SEARCH ALGORITHMS IN AI

Search Algorithms

Uninformed Search

- Breadth-first Search (BFS)
- Uniform-cost Search (UCS)
- Depth-first Search (DFS)
- Depth-limited Search (DLS)
- Iterative Deeping Depth-first Search (IDV)

Informed Search

- Best-first Search
- Greedy Best-first Search
- A* Search

Local Search

- Hill-Climbing
- Simulated Annealing
- Local Beam
- Stochastic Beam Search
- Genetic Algorithms

SEARCH PROBLEM

Initial State

- State is a representation of the problem

Successor Function (possible actions)

- State X, by action A, goes to state Y (step)
- Implicitly defines the possible **state space**
- **Path**, sequence of states, created by actions

Objective Test (goal test)

- Checks if a **state** is **final** (objective state)

Cost Function

- c(X,A,Y) → **cost** of the step of executing A, from state X to Y.
- Path cost → sum of the cost of steps

Solving a search problem

- **Path** from the **initial** state to a **final** state (goal state)
- Optimal solution → has the least cost path

SEARCH ALGORITHMS IN AI

Search Algorithms

Uninformed Search

Breadth-first Search (BFS)

Uniform-cost Search (UCS)

Depth-first Search (DFS)

Depth-limited Search (DLS)

Iterative Deeping Depth-first Search (IDV)

B → Breadth First

U → Uniform Cost

D → Depth First

D → Depth Limited

I → Iterative Deeping Depth- First

Informed Search

Best-first Search

Greedy Best-first Search

A* Search

G → Greedy Best-First

A → A*

B → Best First

Local Search

Hill-Climbing

Simulated Annealing

Local Beam

Stochastic Beam Search

Genetic Algorithms

G → Genetic Alg.

H → Hill Climbing

S → Simulated Annealing

S → Stochastic Beam

L → Local Beam

SEARCH PROBLEM

Initial State

- State is a representation of the problem

Successor Function (possible actions)

- State X, by action A, goes to state Y (step)
- Implicitly defines the possible **state space**
  ↳ olası durum uzayını dolaylı olarak tanımlar
- **Path**, sequence of states, created by actions

Objective Test (goal test)

- Checks if a **state** is **final** (objective state)

Cost Function

- c(X,A,Y) → **cost** of the step of executing A, from state X to Y.
- Path cost → sum of the cost of steps

Solving a search problem

- **Path** from the **initial** state to a **final** state (goal state)
- Optimal solution → has the least cost path

- initial state → where to begin
- Path → sequence of states (by actions)
- Path cost → sum of the cost of space
- Goal test → check if the state is final
- Optimal solution → has the least cost path

FIND SOLUTION – SEARCH TREE

→ possible actions

Generated from initial state and successors function

- **Root** is the search node with the **initial state**
- **Test** if this node is final (if yes, found the solution)
- **Expand** current state: apply successor function → **generate** new states (new search nodes)
- From all the new states (the ones that were generated) **choose** one of them – **search strategy** – and repeat the process (Test, expand, generate, choose)

root → search node with initial state

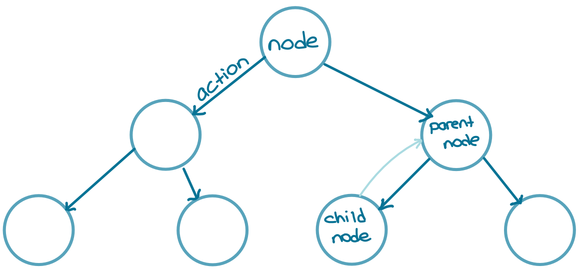
test → if the node is final

expand current state: apply sucessor function

↳ generate new states

Representation of the **search node**

- The **state** (of the state space) associated with it
- The **Parent node**
- The **action** that was applied to the parent to generate it
- The **cost of path** from the initial state (root) – **g(n)**
- The **depth** (number of steps from the root)



Set of generated and not yet expanded nodes – **fringe**

↓ not expanded yet

- New nodes and are in the tree leaves
- Stored in a queue (queue type → search strategies)

FIND SOLUTION – PERFORMANCE

It is necessary to measure the performance of the search algorithm in solving problems

- **Complete:** If the solution is exists, it is found
- **Optimal:** Ensures that it finds the optimal solutions
- **Complexity** in terms of **time** → by the number of nodes generated
- **Complexity** in terms of **space** ↳ by the number of nodes stored in

Complexity is expressed by 3 values

- **b**, branching factor
- **d**, depth of the smallest objective node
- **m**, maximum length of any path

Complexity is measured

- **Time**, by the number of nodes generated
- **Space**, by the number of nodes stored in

complete → if the solution is exists

optimal → find the optimal solution

complexity (time) → by the number of nodes generated

complexity (space) → by the number of nodes stored in

b → max branching factor in a tree

d → the depth of the least cost solution

m → max depth state space

17:40

UNINFORMED SEARCH (BLIND SEARCH)

There is no additional information about the states of the world beyond that given by the problem definition.

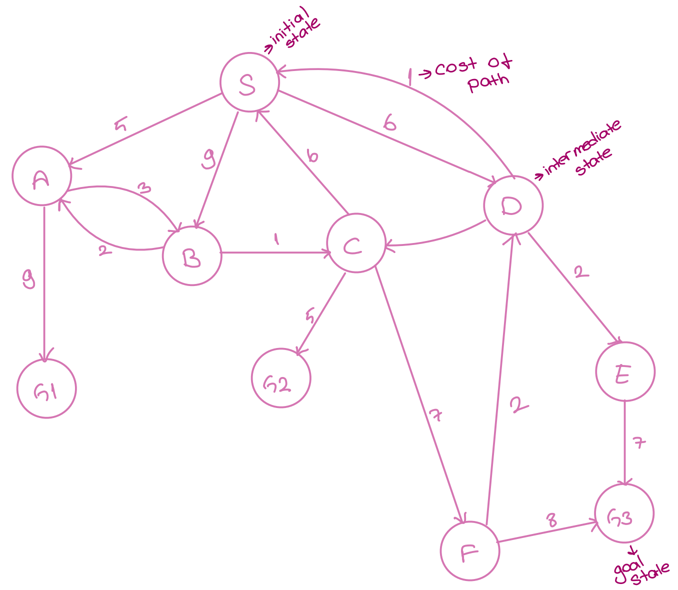
- Avoid repeated states
  - Expand states that were previously expanded search in graphs
  - **Closed list:** stores all nodes already expanded
  - **Open list**: nodes at the fringe of the search tree
  - If the current node is already in the Closed list, it is not expanded.

uninformed search → no info about the current state to the goal state

open list → generated but not expanded

closed list → expanded

Search Space



BREADTH-FIRST SEARCH (BFS)

Complete

PROCURA EM LARGURA PRIMEIRO

- All nodes of a given level are expanded before nodes of the next level are expanded.
- Uses a FIFO (queue) strategy for the selection of nodes at the fringe of the search tree.

Complete

- If the objective but shallow (small) node is at level d, it will be found when that level is expanded.
  → depth of the least cost solution

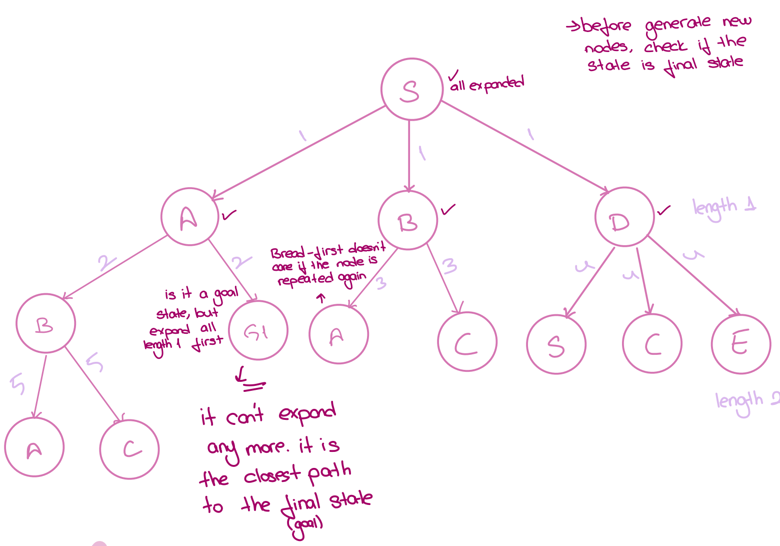
Excellent

- If the path cost is a non-decreasing function of depth. (for example, all actions have the same cost)

- Required memory is a bigger problem than runtime

Exponential complexity problems cannot be solved by uninformed methods, except for small instances.

→ must explore all the paths at level n before move to level (n+1).



final state → S – A – G1

UNIFORM COST SEARCH (UCS)

Optimal

PROCURA DE CUSTO UNIFORME

- Expands the node with the lowest path cost.
  - If all steps are of equal cost this method is equal to breadth-first search.

$F(n) = g(n)$ → gerçekler

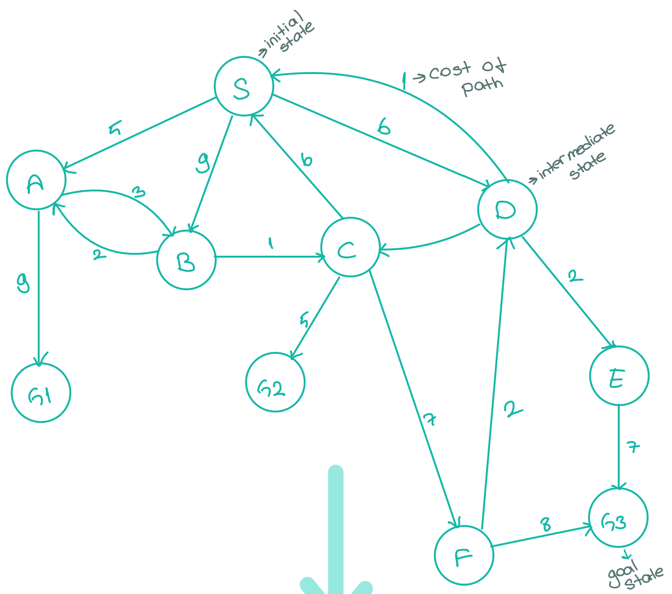
↓ arama kriterini belirleyen fonks.

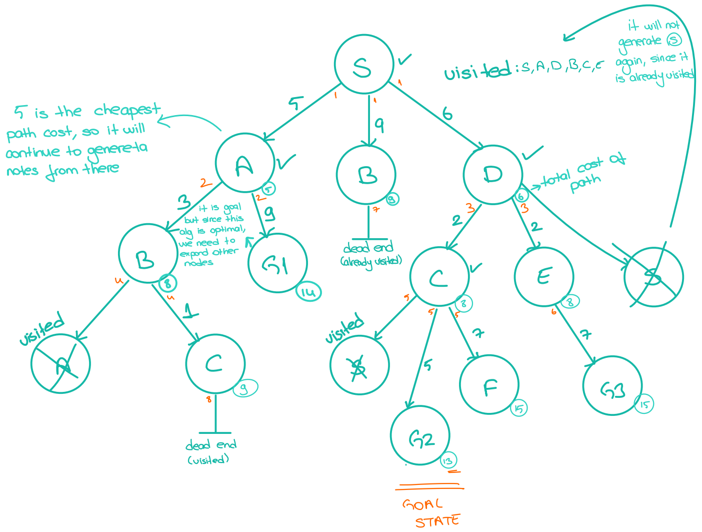
- This strategy is complete as long as it is guaranteed that the cost of each step is greater than or equal to any small positive constant value.

- This condition is also sufficient to guarantee that the strategy is optimal.

→ it will keep a visited list of all nodes, because no need to visit them again

Search Space





GOAL → S – D – C – G2 13

DEPTH-FIRST SEARCH (DFS)

PROCURA EM PROFUNDIDADE PRIMEIRO

- Expands the deepest node at the fringe of the search tree.

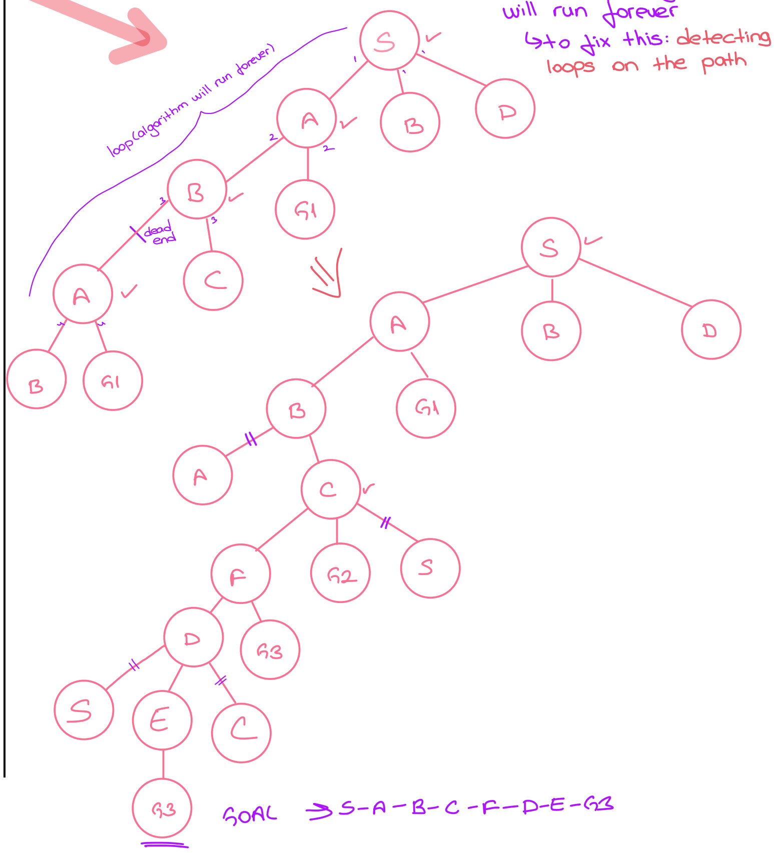
- It uses a LIFO (fstack) strategy for the selection of nodes at the fringe of the search tree.

- Save only expanded nodes between the tree root and a leaf node
  - Path between the root and a leaf node and the nodes not yet expanded.
  - O(bm), **b** is the branching factor and **m** is the maximum depth.
  - Backtracking search, only one successor is expanded at a time, O(m).

→ not optimal in terms of cost
→ not optimal in terms of number of action
→ use a lot of space in memory

- Not complete
  - Unlimited depth.
- Not great
  - There may be another solution closer to the root, in a subtree that has not yet been searched.

→ when the algorithm will run forever
↳ to fix this: detecting loops on the path



DEPTH-LIMITED SEARCH

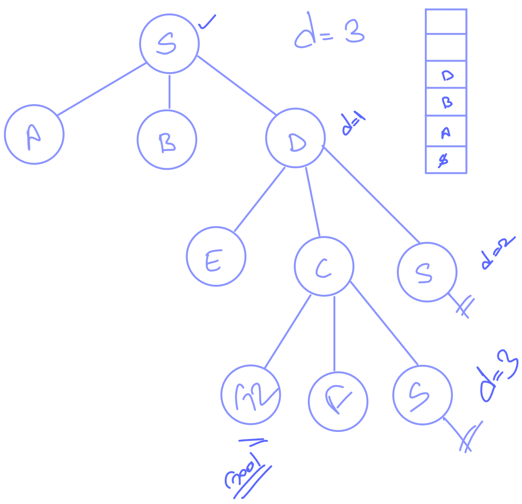
DLS = DFS + limit for the level

PROCURA EM PROFUNDIDADE LIMITADA

- Unlimited depth trees problem
  - Can be solved by considering the depth search first with a pre-established depth limit **l.**
  - Nodes at level 1 are treated as having no successors.
  - Solves the problem of unlimited paths.

- This limit adds a new source of incompleteness
  - If **l < d**, the shallowest (smallest) objective state is beyond the imposed depth limit (it is antural when **d** is not known)

- If **l > d** the search is also not optimal.



ITERATIVE DEEPENING DEPTH-FIRST SEARCH (IDF)

PROCURA EM PROFUNDIDADE PRIMEIRA ITERATIVA

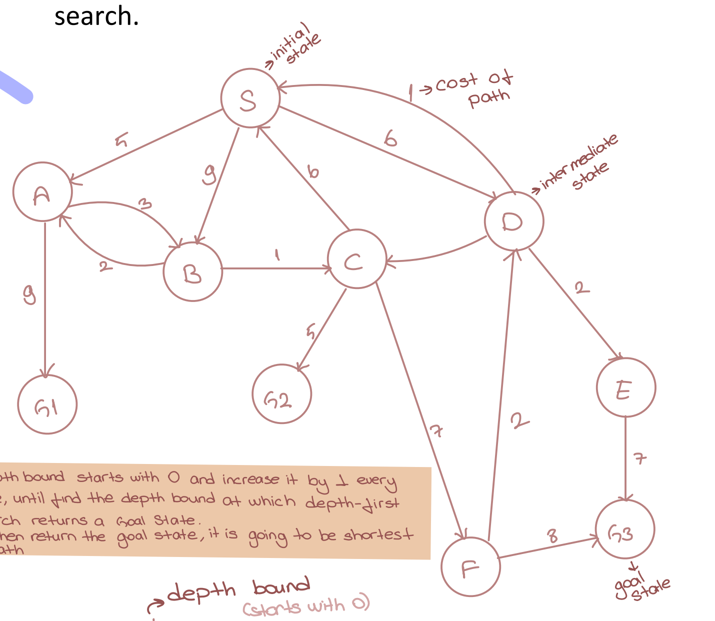
- Apply depth-limited search by gradually increasing the limits (0,1,2,3, ...) until you find the solution.

→ Don't check the loops

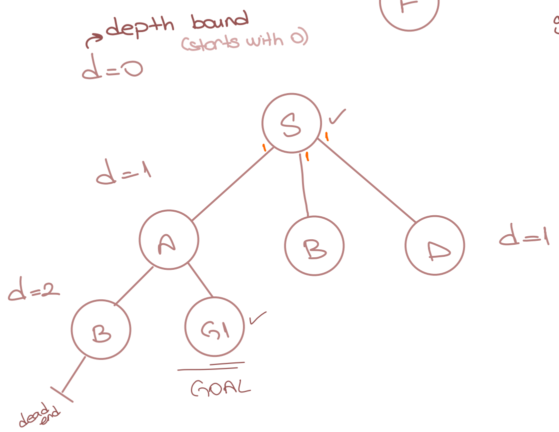
- Complete →branching factor
  - When **b** is finite.
- Excellent
  - When the path cost is a non-decreasing function of the node depth.

it will run depth-first with a depth bound → when reach to depth-bound it will treat as a dead end

- Memory: O(bd)
- Time: generate fewer nodes than breadth-first search.



depth bound starts with 0 and increase it by 1 every time, until find the depth bound at which depth-first search returns a Goal State.
→ when return the goal state, it is going to be shortest path



it can find a optimal plan sometimes faster than breadth-first search

INFORMED SEARCH

- Informed search algorithm contains an array of knowledge such as how far from the goal, path cost, how to reach to goal node, etc.
  - This knowledge help agents to explore less to the search space and find more efficiently the goal node.

– Informed Search –
→ contains an array of knowledge.
• how far from the goal
• path cost
• how to reach to goal node

HEURISTICS FUNCTION

- It takes the current state of the agent as its unput and produces the estimation of how close agent is from the goal.

- Heuristic function estimates how close a state is to the goal.

→ heuristic function

- It is represented by **h(n)**, and it calculates the cost of an optimal path between the pair of states.
  - The value of the heuristic function is always positive.

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

heuristic cost ↳ estimated cost

h(n) = heuristic cost

h*(n) = estimated cost

BEST-FIRST SEARCH → Greedy Search)

NOT COMPLETE

PROCURO MELHOR PRIMEIRO

→ sezgiseli (heuristic) önemliyoruz ve gerçek maliyeti dikkate almıyoruz

- It is a node that selected for expansion based on an evaluation function, f(n).

- Traditionally the node with the lowest rating is selected to be expanded, because the rating function measures the distance to the target.

- What we do is choose the node that seems to be the best (lowest cost), according to the evaluation function.

arama kriterini belirleyen fonks.

$F(n) = h(n)$

↳ heuristic (sezgisel)

- An important part of these algorithms is the heuristic function:
  - **h(n)** = estimated cost of the best path between node **n** and the objective node
  - If **n** is an objective node then **h(n) = 0**

→ it always selects the path which appears best at the moment.

→ it is a combination of DFS and BFS

DFS ↓ Depth-First Search
→ expands the deepest node at the fringe of the search tree

BFS ↓ Breadth-First Search
→ must explore all the paths at level n before moving

→ it uses the heuristic function
h(n) ≤ h*(n)

→ it is implemented by priority queue

GREEDY BEST-FIRST SEARCH

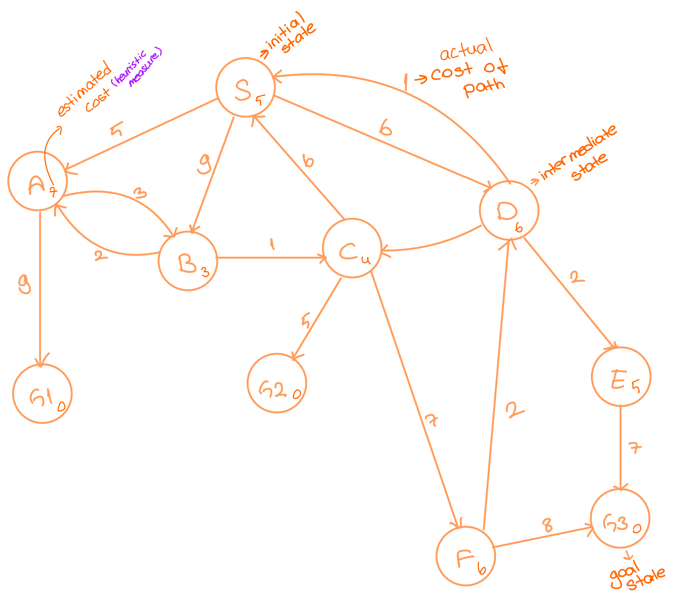
PROCURO MELHOR PRIMEIRO GREEDY

- It tries to expand the node that is closest to the goal, based on the assumption that through that node the solution is reached more quickly.
  - **f(n) = h(n)**

- The algorithm is called Greedy because at each step it tries to get as close to the goal as possible.

- This strategy is similar to **depth-first search**, as it prefers to follow a single path to the goal, backtracking when it reaches a terminal node.
  - It is therefore not optimal and incomplete.

Greedy version



A* SEARCH

o düğüme ulaşmanın maliyeti + o düğüme ulaşmaktan hedefe ulaşmanın maliyeti

- Minimize the estimated total cost for the solution nodes are evaluated by combining the cost of getting to that node, **g(n)**, and the cost of getting from that node to the goal, **h(n)**:
  - **f(n) = g(n) + h(n)** → gerçek maliyet ile sezgisel maliyet

→ heuristic never overestimate the cost

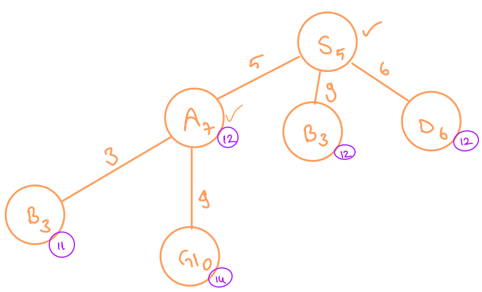
- The node with the lowest value of **g(n)(custo) + h(n)(estimativa)** is chosen, since it has the estimate of the least cost path.

- A* using **Tree-Search** is **optimal** if **h(n)** admissible (kabul edilebilir) heuristic
  - That is, since **h(n)** never overestimates the cost of reaching the goal.
  - In a city map an admissible heuristic is the distance in a straight line.

- A* using **Graph-Search** is **optimal** if **h(n)** consistent (tutarlı) heuristic
  - If for all node **n** and all successors **n'** of **n** generated by an action **a**, the estimated cost of reaching the goal from **n** is not greater tha the step up to **n'** plus the estimated cost of **n'** to the goal: $h(n) \leq c(n, a, n') + h(n')$

A* score = cost of path + heuristic

Visited: S(5), A(12),



LOCAL SEARCH

- In many optimization problems, the path to the goal is irrelevant; the objective state itself is the solution (e.g., n-queens)

- In these cases we can use local search

- Maintains a single “current state”; paths are not are memorized

- In each iteration it seeks to “improve” the current state; useful in optimization

- Typically, a state transitions to “neighboring” states

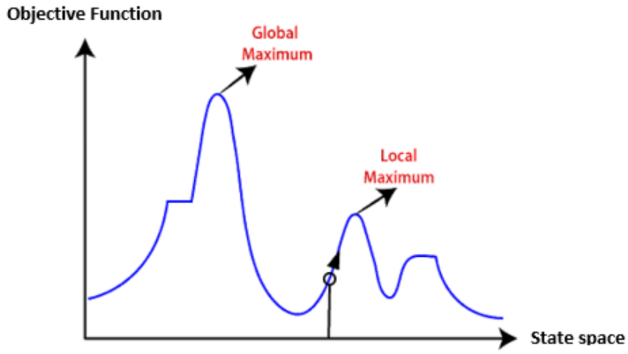
PROBLEM: Typically not complete!

HILL-CLIMBING (GREEDY LOCAL SEARCH)

TREPA COLINAS

- It is a simple cycle that continually moves towards a better value. Ends when no successor has better values.

PROBLEM: Depending on the initial state, it may get stuck at a local maximum.



HILL-CLIMBING VARIANTS

1. **Stochastic Hill-Climbing:** Randomly choose from the best successors

2. **First-choice Hill-Climbing:** Generates the successors randomly unit it finds the first one with better values than the current state and that. Is the one that is chosen (it’s handy if a state has thousands of possible successors)

3. **Random-restart Hill-Climbing:** Conducts a series of searches from different, randomly generated initial states; stops when the target is reached

SEARCH SIMULATED ANNEALING

PROCURA SIMULATED ANNEALING

Idea: Escape from local minima allowing “bad” movements to be made, but gradually decreasing their frequency

- Instead of choosing the best successor, choose a successor at random that is typically “accepted” if the situation improves
- Simulated tempera.

- It is possible to prove that if temperature T decreases slowly enough (as a function of the schedule), then the simulated annealing search will find a global maximum with probability close to 1.

Metaphor: Imagine the task of putting a ping-pong ball into the deepest hole in a surface full of holes

- One solution is to let the ball land at a local minimum and then shake the surface to get it out of the local minimum.
- Simulated annealing starts by “waving” a lot at the beginning and then it starts shaking less and less.

LOCAL BEAM

PROCURA EM BANDA

- Stores reference to **k** states instead of 1
  - Starts with **k** randomly generated states

- In each iteration, all successors of the **k** states are generated.

- If any is an objective state, stop; otherwise choose the **k** best successors and repeat.

Note that this algorithm is more than running **k** Random-Restart Hill Climbings in parallel!

- Successors from all states do not have to be chosen
- If one state generates several good successors and the other **k-1** states do not, the less promising states are abandoned

- However, it can also have problems: there can be little diversity in the **k** state.
  - **Stochastic Beam Search:** **k** successors are chosen at random.

GENETIC ALGORITHMS

ALGORITMOS GENÉTICOS

- Variant of the **Stochastic Beam Search**

- Starts with **k** randomly generated states (**population**) such as in-band search
  - A state is represented as a string over a finite alphabet (usually {0,1})

- The successor state is generated by combining two states (**parents**)
  - Produces the next generation of states by selection, crossover and mutation
  - The **fitness function** gives higher values to the best states

SAT AND CSP

They are identical problems

- There is a set of variables
- Constraints between these variables
- Need to find values for variables
  - They do not violate restrictions

SAT

PROPOSITIONAL SOLVENCY PROBLEMS

- Binary variables

- Restrictions; formula for propositional calculus

- Assignment that satisfies the formula

CSP

- Each variable has a specific domain
- Various type of constraints between variables
- Assign values to variables that satisfy constraints

SAT - DEFINITION

- Given a formula of propositional calculus
  - Represented by the Boolean function **F(x)**

- Identify assignments for variables
  - $\mathbf{X^* = \{x^1 = 0, x^2 = 1, \dots\}}$

- That satisfy the formula
  - $\mathbf{F(X^*) = 1}$

- Or, prove that no assignment satisfies the formula
  - **F(X) =** 0, **for all possible assignments**

- Formula represented in **Conjunctive Normal Form (CNF)**
  - Conjunction of disjunctions

    `( x ∨ y ) ∧ ( x ∨ ¬z ∨ ¬y ) ∧ (z ∨ ¬y`

  - **Clauses** (disjunctions)
  - Positive and negative **literals** (variables)

Consider the formula;

`( x ∨ y) ∧ ( x ∨ ¬z) ∧ (z ∨ ¬y)`

- **The assignment:** `{x=0, z=1}`

- **Literals:** 0 ∧ 0 ∧ 1 = 0
  - True
  - False
  - Free

`( x ∨ y) ∧ ( x ∨ ¬z) ∧ (z ∨ ¬y)`

- **Clause**
  - **Satisfied:** If at least 1 of your literals is true (3^{rd})
  - **Not satisfied:** If all its literals have the valuye false (2^{nd})
  - **Unresolved:** Otherwise (1^{st}) (unitary = with 1 free literal)

- **Formula**
  - **Satisfied:** If all your clauses are satisfied
  - **Not satisfied:** If at least one clause is unsatisfied
  - **Unresolved:** If any of the clauses are unresolved

CSP

CONSTRAINTS SATISFACTION PROBLEMS

- Each variable has a specific domain
- Various type of constraints between variables
- Assign values to variables that satisfy constraints

A CSP is a set of:

- **Variables**: $X=\{x_1,\ldots,x_n\}$
- The respective **domains**: $D=\{d_1,\ldots,d_n\}$
- **Restrictions**: $C=\{c_1,\ldots,c_m\}$

Every restriction;

- A subset of X
- With the specification of the allowed values

- An **assignment** of values to variables $\{x_i=v_i,\ x_j=v_j\}$
  - **V**$_i$ is a value of the domain **d**$_i$
  - **Complete:** if all variables have value
  - **Consistent:** if not violating restrictions
  - **Inconsistent:** otherwise
- **Solution** is a complete and consistent assignment

CSP - RESTRICTIONS

Cryptarithmetic Example

$$\begin{array}{r} \text{SEND} \\ \text{+MORE} \\ \hline \text{MONEY} \end{array}$$

Replace each letter with a different number so that the sum is correct.

Definition of the constraint satisfaction problem

- Variables: {S,E,N,D,M,O,R,Y}
- All have the same domain: {0,1,2,3,4,5,6,7,8,9}

Restrictions

- S and M cannot be zero: {S≠0, M≠0≈ (**unary**)
- The variables are all different: 28 inequality x≠y (binary)

- Correct sum:
  - 1000S + 100E + 10N + D + 1000M + 100O + 10R + E = 10000M + 1000O + 100N + 10E + Y

- Global restriction
  - The previous 28 restrictions can be replaced by
    all_different (S,E,N,D,M,O,R,Y)
    - All variables are different two by two
  - A global constraint is applied on a sequence of variables
  - Used for efficiency reasons
  - And for reasons of simplicity of the model

- Numerical restrictions

- High level restrictions
  - About complex data structures (list, trees)
  - Meta-constraints, combine constraints (implication)

- The choice of a model influences the resolution of the CSP

- CSPs have a higher expressive power than SAT
  - In SAT we lose structure information

SAT AND CSP – ALGORITHMS

We can classify them into

Complete

- If there is a solution, it is found
- Allows you to prove that a problem has no solution
  - If the algorithm ends without finding a solution
- Search with backtracking

Incomplete

- Do not guarantee to find a solution
- Do not allow to prove that a problem has no solution
- Local minima problems
- Local search

BACKTRACKING SEARCH

- Search space
  - Defined by possible assignments to variables

- Depth search
  - At each node the Heuristic chooses the variable to be assigned
    - In CSP this is done in 2 phases:
      - 1st choose the variable
      - 2nd choose the value to assign
  - Propagation of constraints (reduces the search space)
    - SAT: unit clause rule (BCP)
    - CSP: Arc consistency
  - Detects conflict and analyzes it
    - Non-chronological rewind
    - Learning
      - SAT: clause registration
      - CSP: nogoods register (not yet used)
  - Defines a search tree

BACKTRACKING SEARCH – SAT

The evolution in this area was marked by the emergence of increasingly competitive (autonomous) solvers

- GRASP (1996), techniques that reduce the search space (BCP, conflict analysis, etc.)
- zChaff (2001), BCP optimization, lightweight conflict-based heuristics, frequent restarts
- Others, but based on the zChaff architecture:
  - MiniSat (2003)
  - SatElite (2005)
  - Rsat (2007)
  - PicoSat (2008) (new quick restarts policy)

BACKTRACKING SEARCH – CSP

- The evolution in this area was towards creating tools (ILOG)
- And build applications that use them
- Problem dependent application
  - Heuristics use problem domain knowledge
  - Conflict analysis is problem specific
    - Difficult to build generic learning modules
- Generic solvers
  - Start to appear
  - XCSP

LOCAL SEARCH – SAT

- The evolution in this area was marked by the emergence of increasingly competitive (autonomous) solvers.
- Start with a full assignment
  - They are changing (altering) the value of the variables
- GSAT (1992)
  - Performs multiple attempts (restarts)
  - Weights to escape local minimums
- WalkSat (1994)
  - GSAT + other exchange policies
  - Adaptive Novelty+ (2002)
- Genetic Algorithms
  - Various potential solutions
  - Computationally heavy
    - Improved with the use of Parallel Hardware (2006)

LOCAL SEARCH – CSP

- Start with a full assignment
- Use a cost function
- Change the value of a variable or swap it with another variable
- GSAT type heuristic
- The stop conditions
  - Maximum number of iterations
  - Variations of the cost function
- Use Restarts
  - To exit local minima

HEURISTIC DECISIONS

- The use of backtracking search algorithms
  - Making decisions about the variable to choose (and value)
  - Advance the search
  - In an informed way
  - Trying to direct the search to the solution

- A bad initial decision
  - Diverts the search
  - Lead to combinatorial explosions in the search tree

- The use of good heuristic decisions
  - Crucial for solving many problems
  - Reduce combinatorial explosion

- There are no 100% informed heuristics

HEURISTIC DECISIONS – SAT

- DLIS
  - Literals appearing in more unresolved clause

- VSIDS (zChaff)
  - Counter for each literal, of conflict clauses
  - Periodically the counters are normalized
  - Prefers to satisfy (most recent) conflicts first
  - Very fast!

- Others based on VSIDS
  - BerkMin, also considers literals that contributed to the conflict
  - MiniSat, makes normalizations smoother (better results)
  - VMTF, uses a priority queue scheme
    - Shifting start literals that appear in recent conflicts

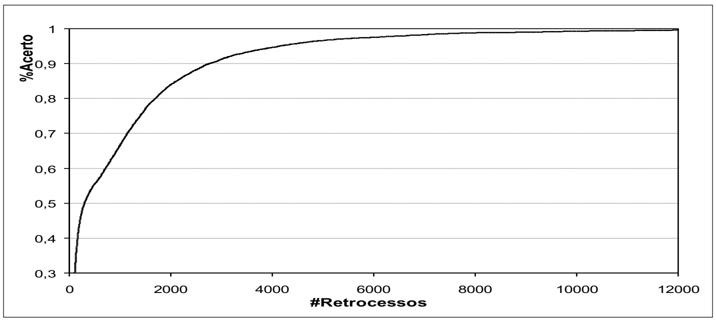
- All these heuristics use randomness
  - Tie cases or to minimize bad choices

HEURISTIC DECISIONS – CSP

- 2 fases
  - Choice of variable
  - Choice of value (considered of marginal importance)
- Fail first principle
  - First try where it is likely to fail
- Dom – smallest number of remaining values
  - Chooses variables with less hypothesis (fewer values in the domain)
- Dom/ddeg
  - Privileges variables with small domains and many links
- Dom/wdeg
  - The importance of links is weighted with a weight
  - Weight increases whenever a constraint (link) is violated
  - Prioritizes heavily violated restrictions (conflict oriented)

RESTARTS

- The runtimes of backtracking algorithms are characterized by **heavy-tail** distributions.
  - Using random heuristics
  - Sometimes the algorithm can take a long time



- Restarts strategy
  - Restarts the search whenever a setback threshold value is reached (**cutoff**)
    - Indicates that the algorithm is lost
  - Increments the cutoff value after each remainder
  - Keep the conflict clause between restarts (**learning**)

RESTARTS – USAGE

- In SAT it is used in the best algorithms (zChaff)
  - Quick restarts
  - Learning between restarts (registration of conflict clauses)
  - Randomization (controlled)

- In CSP it is used in the best algorithms
  - Starts to be used
  - But with some problems
    - Heuristics without randomization
    - Conflict heuristics, with no-goods, but restarts
    - Generally, there is a deficient combination of techniques

-