Chapter S:I

I. Introduction

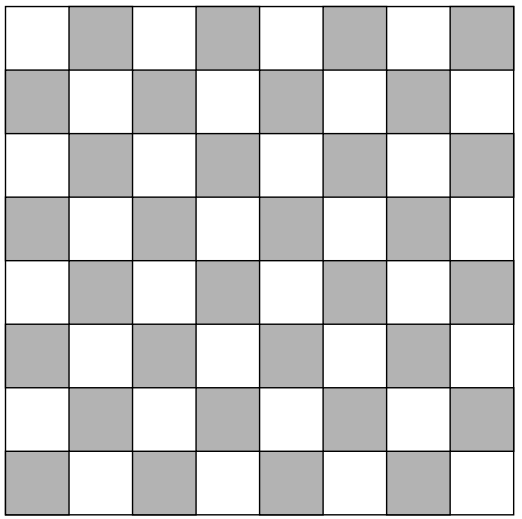
- Examples for Search Problems
- Search Problem Abstraction
- Examples for AND-OR Search Problems

Examples for Search Problems

8-Queens Problem: Local Search

Task: Placement of queens on a board without threats.

Simplest method: Trial and Error

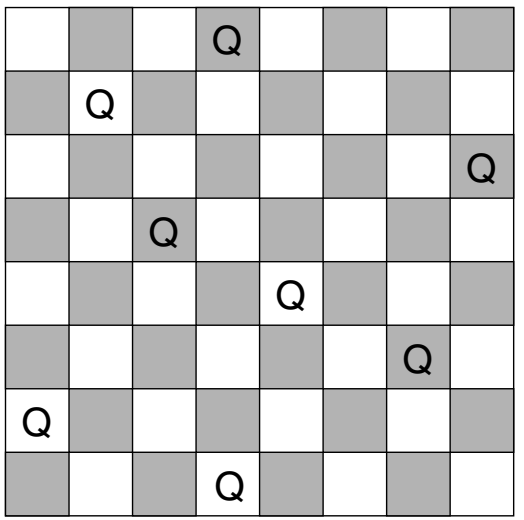


Examples for Search Problems

8-Queens Problem: Local Search (continued)

Task: Placement of queens on a board without threats.

Simplest method: Trial and Error



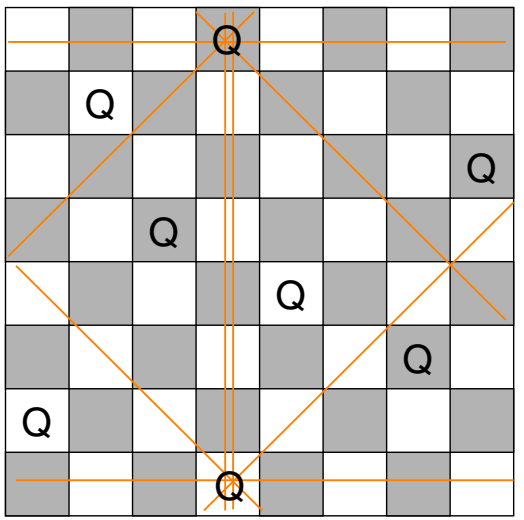
- A solution candidate is a placement of 8 queens.
- Solution candidates can be generated easily.
- Solution candidates can be checked easily.
- Problem: Search space of solution candidates is large, number of solutions is small.
- Systematic approach: Enumerate solution candidates thus exhausting the search space.

Examples for Search Problems

8-Queens Problem: Local Search (continued)

Task: Placement of queens on a board without threats.

Simplest method: Trial and Error



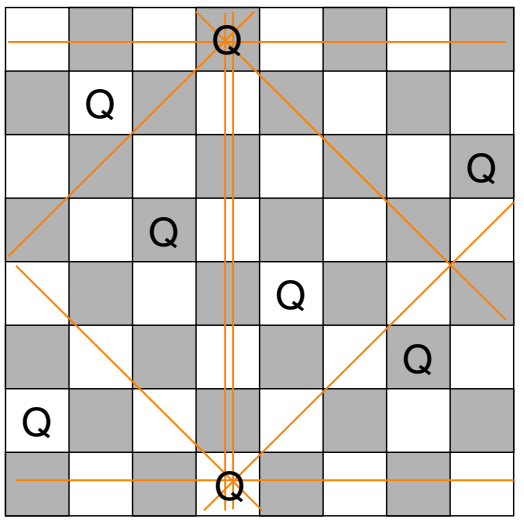
- A solution candidate is a placement of 8 queens.
- Solution candidates can be generated easily.
- Solution candidates can be checked easily.
- Problem: Search space of solution candidates is large, number of solutions is small.
- Systematic approach: Enumerate solution candidates thus exhausting the search space.

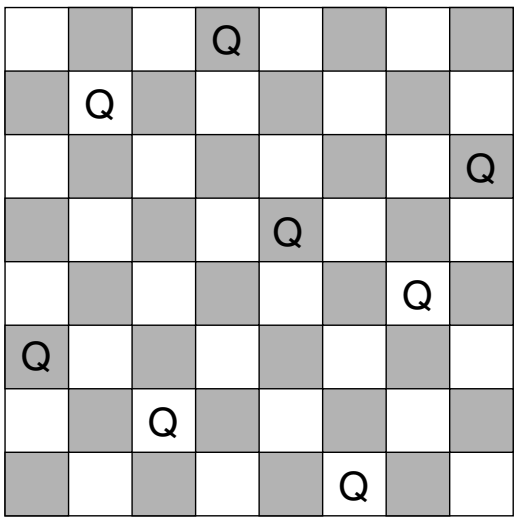
Examples for Search Problems

8-Queens Problem: Local Search (continued)

Task: Placement of queens on a board without threats.

Simplest method: Trial and Error





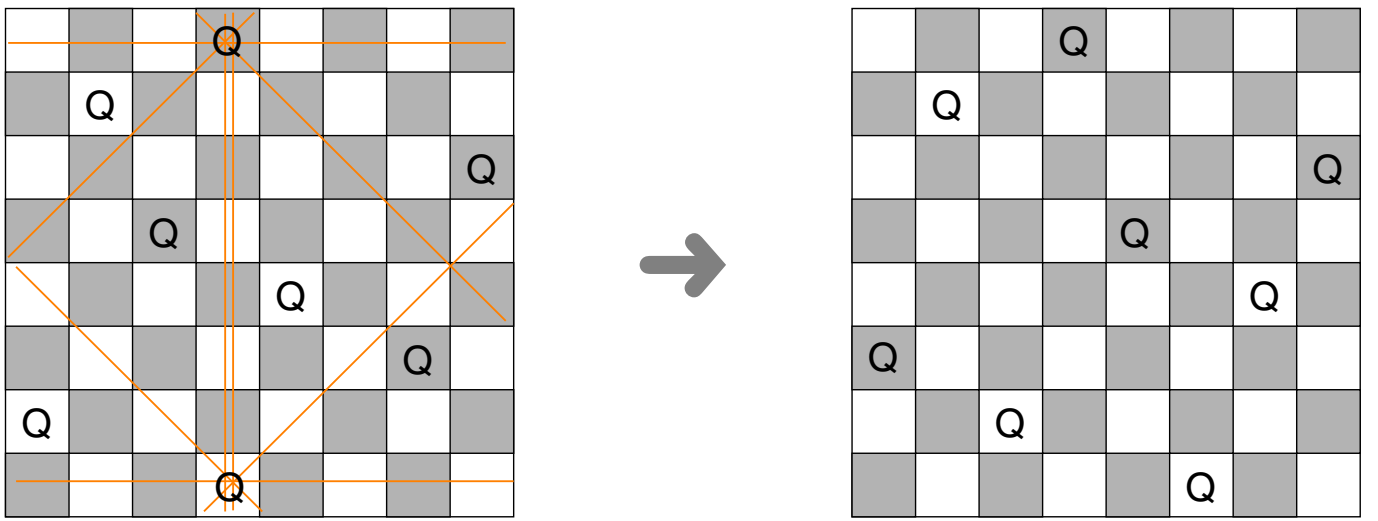
- A solution candidate is a placement of 8 queens.
- Solution candidates can be generated easily.
- Solution candidates can be checked easily.
- Problem: Search space of solution candidates is large, number of solutions is small.
- Systematic approach: Enumerate solution candidates thus exhausting the search space.

Examples for Search Problems

8-Queens Problem: Local Search (continued)

Task: Placement of queens on a board without threats.

Local search approach: Test and Improve



- A solution candidate is a placement of 8 queens. An initial candidate is given.
- Operators generate a neighborhood of solution candidates; one of it is selected.
- A heuristic function determines the cost/merit of a solution candidate.
- Local search algorithm example: Hill-Climbing
- Problem: Local optima cannot be overcome.

Remarks:

- The 'minimum-conflicts' heuristic – moving the piece with the largest number of conflicts to the square in the same column where the number of conflicts is smallest – is particularly effective: it finds a solution to the 1,000,000 queen problem in less than 50 steps on average. . . . A 'reasonably good' starting point can for instance be found by putting each queen in its own row and column so that it conflicts with the smallest number of queens already on the board. [Wikipedia (access 2021/09/08)]
- A hill-climbing algorithm will try to minimize the number of conflicts on the board. The resulting configuration of the queens may be a local minimum in its neighborhood.

Examples for Search Problems

8-Queens Problem: Local Search (continued)

Problem: n-Queens (8-Queens as special case)

Instance: Empty chessboard of size $n \times n$ and n queens to place.

Solution: Board positions for all queens
so that no two queens threaten each other.

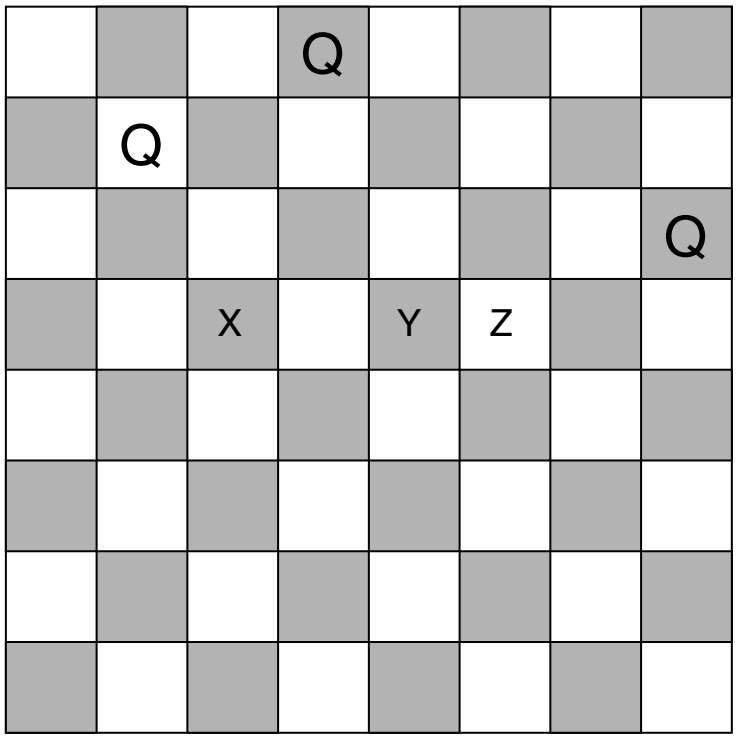
Local search algorithmization

1. Encoding of solution candidates:
   (A2, B5, C3, D8, E7, F6, G4, H1)
2. Cost/merit function for solution candidates:
   Number of queens in conflict.
3. Operators:
   Compute neighboring solution candidates, e.g. by changing the position of a queen in its row.
4. Hillclimbing algorithm:
   Continue with a most promising solution candidate in the neighborhood.

Examples for Search Problems

8-Queens Problem: Solution Construction

Constructive approach: Incremental placement of queens.

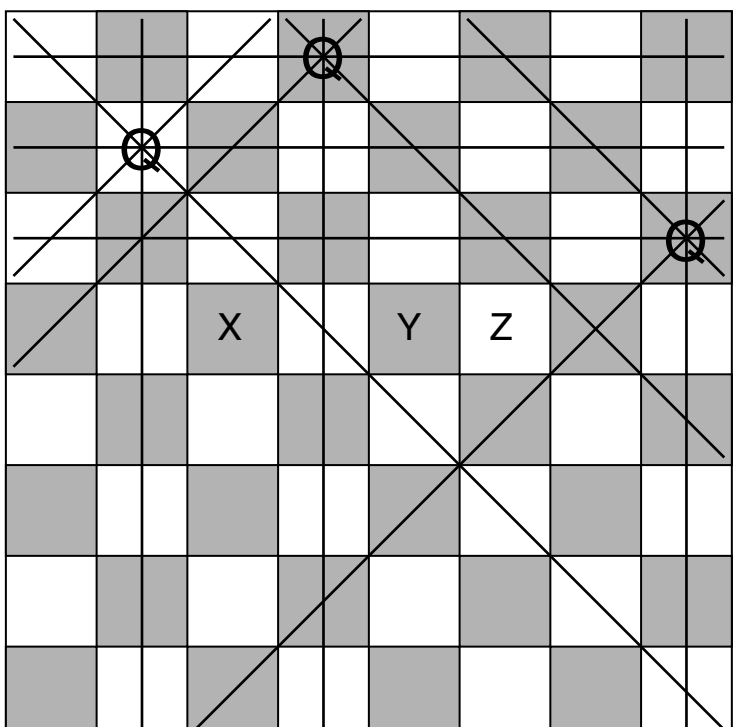


Examples for Search Problems

8-Queens Problem: Solution Construction (continued)

Constructive approach: Incremental placement of queens.

Analysis of the current board: Solution found? Meaningful next placements?

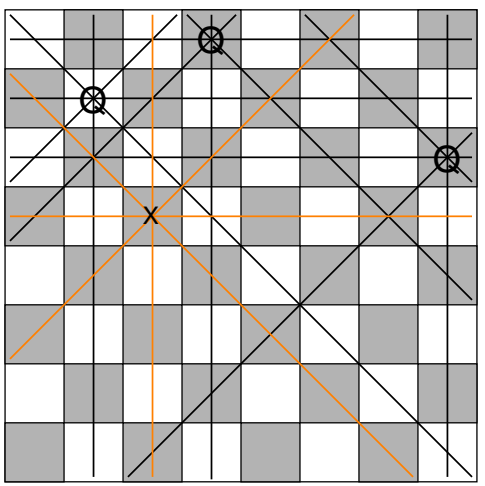


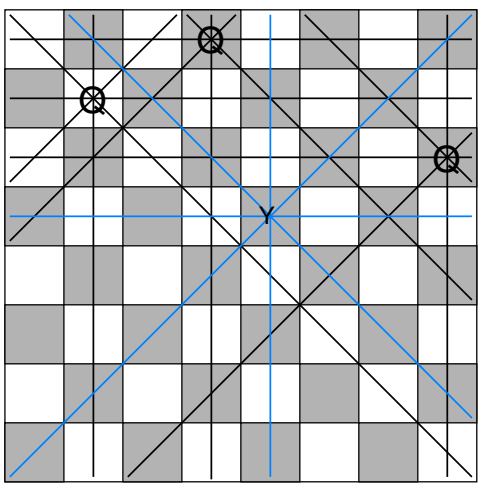
Examples for Search Problems

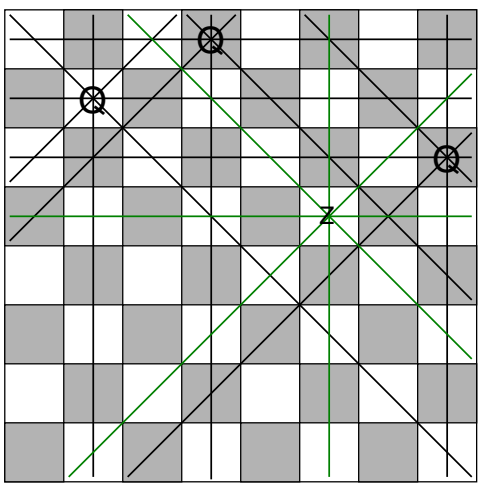
8-Queens Problem: Solution Construction (continued)

Constructive approach: Incremental placement of queens.

Analysis of possible successor boards: Best successor?







- A candidate is a placement of some first queens.
- Each queen defines *constraints* (restrictions).
- *Monotone* situation: constraint violations cannot be repaired.

Examples for Search Problems

8-Queens Problem: Solution Construction (continued)

Desired: A heuristic for the next queen's placement.

Idea: Which placement x is least restrictive regarding future decisions?

Examples for Search Problems

8-Queens Problem: Solution Construction (continued)

Desired: A heuristic for the next queen's placement.

Idea: Which placement x is least restrictive regarding future decisions?

- Heuristic 1: Maximize $h_1(x) =$ number of unattacked cells.

$$\begin{aligned} h_1(X) &= 8 \\ h_1(Y) &= 9 \\ h_1(Z) &= 10 \end{aligned}$$

- Heuristic 2: Maximize $h_2(x) = \min\{uc(x,r) \mid r \text{ is row without queen}\}$, where $uc(x,r)$ is the number of unattacked cells in row r if queen on x.

$$\begin{aligned} h_2(X) &= 1 \\ h_2(Y) &= 1 \\ h_2(Z) &= 2 \end{aligned}$$

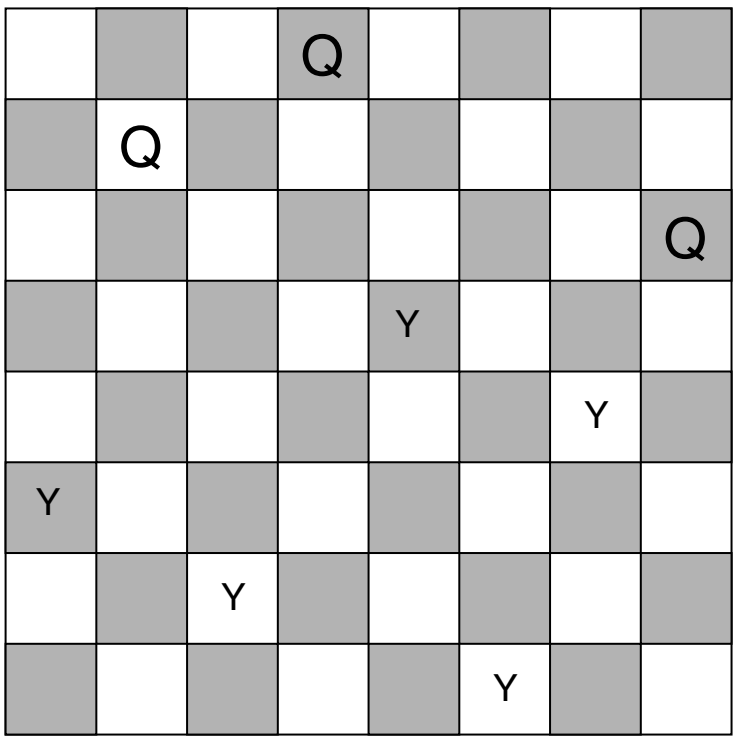
Remarks:

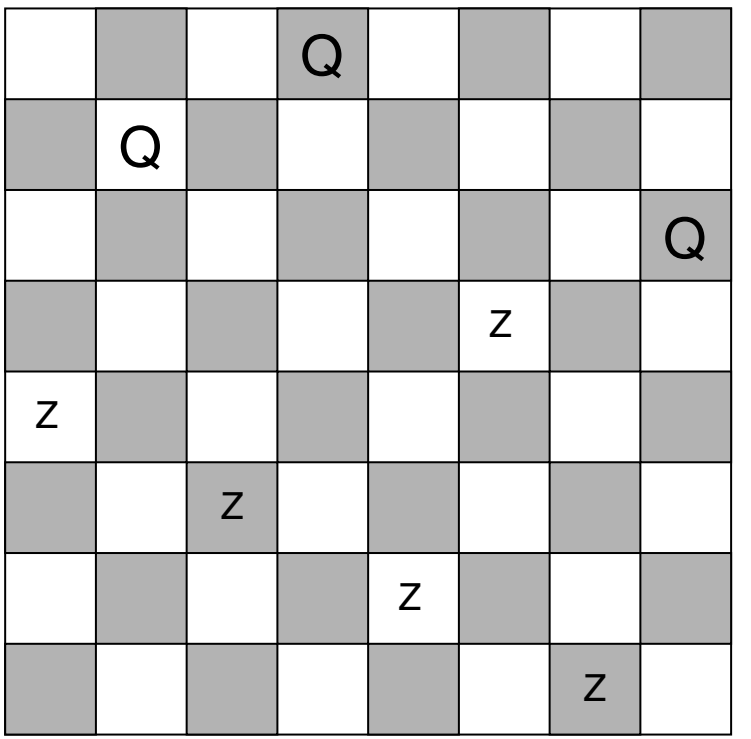
- The described idea became known in the field of Artificial Intelligence as the *Principle of Least Commitment*.
- Both 8-Queens heuristics assess the board after the next placement.
- Both 8-Queens heuristics compute merit values (not costs).
- These heuristics are helpful when solving the 8-Queens problem by greedy best-first search or informed depth-first search: successor states are processed in order of their promise which is defined by the heuristic.
- The 8-queens problem is an example of a problem that ideally is solved with a direct search approach. Among all placements of 8 queens we want to minimize the number of threatened queens. We translate this problem into one that allows a constructive approach: "Find a sequence of placements of single queens that ends with a desired board configuration." So, we are searching for a sequence of steps, although we are only interested in where the last step leads us to.

Examples for Search Problems

8-Queens Problem: Solution Construction (continued)

Possible solutions:





Examples for Search Problems

8-Queens Problem (continued)

Comparison of h_1 and h_2:

- consider computational effort
- consider early recognition of dead ends:

  Predicate $\perp_f (x)$, $x \in \{X, Y, Z\}$: "x is a dead end under f"

$$\perp_{h_1} (x) = \begin{cases} \textit{True} & \text{If } h_1(x) < \text{number of remaining rows,} \\ \textit{False} & \text{Otherwise.} \end{cases}$$

$$\perp_{h_2} (x) = \begin{cases} \textit{True} & \text{If } h_2(x) = 0, \\ \textit{False} & \text{Otherwise.} \end{cases}$$

Heuristic 2 is more general than Heuristic 1: $\perp_{h_1} (x) \Rightarrow \perp_{h_2} (x)$

Compare the definition of "more general" in [ML:II Concept Learning: Search in Hypothesis Space].

Examples for Search Problems

8-Queens Problem: Solution Construction (continued)

Problem: n-Queens (8-Queens as special case)

Instance: Empty chessboard of size $n \times n$ and n queens to place.

Solution: Board positions for all queens
so that no two queens threaten each other.

Algorithmization of the constructive approach

1. Encoding of partial solutions (facilitated by encoding of candidates):
   (A2, B5, C3, *, *, *, *, *)
2. Cost/merit function for partial solutions:
   E.g. function h_1 or h_2.
3. Operators:
   Placement of a queen in the next empty row (without resulting threats).
4. Greedy algorithm:
   Continue with a most promising placement of the next queen.

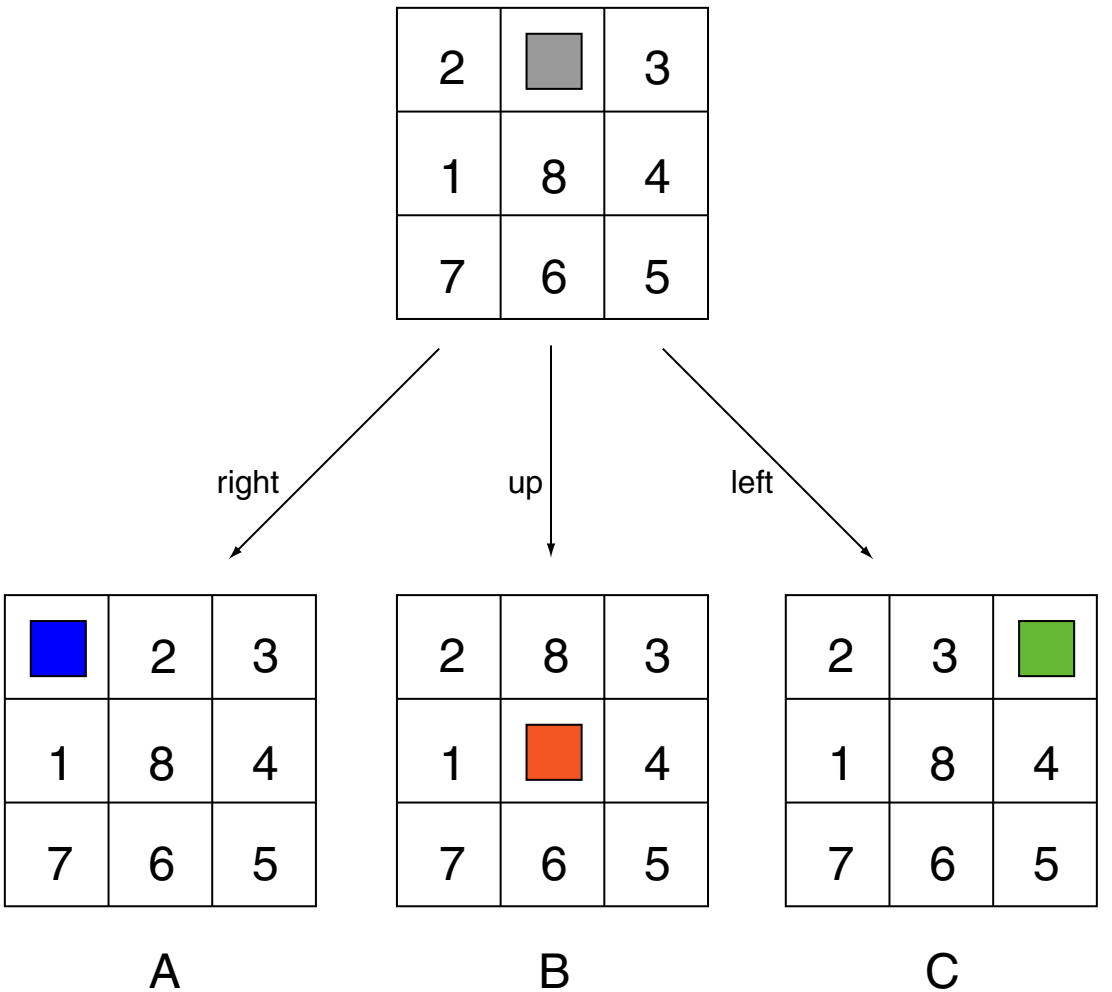
Remarks:

- Obviously, the encoding of partial solutions is based on the encoding of solution candidates: (A2, B5, C3, D8, E7, F6, G4, H1)
- A greedy algorithm will also try to minimize the number of conflicts on the board. The resulting configuration of the queens, however, might not be a local minimum in its neighborhood, if neighborhood is defined by changing positions of not only the last queen.
- The 8-Queens problem can be generalized by allowing boards of different sizes or shapes. Then, additional parameters are needed in a description of a problem instance (not just $n \in \mathbf{N}$ for $n \times n$ chessboards and n queens).
- In the 8-Queens problem we are interested in board configurations. Such a configuration is easily computed from the sequence of placements, but the sequence contains additional information that is not needed (e.g., the order of placements). The encoding of solution candidates as lists enables a step-by-step construction of solution candidates.

Examples for Search Problems

8-Puzzle Problem

Task: Moving tiles on the board to reach a target state.



Examples for Search Problems

8-Puzzle Problem (continued)

Constructive approach: Adding moves one by one to an initially empty sequence.

Desired: A heuristic for the next move.

Idea: Which move x minimizes the “distance” to the goal state?

Examples for Search Problems

8-Puzzle Problem (continued)

Constructive approach: Adding moves one by one to an initially empty sequence.

Desired: A heuristic for the next move.

Idea: Which move x minimizes the "distance" to the goal state?

- Heuristic 1: Minimize $h_1(x)$, with $h_1(x) =$ number of non-matching tiles.

$$\begin{aligned} h_1(A) &= 2 \\ h_1(B) &= 3 \\ h_1(C) &= 4 \end{aligned}$$

- Heuristic 2: Minimize $h_2(x)$, with $h_2(x) =$ sum of city block (Manhattan) distances of non-matching tiles.

$$\begin{aligned} h_2(A) &= 2 \\ h_2(B) &= 4 \\ h_2(C) &= 4 \end{aligned}$$

Remarks:

- In the 8-Puzzle problem the goal state (final configuration) is known, while in the 8-Queens problem the goal state is unknown and has to be determined.
- Both 8-Puzzle heuristics assess the unsolved rest problem.
- Both 8-Puzzle heuristics compute cost values (not merits).
- “Wrong” decisions can be repaired.
- Infinitely long move sequences are possible.

Examples for Search Problems

8-Puzzle Problem (continued)

Problem: $(n^2 - 1)$-Puzzle (8-Puzzle as a special case, fixed set of possible moves)

Instance: q_s. Initial board configuration.
q_γ. Final board configuration.

Solution: A sequence of moves that transforms the initial configuration q_s into the final configuration q_γ.

Algorithmization of the constructive approach

1. Encoding of partial solutions:
   $(m_i)_{i=1}^N$ with $m_i \in \{\text{up}, \text{down}, \text{left}, \text{right}\}$, $N \in \mathbf{N}$ (initial part to be continued)
2. Cost/merit function for partial solutions:
   E.g. function h_1 or h_2.
3. Operators:
   Extend the sequence of moves by a legal move.
4. Greedy algorithm:
   Continue with a most promising next move.

Remarks:

- Obviously, the number of possible sequences of moves is infinite. Therefore, enumerating and testing the sequences is an approach to solving the problem without guarantee of termination. But since the number of board configurations is finite, this number can serve as an upper bound to the length of move sequences we have to consider. If a board configuration occurs twice when performing the moves, there is a shorter sequence resulting in the same outcome.
- The 8-Puzzle problem can be generalized by allowing boards of different sizes or shapes. Then, additional parameters are needed in a description of a problem instance (e.g., $m, n \in \mathbf{N}$ for rectangular boards with m rows and n columns).
- Additional restrictions can be placed on solutions to the 8-Puzzle problem. The task can be the identification of a shortest sequence or of a sequence that is shorter than a given maximum length.
- The following decision problem is NP-complete:

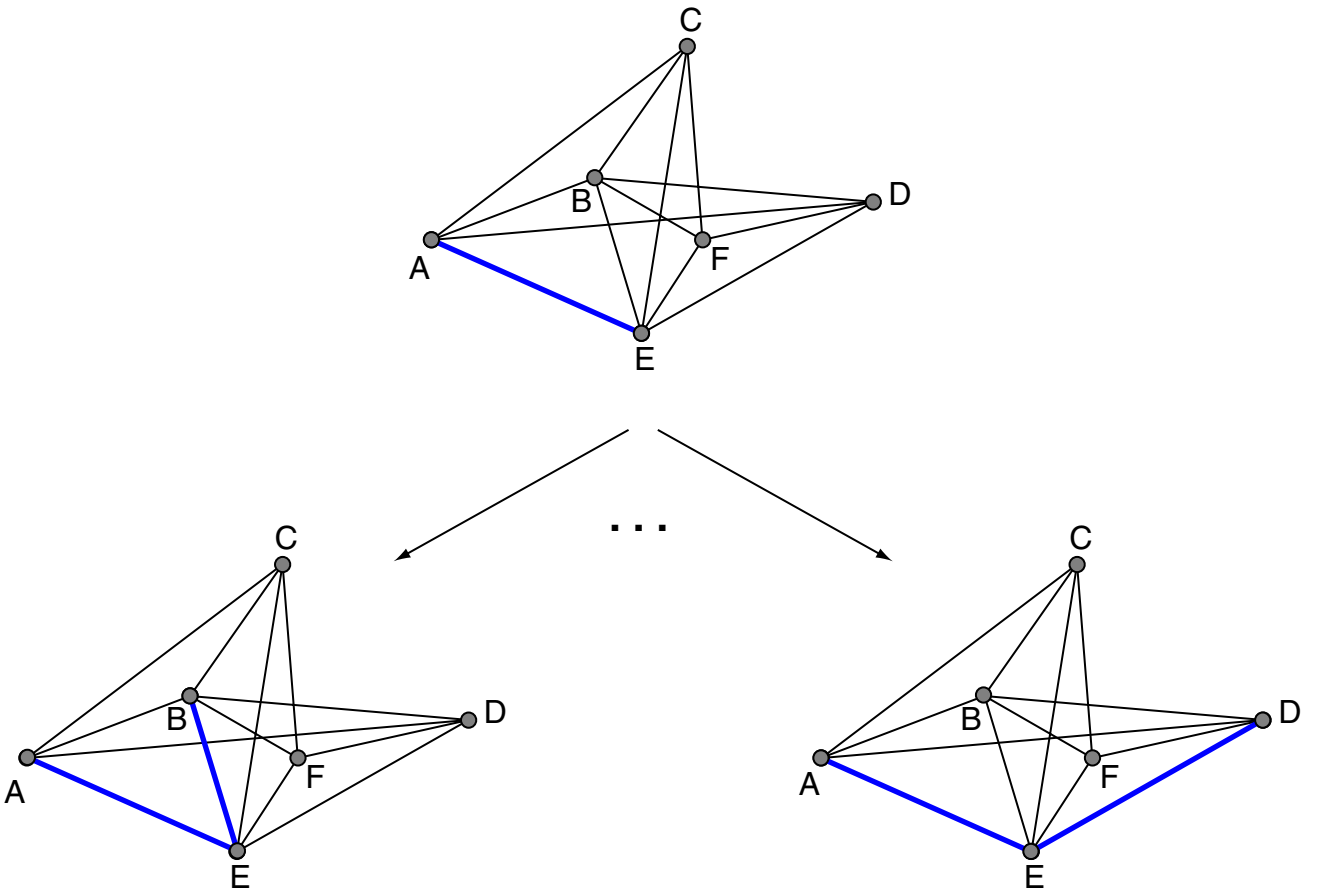
  Let be given an initial configuration and a target configuration of $n^2 - 1$ tiles on an $n \times n$ board, $n \in \mathbf{N}$, and let be given a bound k, $k \in \mathbf{N}$.
  Is there a sequence of at most k moves that transforms the initial configuration into the target configuration?

  The corresponding optimization problem of finding a shortest move sequence is NP-hard.
  [Ratner, Warmuth: *Finding a Shortest Solution for the $N \times N$ Extension of the 15-PUZZLE Is Intractable*. AAAI-86, pp. 168–172.]

Examples for Search Problems

Traveling Salesman Problem (TSP)

Task: Visiting each town exactly once in a round trip (Hamiltonian cycle).



Examples for Search Problems

Traveling Salesman Problem (continued)

Constructive approach: Adding towns one by one to a trip (=seq. of towns to visit).

Desired: A heuristic for the most suited next town x.

Idea: What is a lower bound for the shortest round trip possible then?

Examples for Search Problems

Traveling Salesman Problem (continued)

Constructive approach: Adding towns one by one to a trip (=seq. of towns to visit).

Desired: A heuristic for the most suited next town x.

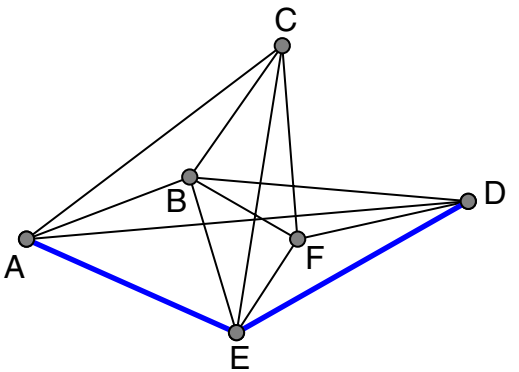
Idea: What is a lower bound for the shortest round trip possible then?

- Heuristic 1: For the remaining vertices $V' \subset V$ minimize the edge weight of a subgraph on V' whose degree is ≤ 2.

- Heuristic 2: For the remaining vertices $V' \subset V$ compute the edge weight of a minimum spanning tree.

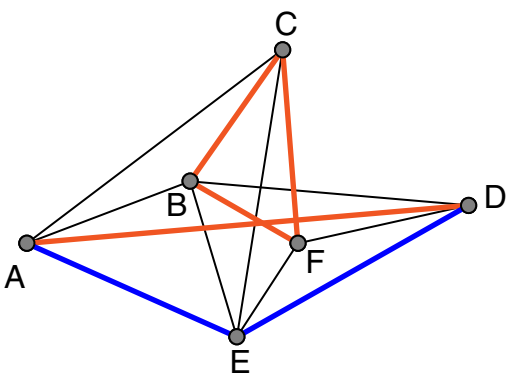
Examples for Search Problems

Traveling Salesman Problem (continued)

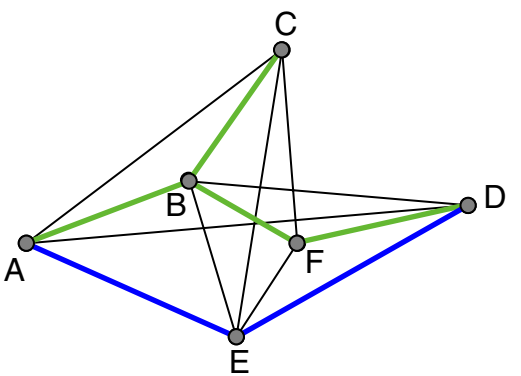


Current State

Estimation of
Completion Cost



Heuristic 1:
Cheapest Degree-2 Graph



Heuristic 2:
Minimum Spanning Tree

Remarks:

- Both TSP heuristics compute cost values (not merits).
- Both TSP heuristics are *optimistic*: they underestimate the cost of the rest tour.
- The cost of a candidate is computed as the sum of true cost of the completed part of the tour plus the estimated cost of the rest tour.

Examples for Search Problems

Traveling Salesman Problem (continued)

Problem: Traveling Salesman Problem

Instance: G. A weighted finite graph.
A. Start vertex for the round-trip.

Solution: A shortest cycle (path starting and ending in A) that visits each other vertex of G exactly once.

Algorithmization of the constructive approach

1. Encoding of partial solutions:
   $(v_i)_{i=0}^{N}$ with $v_0 = A$, v_i vertices in G, $N \in \mathbf{N}$ (initial part to be continued)
2. Cost/merit function for partial solutions:
   E.g. function h_1 or h_2.
3. Operators:
   Extend a path by appending some adjacent vertex.
4. Greedy algorithm:
   Continue with a most promising next vertex.

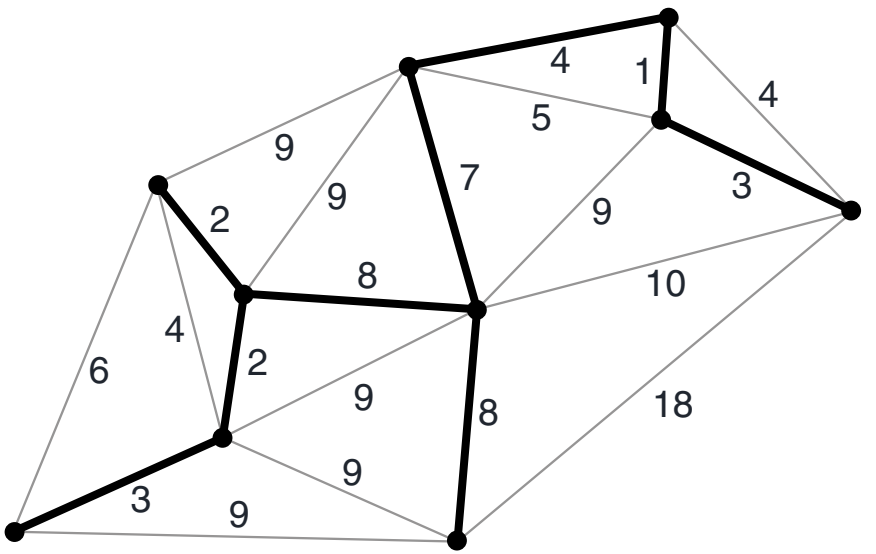
Remarks:

- Obviously, there is only a finite number of paths without cycles. Therefore, enumerating and testing these paths is a possible approach to solving the problem.
- The traveling salesman problem TSP is NP-hard. Additional restrictions can be placed on solutions to the problem. The task can be the identification of a round-trip that is shorter than a given maximum length. This decision problem is NP-complete.

Examples for Search Problems

Minimum Spanning Tree Problem

Task: Find a spanning tree of minimum weight in an undirected graph.



Constructive approach: Adding edges one by one to create a subtree of the graph.

Desired: A heuristic for the most suited next edge e.

Idea: Add a cheapest edge to some vertex not already in the subtree?

Examples for Search Problems

Minimum Spanning Tree (continued)

Prim's Algorithm: (assuming a connected graph)

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge:
   of the edges that connect the tree to vertex not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 until all vertices are in the tree.

Kruskal's Algorithm: (assuming a connected graph)

1. Create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. Create a set S containing all the edges in the graph.
3. Decrease the size of the forest considering edge by edge:
   Remove an edge with minimum weight from S and, if the removed edge connects two different trees in F, then replace these trees in F by the combined two tree (using the edge).
4. Repeat step 3until S is empty or F consists of a single tree..

Q. How to characterize these algorithms with respect to

- the type of search and
- the heuristics used?

Remarks:

- For a graph $G = (V, E)$ Prim's algorithm runs in time $\mathcal{O}(|V|^2)$ (using an adjacency matrix) or $\mathcal{O}(|E| + |V| \log |V|)$ (using Fibonacci heaps) and Kruskal's algorithm in time $\mathcal{O}(|E| \log |E|)$. [Wikipedia]

Examples for Search Problems

MaxSAT Problem

Task: Find a truth assignment that satisfies most of the clauses.

Propositional formula α in conjunctive normal form (CNF)

$$
\begin{array}{lllllllllll}
 & (& \neg x_1 & \vee & x_3 & \vee & x_7 & \vee & \neg x_9 & &) \\
\wedge & (& x_2 & \vee & \neg x_3 & \vee & \neg x_5 & \vee & x_6 & \vee & \neg x_8 &) \\
\wedge & (& \neg x_2 & \vee & x_7 & \vee & x_8 & & & &) \\
\wedge & (& x_1 & \vee & \neg x_2 & \vee & \neg x_3 & \vee & x_9 & &) \\
 & & & & & \vdots & & & & & \\
\wedge & (& \neg x_3 & \vee & x_4 & \vee & \neg x_6 & \vee & x_7 & \vee & \neg x_8 &)
\end{array}
$$

Examples for Search Problems

MaxSAT Problem (continued)

Task: Find a truth assignment that satisfies most of the clauses.

Propositional formula α in conjunctive normal form (CNF)

$$
\begin{array}{llllllllll}
 & (\neg x_1 & \vee & x_3 & \vee & x_7 & \vee & \neg x_9 & &) \\
\wedge & (x_2 & \vee & \neg x_3 & \vee & \neg x_5 & \vee & x_6 & \vee & \neg x_8) \\
\wedge & (\neg x_2 & \vee & x_7 & \vee & x_8 & & & &) \\
\wedge & (x_1 & \vee & \neg x_2 & \vee & \neg x_3 & \vee & x_9 & &) \\
 & & & & \vdots & & & & & \\
\wedge & (\neg x_3 & \vee & x_4 & \vee & \neg x_6 & \vee & x_7 & \vee & \neg x_8) \\
\end{array}
$$

Local Search: Start from an arbitrary truth assignment for the variables in α and iteratively flip truth values assigned to variables.

Heuristics:

GSAT: Flip the truth value of a variable so that the number of satisfied clauses increases the most.

WalkSAT: Flip the truth value of a variable in an unsatisfied clause.

Examples for Search Problems

MaxSAT Problem (continued)

Problem: MaxSAT

Instance: α. Propositional formula in CNF.

Solution: Truth assignment that satisfies most of the clauses in α.

Local search algorithmization

1. Encoding of solution candidates:
   $(x_1 \mapsto \textit{true}, x_2 \mapsto \textit{false}, \ldots, x_n \mapsto \textit{true})$
2. Cost/merit function for solution candidates:
   Number of clauses satisfied.
3. Operators:
   Flip the truth assignment of a variable.
4. Hillclimbing algorithm:
   Try a most promising flipping.

Remarks:

- There is no unique way of constructing an assignment by iterated flipping of truth assignments for variables.
- Restarting the algorithm with a new random assignment can help when getting stuck in a local maximum of numbers of satisfied clauses.
- The optimization problem MaxSAT requires for a given CNF formula to determine the maximum number of clauses that can be satisfied by any truth assignment. Since this maximum number is not known in advance, the only termination criterion is that all clauses are satisfied. Therefore, either all possible truth assignments have to be tested or the algorithm may have found only a local maximum.
- The satisfiability problem SAT (Decision problem: Is a given CNF Formula satisfiable?) is NP-complete, the optimization problem MaxSAT is NP-hard. (If there were a polynomial time algorithm for MaxSAT, we would be able to decide SAT in polynomial time.)

Examples for Search Problems

MaxSAT Problem (continued)

Task: Find a truth assignment that satisfies most of the clauses.

Propositional formula α in conjunctive normal form (CNF)

$$
\begin{array}{lllllllllll}
 & (\neg x_1 & \vee & x_3 & \vee & x_7 & \vee & \neg x_9 & & &) \\
\wedge & (x_2 & \vee & \neg x_3 & \vee & \neg x_5 & \vee & x_6 & \vee & \neg x_8 &) \\
\wedge & (\neg x_2 & \vee & x_7 & \vee & x_8 & & & & &) \\
\wedge & (x_1 & \vee & \neg x_2 & \vee & \neg x_3 & \vee & x_9 & & &) \\
 & & & & & \vdots & & & & & \\
\wedge & (\neg x_3 & \vee & x_4 & \vee & \neg x_6 & \vee & x_7 & \vee & \neg x_8 &)
\end{array}
$$

Constructive approach: Incrementally assign truth values to variables until all variables in α have a truth value.

Heuristics:

E.g. assign a truth value to a variable that satisfies most of the unsatisfied clauses.

Examples for Search Problems

MaxSAT Problem (continued)

Problem: MaxSAT

Instance: α. Propositional formula in CNF.

Solution: Truth assignment that satisfies most of the clauses in α.

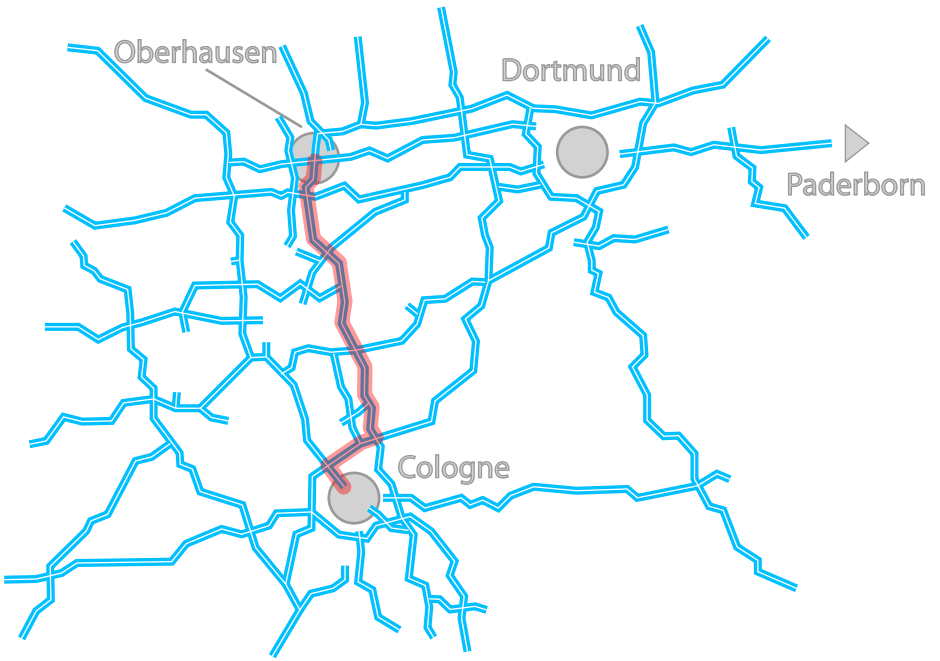
Algorithmization of the constructive approach

1. Encoding of partial solutions:
   $(x_1 \mapsto \textit{true}, x_2 \mapsto \textit{false}, x_3 \mapsto *, \ldots, x_n \mapsto *)$
2. Cost/merit function for partial solutions:
   Number of clauses satisfied.
3. Operators:
   Assign a truth value to an unassigned variable.
4. Greedy algorithm:
   Continue with a most promising assignment.

Examples for Search Problems

Road-Map Problem

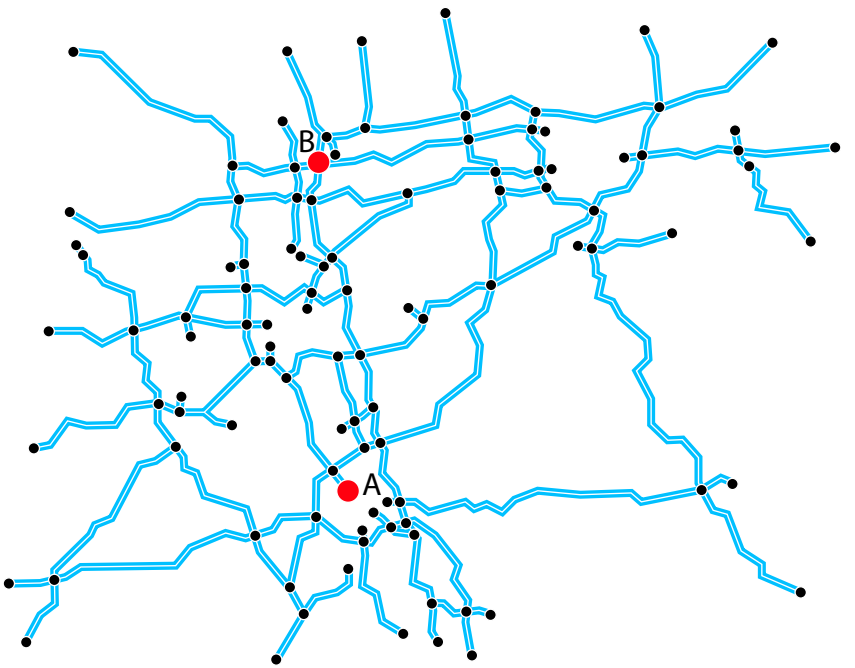
Task: Find for a shortest freeway connection from Cologne to Oberhausen.



Examples for Search Problems

Road-Map Problem (continued)

Task using a graph model: Find a shortest path from A to B.



- Vertices represent crossings, junctions, endpoints, ...
- Edges represent roads in between.
- Edge labels denote the length of a road between two vertices.

Examples for Search Problems

Road-Map Problem (continued)

Problem: Road-Map Problem

Instance: G. A weighted finite graph representing a map with distances.
A, B. Start vertex and end vertex for the trip.

Solution: A shortest path starting in A and ending in B.

Local search algorithmization

1. Encoding of solution candidates:
   vertex sequences $(v_i)_{i=0}^{N}$ with $v_0 = A$, $v_N = B$, v_i vertices in G, $N \in \mathbf{N}, N \geq 1$
2. Cost/merit function for solution candidates:
   Sum of road section lengths for pairs (v_i, v_{i+1}) in the vertex sequence (∞ if there is no edge between v_i and v_{i+1})
3. Operators:
   Replace a subsequence of vertices by some other sequence of vertices.
4. Heuristic:
   Try a most promising sequence first.

Remarks:

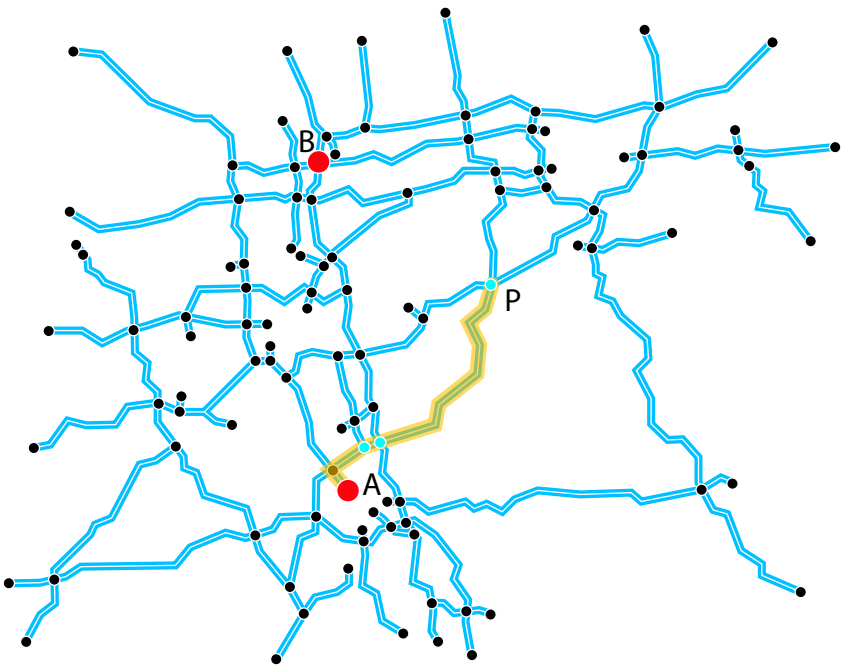
- It would be helpful, if we could use paths (or even acyclic paths from A to B) instead of arbitrary sequences of vertices. In this case, however, the construction of solution candidates and their neighborhood is no longer trivial.
- Operators could be more deliberately designed so that they do not result in sequences containing multiple occurrences of a single vertex.
- Instead of using ∞ as distance for missing edges we can use a ¨high¨ value.
- The step-by-step construction of paths from A to B is the more natural approach.

Examples for Search Problems

Road-Map Problem (continued)

Task using a graph model: Find a shortest path from A to B.

Constructive approach: Adding a further edge to a path.



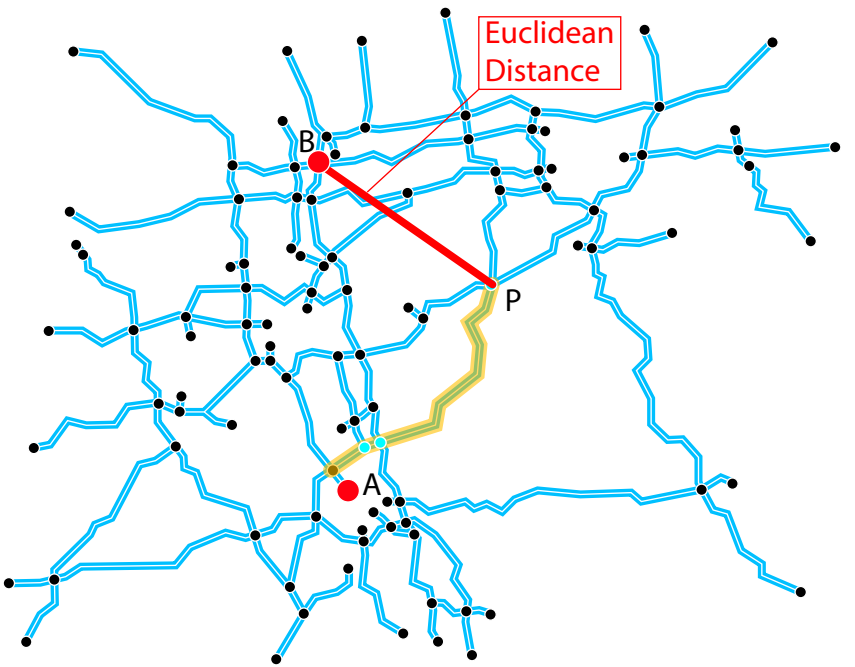
Desired: A heuristic for selection of a promising path to continue.

Examples for Search Problems

Road-Map Problem (continued)

Task using a graph model: Find a shortest path from A to B.

Constructive approach: Adding a further edge to a path.



Desired: A heuristic for the selection of a promising path to continue.

Idea: Which path takes us next to our target?

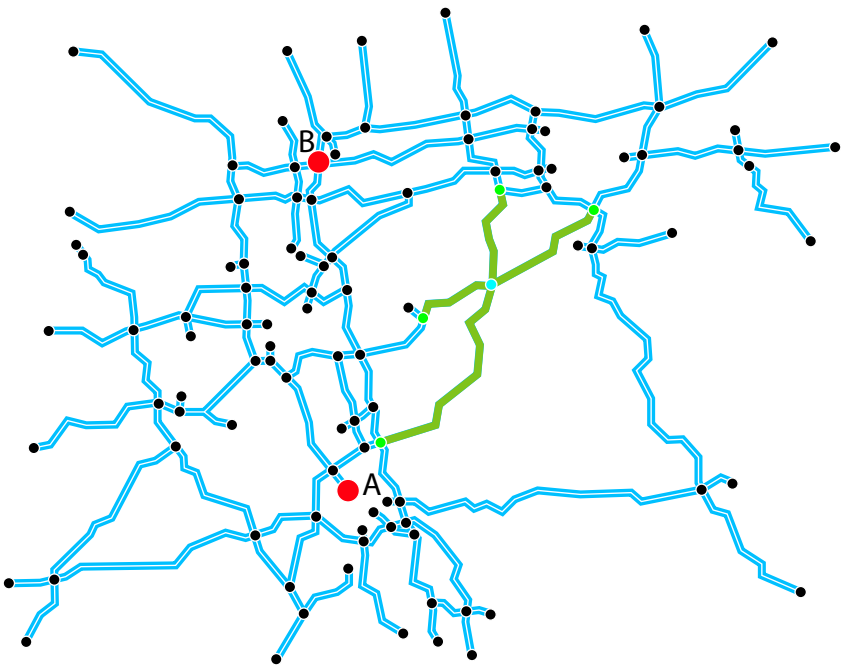
Heuristic: Minimize Euclidean distance between current position and target.

Examples for Search Problems

Road-Map Problem (continued)

Task using a graph model: Find a shortest path from A to B.

Constructive approach: Adding a further edge to a path.



Possible continuations of the selected path will be considered as new candidates.

Examples for Search Problems

Road-Map Problem (continued)

Problem: Road-Map Problem

Instance: G. A weighted finite graph representing a map with distances.
A, B. Start and end vertices for the trip.

Solution: A shortest path starting in A and ending in B.

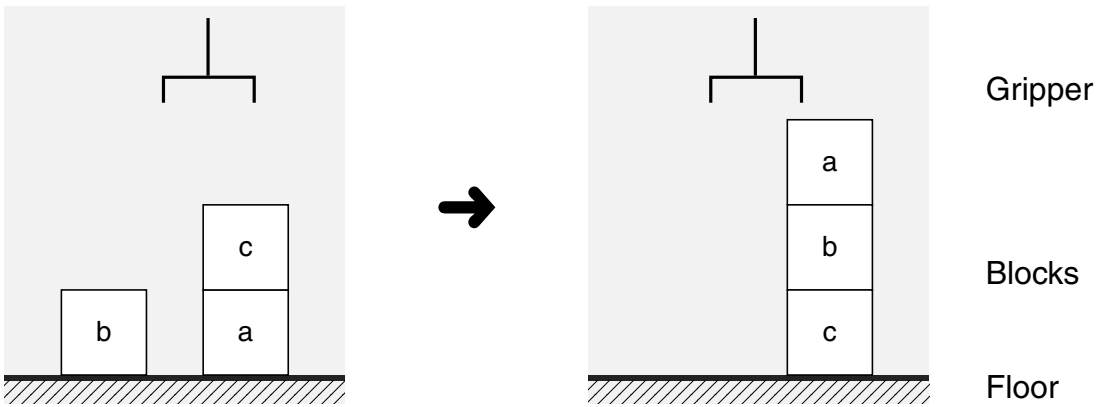
Algorithmization of the constructive approach

1. Encoding of partial solutions:
   Paths $(v_i)_{i=0}^{N}$ in G with $v_0 = A$, v_i vertices in G, $N \in \mathbf{N}, N \geq 1$
2. Cost/merit function for partial solutions:
   Euclidean distance between v_N and B
   (neglecting the length of the path to v_N).
3. Operators:
   Extend the path by a move to an adjacent junction.
4. Greedy algorithm:
   Continue with a most promising junction.

Examples for Search Problems

Blocks-World Planning

Task: Rearrange the blocks.

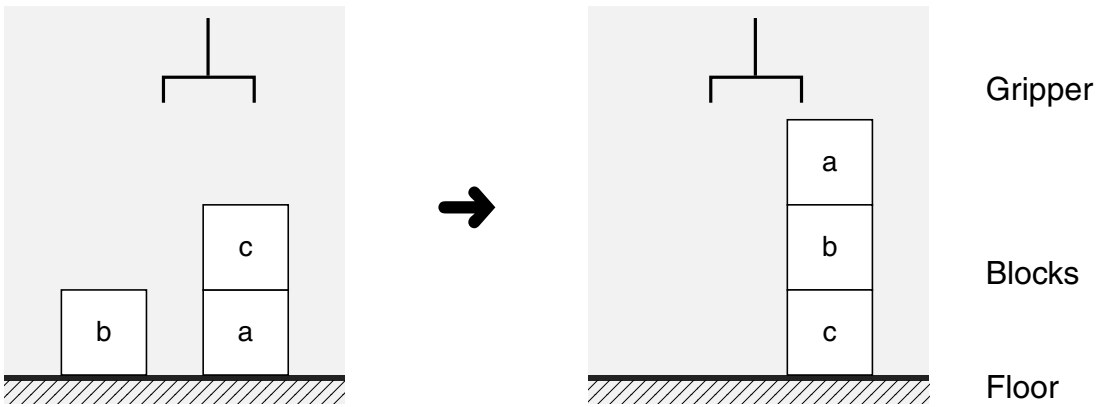


A gripper moves single blocks (clear blocks with no other block on top) to new positions either on top of a clear block or on the floor.

Examples for Search Problems

Blocks-World Planning

Task: Rearrange the blocks.



A gripper moves single blocks (clear blocks with no other block on top) to new positions either on top of a clear block or on the floor.

Desired: A heuristic for the next move.

Idea: Which move x minimizes the "distance" to the target configuration?

Heuristic: E.g., use the number of moves needed ia a simplified version of the problem. Gross simplification: Moves have no preconditions.

Examples for Search Problems

Blocks-World Planning (continued)

Problem: Blocks-World Planning (fixed set of possible moves)

Instance: q_s. Initial block configuration.
q_γ. Final block configuration.

Solution: A sequence of moves that transforms the initial configuration q_s into the final configuration q_γ.

Algorithmization of the constructive approach

1. Encoding of partial solutions:
   $(op_i)_{i=1}^N$ with op_i a gripper operation on some block, $N \in \mathbf{N}$ (initial part to be continued)
2. Cost/merit function for partial solutions
3. Operators:
   Extend the sequence of gripper operations by some possible next operation.
4. Greedy algorithm:
   Continue with a most promising next operation.

Remarks:

- Logical descriptions of Blocks World Planning states use predicates "*on*(x, y)" (semantics: block x is *on* block y) and "*onfloor*(x)" for the position of blocks.
- The problem instance depicted is called Sussman Anomaly. This anomaly illustrates the weakness of planning algorithms that try to solve a task by appending plans for achieving single properties (e.g., "*on*(b, c)" and "*on*(a, b)"). Optimum plans for these subgoals cannot be appended to each other to form a solution of the depicted instance.

Search-Problem Abstraction

Local Search Approach

Observations

- Problem settings are characterized by solution candidates.
- Task is to find a candidate satisfying given properties, e.g. solution or "best" solution.
- There is a set of well-defined steps to traverse from one candidate to another. By applying these steps the "neighborhood" of a candidate can be computed.
- Problem solving is done stepwise
- Starting candidates can be chosen arbitrarily.

➜ Systematic Approach:
Enumeration of candidates.

➜ Heuristic Approach:
Hill-Climbing with restarts.

Search Problem Abstraction

Local Search Approach (continued)

Formal Representation

(S, T, s, F)	Transition System (aka. State Transition System)
S	A set of solution candidates of a problem.
$T \subseteq S \times S$	A transition relation defining the neighboring solution candidates.
$s \in S$	An initial solution candidate to start from.
$F \subseteq S$	A set of solution candidates that are solutions.

Task: Find a solution candidate γ with $\gamma \in F$.

Search: Traverse S using T starting from s.

Knowledge: A function $f^* : S \to \mathbf{R}$ assigning cost / merit / utility values to candidat
such that F is the set of (nearly) optimum candidates in S.

Remarks:

- Candidates in S are often represented by feature vectors, e.g. a vector of positions (A2, B5, C3, D8, E7, F6, G4, H1) in the 8-queens example.
- Transitions in T are often defined by generic operators working on candidate representations, e.g. an operator can compute the next candidate in an enumeration:

  (A2, B5, C3, D8, E7, F6, G4, H2) for (A2, B5, C3, D8, E7, F6, G4, H1) and

  (A2, B5, C3, D8, E7, F6, G5, H1) for (A2, B5, C3, D8, E7, F6, G4, H8).
- The idea of using knowledge to guide search is that an evaluation function f^* describes a smooth landscape for s with respect to T.
- In constraint satisfaction problems (8-Queens), f could determine the number of threats in a solution candidate and thus estimate the closeness to a solution. Distance to a solution, on the other hand, could be used as an optimization criterion making a constraint satisfaction problem an optimization problem.

Search Problem Abstraction

Solution Construction

Observations

- Problem settings are characterized by (problem) states / (rest) problems.
- There is a set of well-defined steps to traverse from one state to another (problem solving steps).
- Task is to find a sequence / an "optimum" sequence of steps leading to a state satisfying given properties, e.g. goal state (solved problem).
- A state can be characterized by the question "What is (left) to be done?" (States correspond to remaining problems.)

➜ Systematic Approach:
Enumeration of step sequences.

➜ Heuristic Approach:
Greedy Search.

Search Problem Abstraction

Solution Construction (continued)

Formal Representation

(S, T, s, F)	Transition System (aka. State Transition System)
S	A set of states (the remaining problems).
$T \subseteq S \times S$	A transition relation defining the solution steps to solve the problem (problem solving / simplification steps).
$s \in S$	An initial state (start problem) to start from.
$F \subseteq S$	A set of states that are goals (solved problems).
Task:	Find a sequence of transitions leading from s to some γ with $\gamma \in F$ (sequence of steps solving the start problem). ➜ Find a path from s to γ in graph (S, T).
Search:	Traverse set of paths in graph (S, T) that are starting in s.
Knowledge:	A function $f : S^{<\mathbf{N}} \to \mathbf{R}$ assigning estimated cost / merit values to finite transition sequences (a sequence of problem solving / simplification steps given as state sequences) starting in s.

Remarks:

- States in S are often represented by feature vectors, e.g. a vector of positions (A2, B5, C3, *, *, *, *, *) in the 8-queens example. Remaining problem: Replace wildcards * by positions.
- Transitions in T are often defined by generic operators working on state representations, e.g. an operator can compute a next positioning of a queen from the current board:

  (A2, B5, C3, D8, *, *, *, *) for (A2, B5, C3, *, *, *, *, *).

  (Of course, this can be done in more or less sophisticated ways.)
- The idea of using knowledge to guide search is that an evaluation function f estimates optimum cost / merit of transition sequences continuing the given one and ending in a goal state.

Remarks:

- There is no guarantee that S is finite, not even enumerable. Usually, we deal with approximations (discretizations) of real world solution candidate sets.
- F must not be known explicitly. Usually, a test procedure is given implementing a function that returns `true` given some state s if and only if $s \in F$. For the 8-Puzzle Problem the goal state is known (solved problem: current configuration on the board is target configuration), for the 8-Queens Problem we usually don't know goal states (solved problem: positions for 8 queens without threats). In both examples the sequence of transitions (problem solving / simplification steps) that has to be determined is unknown.
- In planning, description languages are used to specify the state transition systems underlying a planning problem. A commonly used example is the Planning Domain Definition Language (PDDL), which was inspired by STRIPS and ADL.
  States are characterized by their properties and transitions (or actions) describe the changes in the properties of a state.
- In all above examples, a solution is incrementally constructed as a sequence of operator applications (next move, next town, ...).

Search Problem Abstraction

Solution Construction: Abstract Problem Setting

Problem: Reachability Problem for State Transition Systems

Instance: (S, T, s, F). A transition system.
s. Initial state.

Solution: A sequence of transitions leading from s to a state in F.

Algorithmization of the constructive approach

1. Encoding of partial solutions:
   $(s_i)_{i=0}^N$ with $s_0 = s$, s_i states in S, (s_i, s_{i+1}) in T (initial seq. to be continued).
2. Cost/merit function f for partial solutions.
3. Operators:
   Extend the sequence of states by a state reachable via a next transition in T.
4. Greedy algorithm:
   Continue with a most promising sequence.

Search Problem Abstraction

Questions in Problem Solving as Search

- What is the original task?
- What is the structure of solution candidates?
  How can solution candidates be constructed?
- What is a useful problem-solving view on the original task?
- What is the initial problem in a constructive approach?
- Which operations (solution steps) are possible to simplify a problem?
- Which rest problems result from such problem simplifications?
- What knowledge can guide the problem solving process / the search?

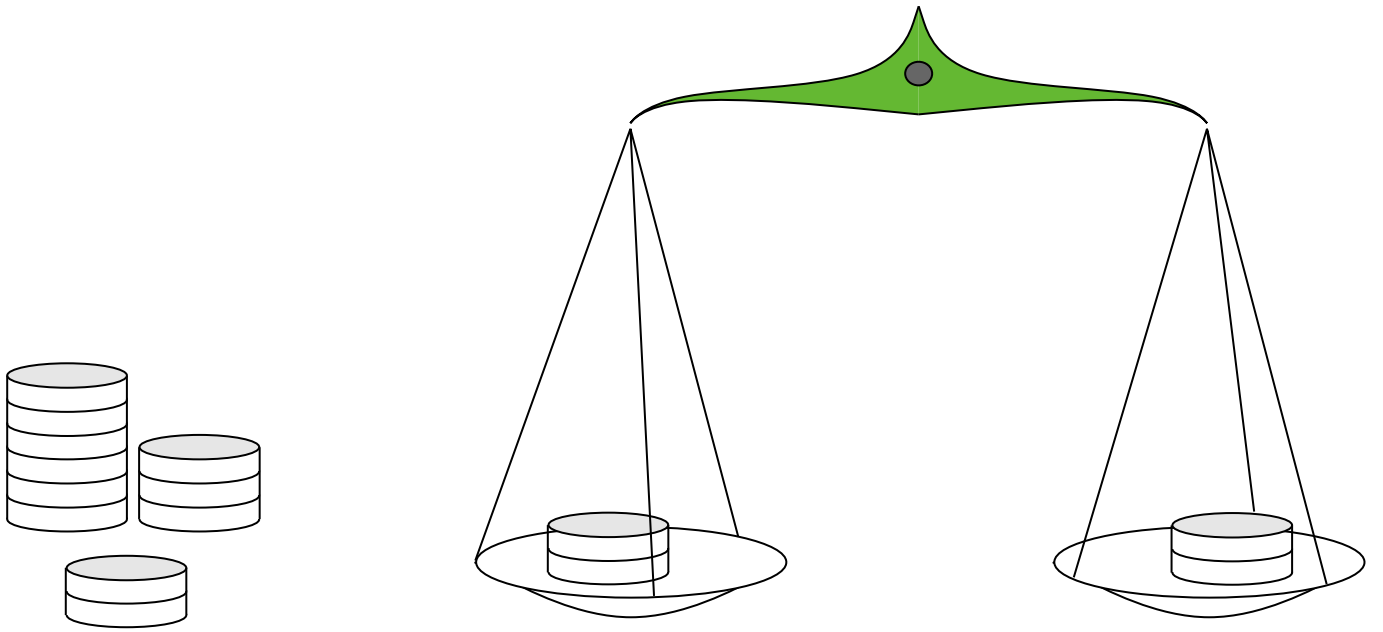
➜ How can the original task be modeled

- by adequate abstraction and encoding of solution candidates and
- by providing problem solving knowledge

in such a way that efficient search algorithms can be formulated?

Examples for AND-OR Search Problems

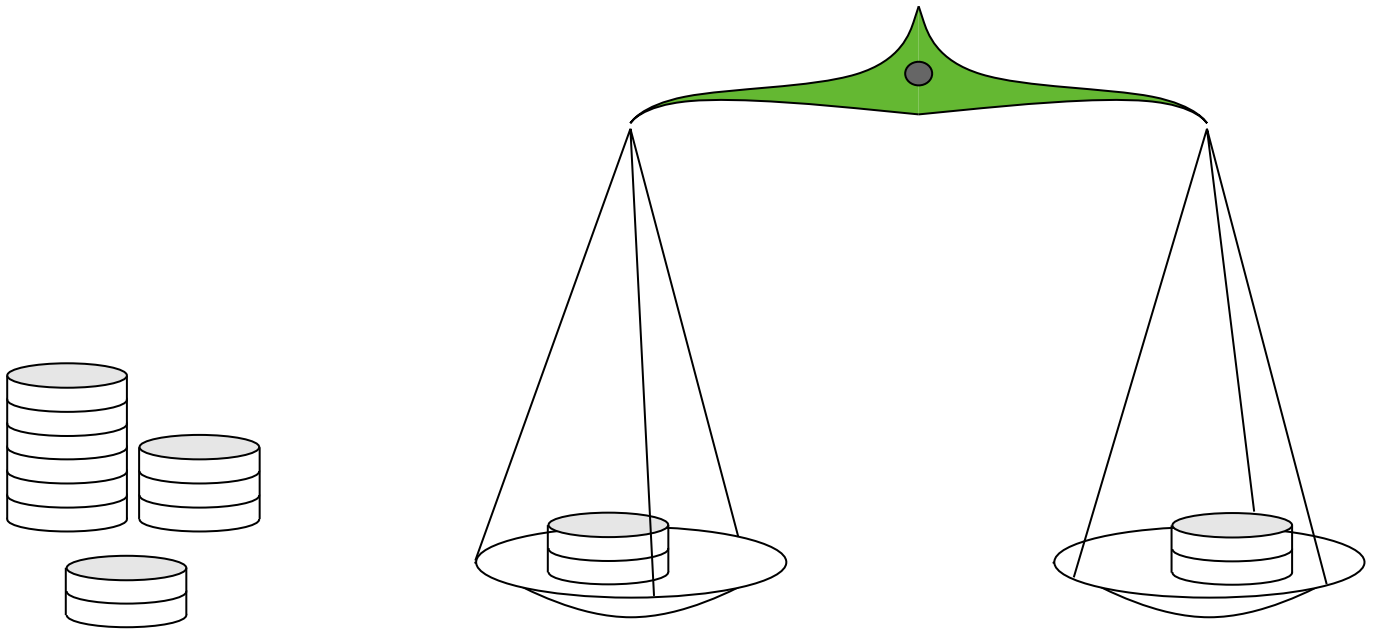
The Counterfeit Coin Problem



- 12 coins are given, one of which is known to be heavier or lighter.
- 3 weighing tests are allowed to find the counterfeit coin.

Examples for AND-OR Search Problems

The Counterfeit Coin Problem



- Task: Find a weighing strategy.
  What to weigh first, second, etc.?
  How to process different weighing outcomes?
- Problem decomposition:
  Different weighing outcomes are handled as independent cases.
- The role of heuristics: focus on the most promising weighing strategy.

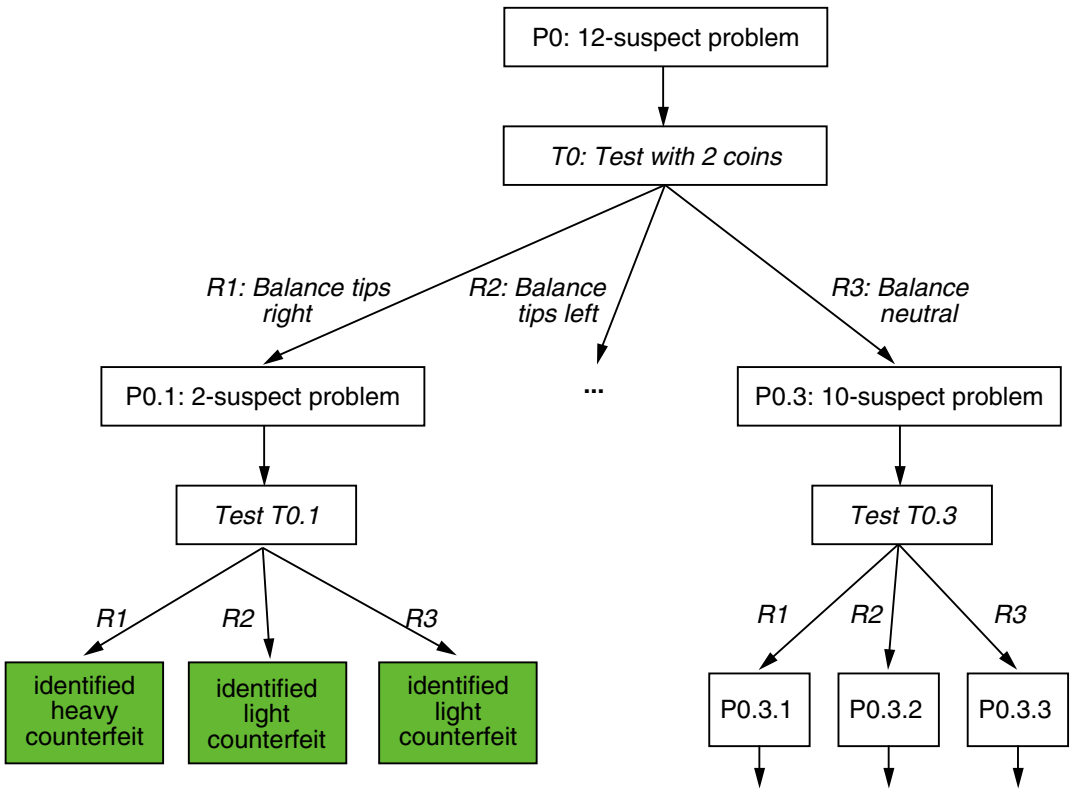
Remarks:

- Each rest problem (specified as state) can be described by knowledge already acquired through the preceding tests. A coin belongs to one of the following four categories: not-heavy, not-light, not-suspect, unknown (= none of the first three categories).
- A weighing action is the selection of a number of coins (for each pan the same number) and to test the selection.
- After each test, the category information of each coin is updated. For instance, if some coin was assigned to “not-light” and from the weighing outcome follows “not-heavy”, its new category becomes “not-suspect”.
- Each weighing action leads to one of three possible outcomes all of which must be dealt with. The most suitable weighing strategy depends on the objective to be optimized:
  1. If the maximum number of weighing actions is to be minimized, treat the most difficult outcome next.
  2. If the expected number of weighing actions is to be minimized, treat the most likely outcome next.

Examples for AND-OR Search Problems

The Counterfeit Coin Problem (continued)

Representation of a weighing strategy



- Challenge: Avoid solving the same problem again and again.
- What is useful information about coins for representing remaining problems?

Remarks:

- Weighing strategies can be represented as trees (or acyclic directed graphs).
- For a problem at hand, we first constrain the problem by deciding which test to perform and then we consider the possible outcomes of that test (the remaining problems).
- If the balance tips to one side, the counterfeit coin is among the coins just weighed. In the example depicted, test $T0.1$ is then to weigh one of the two "suspect" coins against one of the remaining ten "not-suspect" coins.
- Following a weighing strategy, the subset of "suspect" coins (and "not-light", "not-heavy", "not-suspect") changes over time. For a test, the participants are selected randomly from these sets.
- Of course, more than two coins can be used in a weighing.

Examples for AND-OR Search Problems

The Counterfeit Coin Problem (continued)

Problem: The Counterfeit Coin Problem

Instance: C. A finite set of coins containing exactly one counterfeit coin.

Solution: A weighing strategy that identifies the counterfeit coin and its failing ("too-light" or "too-heavy").

Examples for AND-OR Search Problems

The Counterfeit Coin Problem (continued)

Problem: The Counterfeit Coin Problem

Instance: C. A finite set of coins containing exactly one counterfeit coin.

Solution: A weighing strategy that identifies the counterfeit coin and its failing ("too-light" or "too-heavy").

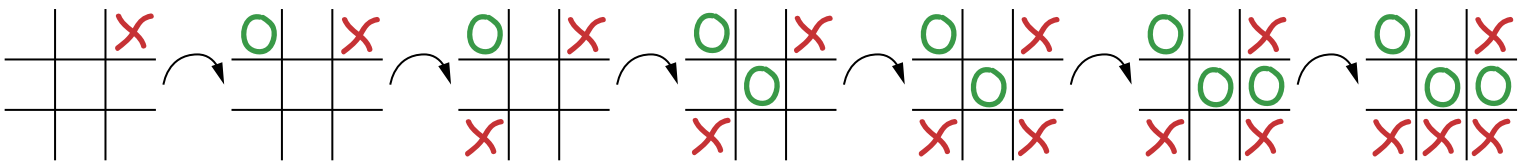
Algorithmization

1. Encoding of solution candidates:
   Finite trees of the above type ($=$ weighing strategies) with leaves in which the counterfeit coin and its failing are known.
2. Encoding of partial solutions:
   Finite trees of the above type without leaf condition.
3. Operators:
   Extend a branch by a vertex representing a weighing decision and its successors for the possible outcomes.
4. Greedy algorithm:
   Extend a most promising partial weighing strategy.

Examples for AND-OR Search Problems

Tic-Tac-Toe Game [Wikipedia]

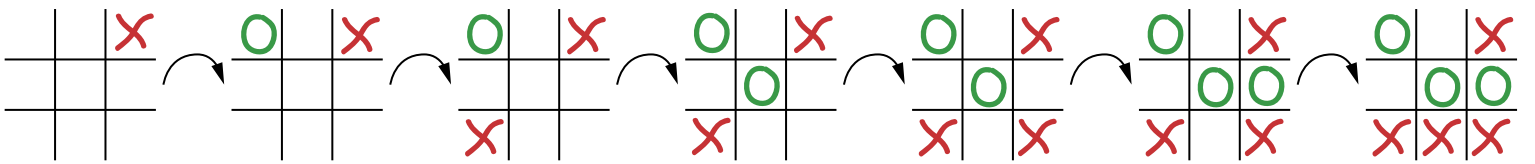
Tic-tac-toe is a game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



Examples for AND-OR Search Problems

Tic-Tac-Toe Game [Wikipedia]

Tic-tac-toe is a game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



- Task: Search a winning strategy for the first player (X)

  Where to put the marks, first, second, etc.?

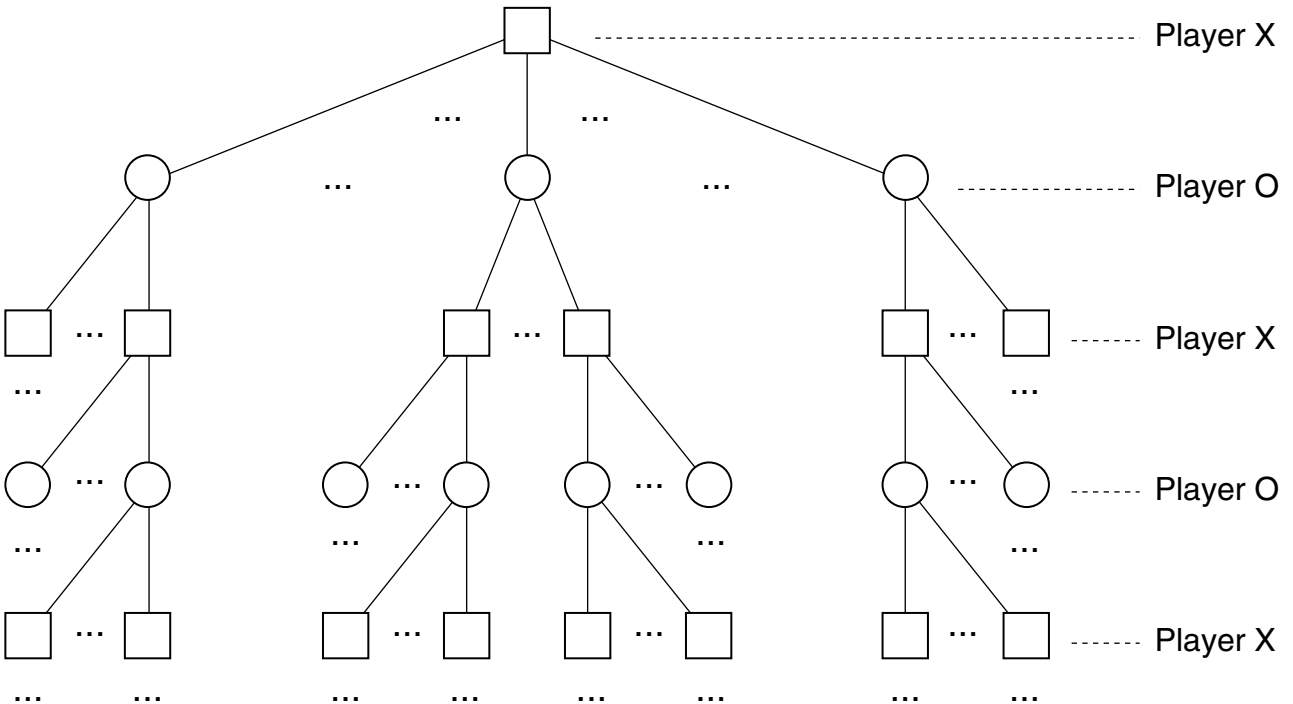
  How to react to different moves of the opponent?

- Problem decomposition: different possible oponent moves are handled as independent cases.

Examples for AND-OR Search Problems

Tic-Tac-Toe Game (continued)

Representation of a game tree



- Challenge: Game trees can be huge.
- How can the winning potential of a game situation be determined?

Examples for AND-OR Search Problems

Tic-Tac-Toe Game (continued)

Problem: Tic-Tac-Toe Game

Instance: Empty 3×3 grid, player X goes first.

Solution: A winning strategy for player X.

Examples for AND-OR Search Problems

Tic-Tac-Toe Game (continued)

Problem: Tic-Tac-Toe Game

Instance: Empty 3×3 grid, player X goes first.

Solution: A winning strategy for player X.

Algorithmization

1. Encoding of solution candidates:
   Finite trees of the above type with leaves in which the outcome of the game (win/draw/loss) is clear.
2. Encoding of partial solutions:
   Finite trees of the above type without leaf condition.
3. Operators:
   Extend a branch by a possible move of the next player.
4. Greedy algorithm:
   Extend a most promising partial game strategy.

Examples for AND-OR Search Problems

UNSAT Problem

Decide whether a propositional formula in CNF is unsatisfiable:

$$
\begin{array}{llllllllll}
 & (\neg x_1 & \vee & x_3 & \vee & x_7 & \vee & \neg x_9 & &) \\
\wedge & (x_2 & \vee & \neg x_3 & \vee & \neg x_5 & \vee & x_6 & \vee & \neg x_8) \\
\wedge & (\neg x_2 & \vee & x_7 & \vee & x_8 & & & &) \\
\wedge & (x_1 & \vee & \neg x_2 & \vee & \neg x_3 & \vee & x_9 & &) \\
 & & & & \vdots & & & & & \\
\wedge & (\neg x_3 & \vee & x_4 & \vee & \neg x_6 & \vee & x_7 & \vee & \neg x_8) \\
\end{array}
$$

Examples for AND-OR Search Problems

UNSAT Problem (continued)

Decide whether a propositional formula in CNF is unsatisfiable:

$$
\begin{array}{llllllllll}
 & (& \neg x_1 & \vee & x_3 & \vee & x_7 & \vee & \neg x_9 &) \\
\wedge & (& x_2 & \vee & \neg x_3 & \vee & \neg x_5 & \vee & x_6 \vee \neg x_8 &) \\
\wedge & (& \neg x_2 & \vee & x_7 & \vee & x_8 & & &) \\
\wedge & (& x_1 & \vee & \neg x_2 & \vee & \neg x_3 & \vee & x_9 &) \\
 & & & & & \vdots & & & & \\
\wedge & (& \neg x_3 & \vee & x_4 & \vee & \neg x_6 & \vee & x_7 \vee \neg x_8 &)
\end{array}
$$

Constructive approach: Incrementally assign truth values to variables until α evaluates to true/false; possible truth values are handled as independent cases.

Heuristics:

DPLL: E.g. assign a truth value to a variable that occurs most often (product of no. of positive and negative occurrences) in unsatisfied clauses so that the unsatisfied parts of the formula resulting from the assignment (true/false) are similar in size (reduction of tree size).

Remarks:

- For an unsatisfiability result, we have to ensure that all truth assignments have been tested.
- The complement of the satisfiability problem UNSAT (Decision problem: Is a given CNF Formula not satisfiable?) is coNP-complete, i.e., it is the complement problem of an NP-complete problem.
- In fact, any order of truth value assignments to variables leads to the same result: either the formula is satisfied by some assignment or no assignment satisfies the formula.
  Therefore, it is of interest to have a smallest strategy tree, i.e., a tree that minimal with respect to external path length. The DPLL strategy tries to order the truth assignments in such a way that the truth value of the formula can be decided "on average early".

Examples for AND-OR Search Problems

UNSAT Problem (continued)

Problem: UNSAT

Instance: α. Propositional formula in CNF.

Solution: A selection strategy for variables to assign truth values to such that in all cases the formula evaluates to false. (Proof of unsatisfiability)

Algorithmization of the constructive approach

1. Encoding of partial solutions:
   $(x_1 \mapsto \textit{true}, x_2 \mapsto \textit{false}, x_3 \mapsto *, \ldots, x_n \mapsto *)$
2. Cost/merit function for partial solutions:
   Number of clauses not satisfied.
3. Operators:
   Assign a truth value to an unassigned variable.
4. Greedy algorithm:
   Continue with a most promising variable.