Chapter S:IV

IV. Search Space Representation

- Problem Solving
- Systematic Search
- Search Space Encoding
- State-Space Representation

- Problem-Reduction Representation
- Choosing a Representation
- Relation to Dynamic Programming
**Problem-Reduction Representation**

**Problem Solving: Constructing Solutions**

<table>
<thead>
<tr>
<th>Example</th>
<th>Solution Step</th>
<th>Rest Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-Queens</td>
<td>Fix position of next queen.</td>
<td>Positioning of remaining queens on the unattacked fields of the board.</td>
</tr>
<tr>
<td>8-Puzzle</td>
<td>Fix next move.</td>
<td>Resulting board configuration that must be transformed into the target configuration.</td>
</tr>
<tr>
<td>TSP</td>
<td>Fix next town to visit.</td>
<td>Find trip from this town back to the starting point.</td>
</tr>
<tr>
<td>Blocks World</td>
<td>Fix next action (block move).</td>
<td>Resulting block configuration on floor must be transformed into target configuration.</td>
</tr>
</tbody>
</table>

**Previous Assumption:**

Problem solutions can be described by a linear sequence of applications of (relatively simple) operators that transform a problem into a (hopefully) simpler rest problem.

Q. Is there anything else we may want to do?
Problem-Reduction Representation

Disadvantage of State-Space Search

A state space search can only determine solutions that are described as paths. Solutions are constructed in a linear way.

Desired solutions can be structured:

- **Action-Reaction-Problems**
  Solutions are AND-OR trees with AND nodes that represent the reactions of the environment to actions of an agent. A solution describes effective behavior for each of the unpredictable reactions of the environment.

- **2-Player Games**
  Solutions are AND-OR trees with AND nodes that represent the reactions of the opponent. A solution describes a game strategy that determines the best move in every game situation.

- **Means-end Analysis**
  A goal-oriented analysis of the problem leads to a decomposition into a sequence of subproblems. Solutions can be constructed as sequences of solutions for the subproblems.

→ Independent subproblems can be solved in any order.

→ Subsolutions can be shared.
A. **Construction**

Construction algorithms generate a solution by adding individual components one at a time until a feasible solution is obtained.

⇒ OR Graph Search (= State Space Search)

B. **Improvement**

Improvement heuristics begin with a feasible solution and successively improve it by a sequence of exchanges or mergers in a local search.

⇒ Local search algorithms like Hill-Climbing

E. **Partitioning (Decomposition [Pearl])**

Partitioning algorithms break or ‘partition’ a problem into smaller subproblems, each of which is solved independently.

⇒ AND-OR Graph Search
Problem-Reduction Representation

Problem Solving: Solution Steps

- Problem Transformation (Simplification)
  A problem is transformed into a single (simpler) problem.

- Problem Decomposition
  A problem is decomposed into a finite number of (sub-) problems.
  - Problem Transformation (Simplification)
    A problem is transformed into a (simpler) problem.
  - Means-End Analysis
    A problem is decomposed into a finite number of (sub-) problems.
The application of an operator may lead to different kinds of subproblems. A graph that captures such a search space structure has two kinds of links:

1. **AND links.**
   Lead to independent subproblems all of which, however, must be solved in order to solve the problem associated with the parent node. AND links of a decomposition step are marked as sibling links.

2. **OR links.**
   Lead to alternative subproblems one of which has to be solved in order to solve the problem associated with the parent node.

A graph of this type is called *problem-reduction graph* or *AND-OR graph.*
Remarks:

- As in the state-space graph: OR links encode applied operators (problem transformation).
- As in the state-space graph: A sequence of OR links along a path that starts at the root node \( s \) defines a solution base.
- Q. Does a path from the root node to a goal node represent a solution to the problem?
- AND links model a problem decomposition. They are employed to decompose a problem according to possible outcomes (as in the counterfeit problem) or to decompose a complex problem into less complex subproblems. [Towers of Hanoi]
- Several possibilities may exist to decompose a given rest problem (at a parent node) into subproblems. The AND links of those subproblems that belong together (sibling links) are hence marked as such.
Problem-Reduction Representation

Counterfeit Problem

- **P0**: 12-suspect problem
  - **T1**: Test with 12 coins
  - **T6**: Test with 2 coins

- **P0+T6**: All problems that identified heavy counterfeit

- **P0+T6**: All problems that identified light counterfeit

- **P1**: 2-suspect problem
  - **Test T1**
  - **Test Ti**
  - **P1+Ti**: All problems that identified heavy counterfeit

- **P2**: 10-suspect problem
  - **Test T1**
  - **Test Tj**
  - **P2+Tj**: All problems that identified light counterfeit

- **R1**: Balance tips right
- **R2**: Balance neutral
- **R2**: Balance tips left

- **P3**: Identified light counterfeit
- **P4**: Identified light counterfeit
- **P5**: Identified light counterfeit

S:IV-69  Search Space Representation  © STEIN/LETTMANN 2023
Problem-Reduction Representation

Counterfeit Problem

- **P0**: 12-suspect problem
- **P1**: 2-suspect problem
- **P2**: 10-suspect problem
- **P3**, **P4**, **P5**: Identified counterfeit

Tests:
- **T1**: Test with 12 coins
- **T6**: Test with 2 coins
- **T1** and **T6**: All problems that identified counterfeit

Checks:
- **R1**: Balance tips right
- **R2**: Balance neutral
- **R3**: Balance tips left

- **P1+T1**: All problems that identified heavy counterfeit
- **P2+Tj**: All problems that identified light counterfeit

- **P0+T6**: All problems that identified counterfeit
Remarks:

- In the 2-suspect problem, we have two coins for which the balance tips one way or another, along with ten remaining coins from which we know that they are honest.
- The 2-suspect problem (for instance) may occur several times in a weighing strategy.
- The 2-suspect problem can be solved independently from all other weighing problems.
- The solution of the 2-suspect problem can be determined, saved, and reused.
Problem-Reduction Representation
The Different Node types

The application of an operator may lead to different kinds of subproblems. A canonical graph that captures such a search space structure has two kinds of nodes:

1. **AND nodes.**
   - Nodes with outgoing AND links only; more precisely: AND links that belong together, sibling AND links.
   
   Examples: decisions of an opponent (Keyword: game tree search), random processes in the nature, (artificial) experiments, results of external computations, all kinds of events controlled by a third party.

2. **OR nodes.**
   - Nodes with outgoing OR links only.

   Examples: the own decisions, the actual strategy.
Remarks:

- The AND-OR graph of the counterfeit problem has a canonical representation by nature.
- In the AND-OR graph of the counterfeit problem the child nodes of an OR node are AND nodes and vice versa. This alternation of node types is typical for action-reaction problems, e.g. represented in the form of a game tree. [S:VII Game Playing Introduction].
Problem-Reduction Representation

Search Building Blocks [State-Space]

Definition 5 (Problem-Reduction Graph, AND-OR Graph)

Consider the set of all problems that can be generated by applying either the operators or a problem decomposition to the database.

One obtains the problem-reduction graph or AND-OR graph by

1. connecting each parent node with its generated successors using directed links,

2. labeling those links that belong to an operator with the applied operator,

3. comprising those links that belong to a problem decomposition as sibling links, and

4. introducing additional intermediate nodes and OR-links for each set of sibling AND-links if a node would have more than one set of outgoing sibling links or outgoing AND-links together with outgoing OR-links.
Remarks:

- Usually, solved-labeling is not applied to an underlying search space graph $G$. Instead, it is applied to the finite, explored subgraph of $G$ as a termination test: If the root node $s$ is labeled “solved”, the explored part of $G$ contains a solution graph.
Problem-Reduction Representation

Algorithm: solved-labeling

Input: 
- $G$. A finite, acyclic AND-OR graph.
- $n$. A node in $G$.

- $\text{successors}(n)$. Returns the successors of node $n$.
- $\star(n)$. Predicate that is $True$ if $n$ is a goal node.

Output: The symbol $True$ if a solution exists, $False$ otherwise.
Problem-Reduction Representation

Algorithm: solved-labeling (without labeling)

Input: 
- \( G \). A finite, acyclic AND-OR graph.
- \( n \). A node in \( G \).
- \( \text{successors}(n) \). Returns the successors of node \( n \).
- \( \star(n) \). Predicate that is \( True \) if \( n \) is a goal node.

Output: The symbol \( True \) if a solution exists, \( False \) otherwise.

\( \text{solved-labeling}(G, n, \text{successors}, \star) \)

1. IF \( |\text{successors}(n)| = 0 \) THEN
   IF \( \star(n) \) THEN RETURN(\( True \));
   ELSE RETURN(\( False \));
Problem-Reduction Representation

Algorithm: solved-labeling (without labeling)

Input:
- $G$: A finite, acyclic AND-OR graph.
- $n$: A node in $G$.
- $\text{successors}(n)$: Returns the successors of node $n$.
- $\star(n)$: Predicate that is True if $n$ is a goal node.

Output: The symbol True if a solution exists, False otherwise.

\[ \text{solved-labeling}(G, n, \text{successors}, \star) \]

1. IF $|\text{successors}(n)| = 0$ THEN
   IF $\star(n)$ THEN RETURN(True);
   ELSE RETURN(False);
2. FOREACH $n'$ IN $\text{successors}(n)$ DO
   IF \text{solved-labeling}(G, $n'$, $\text{successors}$, $\star$) THEN
     IF $\text{OR\_node}(n)$ THEN RETURN(True); // One success is enough.
     ELSE IF $\text{AND\_node}(n)$ THEN RETURN(False); // One fail is one too much.
   ENDDO
Problem-Reduction Representation

Algorithm: solved-labeling (without labeling)

Input: 
- $G$. A finite, acyclic AND-OR graph.
- $n$. A node in $G$.
- $\text{successors}(n)$. Returns the successors of node $n$.
- $\star(n)$. Predicate that is True if $n$ is a goal node.

Output: The symbol True if a solution exists, False otherwise.

\begin{verbatim}
solved-labeling($G, n, \text{successors}, \star$)

1. IF $|\text{successors}(n)| = 0$ THEN
   IF $\star(n)$ THEN RETURN(True);
   ELSE RETURN(False);
2. FOREACH $n'$ IN $\text{successors}(n)$ DO
   IF solved-labeling($G, n', \text{successors}, \star$) THEN
     IF $\text{OR\_node}(n)$ THEN RETURN(True); // One success is enough.
     ELSE IF $\text{AND\_node}(n)$ THEN RETURN(False); // One fail is one too much.
   ENDDO
3. IF $\text{OR\_node}(n)$ THEN RETURN(False); // OR\_node n has no solvable successor.
   ELSE RETURN(True); // All successors of AND\_node n are solvable.
\end{verbatim}
Algorithm: solved-labeling (with labeling)

Input:  
- $G$. A finite, acyclic AND-OR graph.
- $n$. A node in $G$.
- successors$(n)$. Returns the successors of node $n$.
- $\star(n)$. Predicate that is True if $n$ is a goal node.

Output: The symbol True if a solution exists, False otherwise.

solved-labeling$(G, n, \text{successors}, \star)$

1. IF solved$(n)$ THEN RETURN(True);

2. IF $|\text{successors}(n)| = 0$ THEN
   IF $\star(n)$ THEN solved$(n) = \text{True}$, RETURN(True);
   ELSE RETURN(False);

3. FOREACH $n'$ IN successors$(n)$ DO
   IF solved-labeling$(G, n', \text{successors}, \star)$ THEN
     IF OR_node$(n)$ THEN solved$(n) = \text{True}$, RETURN(True);
     ELSE
       IF AND_node$(n)$ THEN RETURN(False);
     END

4. IF OR_node$(n)$ THEN RETURN(False);
   ELSE solved$(n) = \text{True}$, RETURN(True);
Cyclic problem-reduction graphs can entail infinite solution graphs, caused by “unfolding”:
Cyclic problem-reduction graphs can entail infinite solution graphs, caused by “unfolding”:
Cyclic problem-reduction graphs can entail infinite solution graphs, caused by “unfolding”:
Remarks:

- “Unfolding” means that when a node is reached a second time, the state is not reused but a copy is created instead. I.e., we are completely omitting an occurrence check.

- Note that the solution graph definition restricts to acyclic AND-OR graphs.
  Q. How could the solved-labeling procedure be adapted to deal with cyclic AND-OR graphs?

- Usually the search space underlying a problem is only partially explored, and hence the AND-OR graph $H$ does not show the complete picture.
  Q. How could the solved-labeling procedure be adapted to deal with incomplete AND-OR graphs?

- Q. How could an “unsolvable-labeling procedure” be defined, which proves whether a problem is unsolvable?
Problem-Reduction Representation

Hypergraphs

AND-OR graphs can be considered as a generalization of ordinary graphs, called hypergraphs:

- Ordinary graph: Directed links (edges, arcs) connect two nodes.
- Hypergraph: Directed hyperedges, also called “connectors”, connect two sets of nodes.
- AND-OR graph: The links of an AND node form a single hyperedge.

→ The determination of a solution graph corresponds to the determination of a hyperpath between the root node $s$ and a set of goal nodes that represent solved rest problems.
IV. Search Space Representation

- Problem Solving
- Systematic Search
- Search Space Encoding
- State-Space Representation

- Problem-Reduction Representation
- Choosing a Representation
- Relation to Dynamic Programming
Choosing a Representation
Problem-Reduction Graphs with Alternating Node Types

Original problem

Operator 1 → Subproblem 1
Operator 2 → Subproblem 2
Operator k → Subproblem k

Event 1

Subproblem 1_1 → Subproblem 1_1_1 → Subproblem ...
Subproblem 1_1_l → Subproblem 1_1_2 → Subproblem ...

Event l

Operator 1 → Subproblem 1_1_1
Operator 2 → Subproblem 1_1_2
Operator k → Subproblem 1_1_k

Reaction problems

Event 1

Subproblem ...

Event l

Operator 1
Operator 2
Operator k

Reaction problems
Remarks:

- By introducing OR nodes with a single operator resp. by introducing additional AND nodes with a single event, each AND-OR graph can be transformed into a problem-reduction graph with alternating node types. W.l.o.g. the start node of such graphs is an OR node.
Choosing a Representation
Problem-Reduction Graphs with Alternating Node Types (continued)

Interpretation:

- $k$ operators $\sim$ own actions, decisions, strategy
- $l$ events $\sim$ reactions

- **Action problems.**
  Consequences that result from a subproblem under the impact of applicable operators.

- **Reaction problems.**
  Consequences that result from a subproblem under the impact of possible events.

Examples of problem-reduction graphs with strictly alternating node types are balance puzzles and 2-player games like chess.
Choosing a Representation
Transforming Problem-Reduction into State-Space Graphs

Original problem

Operators:
- Operator 1
- Operator 2
- Operator k

Subproblems:
- Orgproblem + Operator 1 + Event 1 → Subproblem 1_1
- Orgproblem + Operator 1 + Event l → Subproblem 1_l
- Orgproblem + Operator 2 + Event 1 → Subproblem 2_1
- Orgproblem + Operator 2 + Event l → Subproblem 2_l

n'
Choosing a Representation
Transforming Problem-Reduction into State-Space Graphs

Original problem

Operator 1

Operator 2

...Operator k

Orgproblem + Operator 1 + Event 1 → Subproblem 1_1
...Orgproblem + Operator 1 + Event l → Subproblem 1_l

Orgproblem + Operator 2 + Event 1 → Subproblem 2_1
...Orgproblem + Operator 2 + Event l → Subproblem 2_l

Subproblem 1_1 + Operator 1
Subproblem 1_2 + Operator 1
...
Subproblem 1_l + Operator 1

Subproblem 1_1 + Operator 2
Subproblem 1_2 + Operator 1
...
Subproblem 1_l + Operator 1

Subproblem 1_1 + Operator k
Subproblem 1_2 + Operator k
...
Subproblem 1_l + Operator k
Choosing a Representation
Transforming Problem-Reduction into State-Space Graphs

Original problem

\[ \text{Orgproblem} + \text{Operator 1} + \text{Event 1} \rightarrow \text{Subproblem 1}\_1 \]
\[ \ldots \]
\[ \text{Orgproblem} + \text{Operator 1} + \text{Event } l \rightarrow \text{Subproblem 1}\_l \]

\[ \text{Orgproblem} + \text{Operator 2} + \text{Event 1} \rightarrow \text{Subproblem 2}\_1 \]
\[ \ldots \]
\[ \text{Orgproblem} + \text{Operator 2} + \text{Event } l \rightarrow \text{Subproblem 2}\_l \]

Subproblem 1\_1 \rightarrow Subproblem ...
Subproblem 1\_2 \rightarrow Subproblem ...
Subproblem 1\_l \rightarrow Subproblem ...

Subproblem 1\_1 + \text{Operator 1} \rightarrow Subproblem ...
Subproblem 1\_1 + \text{Event 1} \rightarrow Subproblem ...
Subproblem 1\_1 + \text{Event } l \rightarrow Subproblem ...

Subproblem 1\_2 + \text{Operator 1} \rightarrow Subproblem ...
Subproblem 1\_2 + \text{Event 1} \rightarrow Subproblem ...
Subproblem 1\_2 + \text{Event } l \rightarrow Subproblem ...

Subproblem 1\_l + \text{Operator 1} \rightarrow Subproblem ...
Subproblem 1\_l + \text{Event 1} \rightarrow Subproblem ...
Subproblem 1\_l + \text{Event } l \rightarrow Subproblem ...

Subproblem 1\_1 + \text{Operator 1} \rightarrow Subproblem ...
Subproblem 1\_2 + \text{Operator 1} \rightarrow Subproblem ...
Subproblem 1\_l + \text{Operator 1} \rightarrow Subproblem ...

Subproblem 1\_1 + \text{Operator 2} \rightarrow Subproblem ...
Subproblem 1\_2 + \text{Operator 2} \rightarrow Subproblem ...
Subproblem 1\_l + \text{Operator 2} \rightarrow Subproblem ...

Subproblem 1\_1 + \text{Operator k} \rightarrow Subproblem ...
Subproblem 1\_2 + \text{Operator k} \rightarrow Subproblem ...
Subproblem 1\_l + \text{Operator k} \rightarrow Subproblem ...

...
Choosing a Representation
Transforming Problem-Reduction into State-Space Graphs (continued)

Transformation rules:

- To the node $s$ the applicable operators are applied.
- A successor node $n'$ of $s$ is comprised of all reaction problems that may result from applying the chosen operator.
- To a node $n'$ all applicable operator combinations are applied, whereas each subproblem $p \in n'$ gets its own operator, i.e., a set of $|n'|$ operators is associated with the link $(n', n'').$
- The successor node $n''$ of $n'$ is comprised of all reaction problems that may result from applying the chosen operator set.

Observe that the node $n'$ has as many successors as operator combinations are possible, given the subproblems $p \in n'$. This number is in $O(k|n'|)$

→ Goal nodes are nodes that contain only trivial (solved) subproblems.
Remarks:

- If a subproblem in a node is already solved, no further operator is applied to it.
- Instead of handling all subproblems in a node simultaneously, one subproblem could be selected at a time and an operator applied to it.
Choosing a Representation
Transformed Counterfeit Problem

- **P0**: 12-suspect problem

- **T1**: Test with 12 coins

- **T6**: Test with 2 coins

**n'**

- **Test T1** if P1
- **Test T1** if P2

- **...**

- **Test Ti** if P1
- **Test Tj** if P2

**n''**

- **P0 + T6 + R1 (balance right)** → P1: 2-subjects problem
- **P0 + T6 + R2 (balance left)** → P1: 2-subjects problem
- **P0 + T6 + R3 (balance neutral)** → P2: 10-subjects problem

- **P1 + Ti + R1 (balance right)** → identified heavy counterfeit
- **P1 + Ti + R2 (balance left)** → identified light counterfeit
- **P1 + Ti + R3 (balance neutral)** → identified light counterfeit

- **P2 + Tj + R1 (balance right)** → P3
- **P2 + Tj + R2 (balance left)** → P4
- **P2 + Tj + R3 (balance neutral)** → P5

- **...**
Choosing a Representation
State-Space versus Problem-Reduction Representation

A state-space representation is advisable if solutions are paths (or nodes).

⇒ The solution graph is a path in the state-space graph.

Examples:

- constraint satisfaction problems
- sequence seeking problems
Choosing a Representation
State-Space versus Problem-Reduction Representation (continued)

A problem-reduction representation is advisable if solutions are trees, graphs, or partially ordered node sets.

→ The solution graph is a tree-like subgraph of the problem-reduction graph.

Examples:
- strategy seeking problems
- symbolic integration
- theorem proving

Characteristics:
- Problems may be decomposed into subproblems that can be solved independently of each other (= in parallel).
- The divide-and-conquer paradigm can be applied to find a global optimum.
Remarks:

- Choosing a search space representation requires the analysis whether and how the solutions of subproblem interact with each other when being composed.

- Q. Is a problem-reduction representation possible for the 8-puzzle problem?

- Q. Is a problem-reduction representation an advisable choice for the 8-puzzle problem?

- There are sequence seeking problems where the solutions of subproblems interact with each other (e.g., the 8-puzzle problem), but where a problem-reduction representation is superior to a state-space representation. Example: The tower of Hanoi problem.
A stack of $n$ disks $D_1, \ldots, D_n$ (different sizes, sorted, smallest on top) is to be moved from peg 1 to peg 3. Only one of the topmost disks can be moved at a time, no disk may be placed on top of a smaller one.
A stack of \( n \) disks \( D_1, \ldots, D_n \) (different sizes, sorted, smallest on top) is to be moved from peg 1 to peg 3. Only one of the topmost disks can be moved at a time, no disk may be placed on top of a smaller one.

Observations:

- The problem can be solved analogously to solving the 8-puzzle problem. A sequence of legal moves of single disks has to be determined.

- The recursive definition of moving a stack of disks \( D_1, \ldots, D_k \) from one peg to another peg can be exploited: problem decomposition.
Choosing a Representation

Tower of Hanoi Problem

A stack of \( n \) disks \( D_1, \ldots, D_n \) (different sizes, sorted, smallest on top) is to be moved from peg 1 to peg 3. Only one of the topmost disks can be moved at a time, no disk may be placed on top of a smaller one.

Observations:

- The problem can be solved analogously to solving the 8-puzzle problem. A sequence of legal moves of single disks has to be determined.
- The recursive definition of moving a stack of disks \( D_1, \ldots, D_k \) from one peg to another peg can be exploited: problem decomposition.

Moving a stack of disks \( D_1, \ldots, D_k \) from peg \( a \) to peg \( b \) can be done in three “steps”:

1. Move the stack of disks \( D_1, \ldots, D_{k-1} \) from peg \( a \) to peg \( c \).
2. Move disk \( D_k \) peg \( a \) to peg \( b \).
3. Move the stack of disks \( D_1, \ldots, D_{k-1} \) steps peg \( c \) to peg \( b \).
Choosing a Representation
Tower of Hanoi Problem (continued)

→ Problem decomposition can be a powerful tool, even if order constraints have to be taken into account.
Choosing a Representation

Tower of Hanoi Problem (continued)

A stack of $n$ disks $D_1, \ldots, D_n$ (different sizes, sorted, smallest on top) is to be moved from peg 1 to peg 3. Only one of the topmost disks can be moved at a time, no disk may be placed on top of a smaller one.

OR Graph Representation:
- Nodes are states (i.e., the stacked disks for each peg). The remaining problem is known.
- Edges correspond to legal moves of some single disk.
- Cyclic paths can be pruned.
Choosing a Representation
Tower of Hanoi Problem (continued)

A stack of \( n \) disks \( D_1, \ldots, D_n \) (different sizes, sorted, smallest on top) is to be moved from peg 1 to peg 3. Only one of the topmost disks can be moved at a time, no disk may be placed on top of a smaller one.

AND-OR Graph Representation:

- Nodes are states (i.e., the stacked disks for each peg). The remaining problem has to be stored explicitly.
- All nodes are AND nodes.
- Families of AND edges correspond to the following problem decomposition:
  - hardest single move problem,
    (Moving the biggest misplaced disk from its current position to its target position.)
  - establishing a state that allows that move,
    (Moving all disks on top of the biggest misplaced disk out of the way.)
  - establishing the target state out of the state resulting from that move.
    (Moving the remaining misplaced disks to their target positions.)

\( \rightarrow \) A solution graph will be constructed directly.
Choosing a Representation
Tower of Hanoi Problem (continued)

A stack of \( n \) disks \( D_1, \ldots, D_n \) (different sizes, sorted, smallest on top) is to be moved from peg 1 to peg 3. Only one of the topmost disks can be moved at a time, no disk may be placed on top of a smaller one.

AND-OR Graph Representation: (continued)

- Start with the most difficult problem: \( D_n \) from peg 1 to peg 3.
  - \( D_n \) must be clear and peg 3 must be empty.
  - Introduce new subproblems: \( D_1, \ldots, D_{n-1} \) from peg 1 to peg 2.

- Solve the remaining problems.
Choosing a Representation
Means-End Analysis

Characteristic for the tower of Hanoi problem is that the set of subproblems \( p_1, \ldots, p_n \) is **serializable**. Formally, this means that there is a sequence \( i_1, \ldots, i_n \) such that for problems \( p_{i_1}, \ldots, p_{i_n} \):

1. \( p_{i_1}, \ldots, p_{i_k} \) can be solved after \( p_{i_{k+1}}, \ldots, p_{i_n} \) have been solved.

2. Solving \( p_{i_1}, \ldots, p_{i_k} \) will not compromise the solutions of \( p_{i_{k+1}}, \ldots, p_{i_n} \).
Choosing a Representation
Means-End Analysis

Characteristic for the tower of Hanoi problem is that the set of subproblems $p_1, \ldots, p_n$ is serializable. Formally, this means that there is a sequence $i_1, \ldots, i_n$ such that for problems $p_{i_1}, \ldots, p_{i_n}$:

1. $p_{i_1}, \ldots, p_{i_k}$ can be solved after $p_{i_{k+1}}, \ldots, p_{i_n}$ have been solved.
2. Solving $p_{i_1}, \ldots, p_{i_k}$ will not compromise the solutions of $p_{i_{k+1}}, \ldots, p_{i_n}$.

If this underlying problem structure is identified, the powerful operator selection strategy “means end analysis” can be applied:

“The basic difference between this method [means end analysis] and the state-space approach is its purposeful behavior: operators are invoked by virtue of their potential in fulfilling a desirable subgoal, and new subgoals are created in order to enable the activation of a desirable operator.”

[Pearl 1984, p.29]
Remarks:

- Planning a trip (e.g. to New York) is another example for a problem that can be tackled by a means end analysis:
  1. New York \(\rightarrow\) plane from Frankfurt
  2. plane from Frankfurt \(\rightarrow\) train to Frankfurt
  3. train to Frankfurt \(\rightarrow\) bus to train station
  4. \ldots

- The power of problem reduction representation under a regime of a means end analysis: AND-linked subproblems can be solved independently, though for processing the global plan a strict order (from left to right) must be obeyed.

- Q. What is the complexity class of the tower of Hanoi problem?
Chapter S:IV

IV. Search Space Representation

- Problem Solving
- Systematic Search
- Search Space Encoding
- State-Space Representation
- Problem-Reduction Representation
- Choosing a Representation
- Relation to Dynamic Programming
Relation to Dynamic Programming
Optimum Solution Cost and Bellman’s Principle of Optimality

Dynamic programming has been developed to solve

- discrete optimization problems and
- planning problems for which an optimum sequence of decisions is sought.

Bellman’s principle of optimality:

“An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.”

[Bellman 1954]
Relation to Dynamic Programming
Optimum Solution Cost and Bellman’s Principle of Optimality

Dynamic programming has been developed to solve

- discrete optimization problems and
- planning problems for which an optimum sequence of decisions is sought.

Bellman’s principle of optimality:

“An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.”

[Bellman 1954]

A stronger version of Bellman’s optimality principle:

Each partial solution of an optimum solution is in turn an optimum solution for the respective subproblem.

Optimum (estimated) solution cost can be defined via systems of equations.
Definition 6 (Bellman Equations for $C^*$ [Overview])

Let $G$ be an acyclic AND-OR graph rooted at $s$. Let $C_H(n)$ be a recursive cost function for $G$ based on $E$ (local properties).

$$C^*(n) = \begin{cases} 
  c(n) & \text{n is goal node and leaf in } G \\
  \infty & \text{n is leaf in } G \text{ but no goal node} \\
  \min_i \{F[E(n), C^*(n_i)]\} & \text{n is inner OR node in } G, \text{n}_i \text{ direct successor of } n \text{ in } G \\
  F[E(n), C^*(n_1), \ldots, C^*(n_k)] & \text{n is inner AND node in } G, \text{n}_i \text{ direct successor of } n \text{ in } G 
\end{cases}$$
Relation to Dynamic Programming
Optimum Solution Cost and Bellman’s Principle of Optimality (continued)

Definition 6 (Bellman Equations for $C^*$ [Overview])

Let $G$ be an acyclic AND-OR graph rooted at $s$. Let $C_H(n)$ be a recursive cost function for $G$ based on $E$ (local properties).

$$C^*(n) = \begin{cases} 
  c(n) & \text{$n$ is goal node and leaf in $G$} \\
  \infty & \text{$n$ is leaf in $G$ but no goal node} \\
  \min_i \{F[E(n), C^*(n_i)]\} & \text{$n$ is inner OR node in $G$, $n_i$ direct successor of $n$ in $G$} \\
  F[E(n), C^*(n_1), \ldots, C^*(n_k)] & \text{$n$ is inner AND node in $G$, $n_i$ direct successor of $n$ in $G$}
\end{cases}$$

If $F$ is properly chosen ("If $F$ obeys Bellman’s principle of optimality"), the equations stated in Definition 6 form necessary and complete conditions for the optimality of the solution cost $C^*$.

"⇒" The values that fulfill the Bellman equations are the optimum solution cost (for the respective node).

"⇐" The optimum solution cost for a node fulfill the Bellman equations.
Remarks:

- The name “dynamic programming” was introduced by Bellman. As with “linear programming”, the terms “program” and “programming” should be read as “plan” and “plan generation” respectively.

- The term “policy” characterizes the process of decision making and corresponds to the principle of selecting a most promising solution base.

- The objective function in discrete optimization problems corresponds to the evaluation functions $f$ (state-space search) or $f_1$ (problem-reduction search).

- Observe the rationale behind the Bellman equations (Definition 6) and optimality:
  1. Given a problem (modeled as AND-OR graph) we “may be lucky” that its solutions obey Bellman’s principle of optimality (consider that we have no free choice of $F$).
     Similarly: The cost measure $F$ imposed by the problem obeys Bellman’s principle of optimality (examples for $F$ include “$+$” and “$\max$”). In this case we should formulate the optimization (= cost) function according to Definition 6.
  2. Then, operationalizing Definition 6 will yield the optimum solution cost for our problem.

Conversely, the fact that the solutions of our problem won’t obey Bellman’s principle of optimality means that we cannot formulate its cost function recursively and equip it with an amenable cost measure (examples for $F$ include “$+$” and “$\max$”). We still may be able to compute a cost function as solution of Bellman’s equations according to Definition 6, but this operationalization will probably not yield the optimum solution cost for our problem.
Relation to Dynamic Programming
Optimum Solution Cost and Bellman’s Principle of Optimality (continued)

Using the face-value principle for partially explored search spaces, Bellman’s principle of optimality induces a system of equations defining estimated optimum solution cost \( \hat{C}(n) \).

**Definition 7 (Bellman Equations for \( \hat{C} \))**

Let \( G \) be an explored subgraph of an acyclic AND-OR graph \( G \) rooted at \( s \). Let \( h \) be an underestimating heuristic function, let \( C_H(n) \) be a recursive cost function for \( G \) based on \( E \) and \( F \), and let \( F \) obey Bellman’s principle of optimality.

\[
\hat{C}(n) = \begin{cases} 
  c(n) & \text{if } n \text{ is goal node and leaf in } G \\
  h(n) & \text{if } n \text{ is leaf in } G \text{ but no goal node} \\
  \min_{i} \{ F[E(n), \hat{C}(n_i)] \} & \text{if } n \text{ is inner OR node in } G, \text{ } n_i \text{ direct successors of } n \text{ in } G \\
  F[E(n), \hat{C}(n_1), \ldots, \hat{C}(n_k)] & \text{if } n \text{ is inner AND node in } G, \text{ } n_i \text{ direct successors of } n \text{ in } G 
\end{cases}
\]

S:IV-115  Search Space Representation © STEIN/LETTMANN 2023
Corollary 8 (Bellman Equations for $\hat{C}$)

A recursive cost function $C_H(n)$ that is based on local properties $E$ and a monotone cost measure $F$ obeys Bellman’s principle of optimality.
Relation to Dynamic Programming

Partially Explored State Spaces

**Definition 9 (Bellman Equations for $\hat{C}$)**

Let $G$ be an explored subgraph of a state-space graph $\mathcal{G}$ rooted at $s$. Let $h$ be an underestimating heuristic function, let $C_P(n)$ be a recursive cost function for $\mathcal{G}$ based on $E$ and $F$, and let $F$ obey Bellman’s principle of optimality.

\[
\hat{C}(n) = \begin{cases} 
  c(n) & \text{if } n \text{ is goal node and leaf in } G \\
  h(n) & \text{if } n \text{ is leaf in } G \text{ but no goal node} \\
  \min_i \{ F(E(n), \hat{C}(n_i)) \} & \text{if } n \text{ is inner OR node in } G, \text{ } n_i \text{ direct successors of } n \text{ in } G
\end{cases}
\]
Definition 9 (Bellman Equations for $\hat{C}$)

Let $G$ be an explored subgraph of a state-space graph $\mathcal{G}$ rooted at $s$. Let $h$ be an underestimating heuristic function, let $C_P(n)$ be a recursive cost function for $G$ based on $E$ and $F$, and let $F$ obey Bellman’s principle of optimality.

\[
\hat{C}(n) = \begin{cases} 
  c(n) & \text{n is goal node and leaf in } G \\
  h(n) & \text{n is leaf in } G \text{ but no goal node} \\
  \min_i \{ F[E(n), \hat{C}(n_i)] \} & \text{n is inner OR node in } G, \text{ } \\
  \text{n_i direct successors of } n \text{ in } G
\end{cases}
\]

Special case A*:

- use top down propagation of path cost values $g(n)$
- consider only paths to nodes in OPEN

\[
\hat{C}(n) = \begin{cases} 
  h(n) & \text{n is on OPEN} \\
  \min_i \{ c(n, n_i) + \hat{C}(n_i) \} & \text{n is on CLOSED, } n_i \text{ direct successors of } n
\end{cases}
\]
Algorithm A* and Bellman’s principle of optimality:

1. **Top-down Propagation.**

   Superior (optimum) paths to non-goal states are extended to superior (optimum) paths to successor states until a goal is reached.

→ **Right-Monotonicity**  ~  **Order Preservation.**

For all nodes \( n' \) and paths \( P_{s-n'}, P'_{s-n'} \) from \( s \) to \( n' \), and for all nodes \( n \) and paths \( P_{n'-n} \) from \( n' \) to \( n \) holds:

\[
\hat{C}_{P_{s-n'}}(s) \leq \hat{C}_{P'_{s-n'}}(s) \quad \Rightarrow \quad \hat{C}_{P_{s-n}}(s) \leq \hat{C}_{P'_{s-n}}(s)
\]

where \( P_{s-n} \) uses subpath \( P_{s-n'} \) and \( P'_{s-n} \) uses \( P'_{s-n'} \).
Algorithm A* and Bellman’s principle of optimality:

2. Bottom-up Propagation.

Starting from optimum solutions for rest problems, optimum solutions for complexer problems are synthesized until the original problem is solved.

→ Left-Monotonicity.

For all nodes $n'$ and paths $P_{n'-\gamma}, P'_{n'-\gamma}$ from $n'$ to $\gamma$, and for all paths $P_{s-n'}$ from $s$ to $n'$ holds:

$$\hat{C}_{P_{n'-\gamma}}(n') \leq \hat{C}_{P'_{n'-\gamma}}(n') \Rightarrow \hat{C}_{P_{s-\gamma}}(s) \leq \hat{C}_{P'_{s-\gamma}}(s)$$