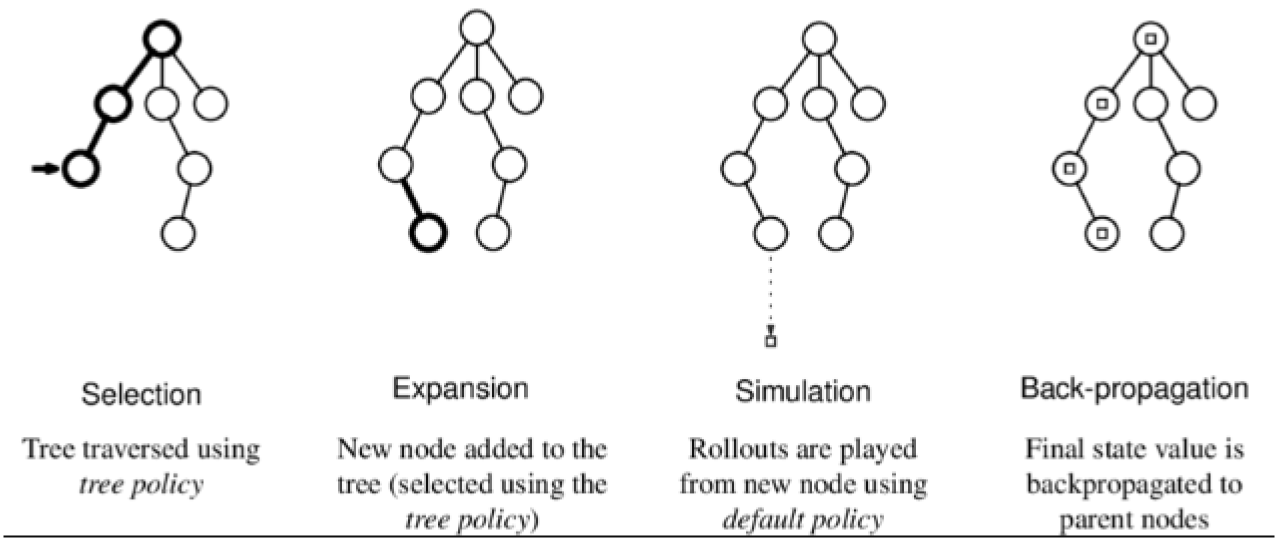
For more complex games, (GO) the branching in the tree is too large to compute in a reasonable time. Furthermore, there are no good enough evaluation functions to explore this large game tree doing optimization (e.g. Hill-Climbing). Thus, Monte Carlo methods were born.

Monte Carlo Tree Search (MCTS)

Main idea:

> Evolve the main game tree close to the current state, creating a partial state.
> Choose one particular node further from the tree if the game does not end,

> simulate further into the game tree from that specific node using random methods (e.g. uniform random moves) until the game ends.
> The result of the game is then backpropagated to update the nodes in the partial tree.


Selection
Tree traversed using *tree policy*
Expansion
New node added to the tree (selected using the *tree policy*)
Simulation
Rollouts are played from new node using *default policy*
Back-propagation
Final state value is backpropagated to parent nodes

This algorithm is an Stochastic Algorithms: an integral part of it is a random distribution.

Important Names:

- Nils Nilsson
- Judea Pearl
- Richard Korf
- Sven Koenig