**Uninformed Search**

Problem Solving as Search

• Search is needed to solve many real-world problem

8-puzzle problem Planning Go

Diagram

Description automatically generated

• Search is a central topic in AI

— Automated reasoning

— More recently: Given that almost all AI formalisms (planning, learning, etc.) are **NP-complete** or worse, some form of search is generally unavoidable (no “smarter” algorithm available).

Define a Search Problem

• A **search problem** consists of:

• **State space**: A picture containing text

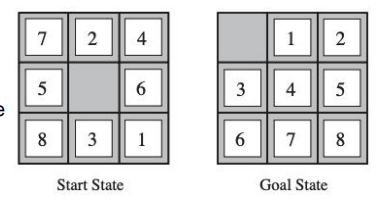
Description automatically generated

• A **successor function**: (action + cost)

• A **start state** and a **goal test**:

• A **path** is any sequence of states connected by a sequence of actions.

• **Path cost** – function that assigns a cost to a path; relevant if more than one path leads to the goal, and we want the shortest path.

• A **solution** is a sequence of actions (a plan) which transforms the **start state** to a **goal state**.

Example: The 8-Puzzle

• **State space**: • The location of each tile

• **Initial state**: • Any state can be the inital state

• **Goal test**: • Whether the state matches the goal state

• **Successor function**: • The movement of the blank space (Left, Right, Up or Down)

• **Path cost**: • Each step costs

A picture containing table

Description automatically generatedExample: Cryptarithmetic

• Find **substitution of digits** **for letters** such that the resulting sum is arithmetically correct.

• Each letter must stand for a different digit.

• **State space**: an 8-tuple indicating a (partial) assignment of digits to letters.

• **Goal test**: all letters have been assigned digits and sum is correct

Diagram, radar chart

Description automatically generated• **Successo**r function: represents the act of assigning digits to letters

• **Path cost**: all solutions are equally valid; step cost = 0

Example: Traveling in Romania

**• State space**: • Cities

• **Initial state**: • Arad

• **Goal test**: • Is state == Bucharest

• **Successor function**: • Roads: go to adjacent cities with cost = distance

• **Path cost**: • The cost of a path

Solving a Search Problem: State Space Search

• **Input:** • Initial state • Goal test • Successor function • Path cost function

• **Output:** path from initial state to goal. Solution quality is measured by the path cost function

• Expanding: apply each legal action to the current state

Diagram

Description automatically generated• The leaf nodes available for expansion is called the frontier

A picture containing logo

Description automatically generatedSearch procedure defines a search **tree**

**root node** — initial state

**Children of a node** — successor states

**Leaves of tree (frontier)** — states not yet expanded

**Search strategy** — algorithm for deciding which leaf node to expand next

Node Data Structure

• Node data structure is used to **keep track of** the search

Diagram

Description automatically generated• Nodes vs. States

• A node is a **bookkeeping data structure** used to represent the search tree

• A state corresponds to a **configuration** of the world

Evaluating a Search Strategy

• Completeness: is the strategy guaranteed to find a solution when there is one?

• Time complexity: how long does it take to find a solution?

• Space complexity: how much memory does it need?

• Optimality: does the strategy find the highest-quality solution when there are several different solutions?

Generic Tree-Search Algorithm

Add **initial state** to the **frontier**

Loop

**node** = remove-frontier( ) -- and save in order to return as part of path to **goal**

if goal-test(**node**) = true return path to goal

S = successors(**node**)

Add **S** to **frontier**

until **frontie**r is empty

return **failure**

A picture containing diagram

Description automatically generatedTree Search vs. Graph Search

• **Tree search** allows a state to be expanded more than once

• Failure to detect repeated states can cause more work

• Advantage: memory-efficient

Graph Search

•Idea: never expand a state twice

• How to implement:

• Tree search + set of expanded states (“closed set”)

• Expand the search tree node-by-node, but…

• Before expanding a node, check to make sure its state has never been expanded before

• If not new, skip it, if new add to closed set

Uninformed Search: BFS

• Use the **first-in, first out or FIFO** queue to store the frontier

Diagram

Description automatically generated• Consider paths of length 1, then of length 2, then of length 3, then of length 4, .

Time and Memory Requirements for BFS

• Let **b = branching factor** -> maximum number of successors of any node

• **d = solution depth** -> the shallowest goal node

• Then the maximum number of nodes generated is: b + b2+ ... + bd = O(bd)

• For graph search, O(bd-1) in the closed set and O(bd) in the frontier

• **branching factor = 10** • 1 million nodes / second • each node requires 1000 bytes of storage

Table

Description automatically generated

Uniform-Cost Search

• **Use BFS**, but always expand **the lowest-cost node** on the frontier as measured by path cost g(n)

• **g(Successor(n)) > g(n)** is a **necessary condition** for completeness and a **sufficient condition** for optimality

Uninformed search: DFS

• Use the last-in, first out or LIFO queue to store the frontier

Diagram

Description automatically generated

Time and Memory Requirements for DFS

• Let **b = branching factor** -> maximum number of successors of any node

Chart, radar chart

Description automatically generated• **m = maximum depth of any node**

• Time: for tree search, then the maximum number of nodes generated is: **O(bm)**

• Space: for tree search, only need to store **O(bm)** nodes

• at depth l < d we have b-1 nodes

• at depth d we have b nodes

• total = (m-1)\*(b-1) + b = O(bm)

DFS vs. BFS

Table

Description automatically generated

**d**: depth of the shallowest solution **m**: maximum depth

• Takeaways:

• If the solution is not far from the root: BFS might be faster

• If the search tree is very deep: DFS may never find solution

• If the search tree is wide: BFS might take too much memory

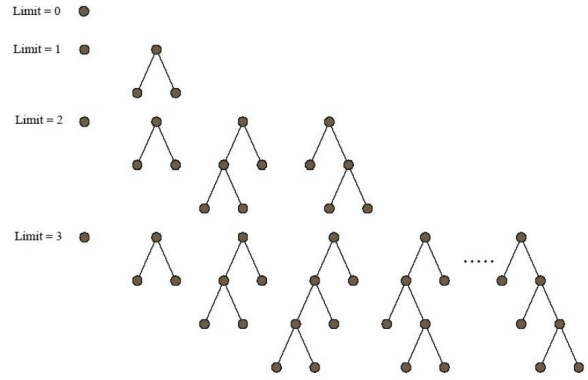
• If d is known: DFS is preferred

• If optimal solution is needed: BFS is preferred

Iterative Deepening [Korf 1985]

• Problem of DFS: cannot avoid infinite loops

• Idea: Use an artificial depth cutoff, **c**. If search to depth **c** succeeds, we are done. if not, increase **c** by 1 and start over. Each iteration searches using DFS. Combine the benefit of the DFS and BFS



Iterative Deepening

• Space requirements: same as DFS.

• Time requirements: would seem very expensive! But not much different from single BFS or DFS to depth d

• Reason: **Almost all work is in the final couple of layers**. E.g., binary tree: 1/2 of the nodes are in the bottom layer. With b = 10, 9/10th of the nodes in the final layer!

Table

Description automatically generated• So, repeated runs are on much smaller trees (i.e., expoentially smaller)

Examples: b = 10, d = 5, the number of nodes generated in a BFS:

b + b2 + ... + bd = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110

For **IDS**: (d)b + (d-1)b2 + ... + (1)bd = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450

Cost of repeating the work at shallow depths is not prohibitive

Cost of Iterative Deepening

• Space: O(bd) as in DFS, time: O(bd)

• Asymptotic ratio of the number of nodes generated by BFS and IDS: • (b+1)/(b-1)

Bidirectional Search

A group of trees

Description automatically generated with low confidence

• When is bidirectional search applicable?

• Generate predecessors is easy • Goal state is clearly specified. (“no queen attacks another queen”)

• **Search forward from the start** state and **backward from the goal** state simultaneously and **stop when the two searches meet the middle**.

• If **branching factor = b** from both directions, and solution exists at **depth d**,

then need only O(2bd/2) = O(bd/2) steps.

• Example: b = 10, d = 6 then BFS needs 1,111,110 nodes and bidirectional search needs only 2,220.

Limitations

• What are the problems of all the methods? **Slow**!

• The search is blind in the sense that the information of the goal state is not used

• Informed search: • with the guidance of the goal state.

**Informed Search**

Define a Search Problem

• A **search problem** consists of:

• **State space**: A picture containing text

Description automatically generated

• A **successor function**: (action + cost)

• A **start state** and a **goal test**:

**b: branching factor** **d: depth of the shallowest solution** **m: maximum depth**

Table

Description automatically generated

Informed Methods: Heuristic Search

• Informed methods use problem-specific knowledge

**• The location of the goal**

• Humans rely on informed search!

• We want to have some **estimate** of the distance from **the states** **in the frontier** to **the goal state**.

• Why estimate? Because the states we can reach are based on the actions we can take.

• We use an **evaluation** function f(n) as our **estimate**

• Best-first search:

**• Nodes are selected for expansion** based on the evaluation function, f(n).

• Expand the node with the **lowest** evaluation

• The evaluation function can be complex: f(n) = f1 (n) + f2 (n) + …

Greedy Best-First Search

•One natural component of f(n):

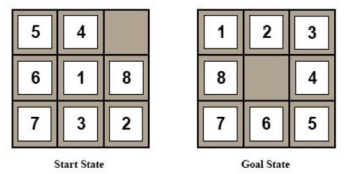
• Heuristic function:

• h(n) = **estimated cost** of the **cheapest** path from the state at **node n** to a **goal state**

Heuristic search is an attempt to search the most promising paths first.

• Greedy Best-First Search

• Expands the node that is “closest” to the goal as measured by h(n)

• A common case:

• Best-first takes you straight to the (wrong) goal

• Worst-case: like a badly guided DFS.

Example: 8-puzzle problem

• Design of heuristic function is important

• One possible heuristic function: • The number of tiles misplaced

Example: Find a Path from Arad to Bucharest

• One possible heuristic function:

• the straight-line distance to Bucharest

Diagram

Description automatically generatedChart, radar chart

Description automatically generated

Example: Find a Path from Arad to Bucharest

• One possible heuristic function: • the **straight-line distance** to Bucharest

• Expand the node **seems “closest”**.

Greedy Best-First Search Can be Suboptimal

• From Arad to Sibiu to Fagaras -- but to Rimnicu would have been better

Diagram, radar chart

Description automatically generated

• What is missing? • **The cost of getting from the start node (Arad) to intermediate nodes**.

Greedy Best-First Search is Incomplete

• Start state: Iasi • Goal state: Fagaras

Diagram, radar chart

Description automatically generated

A\* Search

• Proposed in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael

• Most widely known form of Best-First Search.

• Combining **Uniform-Cost Search** **and Greedy Best-First Search** .

• f(n) = g(n) + h(n)

• g(n): the path cost from the start node to node n

• h(n): the estimated cost of the cheapest path from node n to the goal node

• When h(n) satisfies certain properties, A\* is both complete and optimal!

Diagram

Description automatically generated

• **Uniform-cost** orders by path cost g(n)

Diagram

Description automatically generated

• **Greedy best first search** orders by estimated goal proximity h(n).

Diagram

Description automatically generatedDiagram

Description automatically generated

• **A\* search combines** g(n) and h(n)

When to terminate in A\* Search

• Similar to Uniform Cost Search, the goal test is applied to a node when it is selected for **expansion**,

• The path cost to the goal state may get **update**.

A picture containing text, clock, watch

Description automatically generatedIs A\* Search Optimal?

• What went wrong?

• Actual goal cost < estimated goal cost

• Solution?

Need Some Conditions

• To guarantee that A\* finds an **optimal solution**, we need that h(n) **never overestimates** the cost of reaching the goal

• Called an **admissible** (có thể chấp nhận được) heuristic

• Transfer to f, i.e., f also doesn't **overestimate**.

Formal Definition of Admissibility

• Let h\*(n) be **the actual cost** to reach a goal from n.

•A heuristic function h is **optimistic** or **admissible** if 0 ≤ h(n) ≤ ℎ\*(n) all nodes n.

•If h is **admissible**, then the A\* algorithm will never return a sub-optimal goal node

Example: Admissible Heuristic

• Path finding: • the **straight-line distance** to Bucharest

Chart, radar chart

Description automatically generated

Diagram

Description automatically generatedOptimality of A\* Tree Search Assume:

• A is an **optimal** goal node

• B is a **sub-optimal** goal node

• h is **admissible**

Claim: A will be expanded before B. 1. **f(n) is less or equal to f(A)**

Let h\*(n) be the cheapest cost of getting to A from n

• h is admissible -> h(n) ≤ h\*(n)

Text

Description automatically generated with medium confidence• h\*(n) = g(A) - g(n) -> h(n) ≤ g(A) - g(n) • -> f(n) ≤ f(A)

Diagram

Description automatically generatedProof:

• Imagine B is on the frontier

• Some ancestor n of A is on the frontier, too (maybe A!)

• Claim: n will be expanded before B

1. f(n) is less or equal to f(A)

2. f(A) is less than f(B)

Diagram

Description automatically generatedIcon

Description automatically generated with medium confidence

3. n expands before B

A picture containing polygon

Description automatically generated

• **All ancestors of A expand before B --> A expands before B**

Diagram

Description automatically generatedA\* Graph Search Gone Wrong

A picture containing text, device, gauge

Description automatically generated

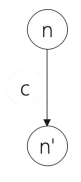
Diagram, schematic

Description automatically generatedConsistency (Tính nhất quán) of Heuristic

• Main idea: estimated heuristic costs ≤ actual costs

• Admissibility: heuristic cost ≤ actual cost to goal **h(A) ≤ actual cost from A to G**

• **Consistency**: heuristic “arc” cost ≤ actual cost for each arc **h(A) – h(C) ≤ cost(A to C)**

Consequence of Consistency

• The f value along a path **never decreases**

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

= g(n) + c + h(n')

≥ g(n) + h(n) consistency

= f(n)

• When A\* **selects a node for expansion**, the **optimal path** to that node has been found.

Diagram

Description automatically generated• Proof:

• Assume g(n) > g\*(n)

• Let n’ be the shallowest node in frontier on optimal path from s to n

• g(n') = g\*(n') and f(n') = g\*(n') + h(n')

• We have f(n') ≤ g\*(n') + c(n' , n) + h(n) **consistency**

• f(n') ≤ g\*(n) + h(n)

• f(n') < f(n) **contradiction**.

Optimality of A\* Graph Search

• Consider what A\* does with **a consistent heuristic**:

• Fact 1: A\* expands nodes in nondecreasing total f value

• Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimal

• Result: **A\* graph search is optimal**.

Summary

• Tree search:

• A\* is optimal if heuristic is **admissible**

• UCS is a special case (h = 0)

• Graph search:

• A\* optimal if heuristic is consistent

• UCS optimal (h = 0 is consistent)

• **Consistency implies admissibility**

• In general, most natural admissible heuristics tend to be consistent.

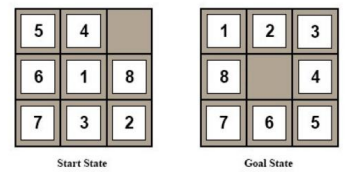
Intuition(trực giác) A\*

• Let f\* be the cost of optimal solution path.

• We have

• A\* expands all nodes with f(n) < f\*

• A\* may then expand some nodes right on “goal contour” , with f(n) = f\* before selecting a goal node.

Diagram

Description automatically generated

• A\* Gradually adds "**f-contours**" of nodes

• Contour i has all nodes with f=fi, where fi < f i+1 .

Design of Heuristics

1. h1 = number of misplaced tiles

2. h2= Manhattan distance

• the sum of the distances of the tiles from their goal positions

Which one should we use? h1 ≤ h2 ≤h\*

Importance of h(n) hc ≤ hm ≤ h\*

• **Prefer** hm

• **Note:** Expand all nodes with f(n) = g(n) + h(n) < f\*?

• **Every node with h(n) < f\* - g(n) will be expanded**

• • Since h1 ≤ h2 , every node expanded with h2 will be expanded by h1 .

• Aside. How would we get an hopt?

Comparison of Search Costs on 8-Puzzle

Table

Description automatically generated

• Effective branch factor b\*: characterize the quality of a heuristic N + 1 = 1 + b\* +(b\*)2 +….+(b\*)d .

Where Does the Heuristics

• Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

• Admissible heuristics: h(n) ≤ℎ\*(n)

• But if ℎ\*(n) is unknown, how can we verify the condition?

• Often, admissible heuristics are solutions to **relaxed problems**, where new actions are available -> **fewer constraints**

• The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original

• The models obtained by such constraint-deletion processes are called **relaxed models**

Example: 8 Queen

• Description of actions: A tile can move from square A to square B if A is adjacent to B and B is blank

(a) A tile can move from square A to square B if A is adjacent to B.

(b) A tile can move from square A to square B if B is blank

(c) A tile can move from square A to square B

Example: Traveling Salesman Problem

• Find the shortest route that visits each city exactly once and returns to the origin city

Diagram, shape

Description automatically generated

• Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

Example: Traveling Salesman Problem

• Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

• Three conditions for the path:

• Being a graph • Being connected • Being degree 2.

• Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

• Three conditions for the path:

• Being a graph -> Optimal Assignment Problem

• Being degree 2

• Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

• Three conditions for the path: • Being a graph • Being connected -> Minimum Spanning Tree Problem

Consistency

• Heuristics come from **relaxed problems** are guaranteed to be **consistent**.

Diagram

Description automatically generated

• h(n) and h(n') stand for the minimum cost of finding solution for some relaxed problem. We have

h(n) ≤ **c'(n, n')** + h(n') relaxed cost -> h(n) ≤ c(n, n') + h(n)

What if We Have Multiple Heuristics

• Suppose we have heuristics h1 (n), h2 (n), h3 (n)

• Every node with h(n) < f\* - g(n) will be expanded

• The final heuristic h(n) = max(h1 (n), h2 (n), h3 (n)).

A\* Applications

• Video games • Pathing / routing problems • Resource planning problems • Robot motion planning • Language analysis • Machine translation • Speech recognition.

A\*: Summary

**Optimal**: Yes A\* is optimally efficient: given the information in h, no other optimal search method can expand fewer nodes

**Complete:** Unless there are infinitely many nodes with f(n) < f\*. Asussme locally finite: (1) finite branching, (2) every operator costs at least ε >0

**Complexity (time and space**): still exponential because of breadth-first nature. Unless |h(n) - h\*| <= O(log(h\*(n)), with h true cost of getting to goal, the time complexity is polynomial.

Link: [https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i\_thu%E1%BA%ADt\_t%C3%ACm\_ki%E1%BA%BFm\_A\*](https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_t%C3%ACm_ki%E1%BA%BFm_A*)

. <https://vi.wikipedia.org/wiki/Heuristic#:~:text=Heuristic%20(%2Fhj%CA%8A%C9%99%CB%88r%C9%AA,%C4%91%E1%BA%A3m%20b%E1%BA%A3o%20l%C3%A0%20t%E1%BB%91i%20%C6%B0u>.

.

<https://en.wikipedia.org/wiki/Consistent_heuristic#:~:text=In%20the%20study%20of%20path,cost%20of%20reaching%20that%20neighbour>.

**Local Search**

Search Algorithms

•BFS, DFS, UCS, A\* etc

• Explore the state-space **systematically** to the goal state

• Need to reconstruct the **path**.

Local Search Methods

• The search spaces for some real-world problems is **enormously big**

• How many combinations can 500 processes be assigned to 100 computers? 100500 states

• There are 3361 states in Go board

• A completely different kind of method is called for: • Local Search Methods

Local Search Methods

• Applicable when we're interested in the Goal State -- **not in how to get there**

• E.g. N-Queens, Course Scheduling Problem or Job-shop Scheduling

• Basic idea:

- use a single current state

- don't save paths followed

- generally move only to sucessors/neighbors of that state

- find the best state according to an objective function or cost function

Local Search Methods

• Advantages:

• Use very little memory

• Can be applied to problems with a large number of states

• Can be applied to problems with changing state spaces

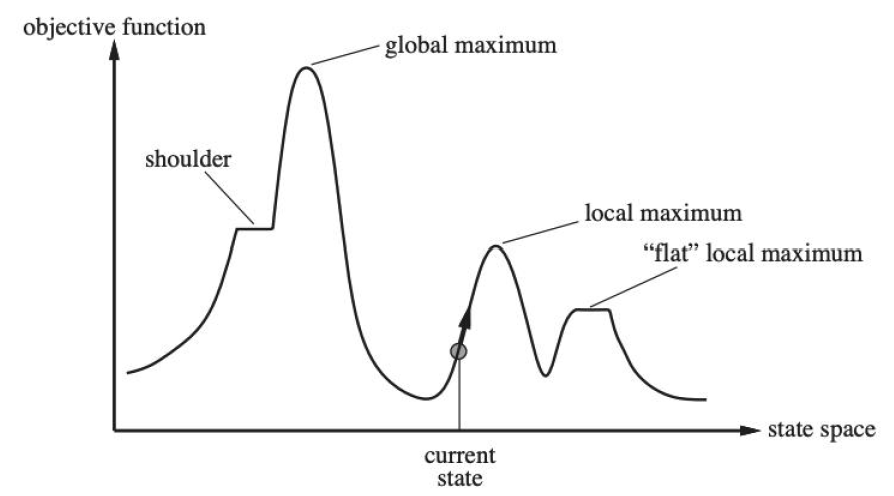
• Disadvantages:

• Cannot recover the path

• The solution may not be optimal

State-Space Landscape

• Local Search Methods rely on an objective function or a cost function to compute the state-space landscape



• Local search algorithms explore this landscape

Hill Climbing Search

• Move in the direction of increasing value

• Terminate when it reaches a “peak” where no neighbor has a higher value.

Graphical user interface, text, application

Description automatically generated

• Only need to record the state and the value of the objective function or cost function

• **Similar to greedy algorithm**, **only check the value of the immediate neighbors**

• Rely on **complete-state formulation** instead of incremental formulation

Complete-State Formulation vs. Incremental Formulation

Diagram

Description automatically generatedA picture containing crossword puzzle

Description automatically generated

• Complete-State Formulation

• **All queens are in the board**. Move any queen in the same column

• **Why used in local search**? -> Divide object function

• Incremental Formulation

• Add each queen to the board one by one (used in many search algorithms we covered before)

Hill Climbing Search Can Get Stuck

• Local maxima

• A peak that is higher than all its neighbors but lower than the global maximum.

• Ridges: • A sequence of local maxima

A picture containing crossword puzzle

Description automatically generatedA picture containing antenna

Description automatically generated

• Plateaux or shoudlers • The objective function is constant

A picture containing text, electronics, calculator, keyboard

Description automatically generatedDiagram

Description automatically generated

Example: 8-queen problem

• Put n queens on an n × n board with no two queens on the same row, column, or diagonal

• What is the **state-space** of this problem?

• How is this problem different from the 8-puzzle problem?

Example: 8-queen problem

• Heuristic cost function: the number of pairs of queens that are attacking each other, either directly or indirectly

• Each time we can move the queen to another square in the same column

Example: 8-queen problem

A picture containing text, crossword puzzle

Description automatically generated• Heuristic cost function: the number of pairs of queens that are attacking each other, either directly or indirectly

• Local minimum with h =1

Table

Description automatically generatedExample: Satisfiability

• Propositional logic:

• Literals (**True or False**) A, B, C, ...

• Logical connectives: A picture containing icon

Description automatically generated

• Clause: a disjunction of literals Chart, scatter chart

Description automatically generated

• Conjunctive normal form: 

Example: Satisfiability

• A wide variety of key CS problems can be translated into a propositional logical formalization

e.g., 

• Solved by finding a truth assignment to the propositional variables (A,B,C,…) that makes it true, i.e., a model.

• If a formula has a model, we say that it is “satisfiable”

Example: Satisfiability Testing

• Best-known method: Davis-Putnam Procedure (1960)

- Backtrack search (DFS) through the space of truth assignments

• Assigning values to variables one by one and simplifying at each step

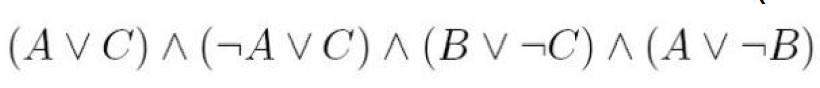
• Cannot be satisfied Shape

Description automatically generated

Example: Satisfiability Testing

Best-known method**: Davis-Putnam Procedure** (1960): <https://www.youtube.com/watch?v=ENHKXZg-a4c>

Diagram

Description automatically generated with low confidence

• To date, Davis-Putnam Procedure is still the fastest sound

and complete method.

• However, there are classes of formulas where the procedure

scales badly.

• Consider an incomplete local search procedure

Greedy Local Search -- GSAT

• Begin with a random truth assignment (assume CNF).

• Flip the value assignment to the variable that **yields the greatest number of satisfied clauses**. (Note: Flip even if there is no improvement.)

• Repeat until a model is found, or have performed a specified maximum number of flips.

• If a model is still not found, repeat the entire process, starting from a different initial random assignment.

• Input: a conjunctive normal form α, MAX-FLIPS, MAX-TRIES

• Output: a satisfying truth assignment if found

• For i = 1 to MAX-TRIES:

T := **a randomly generated truth assignment**

For j = 1 to MAX-FLIPS:

If T satisfies α, then return T

p := **a propositional variable such that a change of its truth leads to the largest number of satisfied clauses**

T := **T with the truth value of p reversed**

return “no satisfying assignment found”

Greedy Local Search -- GSAT



• A: False B: False C: True

Table

Description automatically generated

How well does it work?

• First intuition: It wil get stuck in local minima, with a few unsatisfied clauses.

• Note we are not interested in almost satisfying assignments

• E.g., a plan with one “magic” step is useless.

• Contrast with optimization problems.

• GSAT is not complete.

• **Surprise:** It often finds global minimum!

• i.e., finds satisfying assignments

• Generate a large number of random formulas

• Different number of propositional variables

• Different number of clauses



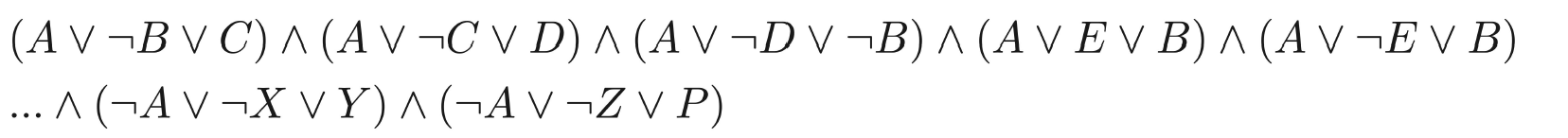
• Run GSAT and Davis-Putnam Procedure on the generated formulas and measure the time to find the right assignment.

Greedy Local Search – GSAT

Table

Description automatically generated

Limitations of GSAT



• Can be satisfied only when A is True

• But GSAT prefers a negative assignment

Graphical user interface, text, application, Word

Description automatically generated

HCS:

<https://www.youtube.com/watch?v=oSdPmxRCWws>

<https://www.youtube.com/watch?v=rTcRLIxCxVs>

Best-known method**: Davis-Putnam Procedure** (1960): <https://www.youtube.com/watch?v=ENHKXZg-a4c>

GSAT: <https://www.youtube.com/watch?v=SAXGKCnOuP8>

**Local Search**

Improvements to Basic Local Search

• Issue: How to move more quickly to successively higher plateaus and avoid getting “ stuck” / local minima

• Idea: Introduce uphill moves (“ noises”) to escape from long plateaus (or true local minima).

• Strategies: • Simulated Annealing • Random-restart hill-climbing • Tabu search • Local beam search • Genetic Algorithms.

Variation on Hill-Climbing

• Random restarts: simply restart at a new random state after a pre-defined number of local steps

• Tabu: prevent returning quickly to the same state. • Implementation: Keep fixed length queue (“tabu list”): add most recent step to queue; drop “oldest” step. Never make step that’s currently on the tabu list

• **Uphill moves are acceptable** if no downhill moves are available.

A picture containing diagram

Description automatically generatedSimulated Annealing

Chart

Description automatically generated with low confidence

• Idea: • Use conventional hill-climbing techniques, but occasionally take a step in a direction **other than that in which the rate of change is maximal**.

Chart, line chart

Description automatically generated

As time passes, • The size of any down-hill step taken is **decreased**.

• The probability that a down-hill step is taken is gradually reduced

Chart, line chart

Description automatically generatedA picture containing chart

Description automatically generated

Chart

Description automatically generatedSimulated Annealing

• Intuition:

A picture containing text, clipart

Description automatically generated

• The probability of move to a bad state should decrease exponentially with the “badness” h(Next) - h(Current)

• The probability decreases as the “temperature” T goes down

• How to schedule the “temperature”? • Initially with a high temperature and gradually decays



SA Algorithm

• Current, Next: nodes/states

• T: “temperature” controlling probability of downward steps

• Schedule: mapping from time to “temperature”

• H: heuristic evaluation function.

Chart

Description automatically generated

current ← initial state

for t ← 1 to inf do

T ← schedule[t]

if T = 0 then return current

next ← randomly selected successor of current

Δh ← h(next) - h(current)

if Δh > 0 then current ←next uphill

else current ← next only with probability eΔh/T downhill.

SA Algorithm: Convergence

• If the schedule lowers T slowly enough, SA will find a global minimum with probability approaching 1

Diagram

Description automatically generated• In practice, reaching the global minimum could take a large number of iterations.

Local Beam Search

• Instead of maintaining one current state, local beam search **keep track of k states**

• **All of the successors** of the k states are generated

• Local beam search selects the **best k states** from all the successors and repeat until the goal state is found.

• Useful information is passed among the parallel search threads

• Disadvantage: • All k states can become stuck in a small region of the state space

Example: Satisfiability

• A wide variety of key CS problems can be translated into propositional logical formalization

e.g., 

• Solved by finding a truth assignment to the propositional variables (A,B,C,…) that make it true, i.e., a model.

• If a formula has a model, we say that it is “satisfiable”

Random Walk SAT

Random walk SAT algorithm: Text

Description automatically generated with medium confidence

I Pick random truth assignment.

II Repeat until all clauses satisfied:

Flip variable from any unsatisfied clause.

Solve 2-SAT (2 variables per clause) in O(n2) flips

Does not work at all for hard k-SAT (k >= 3)

Walk SAT: Mixing Random Walk w/ Greedy Search(Selman et al. 1996)

• With probability p, walk, i.e., pick a variable in some unsatisfied clause and flip it;

• With probability (1-p) make a **greedy** flip, i.e., one that makes greatest decrease in number of unsatisfied clauses.

• Cannot detect **un-satisfiability**.

Experimental Results: Hard Random 3CNF

• Noise: different from Walk SAT, the selected randomly flipped variable is not restricted to be in an unsatisfied clause

Table

Description automatically generated

• Complete methods, such as DP, up to 400 variables

• **Walk SAT** better than **Simulated Annealing** better than **Basic GSAT** better than **Backtracking (Davis-Putnam)**.

Local Search in continuous Spaces

• Originated in the 17th century, after the development of calculus by Newton and Leibniz.

Min f(x1, x2,x3,…). A close-up of a human hand

Description automatically generated with low confidence

• Example: • Place a storage center (x, y) that are close to n cities(ai ,bi).

Text

Description automatically generated with medium confidence

Gradient Descent

• Similar to hill-climbing search but the states are continuous

• Basic idea:

• Use the gradient of the cost function for updating the unknowns. Text

Description automatically generated

• The gradient gives the direction for decreasing the objective function.

Shape

Description automatically generatedText

Description automatically generated

• For small enough α, we have

Diagram

Description automatically generated

Genetic search