Chapter S:IV

IV. Informed Search

- ❑ Best-First Search
- ❑ Best-First Search for State-Space Graphs
- ❑ Best-First Search for AND-OR Graphs
- ❑ Relation between GBF and BF

- ❑ Cost Functions
- ❑ Evaluation of AND-OR Graphs
- ❑ Evaluation of State-Space Graphs
- ❑ Algorithm A*
- ❑ Relation to Dynamic Programming

- ❑ Hybrid Strategies

Cost Functions

Overview

BF and GBF define a schema for the design of search strategies. Up to this point, the evaluation functions f (state-space search) and f_1 (problem-reduction search) remained unspecified.

Questions:

- How to compute f or f_1?
- How to evaluate a solution graph?
- How to evaluate a search space graph?
- How to identify a most promising solution base?

Answering these question gives rise to a taxonomy of Best-First algorithms.

Remarks:

- In fact, implicitly, we have already given answers during the Illustration of GBF before. What will be added in the following are the foundations for these answers, along with a definite semantics.

Cost Functions

Overview (continued)

The answers are developed in several steps by the following concepts:

1. Recursive cost functions (for graphs)
2. Solution cost (for a given solution graph)
3. Optimum solution cost (for a complete search space graph)
4. Estimated solution cost (for a given solution base)
5. Estimated optimum solution cost (for a partial search space graph)

Cost Functions

Overview (continued)

The answers are developed in several steps by the following concepts:

1. Recursive cost functions (for graphs)
2. Solution cost (for a given solution graph)
3. Optimum solution cost (for a complete search space graph)
4. Estimated solution cost (for a given solution base)
5. Estimated optimum solution cost (for a partial search space graph)

Names of the respective cost functions:

		Solution	
		given	optimum searched
Exploration	complete	C_G , C_P	C^*
	partial	$\widehat{C}_G$, $\widehat{C}_P$	$\widehat{C}$

Cost Functions

Overview (continued)

The answers are developed in several steps by the following concepts:

1. Recursive cost functions (for graphs)
2. Solution cost (for a given solution graph)
3. Optimum solution cost (for a complete search space graph)
4. Estimated solution cost (for a given solution base)
5. Estimated optimum solution cost (for a partial search space graph)

Names of the respective cost functions:

		Solution	
		given	optimum searched
Exploration	complete	C_G , C_P	C^*
	partial	$\widehat{C}_G$, $\widehat{C}_P$	$\widehat{C}$

Cost Functions

Overview (continued)

The answers are developed in several steps by the following concepts:

1. Recursive cost functions (for graphs)
2. Solution cost (for a given solution graph)
3. Optimum solution cost (for a complete search space graph)
4. Estimated solution cost (for a given solution base)
5. Estimated optimum solution cost (for a partial search space graph)

Names of the respective cost functions:

		Solution	
		given	optimum searched
Exploration	complete	C_G , C_P	C^*
	partial	$\widehat{C}_G$, $\widehat{C}_P$	$\widehat{C}$

Cost Functions

Overview (continued)

The answers are developed in several steps by the following concepts:

1. Recursive cost functions (for graphs)

2. Solution cost (for a given solution graph)
3. Optimum solution cost (for a complete search space graph)

4. Estimated solution cost (for a given solution base)
5. Estimated optimum solution cost (for a partial search space graph)

Names of the respective cost functions:

		Solution	
		given	optimum searched
Exploration	complete	$C_G(n)$, $C_P(n)$	$C^*(n)$
	partial	$\widehat{C}_G(n)$, $\widehat{C}_P(n)$	$\widehat{C}(n)$

Cost Functions

Overview (continued)

The answers are developed in several steps by the following concepts:

1. Recursive cost functions (for graphs)
2. Solution cost (for a given solution graph)
3. Optimum solution cost (for a complete search space graph)
4. Estimated solution cost (for a given solution base)
5. Estimated optimum solution cost (for a partial search space graph)

Names of the respective cost functions:

		Solution	
		given	optimum searched
Exploration	complete	$C_G(s)$, $C_P(s)$	$C^*(s)$
	partial	$\widehat{C}_G(s)$, $\widehat{C}_P(s)$	$\widehat{C}(s) \rightsquigarrow H$

H = most promising solution base.

Cost Functions

If solution graphs are known, the solution cost for a solution graph can be determined.

Definition 13 (Cost Function $C_H,\ C_P$)

Let G be an acyclic AND-OR graph and let M be an ordered set. A function C_H, which assigns to each solution graph H and each node n in G a cost value $C_H(n)$ in M, is called a cost function (for G).

For an OR-Graph G, the cost function, which assigns each solution path P in G and each node n in G a cost value $C_P(n)$ in M is denoted by C_P.

Usage and notation of C_H and C_P :

- No provisions are made how to compute $C_H(n)$ for a solution graph H. $C_H(s)$ specifies the cost of a solution graph H for s : $f_1(H) = C_H(s)$.
- No provisions are made how to compute $C_P(n)$ for a solution path P. $C_P(s)$ specifies the cost of a solution path P for s : $f(\gamma) = C_P(s)$.

Cost Functions

If solution graphs are known, the solution cost for a solution graph can be determined.

Definition 13 (Cost Function $C_H,\ C_P$)

Let G be an acyclic AND-OR graph and let M be an ordered set. A function C_H, which assigns to each solution graph H and each node n in G a cost value $C_H(n)$ in M, is called a cost function (for G).

For an OR-Graph G, the cost function, which assigns each solution path P in G and each node n in G a cost value $C_P(n)$ in M is denoted by C_P.

Usage and notation of C_H and C_P :

- No provisions are made how to compute $C_H(n)$ for a solution graph H.
  $C_H(s)$ specifies the cost of a solution graph H for s : $f_1(H) = C_H(s)$.
- No provisions are made how to compute $C_P(n)$ for a solution path P.
  $C_P(s)$ specifies the cost of a solution path P for s : $f(\gamma) = C_P(s)$.

Remarks:

- As ordered set M usually $\mathbf{R} \cup \{\infty\}$ is chosen.
- $C_H(n)$ (resp. $C_P(n)$) should be seen as binary function with arguments H (resp. P) and n.
- The cost value $C_H(n)$ (resp. $C_P(n)$) is meaningful only if n is a node in H (resp. P).
- Solution cost does not measure efforts for finding a solution. Solution cost aggregate properties of operations (and decompositions) in a solution graph to form a cost value.
- Instead of cost functions we may employ merit functions or, even more general, weight functions. The respective notations are Q_H and Q_P for merits, and W_H and W_P for weights. E.g., Algorithm HC usually employs a merit function.
- A cost function can be a complex accounting rule, considering the properties of a solution graph or a solution path:
  1. node costs, such as the processing effort of a manufacturing machine,
  2. edge costs, such as the cost for transportation or transmission, and
  3. terminal payoffs, which specify a lump value for the remaining solution effort at leaf nodes.
- At places where the semantics was intuitively clear, we have already used the notation $C_H(n)$ and $C_P(n)$ to denote the solution cost of a problem associated with node n. Definition 13 catches up for the missing notation and semantics.

Cost Functions

If the entire search space graph rooted at a node s is known, the optimum solution cost for the root node s can be determined.

Definition 14 (Optimum Solution Cost C^*, Optimum Solution)

Let G be an acyclic AND-OR graph with root node s and let $C_H(n)$ denote a cost function for G.

The optimum solution cost for a node n in G, $C^*(n)$, is defined as

$$C^*(n) = \inf\{C_H(n) \mid H \text{ is solution graph for } n \text{ in } G\}$$

Similarly, for an OR-Graph G and a cost function $C_P(n)$ for G, the optimum solution cost for n is defined as

$$C^*(n) = \inf\{C_P(n) \mid P \text{ is solution path for } n \text{ in } G\}$$

A solution graph (path) with solution cost $C^*(n)$ is called optimum solution graph (path) for n. The optimum solution cost for s, $C^*(s)$, is abbreviated as C^*.

Cost Functions

If the entire search space graph rooted at a node s is known, the optimum solution cost for the root node s can be determined.

Definition 14 (Optimum Solution Cost C^*, Optimum Solution)

Let G be an acyclic AND-OR graph with root node s and let $C_H(n)$ denote a cost function for G.

The optimum solution cost for a node n in G, $C^*(n)$, is defined as

$$C^*(n) = \inf\{C_H(n) \mid H \text{ is solution graph for } n \text{ in } G\}$$

Similarly, for an OR-Graph G and a cost function $C_P(n)$ for G, the optimum solution cost for n is defined as

$$C^*(n) = \inf\{C_P(n) \mid P \text{ is solution path for } n \text{ in } G\}$$

A solution graph (path) with solution cost $C^*(n)$ is called optimum solution graph (path) for n. The optimum solution cost for s, $C^*(s)$, is abbreviated as C^*.

Remarks:

- If G contains no solution graph (resp. solution path) for n, let $C^*(n) = \infty$.

Cost Functions

If the entire search space graph rooted at a node s is known, the optimum solution cost extending a solution base for s can be determined.

Definition 15 (Optimum Solution Cost C^*_H, C^*_P for a Solution Base)

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The optimum solution cost for node n in G based on a solution base H in G, $C^*_H(n)$, is defined as

$$C^*_H(n) = \inf\{C_{H'}(n) \mid H' \text{ is solution graph in } \mathcal{G} \text{ extending } H\}$$

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the optimum solution cost for a node n in G based on a solution base P, $C^*_P(n)$, is defined as

$$C^*_P(n) = \inf\{C_{P'}(n) \mid P' \text{ is solution path in } \mathcal{G} \text{ extending } P\}$$

Cost Functions

If the entire search space graph rooted at a node s is known, the optimum solution cost extending a solution base for s can be determined.

Definition 15 (Optimum Solution Cost C^*_H, C^*_P for a Solution Base)

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The optimum solution cost for node n in G based on a solution base H in G, $C^*_H(n)$, is defined as

$$C^*_H(n) = \inf\{C_{H'}(n) \mid H' \text{ is solution graph in } \mathcal{G} \text{ extending } H\}$$

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the optimum solution cost for a node n in G based on a solution base P, $C^*_P(n)$, is defined as

$$C^*_P(n) = \inf\{C_{P'}(n) \mid P' \text{ is solution path in } \mathcal{G} \text{ extending } P\}$$

Cost Functions

If the search space graph rooted at a node s is known partially, the optimum solution cost extending a solution base for s can be estimated.

Definition 16 (Estimated Solution Cost $\widehat{C}_H$, $\widehat{C}_P$ for a Solution Base)

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The estimated solution cost for a node n in G based on a solution base H in G, $\widehat{C}_H(n)$, returns an estimate of $C^*_H(n)$.

$\widehat{C}_H(n)$ is optimistic, if and only if $\widehat{C}_H(n) \leq C^*_H(n)$.

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the estimated solution cost for a node n in G based on a solution base P, $\widehat{C}_P(n)$, returns an estimate of $C^*_P(n)$.

$\widehat{C}_P(n)$ is optimistic, if and only if $\widehat{C}_P(n) \leq C^*_P(n)$.

Usage of $\widehat{C}_H$, $\widehat{C}_{P'}$: In BF and GBF we use $f_1(H) = \widehat{C}_H(s)$ resp. $f(n) = \widehat{C}_P(s)$.

Cost Functions

If the search space graph rooted at a node s is known partially, the optimum solution cost extending a solution base for s can be estimated.

Definition 16 (Estimated Solution Cost $\widehat{C}_H$, $\widehat{C}_P$ for a Solution Base)

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The estimated solution cost for a node n in G based on a solution base H in G, $\widehat{C}_H(n)$, returns an estimate of $C^*_H(n)$.

$\widehat{C}_H(n)$ is optimistic, if and only if $\widehat{C}_H(n) \leq C^*_H(n)$.

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the estimated solution cost for a node n in G based on a solution base P, $\widehat{C}_P(n)$, returns an estimate of $C^*_P(n)$.

$\widehat{C}_P(n)$ is optimistic, if and only if $\widehat{C}_P(n) \leq C^*_P(n)$.

Usage of $\widehat{C}_H$, $\widehat{C}_{P'}$: In BF and GBF we use $f_1(H) = \widehat{C}_H(s)$ resp. $f(n) = \widehat{C}_P(s)$.

Cost Functions

If the search space graph rooted at a node s is known partially, the optimum solution cost extending a solution base for s can be estimated.

Definition 16 (Estimated Solution Cost $\widehat{C}_H$, $\widehat{C}_P$ for a Solution Base)

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The estimated solution cost for a node n in G based on a solution base H in G, $\widehat{C}_H(n)$, returns an estimate of $C_H^*(n)$.

$\widehat{C}_H(n)$ is optimistic, if and only if $\widehat{C}_H(n) \leq C_H^*(n)$.

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the estimated solution cost for a node n in G based on a solution base P, $\widehat{C}_P(n)$, returns an estimate of $C_P^*(n)$.

$\widehat{C}_P(n)$ is optimistic, if and only if $\widehat{C}_P(n) \leq C_P^*(n)$.

Usage of $\widehat{C}_H$, $\widehat{C}_{P'}$: In BF and GBF we use $f_1(H) = \widehat{C}_H(s)$ resp. $f(n) = \widehat{C}_P(s)$.

Remarks:

- Estimated optimum solution cost values are computed on basis of the explored subgraph G of the underlying search space graph $\mathcal{G}$. So, values may change over time with G.

- In the setting of Definition 15 we assume

  $$C^*(s) = \inf\{C^*_H(s) \mid H \text{ is solution base in } G\}$$

  and

  $$C^*(s) = \inf\{C^*_P(s) \mid P \text{ is solution base in } G\}$$

  respectively.

  Therefore, it is essential for search algorithms to keep available solution bases that are important for this result. Optimistically estimating $C^*_H(n)$ or $C^*_P(n)$ in GBF and BF respectively will direct the search into promising directions.

Cost Functions

If the search space graph rooted at a node s is known partially, the optimum solution cost for s can be estimated.

Definition 17 (Estimated Optimum Solution Cost $\widehat{C}$ [Overview]**)**

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The estimated optimum solution cost for a node n in G, $\widehat{C}(n)$, is defined as follows:

$$\widehat{C}(n) = \inf\{\widehat{C}_H(n) \mid H \text{ is solution base in } G\}$$

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the estimated solution cost function fora node n in G, $\widehat{C}(n)$, is defined as follows:

$$\widehat{C}(n) = \inf\{\widehat{C}_P(n) \mid P \text{ is solution base in } G\}$$

A solution base H for s with $\widehat{C}_H(s) = \widehat{C}(s)$ (resp. a solution base P for s with $\widehat{C}_P(s) = \widehat{C}(s)$) is called most promising solution base (for s).

Cost Functions

If the search space graph rooted at a node s is known partially, the optimum solution cost for s can be estimated.

Definition 17 (Estimated Optimum Solution Cost $\widehat{C}$ [Overview])

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The estimated optimum solution cost for a node n in G, $\widehat{C}(n)$, is defined as follows:

$$\widehat{C}(n) = \inf\{\widehat{C}_H(n) \mid H \text{ is solution base in } G\}$$

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the estimated solution cost function fora node n in G, $\widehat{C}(n)$, is defined as follows:

$$\widehat{C}(n) = \inf\{\widehat{C}_P(n) \mid P \text{ is solution base in } G\}$$

A solution base H for s with $\widehat{C}_H(s) = \widehat{C}(s)$ (resp. a solution base P for s with $\widehat{C}_P(s) = \widehat{C}(s)$) is called most promising solution base (for s).

Cost Functions

If the search space graph rooted at a node s is known partially, the optimum solution cost for s can be estimated.

Definition 17 (Estimated Optimum Solution Cost $\widehat{C}$ [Overview])

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ with root node s and cost function $C_H(n)$ for $\mathcal{G}$.

The estimated optimum solution cost for a node n in G, $\widehat{C}(n)$, is defined as follows:

$$\widehat{C}(n) = \inf\{\widehat{C}_H(n) \mid H \text{ is solution base in } G\}$$

Similarly, for an an explored subgraph G of an OR-Graph $\mathcal{G}$, a cost function $C_P(n)$ for $\mathcal{G}$, the estimated solution cost function fora node n in G, $\widehat{C}(n)$, is defined as follows:

$$\widehat{C}(n) = \inf\{\widehat{C}_P(n) \mid P \text{ is solution base in } G\}$$

A solution base H for s with $\widehat{C}_H(s) = \widehat{C}(s)$ (resp. a solution base P for s with $\widehat{C}_P(s) = \widehat{C}(s)$) is called most promising solution base (for s).

Cost Functions

Recursive Cost Functions

The computation of the evaluation functions f or f_1 would be infeasible

1. if each possible solution base had to be analyzed in isolation, or
2. if the cost of a solution base had to be computed from scratch for each additionally explored node.

➜ Utilization of recursive cost functions to implement the evaluation function f (state-space search) or f_1 (problem-reduction search).

Efficiency benefits of recursive functions:

1. Shared computation.

   Already computed evaluation results for solution bases H are exploited for other solution bases that contain H.

2. Selective updating.

   Only the predecessors of a newly explored node need to be updated.

Cost Functions

Recursive Cost Functions (continued)

Definition 18 (Recursive Cost Function, Cost Measure)

A cost function C_H for a solution graph H is called recursive, if for each node n in H holds:

$$C_H(n) = F[E(n), C_H(n_1), C_H(n_2), \ldots, C_H(n_k)], \quad \text{where}$$

- $n_1, n_2, \ldots, n_k$ denote the direct successors of n in H,
- $E(n) \in \mathbf{E}$ denotes a set of *local* properties of n with respect to H,
- F is a function that prescribes how local properties of n are accounted (better: combined) with properties of the direct successors of n:

$$F : \mathbf{E} \times M^k \to M, \quad \text{where } M \text{ is an ordered set.}$$

Similarly, a cost function C_P for solution paths P is called recursive, if for each node n in P holds:

$$C_P(n) = \begin{cases} F[E(n)] & n \text{ is goal node and leaf in } P \\ F[E(n), C_P(n')] & n \text{ is inner node in } P \text{ and } n' \text{ direct successor of } n \text{ in } P \end{cases}$$

F is called cost measure.

Cost Functions

Recursive Cost Functions (continued)

Definition 18 (Recursive Cost Function, Cost Measure)

A cost function C_H for a solution graph H is called recursive, if for each node n in H holds:

$$C_H(n) = F[E(n), C_H(n_1), C_H(n_2), \ldots, C_H(n_k)], \quad \text{where}$$

- $n_1, n_2, \ldots, n_k$ denote the direct successors of n in H,
- $E(n) \in \mathbf{E}$ denotes a set of *local* properties of n with respect to H,
- F is a function that prescribes how local properties of n are accounted (better: combined) with properties of the direct successors of n:

$$F : \mathbf{E} \times M^k \to M, \quad \text{where } M \text{ is an ordered set.}$$

Similarly, a cost function C_P for solution paths P is called recursive, if for each node n in P holds:

$$C_P(n) = \begin{cases} F[E(n)] & n \text{ is goal node and leaf in } P \\ F[E(n), C_P(n')] & n \text{ is inner node in } P \text{ and } n' \text{ direct successor of } n \text{ in } P \end{cases}$$

F is called cost measure.

Remarks:

- Observe that for each node n in a solution graph H the complete subgraph of H that is rooted at n forms a solution graph for n.
- The formalization $C_H(n) = F[E(n), C_HG(n_1), \ldots, C_H(n_k)]$ is a shorthand for

$$C_G(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ F[E(n), C_H(n')] & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ F[E(n), C_H(n_1), \ldots, C_H(n_k)] & n \text{ is inner AND node in } H, \\ & n_1, \ldots, n_k \text{ direct successors of } n \text{ in } H \end{cases}$$

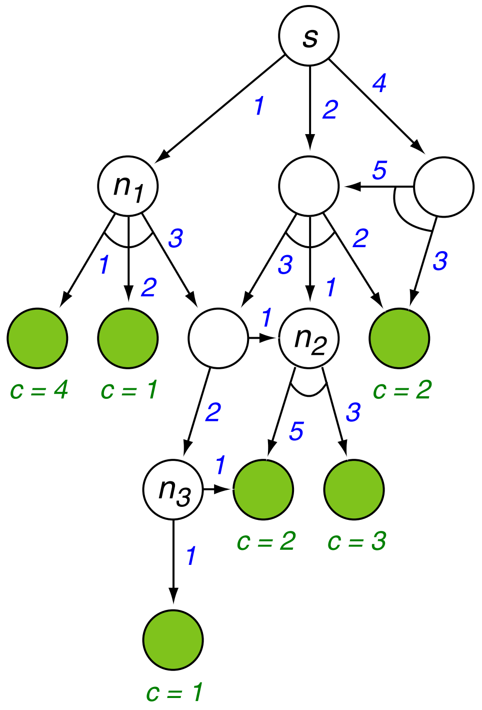
  with $c(n) := F[E(n)]$.
- If for a solution base its merits, quality, or other positive aspects are measured, F is called merit measure.
- The computation of $C_H(n)$ is called cost and merit propagation respectively. If $C_H(n)$ fulfills the conditions of a recursive cost function, it can be computed bottom-up, similar to the solved-labeling procedure.
- The definition of $C_H(n)$ covers the base case as well: the leaf nodes in H have no successors, and $C_H(n)$ depends solely on $E(n)$.
- The function $C_H(n)$ employs the the same $E(n)$ and the same cost measure F for all possible solution graphs of a search space graph.

  As a consequence, cost value computation is context insensitive: If a solution graph for node n is contained in two solution graphs H and H', then $C_H(n) = C_{H'}(n)$ holds.

Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, Overview]

Underlying problem-reduction graph:



$E(n) = c(n, n') =$ cost for edge (n, n')

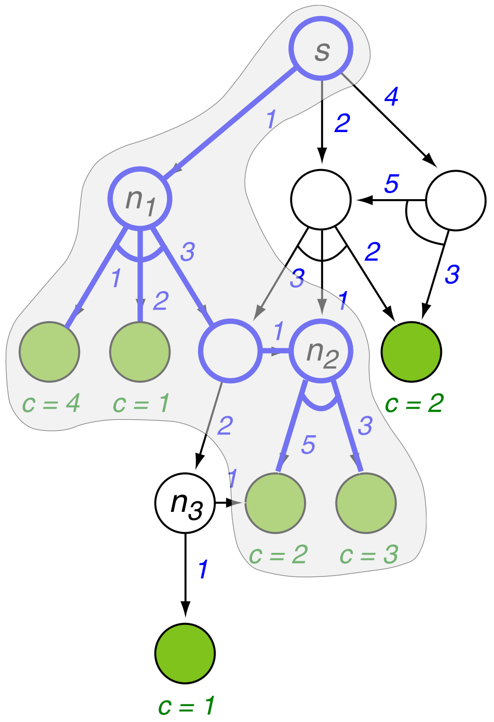
$$C_H(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ c(n, n') + C_H(n') & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ \max_i\{c(n, n_i) + C_H(n_i)\} & n \text{ is inner AND node in } H, \\ & n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, Overview]

$H \equiv$ solution graph for s :

$C_H(s) = 12$



$E(n) = c(n, n') =$ cost for edge (n, n')

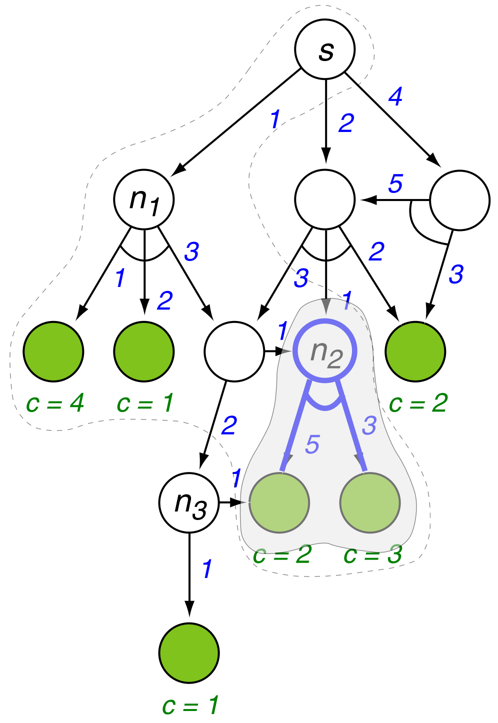
$$C_H(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ c(n,n') + C_H(n') & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ \text{max}_i\{c(n,n_i) + C_H(n_i)\} & n \text{ is inner AND node in } H, \\ & n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, Overview]

Solution graph for n_2 :

$C_H(n_2) = 7$



$E(n) = c(n, n') =$ cost for edge (n, n')

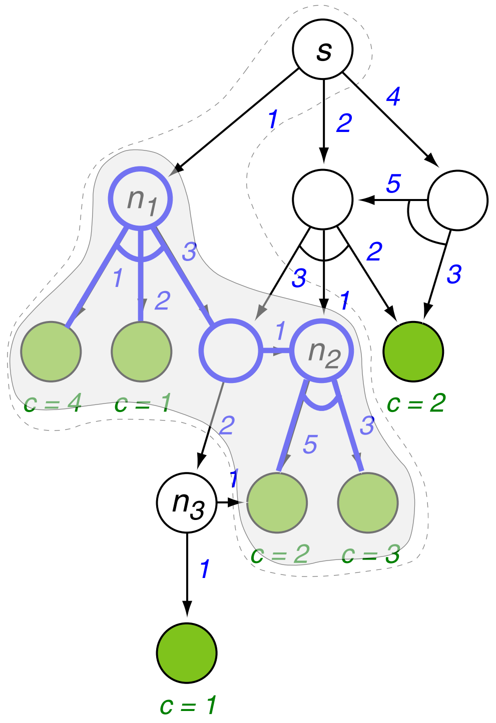
$$C_H(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ c(n, n') + C_H(n') & n \text{ is inner OR node in } H, \; n' \text{ direct successor of } n \text{ in } H \\ \max_i\{c(n, n_i) + C_H(n_i)\} & n \text{ is inner AND node in } H, \; n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, Overview]

Solution graph for n_1 :

$C_H(n_1) = 11$



$E(n) = c(n, n') =$ cost for edge (n, n')

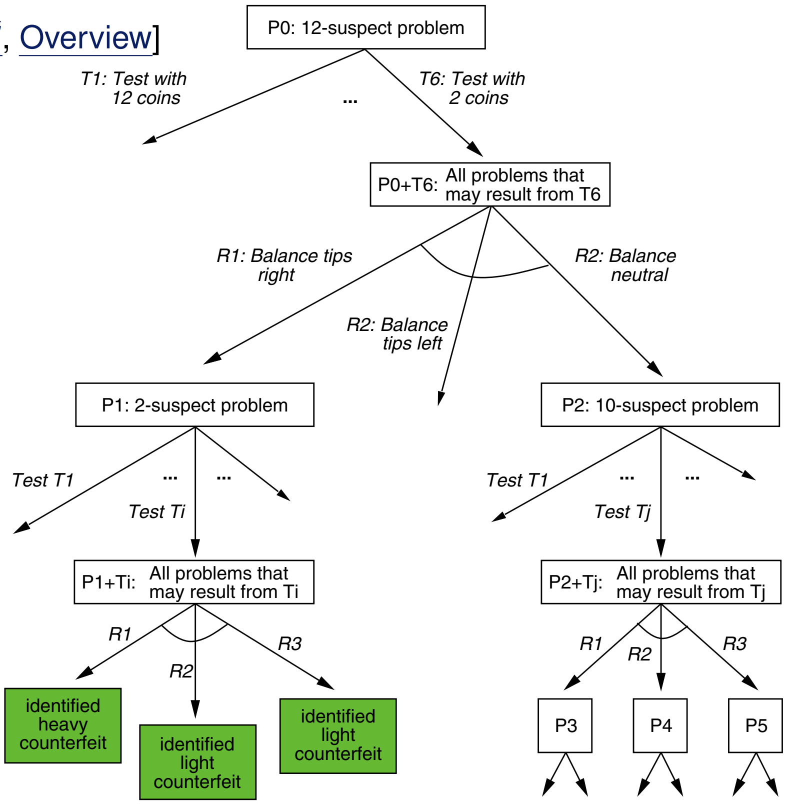
$$C_H(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ c(n, n') + C_H(n') & n \text{ is inner OR node in } H, \; n' \text{ direct successor of } n \text{ in } H \\ \text{max}_i\{c(n, n_i) + C_H(n_i)\} & n \text{ is inner AND node in } H, \; n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, $\widehat{C}$, Overview]

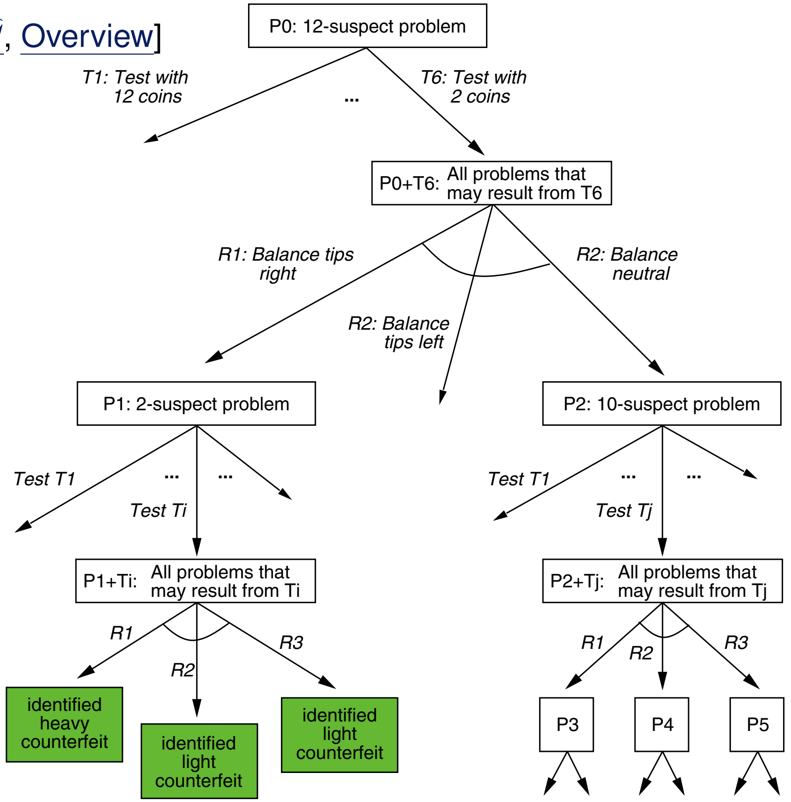
Solution graph for the counterfeit problem:

- Leaf nodes correspond to identified coin.
- OR nodes (action, strategy) specify the chosen test.
- AND nodes (reaction) model the weighing outcomes.



Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, $\widehat{C}$, Overview]



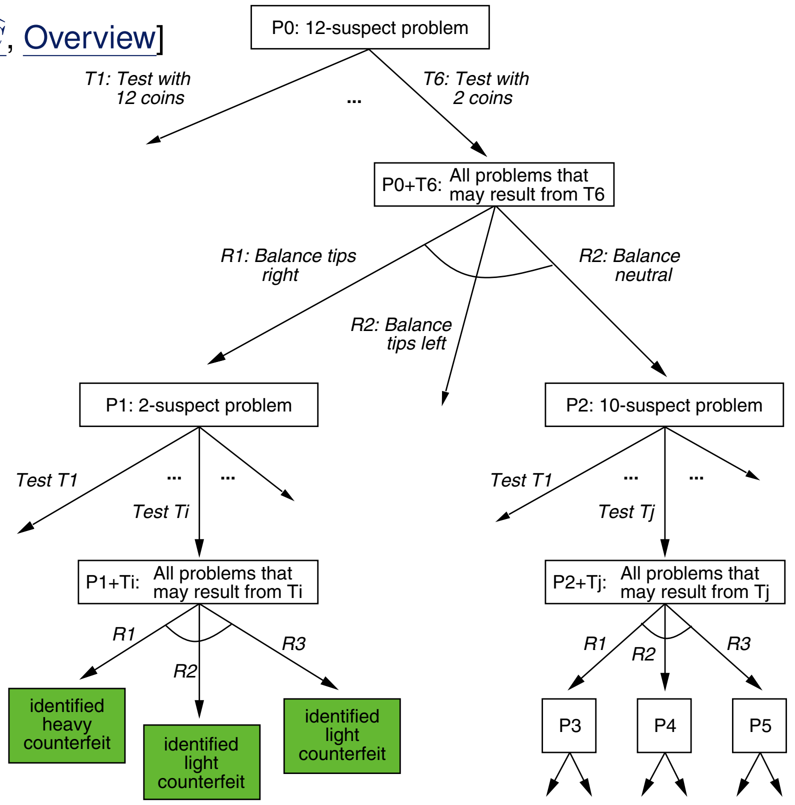
Solution graph for the counterfeit problem:

- Leaf nodes correspond to identified coin.
- OR nodes (action, strategy) specify the chosen test.
- AND nodes (reaction) model the weighing outcomes.

$$C_H(n) = \begin{cases} 0 & n \text{ is goal node and leaf in } H \\ 1 + C_H(n') & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ \max\{C_H(n_i) \mid i = 1, 2, 3\} & n \text{ is inner AND node in } H, \\ & n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Cost Functions

Illustration of C_H for AND-OR Graphs [C^*, $\widehat{C}$, Overview]



Solution graph for the counterfeit problem:

- Leaf nodes correspond to identified coin.
- OR nodes (action, strategy) specify the chosen test.
- AND nodes (reaction) model the weighing outcomes.

$$C_H(n) = \begin{cases} 0 & n \text{ is goal node and leaf in } H \\ 1 + C_H(n') & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ \sum_{i=1,2,3} p_i(n) \cdot C_H(n_i) & n \text{ is inner AND node in } H, \\ & n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Remarks:

- The recursive cost function $C_H(n)$ with max$\{C_H(n_i) \mid i = 1, 2, 3\}$ in the AND-node branch models the maximum possible number of tests (worst case costs) using the weighing operations in the solution graph H.

- The recursive cost function $C_H(n)$ with $\sum_{i=1,2,3} p_i(n) \cdot C_H(n_i)$ in the AND-node branch models the expected test effort. The $p_i, i = 1, 2, 3$ are the probabilities for the test outcomes and could be quantified as follows:

$$p_i(n) = \begin{cases} \frac{1}{2} \cdot \frac{k(n)}{12-u(n)} & i \in \{1, 3\} \\ 1 - \frac{k(n)}{12-u(n)} & i = 2 \end{cases}$$

  where $k(n)$, $k(n) \leq 12$, is number of suspicious coins that are weighed in subproblem n, and $u(n)$, $u(n) < 12$, is the number of coins classified as “not-suspect” in subproblem n.

Cost Functions

Recursive Cost Functions (continued)

If the search space graph rooted at a node s is known partially and a recursive cost function is used, cost estimates for a solution base

1. can be built upon estimates for optimum solution cost of non-goal leaf nodes in this solution base and,
2. can be computed by taking the estimations of h for granted and propagating the cost values bottom-up. Keyword: *Face-Value Principle*

Definition 19 (Heuristic Function h)

Let G be an acyclic AND-OR graph. A function h, which assigns each node n in G an estimate $h(n)$ of the optimum solution cost value $C^*(n)$, the optimum cost of a solution graph for n, is called heuristic function (for G).

Similarly, for an OR-Graph G, a heuristic function $h(n)$ returns an estimate of the optimum solution cost value $C^*(n)$, the optimum cost of a solution path from n to a goal node.

Remarks:

- Assuming $h(n) = \infty$ for unsolvable nodes n, a heuristic function h can also be used for dead end recognition.

Cost Functions

Recursive Cost Functions (continued)

Corollary 20 (Estimated Solution Cost $\widehat{C}_H$, $\widehat{C}_P$ for a Solution Base)

Let G, $\mathcal{G}$, s, and $C_H(n)$ (resp. $C_P(n)$) be defined as for $\widehat{C}_H$ (resp. $\widehat{C}_P$). Let the cost function be recursive based on F and E, and let h be a heuristic function.

Using the face-value principle, the estimated solution cost for solution bases H in G is computed as follows:

$$\widehat{C}_H(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ h(n) & n \text{ is leaf in } H \text{ but no goal node} \\ F[E(n), \widehat{C}_H(n')] & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ F[E(n), \widehat{C}_H(n_1), \ldots, \widehat{C}_H(n_k)] & n \text{ is inner AND node in } H, \\ & n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Similarly, the estimated solution cost $\widehat{C}_P$ for solution bases P is computed as:

$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] & n \text{ is inner node in } P \text{ and } n' \text{ direct successor of } n \text{ in } P \end{cases}$$

Cost Functions

Recursive Cost Functions (continued)

Corollary 20 (Estimated Solution Cost $\widehat{C}_H$, $\widehat{C}_P$ for a Solution Base)

Let G, $\mathcal{G}$, s, and $C_H(n)$ (resp. $C_P(n)$) be defined as for $\widehat{C}_H$ (resp. $\widehat{C}_P$). Let the cost function be recursive based on F and E, and let h be a heuristic function.

Using the face-value principle, the estimated solution cost for solution bases H in G is computed as follows:

$$\widehat{C}_H(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } H \\ h(n) & n \text{ is leaf in } H \text{ but no goal node} \\ F[E(n), \widehat{C}_H(n')] & n \text{ is inner OR node in } H, \\ & n' \text{ direct successor of } n \text{ in } H \\ F[E(n), \widehat{C}_H(n_1), \ldots, \widehat{C}_H(n_k)] & n \text{ is inner AND node in } H, \\ & n_i \text{ direct successors of } n \text{ in } H \end{cases}$$

Similarly, the estimated solution cost $\widehat{C}_P$ for solution bases P is computed as:

$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] & n \text{ is inner node in } P \text{ and } n' \text{ direct successor of } n \text{ in } P \end{cases}$$

Evaluation of AND-OR Graphs

Recursive Cost Functions and Efficiency

If the search space graph is an AND-OR graph rooted at a node s and is known partially and a recursive cost function is used that is defined via a

1. monotone cost measure F, i.e., for $e, c_1, \ldots, c_k, c_1', \ldots, c_k'$ with $c_1 \leq c_1', \ldots, c_k \leq c_k'$ we have

$$F[e, c_1, \ldots, c_k] \leq F[e, c_1', \ldots, c_k']$$

the (estimated) optimum solution cost can be computed bottom-up.

Similarly, a solution base can be determined which has the estimated optimum solution cost as its estimated solution cost.

If additionally the recursive cost function is based on an

2. underestimating heuristic function h, i.e., $h(n) \leq C^*(n)$

then the estimated solution cost $\widehat{C}_H(s)$ is underestimating optimum solution cost $C^*_H(s)$ for a solution base H.

Evaluation of AND-OR Graphs

Recursive Cost Functions and Efficiency (continued)

Corollary 21 (Optimum Solution Cost C^* [BF, 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 and a monotone cost measure F.

The optimum solution cost $C^*(n)$ for a node n in G can be computed as follows:

$$C^*(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } G \\ \infty & n \text{ is unsolvable leaf node in } G \\ \min_i\{F[E(n), C^*(n_i)]\} & n \text{ is inner OR node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \\ F[E(n), C^*(n_1), \ldots, C^*(n_k)] & n \text{ is inner AND node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

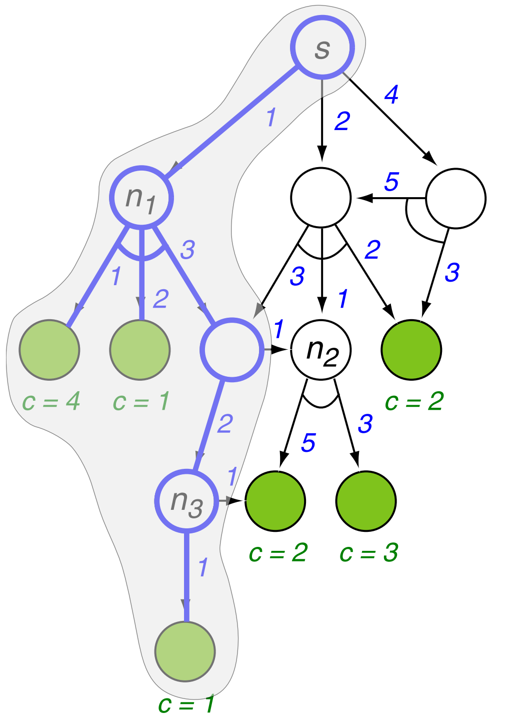
Compare to Bellman's equations.

Evaluation of AND-OR Graphs

Illustration of C^* [C_H, Overview]

Optimum solution graph for s:

$C^*(s) \equiv C^* = 8$



$E(n) = c(n, n') =$ cost for edge (n, n')

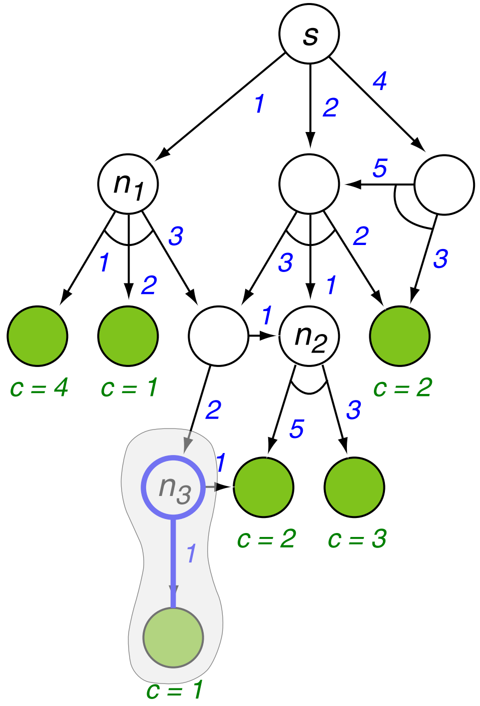
$$C^*(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } G \\ \min_i\{c(n,n_i) + C^*(n_i)\} & n \text{ is inner OR node in } G \\ & n_i \text{ direct successors of } n \text{ in } G \\ \max_i\{c(n,n_i) + C^*(n_i)\} & n \text{ is inner AND node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

Evaluation of AND-OR Graphs

Illustration of C^* [C_H, Overview]

Optimum solution graph for n_3:

$C^*(n_3) = 2$



$E(n) = c(n, n') =$ cost for edge (n, n')

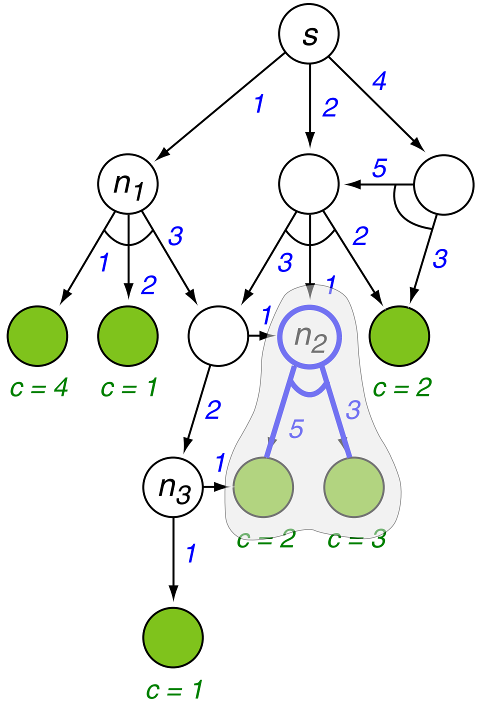
$$C^*(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } G \\ \min_i\{c(n,n_i) + C^*(n_i)\} & n \text{ is inner OR node in } G \\ & n_i \text{ direct successors of } n \text{ in } G \\ \max_i\{c(n,n_i) + C^*(n_i)\} & n \text{ is inner AND node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

Evaluation of AND-OR Graphs

Illustration of C^* [C_H, Overview]

Optimum solution graph for n_2:

$C^*(n_2) = 7$



$E(n) = c(n, n') =$ cost for edge (n, n')

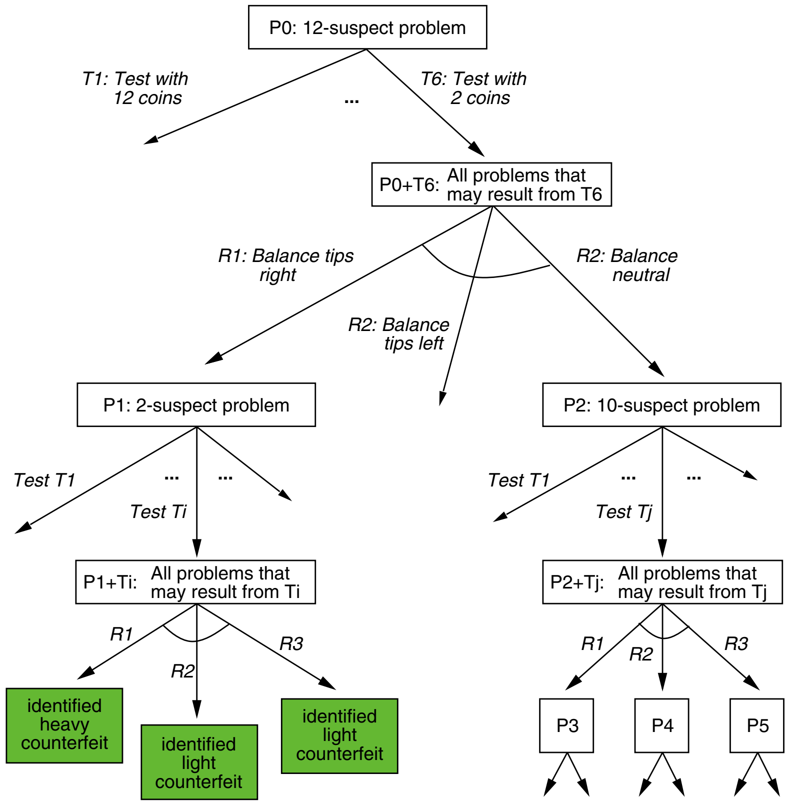
$$C^*(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } G \\ \min_i\{c(n, n_i) + C^*(n_i)\} & n \text{ is inner OR node in } G \\ & n_i \text{ direct successors of } n \text{ in } G \\ \max_i\{c(n, n_i) + C^*(n_i)\} & n \text{ is inner AND node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

Evaluation of AND-OR Graphs

Illustration of C^* [C_G, $\widehat{C}$, Overview]

Search space graph for the counterfeit problem:

- Leaf nodes correspond to identified coin.
- OR nodes (action, strategy) specify the possible tests.
- AND nodes (reaction) model the weighing outcomes.



$$C^*(n) = \begin{cases} 0 & n \text{ is goal node and leaf in } G \\ \min_i\{1 + C^*(n_i)\} & n \text{ is inner OR node in } G \\ & n_i \text{ direct successors of } n \text{ in } G \\ \sum_{i=1,2,3} p_i \cdot C^*(n_i) & n \text{ is inner AND node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

Remarks:

- If the underlying search space graph is infinite, it may be impossible to compute $C^*(n)$. Exploiting properties of E and F for a search space graph can make the computation feasible.

Evaluation of AND-OR Graphs

Recursive Cost Functions and Efficiency (continued)

Corollary 22 (Estimated Optimum Solution Cost $\widehat{C}$ [BF, Overview])

Let G be an explored subgraph of an acyclic AND-OR graph $\mathcal{G}$ rooted at s. Let h be a heuristic function and let $C_H(n)$ be a recursive cost function for $\mathcal{G}$ based on E and a monotone cost measure F.

Using the face-value principle, the estimated optimum solution cost $\widehat{C}(n)$ for a node n in G can be computed as follows:

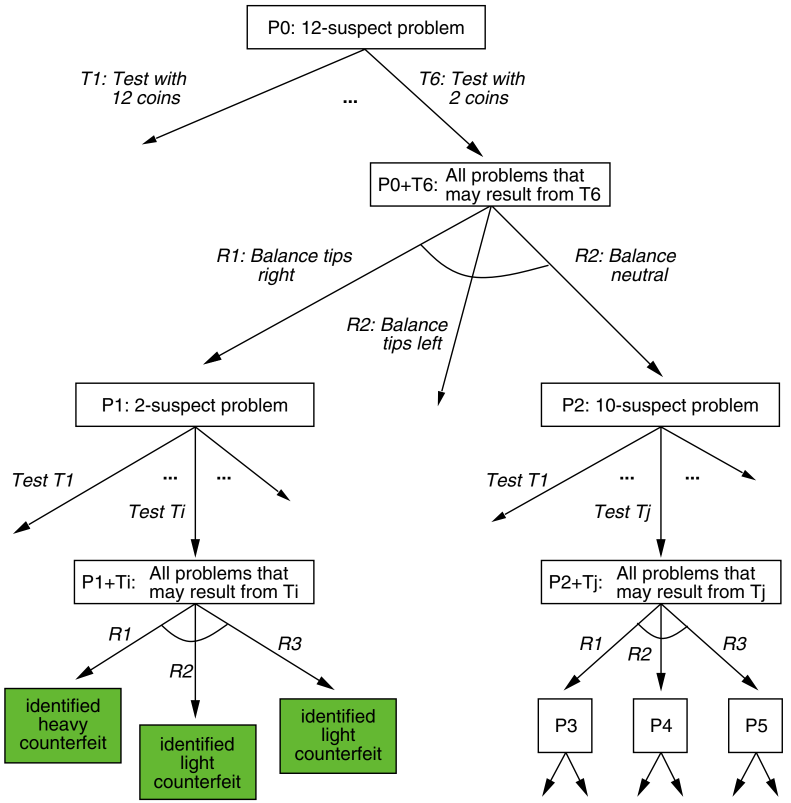
$$\widehat{C}(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } G \\ h(n) & n \text{ is leaf in } G \text{ but no goal node} \\ \min_i\{F[E(n), \widehat{C}(n_i)]\} & n \text{ is inner OR node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \\ & \\ F[E(n), \widehat{C}(n_1),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

Remarks:

- Line 2 of the definition for $\widehat{C}$ requires a useful handling regarding the value ∞. ($h(n) = \infty$ for unsolvable nodes n.) In particular, the cost measure F should return ∞ if one of its arguments is ∞.
- Line 3 and 4 of the definition for $\widehat{C}$ presume the—already computed—estimated optimum solution cost values for the direct successors of n. Their existence is guaranteed here (also for infinitely large search space graphs) since the search (= the computation of $\widehat{C}$) uses a finite portions of the search space graph, the explored subgraphs of the search space graph.
- $\widehat{C}(n)$ computes for a node n the minimum of the estimated costs among all solution bases rooted at n. In particular, $\widehat{C}(s)$ computes the estimated optimum solution cost for the entire problem, and it hence defines a most promising solution base H for s.
- Finally, cost propagation is reasonable (also in the sense of "is efficient") only for problems whose solution obey Bellman's principle of optimality.

Evaluation of AND-OR Graphs

Illustration of $\widehat{C}(n)$ [C_G, C^*, Overview]



Search space graph G for the counterfeit problem:

- Leaf nodes correspond to identified coin.
- OR nodes (action, strategy) specify the possible tests.
- AND nodes (reaction) model the weighing outcomes.

G is explored subgraph of G generated so far.

$$\widehat{C}(n) = \begin{cases} 0 & n \text{ is goal node and leaf in } G \\ h(n) & n \text{ is leaf in } G \text{ but no goal node} \\ \min_i\{1 + \widehat{C}(n_i)\} & n \text{ is inner OR node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \\ \max\{\widehat{C}(n_i) \mid i = 1, 2, 3\} & n \text{ is inner AND node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

Evaluation of AND-OR Graphs

Relation to the Algorithm GBF

```
GBF$^*(s, \textit{successors}, \perp, \star, f_1, f_2)$
      ...
 2.  LOOP
 3.    IF (OPEN $= \emptyset$) THEN RETURN($\textit{Fail}$);
4.a    $H = \textit{min\_solution\_base}(s, f_1)$;    // Find most promising solution base.
       ...
```

Evaluation of AND-OR Graphs

Relation to the Algorithm GBF

```
GBF*($s$, successors, $\bot$, $\star$, $f_1$, $f_2$)
      ...
  2.  LOOP
  3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
 4.a    $H$ = min_solution_base($s$, $f_1$);     // Find most promising solution base.
        ...
```

Corollary 23 (Most Promising Solution Base H [Overview, Solution base]**)**

Let G, $\mathcal{G}$, h, E, F and $C_H(n)$ be defined as before.

Using the face-value principle, a most promising solution base for s, H, can be characterized by the following conditions:

1. If H contains an inner OR node n, then H contains exactly one link to a direct successor n' in H, where

$$F[E(n), \widehat{C}(n')] = \min_i\{F[E(n), \widehat{C}(n_i)]\} \quad n_i \text{ direct successor of } n$$

2. If H contains an inner AND node, then H contains all links to its direct successors in H.

Evaluation of AND-OR Graphs

Relation to the Algorithm GBF [BF]

```
GBF*($s$, successors, $\bot$, $\star$, $f_1$, $f_2$)
      ...
  2.  LOOP
  3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
 4.a    $H$ = min_solution_base($s$, $f_1$);
        solved_labeling($H$);    // Check if $H$ is a solution graph.
        IF $\star(s)$ THEN RETURN($H$);    // Delayed termination.
        ...
```

Define $f_1(H)$ as $\widehat{C}_H(s)$:

- ➜ f_1 is a recursive evaluation function.
- ➜ Algorithm GBF becomes Algorithm AO.

Delayed termination:

- ➜ Algorithm GBF becomes Algorithm GBF*.
- ➜ Algorithm AO becomes Algorithm AO*.

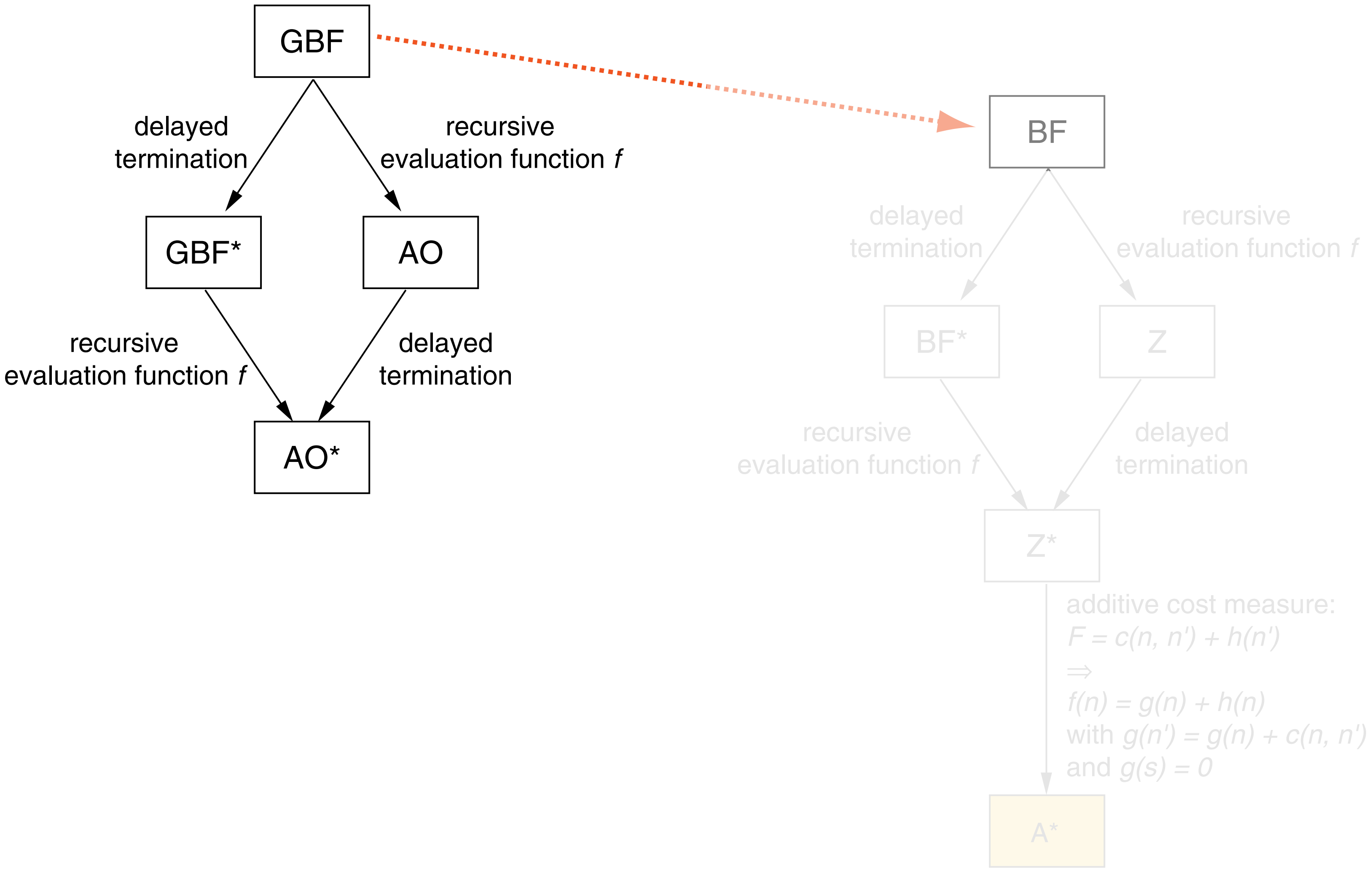
Remarks:

- $\widehat{C}$ is a recursive cost function. Hence, the determination of a most promising solution base H can happen bottom-up along with the computation of $\widehat{C}(n)$ for n in G, simply by propagating the cost of the cheapest OR node.
- The bottom-up propagation yields a minimal solution graph (or solution base) for each node in G.
- Leaf nodes in G are either solved rest problems (and are on CLOSED) or nodes on OPEN that are currently not labeled “unsolvable”.
- Recall that $\widehat{C}_H(s)$ computes the estimated solution cost for a given solution base G. If h is optimistic and if, in addition, F is monotone, then $f_1(H)$ (defined as $\widehat{C}_H(s)$) is optimistic for all solution bases H (not only for a most promising solution base).

  A proof of this claim is given in the lab class.

Evaluation of AND-OR Graphs

Taxonomy of Best-First Algorithms



Evaluation of State-Space Graphs

Recursive Cost Functions and Efficiency

If the search space graph is an OR graph rooted at a node s is known partially and a recursive cost function is used that is defined via a

1. monotone cost measure F, i.e., for e, c, c' with $c \leq c'$ we have
$$F[e, c] \leq F[e, c']$$

the (estimated) optimum solution cost can be computed bottom-up.

If additionally the recursive cost function is based on an

2. underestimating heuristic function h, i.e., $h(n) \leq C^*(n)$

then the estimated solution cost $\widehat{C}_P(s)$ is underestimating optimum solution cost $C^*_P(s)$ for a solution base P.

Evaluation of State-Space Graphs

Recursive Cost Functions and Efficiency (continued)

Corollary 24 (Optimum Solution Cost C^* [GBF, Overview])

Let G be an OR graph rooted at s. Let $C_P(n)$ be a recursive cost function for G based on E and a monotone cost measure F.

The optimum solution cost $C^*(n)$ for a node n in G can be computed as follows:

$$C^*(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } G \\ \infty & n \text{ is unsolvable leaf node in } G \\ \min_i\{F[E(n), C^*(n_i)]\} & n \text{ is inner OR node in } G, \\ & n_i \text{ direct successors of } n \text{ in } G \end{cases}$$

Evaluation of State-Space Graphs

Recursive Cost Functions and Efficiency (continued)

Corollary 25 (Estimated Optimum Solution Cost $\widehat{C}$ [GBF, Overview])

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

Using the face-value principle, the estimated optimum solution cost $\widehat{C}(n)$ for a node n in G can be computed as follows:

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

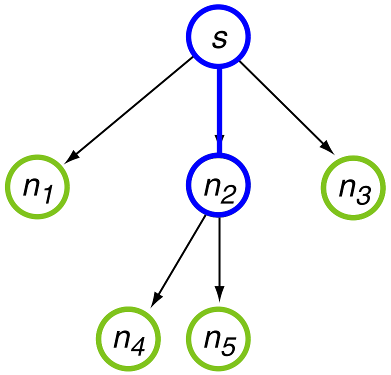
Compare to Bellman's equations.

Remarks:

- Computing $\widehat{C}(n)$ is based on an explored subgraph G of a state-space graph. That means, all solution bases in G are considered and not only those, maintained by BF in its traversal tree.
- If algorithm BF were equipped with a dead end recognition function $\perp(n)$, no unsolvable node would be stored. A dead end recognition could also be incorporated in h in such a way that h returns ∞ for unsolvable nodes.
- $\widehat{C}(n)$ computes for a node n the minimum of the estimated costs among all solution bases rooted at n (paths from n to leaf node in G). In particular, $\widehat{C}(s)$ computes the estimated optimum solution cost for the entire problem, and it hence defines a most promising solution base.
- $\widehat{C}_P$, the estimated solution cost for a solution base P is a recursive cost function. Hence, $\widehat{C}_{P_{s-n}}(s)$ can be computed bottom-up, from n to s along path P_{s-n}.

Evaluation of State-Space Graphs

Illustration of $\widehat{C}(n)$, $\widehat{C}_P(n)$



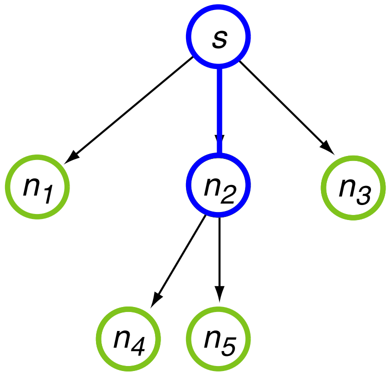
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Evaluation of State-Space Graphs

Illustration of $\widehat{C}(n)$, $\widehat{C}_P(n)$

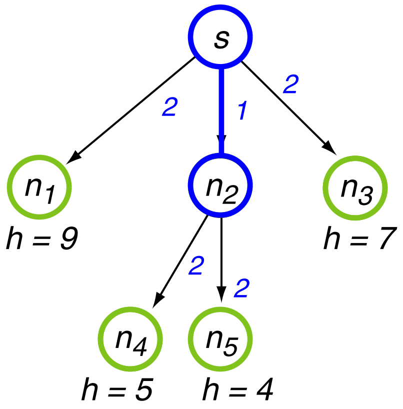


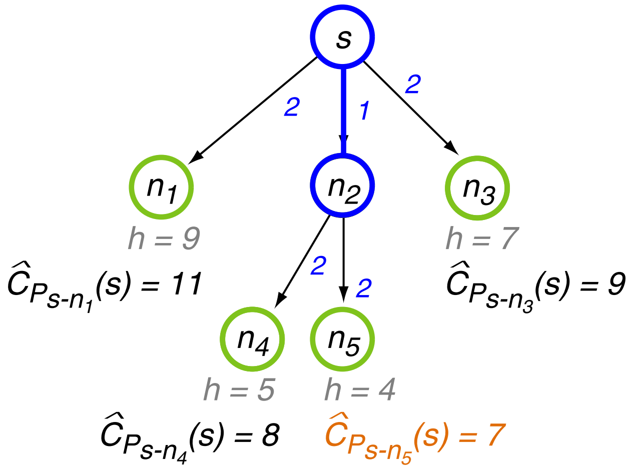
○ Node on OPEN

○ Node on CLOSED

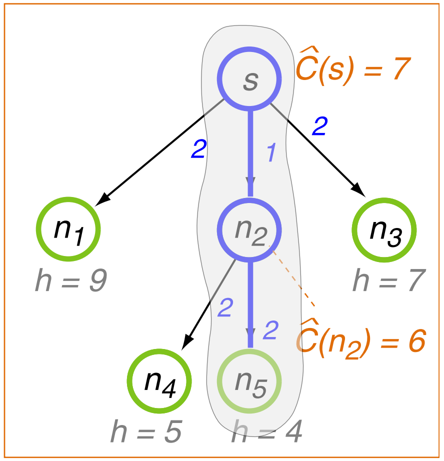
● Solved rest problem

Computation of $\widehat{C}_{P_{s-n}}(s)$ for each node n on OPEN:





s
2
1
2
n1
n2
n3
h = 9
h = 7
$\widehat{C}_{P_{s\text{-}n_1}}(s) = 11$
$\widehat{C}_{P_{s\text{-}n_3}}(s) = 9$
2
2
n4
n5
h = 5
h = 4
$\widehat{C}_{P_{s\text{-}n_4}}(s) = 8$
$\widehat{C}_{P_{s\text{-}n_5}}(s) = 7$



Most promising solution base

Evaluation of State-Space Graphs

Additive Cost Measures

To compute $\widehat{C}_{P_{s-n}}(s)$, a bottom-up propagation from n to s may not be necessary. Dependent on the cost measure F, it can be sufficient to pass a single (several) parameter(s) *top-down*, from a node to its direct successors.

Illustration for F = “$+$” and a path $P_{s-n} = (s, n_1, \ldots, n_k, n)$ from s to n:

$$\begin{aligned}
\widehat{C}_{P_{s-n}}(s) &= F[E(s), \widehat{C}_{P_{s-n}}(n_1)] \\
&= F[E(s),\ F[E(n_1),\ F[E(n_2),\ \ldots,\ F[E(n_k), h(n)] \ldots]\,]\,] \\
&= c(s, n_1) + c(n_1, n_2) + \ \ldots\ + c(n_k, n) + h(n) \\
&= g_{P_{s-n}}(n) + h(n)
\end{aligned}$$

Evaluation of State-Space Graphs

Additive Cost Measures

To compute $\widehat{C}_{P_{s-n}}(s)$, a bottom-up propagation from n to s may not be necessary. Dependent on the cost measure F, it can be sufficient to pass a single (several) parameter(s) *top-down*, from a node to its direct successors.

Illustration for F = "$+$" and a path $P_{s-n} = (s, n_1, \ldots, n_k, n)$ from s to n:

$$\begin{aligned}
\widehat{C}_{P_{s-n}}(s) &= F[E(s), \widehat{C}_{P_{s-n}}(n_1)] \\
&= F[E(s),\ F[E(n_1),\ F[E(n_2),\ \ldots,\ F[E(n_k), h(n)] \ldots]\,]\,] \\
&= c(s, n_1) + c(n_1, n_2) + \ \ldots\ + c(n_k, n) + h(n) \\
&= g_{P_{s-n}}(n) + h(n)
\end{aligned}$$

Definition 26 (Additive Cost Measure)

Let n be an OR node in a search space graph, n' a direct successor of n, and F a cost measure. F is called additive cost measure iff ($\leftrightarrow$) it is of the following form:

$$F[E(n), \widehat{C}(n')] = E(n) + \widehat{C}(n')$$

Remarks:

- $g_{P_{s-n}}(n)$ is the sum of the edge costs of a path $P_{s-n} = (s, n_1, \ldots, n_k, n)$ from s to n.
- $h(n)$ estimates the rest problem cost at node n.
- Here, we use the computation of estimated optimum solution cost extending a solution base for recursive cost functions.

Evaluation of State-Space Graphs

Relation to the Algorithm BF [GBF]

```
BF*($s$, successors, $\star$, $f$)
      ...
  2.  LOOP
  3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
  4.    $n = min(\text{OPEN}, f)$;    // Find most promising solution base.
        IF $\star(n)$ THEN RETURN($n$);    // Delayed termination.
```

Define $f(n)$ as $\widehat{C}_P(s)$:

- ➜ $f(n)$ is a recursive evaluation function.
- ➜ Algorithm BF becomes Algorithm Z.

Delayed termination:

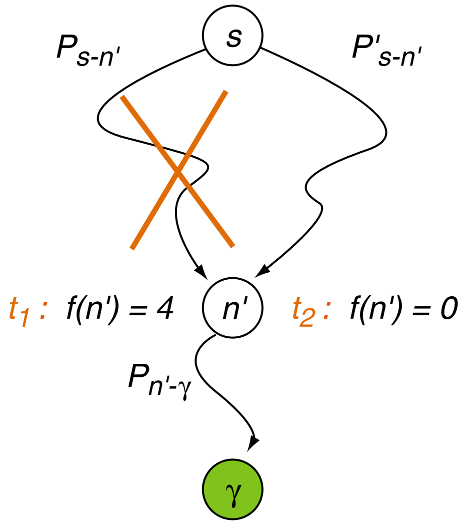
- ➜ Algorithm BF becomes Algorithm BF*.
- ➜ Algorithm Z becomes Algorithm Z*.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation [S:IV Relation between GBF and BF]

Recall that BF discards the inferior of two paths leading to the same node:

```
5.    FOREACH $n'$ IN successors$(n)$ DO
        ...
        IF $(n' \notin$ OPEN AND $n' \notin$ CLOSED$)$
        THEN ...
        ELSE
          IF $\boxed{(f_n(n') < f(n'))}$
          THEN
            update_backpointer$(n', n)$;
            IF $n' \in$ CLOSED THEN ...ENDIF
          ENDIF
```

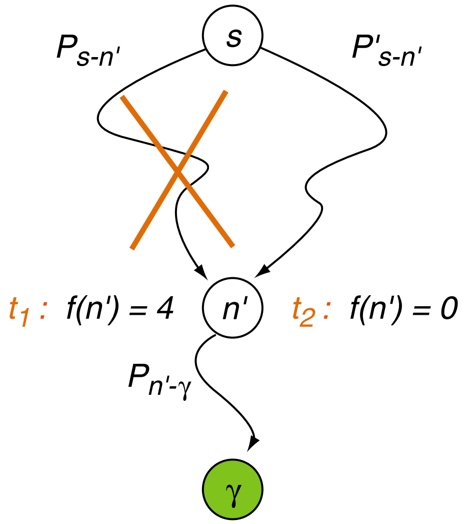


Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation [S:IV Relation between GBF and BF]

Recall that BF discards the inferior of two paths leading to the same node:

```
5.   FOREACH $n'$ IN $successors(n)$ DO
       ...
       IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
       THEN ...
       ELSE
         IF $\boxed{(f_n(n') < f(n'))}$
         THEN
           $update\_backpointer(n', n)$;
           IF $n' \in$ CLOSED THEN ...ENDIF
         ENDIF
```



➜ An optimistic evaluation function f is not sufficient for Z* to be optimum.

➜ Necessary: cost estimations for alternative solution bases must be independent of their shared continuation.

Formally: The cost function $\widehat{C}_P(s)$ must be *order-preserving*.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 27 (Order-Preserving)

A cost function $\widehat{C}_P(s)$ is called order-preservingif 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:

$$\widehat{C}_{P_{s-n'}}(s) \;\leq\; \widehat{C}_{P'_{s-n'}}(s) \quad \Rightarrow \quad \widehat{C}_{P_{s-n}}(s) \;\leq\; \widehat{C}_{P'_{s-n}}(s)$$

P_{s-n} and P'_{s-n} denote the paths from s to n that result from concatenating the paths $P_{s-n'}$ and $P'_{s-n'}$ with $P_{n'-n}$.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 27 (Order-Preserving)

A cost function $\widehat{C}_P(s)$ is called order-preservingif 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:

$$\widehat{C}_{P_{s-n'}}(s) \;\leq\; \widehat{C}_{P'_{s-n'}}(s) \quad\Rightarrow\quad \widehat{C}_{P_{s-n}}(s) \;\leq\; \widehat{C}_{P'_{s-n}}(s)$$

P_{s-n} and P'_{s-n} denote the paths from s to n that result from concatenating the paths $P_{s-n'}$ and $P'_{s-n'}$ with $P_{n'-n}$.

Corollary 28 (Order-Preserving)

If a cost function $\widehat{C}_P(s)$ is order-preserving, then 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:

$$\widehat{C}_{P_{s-n}}(s) \;>\; \widehat{C}_{P'_{s-n}}(s) \quad\Rightarrow\quad \widehat{C}_{P_{s-n'}}(s) \;>\; \widehat{C}_{P'_{s-n'}}(s)$$

and

$$\widehat{C}_{P_{s-n'}}(s) \;=\; \widehat{C}_{P'_{s-n'}}(s) \quad\Rightarrow\quad \widehat{C}_{P_{s-n}}(s) \;=\; \widehat{C}_{P'_{s-n}}(s)$$

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 29 (Order-Preserving for Solution Paths)

A cost function $\widehat{C}_P(s)$ is called order-preserving for solution paths if for all nodes n' and paths $P_{s-n'}$, $P'_{s-n'}$ from s to n', and for all nodes γ and paths $P_{n'-\gamma}$ from n' to γ holds:

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

$P_{s-\gamma}$ and $P'_{s-\gamma}$ denote the paths from s to γ that result from concatenating the paths $P_{s-n'}$ and $P'_{s-n'}$ with $P_{n'-\gamma}$.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 29 (Order-Preserving for Solution Paths)

A cost function $\widehat{C}_P(s)$ is called order-preserving for solution paths if for all nodes n' and paths $P_{s-n'}$, $P'_{s-n'}$ from s to n', and for all nodes γ and paths $P_{n'-\gamma}$ from n' to γ holds:

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

$P_{s-\gamma}$ and $P'_{s-\gamma}$ denote the paths from s to γ that result from concatenating the paths $P_{s-n'}$ and $P'_{s-n'}$ with $P_{n'-\gamma}$.

Corollary 30 (Order-Preserving for Solution Paths)

An order-preserving cost function $\widehat{C}_P(s)$ is order-preserving for solution paths.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Lemma 31 (Order-Preserving)

Evaluation functions f that rely on additive cost measures $F[e, c] = e + c$ are order-preserving.

Define $f(n)$ as $\widehat{C}_{P_{s-n}}(s) = g_{P_{s-n}}(n) + h(n)$ ($f = g + h$ for short):

➜ Algorithm Z* becomes Algorithm A*.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Lemma 31 (Order-Preserving)

Evaluation functions f that rely on additive cost measures $F[e, c] = e + c$ are order-preserving.

Define $f(n)$ as $\widehat{C}_{P_{s-n}}(s) = g_{P_{s-n}}(n) + h(n)$ ($f = g + h$ for short):

➜ Algorithm Z* becomes Algorithm A*.

Proof (of Lemma)

Let $P_{s-n'} = (s, n_{1,1}, \ldots, n_{1,k}, n')$, $P'_{s-n'} = (s, n_{2,1}, \ldots, n_{2,l}, n')$ be paths from s to n', where

$$\widehat{C}_{P_{s-n'}}(s) = c(s, n_{1,1}) + \ldots + c(n_{1,k}, n') + h(n') \;\leq\; c(s, n_{2,1}) + \ldots + c(n_{2,l}, n') + h(n') = \widehat{C}_{P'_{s-n'}}(s)$$

Let $P_{n'-n} = (n', n_1, \ldots, n_r, n)$ be a path from n' to n. Then follows:

$$\begin{array}{rrcl} & c(s, n_{1,1}) + \ldots + c(n_{1,k}, n') + & & c(s, n_{2,1}) + \ldots + c(n_{2,l}, n') + \\ & c(n', n_1) + \ldots + c(n_r, n) + h(n) & \leq & c(n', n_1) + \ldots + c(n_r, n) + h(n) \\ \Leftrightarrow & \widehat{C}_{P_{s-n}}(s) & \leq & \widehat{C}_{P'_{s-n}}(s) \end{array}$$

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Lemma 31 (Order-Preserving)

Evaluation functions f that rely on additive cost measures $F[e, c] = e + c$ are order-preserving.

Define $f(n)$ as $\widehat{C}_{P_{s-n}}(s) = g_{P_{s-n}}(n) + h(n)$ ($f = g + h$ for short):

➜ Algorithm Z* becomes Algorithm A*.

Proof (of Lemma)

Let $P_{s-n'} = (s, n_{1,1}, \ldots, n_{1,k}, n')$, $P'_{s-n'} = (s, n_{2,1}, \ldots, n_{2,l}, n')$ be paths from s to n', where

$$\widehat{C}_{P_{s-n'}}(s) = c(s, n_{1,1}) + \ldots + c(n_{1,k}, n') + h(n') \;\le\; c(s, n_{2,1}) + \ldots + c(n_{2,l}, n') + h(n') = \widehat{C}_{P'_{s-n'}}(s)$$

Let $P_{n'-n} = (n', n_1, \ldots, n_r, n)$ be a path from n' to n. Then follows:

$$\begin{array}{lrcl} & c(s, n_{1,1}) + \ldots + c(n_{1,k}, n') + & & c(s, n_{2,1}) + \ldots + c(n_{2,l}, n') + \\ & c(n', n_1) + \ldots + c(n_r, n) + h(n) & \le & c(n', n_1) + \ldots + c(n_r, n) + h(n) \\ \Leftrightarrow & \widehat{C}_{P_{s-n}}(s) & \le & \widehat{C}_{P'_{s-n}}(s) \end{array}$$

Remarks:

- $g(n)$ denotes the sum of the edge cost values along the pointer path from s to n. Since A* as BF* variant maintains for each node generated at each point in time a unique backpointer, there is only one solution base for each terminal node in the explored subgraph G of the search space graph for which a cost value has to be computed. Therefore, $g_{P_{s-n}}(n)$ can be seen as a function $g(n)$ that is only dependent from n.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Example for a cost function that is recursive but *not* order-preserving:

$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] = |c(n,n') + \widehat{C}_P(n') - 5| & n \text{ is inner node in } P, \\ & n' \text{ direct successor of } n \text{ in } P \end{cases}$$

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

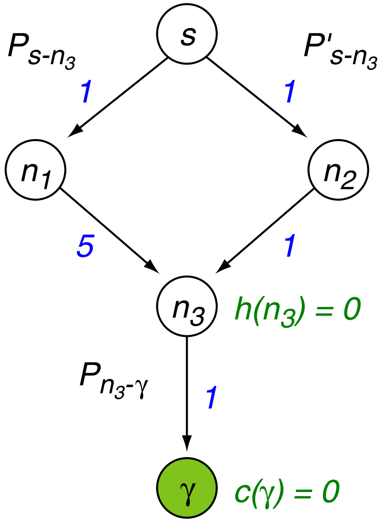
Example for a cost function that is recursive but *not* order-preserving:

$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] = |c(n,n') + \widehat{C}_P(n') - 5| & n \text{ is inner node in } P, \\ & n' \text{ direct successor of } n \text{ in } P \end{cases}$$

$$P_{s-n_3} = (s, n_1, n_3)$$

$$P'_{s-n_3} = (s, n_2, n_3)$$

$$P_{n_3-\gamma} = (n_3, \gamma)$$



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Example for a cost function that is recursive but *not* order-preserving:

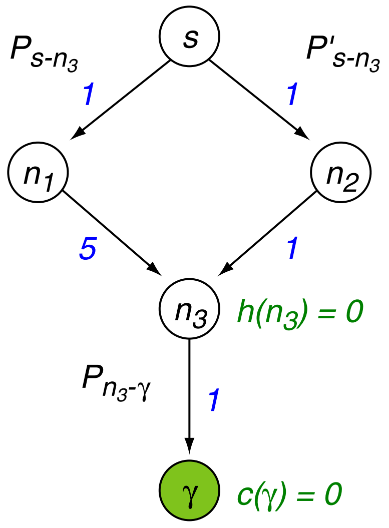
$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] = |c(n,n') + \widehat{C}_P(n') - 5| & n \text{ is inner node in } P, \\ & n' \text{ direct successor of } n \text{ in } P \end{cases}$$

$$P_{s-n_3} = (s, n_1, n_3)$$

$$P'_{s-n_3} = (s, n_2, n_3)$$

$$P_{n_3-\gamma} = (n_3, \gamma)$$

$$\widehat{C}_{P_{s-n_3}}(s) = |1 + |5 + 0 - 5| - 5| = 4$$



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Example for a cost function that is recursive but *not* order-preserving:

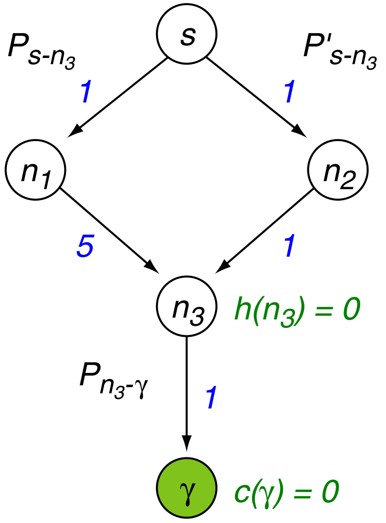
$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] = |c(n,n') + \widehat{C}_P(n') - 5| & n \text{ is inner node in } P, \\ & n' \text{ direct successor of } n \text{ in } P \end{cases}$$

$$P_{s-n_3} = (s, n_1, n_3)$$

$$P'_{s-n_3} = (s, n_2, n_3)$$

$$P_{n_3-\gamma} = (n_3, \gamma)$$

$$\widehat{C}_{P_{s-n_3}}(s) = |1 + |5 + 0 - 5| - 5| = 4$$



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Example for a cost function that is recursive but *not* order-preserving:

$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] = |c(n,n') + \widehat{C}_P(n') - 5| & n \text{ is inner node in } P, \\ & n' \text{ direct successor of } n \text{ in } P \end{cases}$$

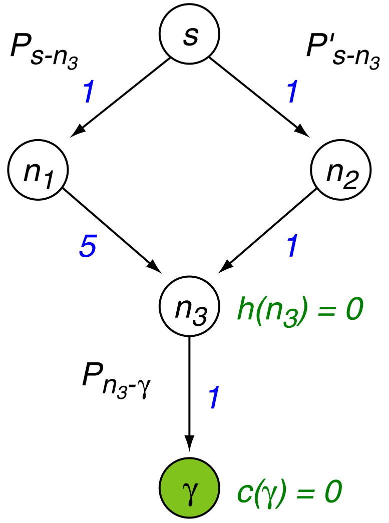
$$P_{s-n_3} = (s, n_1, n_3)$$

$$P'_{s-n_3} = (s, n_2, n_3)$$

$$P_{n_3-\gamma} = (n_3, \gamma)$$

$$\widehat{C}_{P_{s-n_3}}(s) = |1 + |5 + 0 - 5| - 5| = 4$$

$$\widehat{C}_{P'_{s-n_3}}(s) = |1 + |1 + 0 - 5| - 5| = 0$$



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Example for a cost function that is recursive but *not* order-preserving:

$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] = |c(n,n') + \widehat{C}_P(n') - 5| & n \text{ is inner node in } P, \\ & n' \text{ direct successor of } n \text{ in } P \end{cases}$$

$$P_{s-n_3} = (s, n_1, n_3)$$

$$P'_{s-n_3} = (s, n_2, n_3)$$

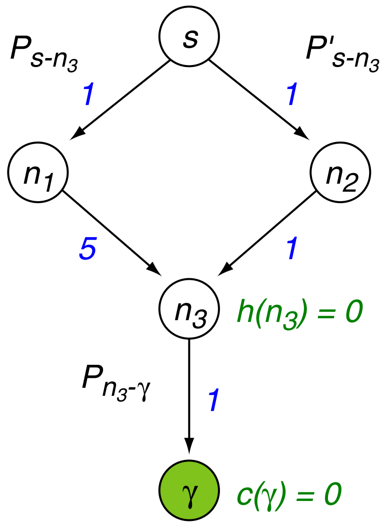
$$P_{n_3-\gamma} = (n_3, \gamma)$$

$$\widehat{C}_{P_{s-n_3}}(s) = |1 + |5 + 0 - 5| - 5| = 4$$

$$\widehat{C}_{P'_{s-n_3}}(s) = |1 + |1 + 0 - 5| - 5| = 0$$

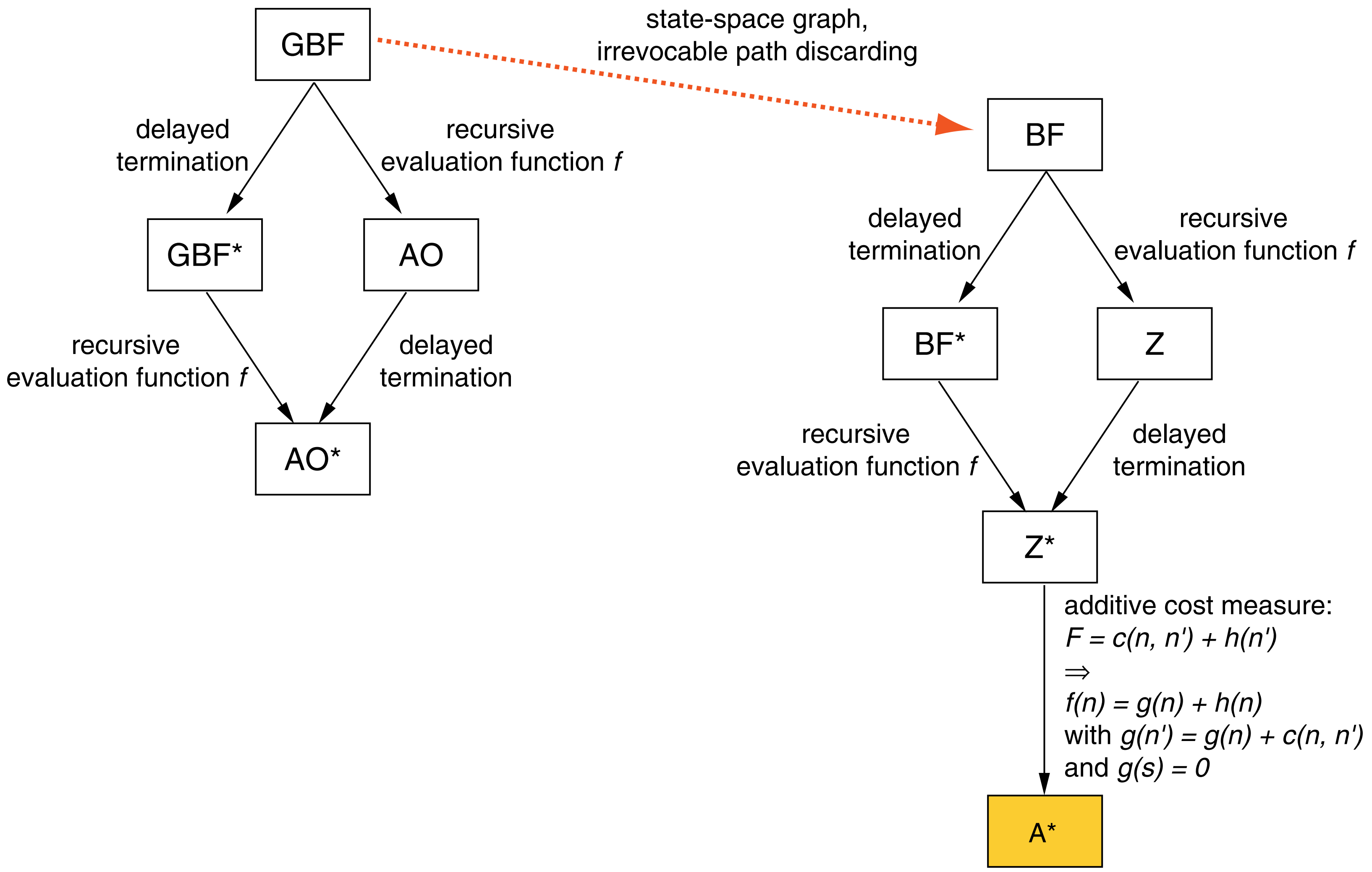
$$\widehat{C}_{P_{s-\gamma}}(s) = |1 + |5 + |1 + 0 - 5| - 5| - 5| = 0$$

$$\widehat{C}_{P'_{s-\gamma}}(s) = |1 + |1 + |1 + 0 - 5| - 5| - 5| = 4$$



Evaluation of State-Space Graphs

Taxonomy of Best-First Algorithms (continued)


GBF
delayed termination
recursive evaluation function *f*
GBF*
AO
recursive evaluation function *f*
delayed termination
AO*
state-space graph, irrevocable path discarding
BF
delayed termination
recursive evaluation function *f*
BF*
Z
recursive evaluation function *f*
delayed termination
Z*
additive cost measure:
F = c(n, n') + h(n')
⇒
f(n) = g(n) + h(n)
with *g(n') = g(n) + c(n, n')*
and *g(s) = 0*
A*

Algorithm A*

Algorithm: A*

Input: s. Start node representing the initial problem.
successors(n). Returns the successors of node n.
$\star(n)$. Predicate that is *True* if n is a goal node.
$h(n)$. Heuristic cost estimation for node n, where $f(n) = \widehat{C}_{P_{s-n}}(s) = g(n) + h(n)$

Output: An optimum goal node or the symbol *Fail*.

Algorithm A* [BF, BF$\star$]

```
A*(s, successors, ⋆, h)

 1.  insert(s, OPEN);
     g(s) = 0;
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
 4.    n = min(OPEN, g + h);     // Most promising solution base minimizes f(n).
       IF ⋆(n) THEN RETURN(n);     // Delayed termination.
       remove(n, OPEN); push(n, CLOSED);
 5.    FOREACH n' IN successors(n) DO     // Expand n.
         g_new = g(n) + c(n, n');
         IF (n' ∉ OPEN AND n' ∉ CLOSED)
         THEN   // n' encodes a new state.
           add_backpointer(n', n);  g(n') = g_new;
           insert(n', OPEN);
         ELSE   // n' encodes an already visited state.
           IF (g_new < g(n'))
           THEN   // The state of n' is reached via a cheaper path.
             update_backpointer(n', n);  g(n') = g_new;
             IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
           ENDIF
         ENDIF
       ENDDO
 6.  ENDLOOP
```

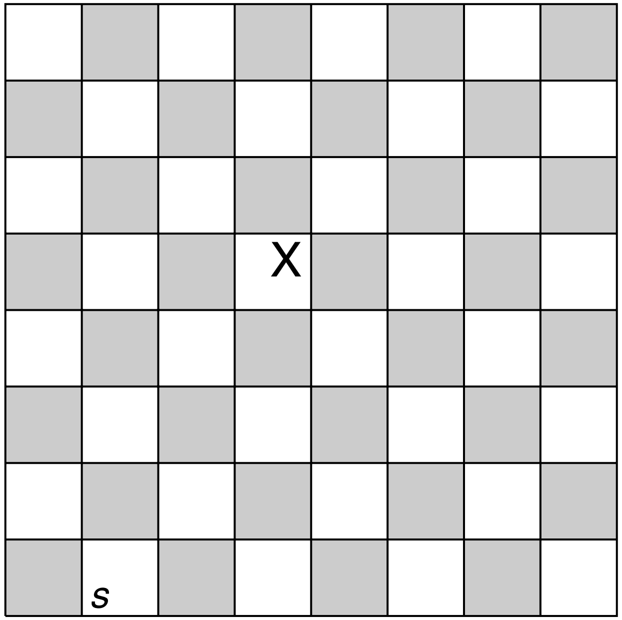
Remarks:

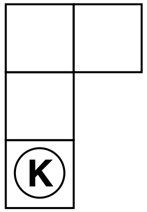
- $g(n)$ is the sum of the edge costs of the current backpointer path $P_{s-n} = (s, \ldots, n)$ from s to n.
- $h(n)$ estimates the optimum rest problem cost at node n.
- $h(n) = c(n)$ is assumed for all goal nodes n. Often, we even have $h(n) = c(n) = 0$ for all goal nodes n.

Algorithm A*

Example: Knight Moves

Search a shortest sequence of knight moves leading from s to X.





Knight move

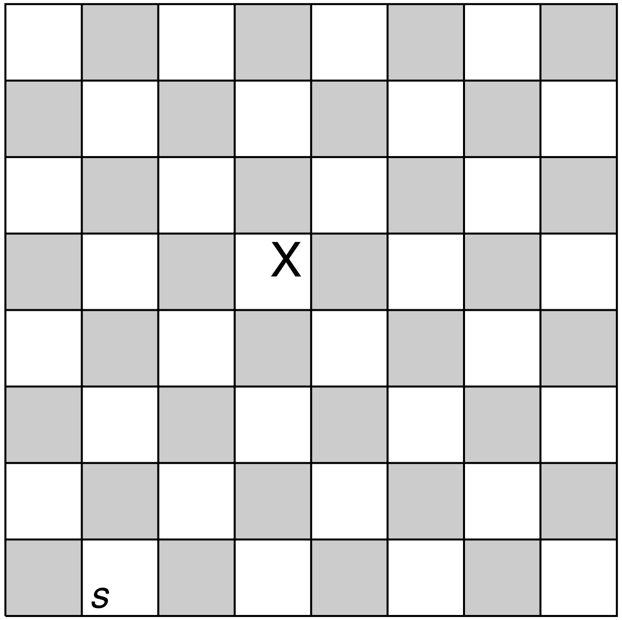
Let n' be a direct successor of n.

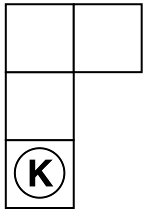
- $f(n') = g(n') + h(n')$
- $g(n') = g(n) + c(n, n')$
- $g(s) = 0$
- $c(n, n') = 1$

Algorithm A*

Example: Knight Moves

Search a shortest sequence of knight moves leading from s to X.





Knight move

Let n' be a direct successor of n.

- $f(n') = g(n') + h(n')$
- $g(n') = g(n) + c(n, n')$
- $g(s) = 0$
- $c(n, n') = 1$

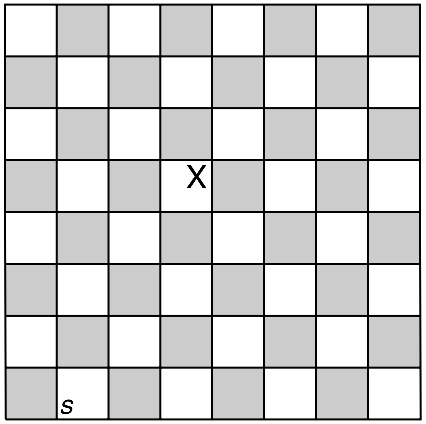
$$h_1 = \left\lceil \frac{\#rows}{2} \right\rceil$$

$$h_2 = \left\lceil \frac{\max\{\#rows,\ \#columns\}}{2} \right\rceil$$

$$h_3 = \left\lceil \frac{\#rows\ +\ \#columns}{3} \right\rceil$$

Algorithm A*

Example: Knight Moves (continued)



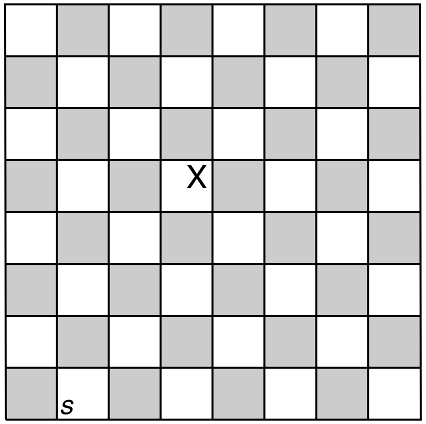
OPEN	CLOSED
$\{s\}$	$\{\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2

$h = h_1 = \lceil \frac{\#rows}{2} \rceil$

Algorithm A*

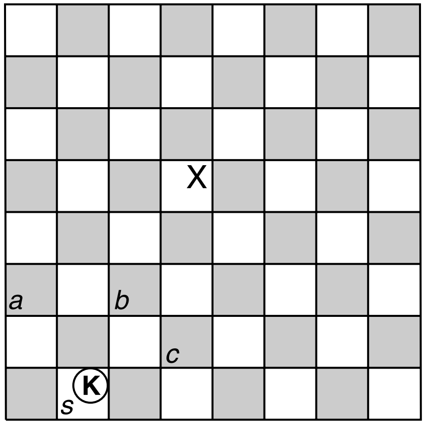
Example: Knight Moves (continued)



OPEN	CLOSED
$\{s\}$	$\{\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2

$h = h_1 = \lceil \frac{\#rows}{2} \rceil$

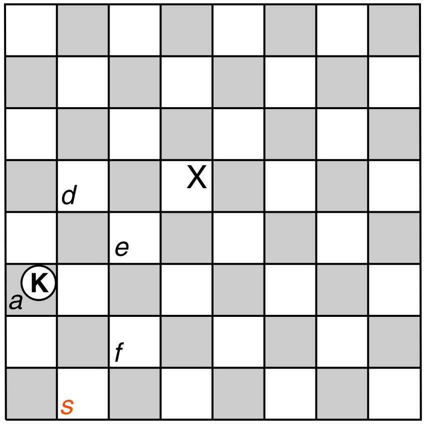


OPEN	CLOSED
$\{a, b, c\}$	$\{s\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3

Algorithm A*

Example: Knight Moves (continued)

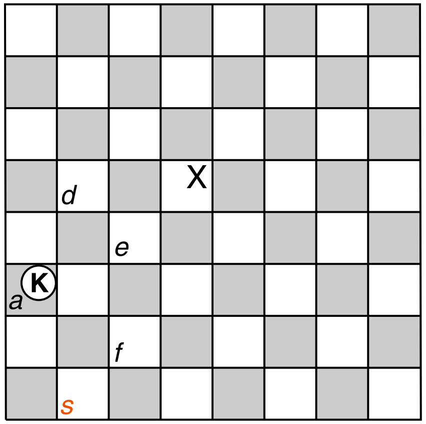


OPEN	CLOSED
$\{d, b, c, e, f\}$	$\{a, s\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3
d	2	0	2
e	2	1	3
f	2	2	4

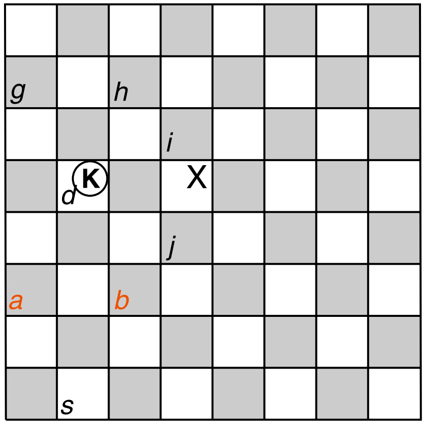
Algorithm A*

Example: Knight Moves (continued)



OPEN	CLOSED
$\{d, b, c, e, f\}$	$\{a, s\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3
d	2	0	2
e	2	1	3
f	2	2	4

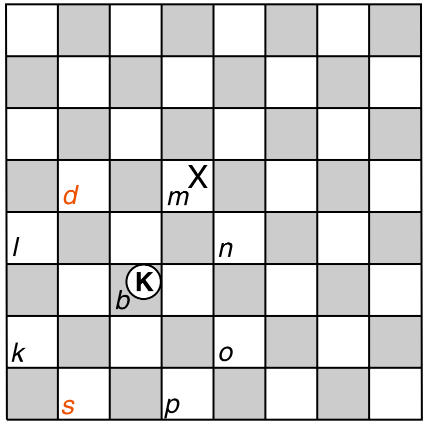


OPEN	CLOSED
$\{b, c, e, f, g, h, i, j\}$	$\{d, a, s\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3
d	2	0	2
e	2	1	3
f	2	2	4
g	3	1	4
h	3	1	4
i	3	1	4
j	3	1	4

Algorithm A*

Example: Knight Moves (continued)

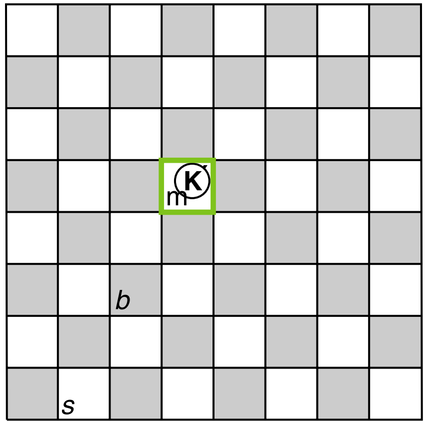


OPEN	CLOSED
$\{m, c, e, l, n, f, g, h, i, j, k, o, p\}$	$\{b, d, a, s\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3
d	2	0	2
e	2	1	3
f	2	2	4
g	3	1	4
h	3	1	4
i	3	1	4
j	3	1	4
m	2	0	2
l	2	1	3
n	2	1	3
k	2	2	4
o	2	2	4
p	2	2	4

Algorithm A*

Example: Knight Moves (continued)



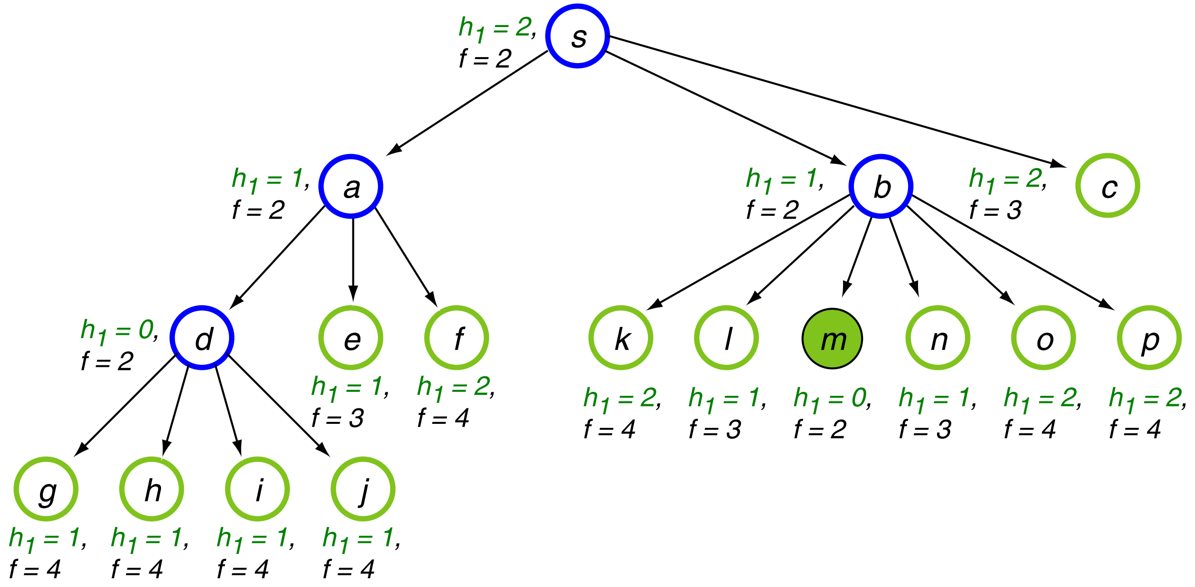
OPEN	CLOSED
$\{c, e, l, n, f, g, h, i, j, k, o, p\}$	$\{m, b, d, a, s\}$

n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3
d	2	0	2
e	2	1	3
f	2	2	4
g	3	1	4
h	3	1	4
i	3	1	4
j	3	1	4
m	2	0	2
l	2	1	3
n	2	1	3
k	2	2	4
o	2	2	4
p	2	2	4

Algorithm A*

Example: Knight Moves (continued)

Analyzed part of the search space graph:



○ Node on OPEN

○ Node on CLOSED

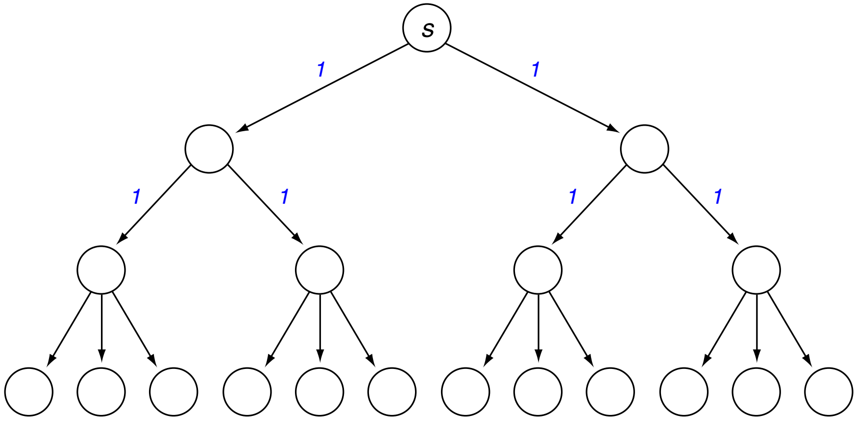
● Solved rest problem

Algorithm A*

Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.

Algorithm A*

Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.



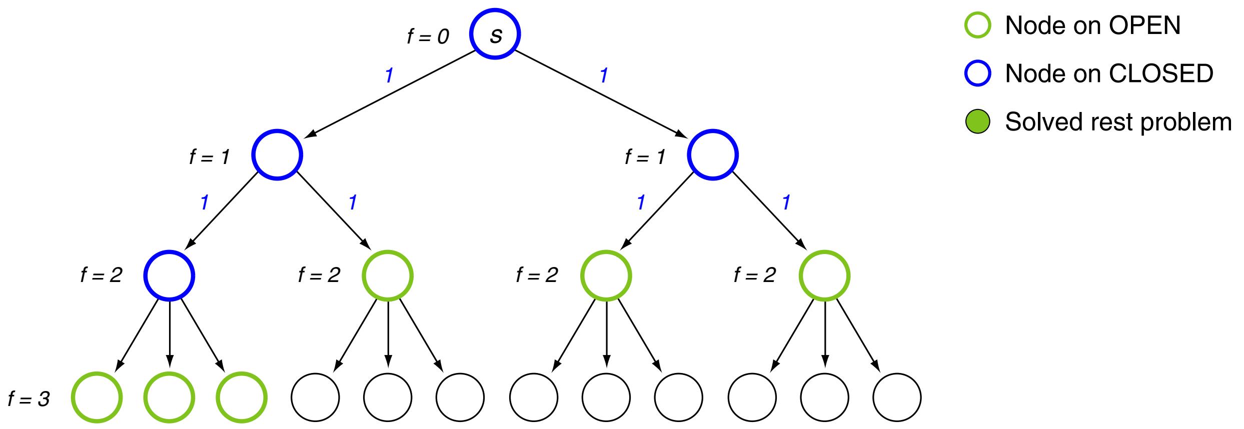
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

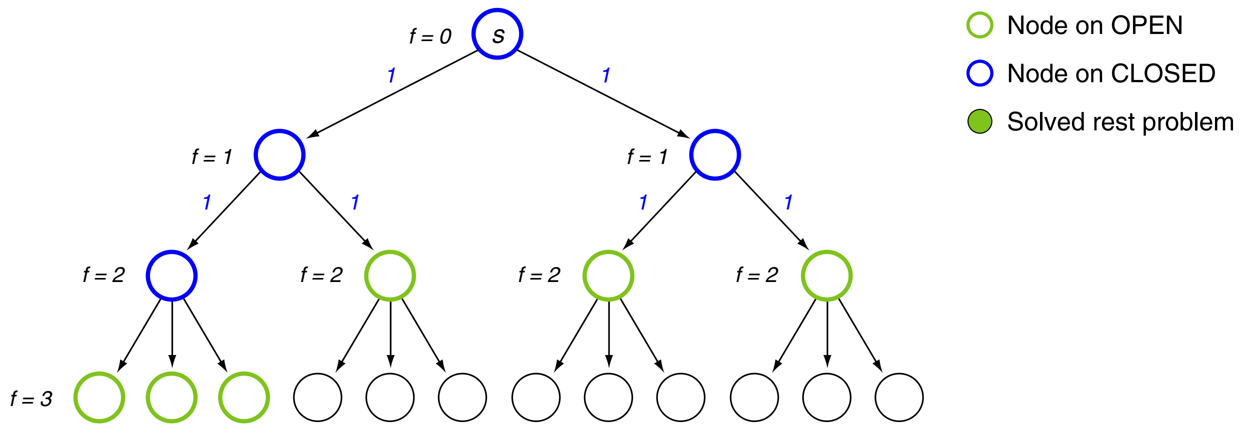
Algorithm A*

Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.



Algorithm A*

Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.



Proof (sketch)

1. $g(n)$ defines the depth of n (consider path from n to s).
2. $f(n) = g(n)$.
3. Breadth-first search $\equiv$ the depth difference of nodes on OPEN is ≤ 1.
4. Assumption: Let n_1, n_2 be on OPEN, having a larger depth difference: $f(n_2) - f(n_1) > 1$.
5. $\Rightarrow$ For the direct predecessor n_0 of n_2 holds: $f(n_0) = f(n_2) - 1 > f(n_1)$.
6. $\Rightarrow$ n_1 must have been expanded before n_0 (consider minimization of f under A*).
7. $\Rightarrow$ n_1 must have been deleted from OPEN. Contradiction to 4.

Algorithm A*

Uniform-cost search is a special case of A*, where $h = 0$.

Proof (sketch)

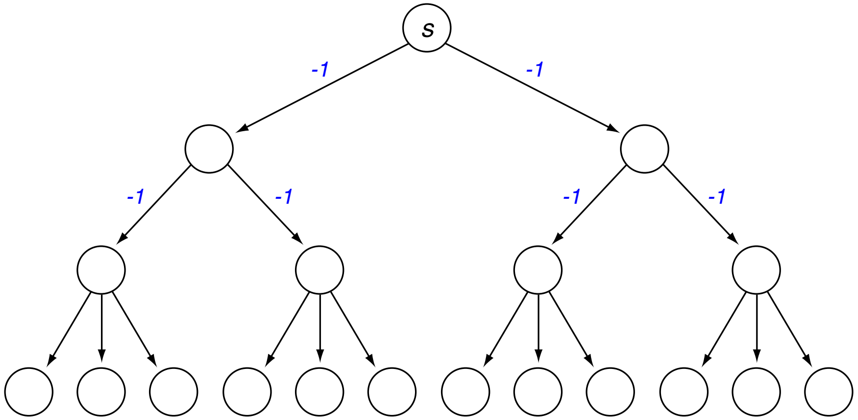
See lab class.

Algorithm A*

Depth-first search is a special case of Z*, where $f(n') = f(n) - 1$, $f(s) = 0$, for all successors n' of n.

Algorithm A*

Depth-first search is a special case of Z*, where $f(n') = f(n) - 1$, $f(s) = 0$, for all successors n' of n.



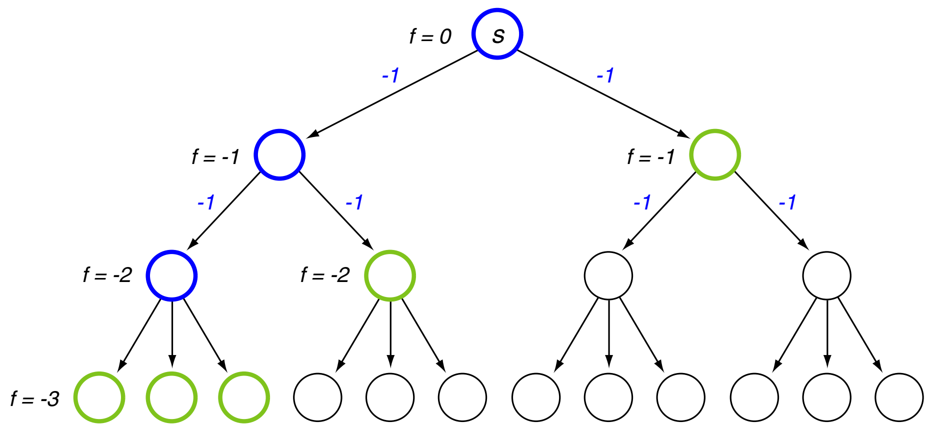
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Algorithm A*

Depth-first search is a special case of Z*, where $f(n') = f(n) - 1$, $f(s) = 0$, for all successors n' of n.



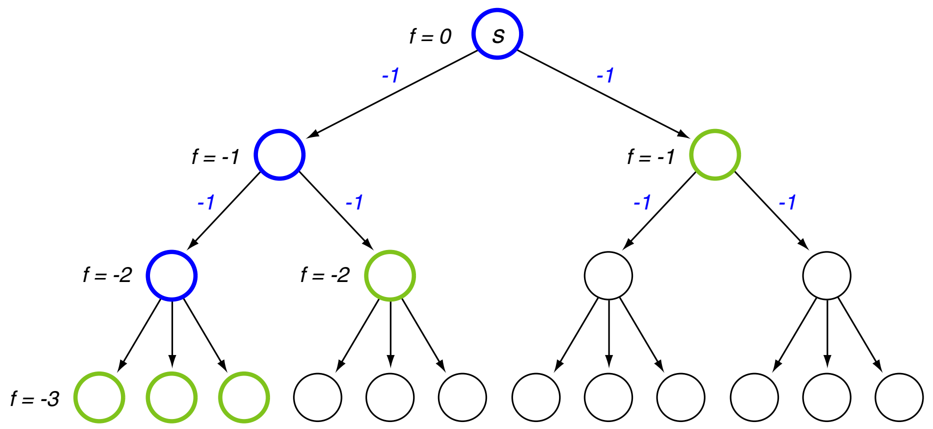
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Algorithm A*

Depth-first search is a special case of Z*, where $f(n') = f(n) - 1$, $f(s) = 0$, for all successors n' of n.



◯ Node on OPEN

◯ Node on CLOSED

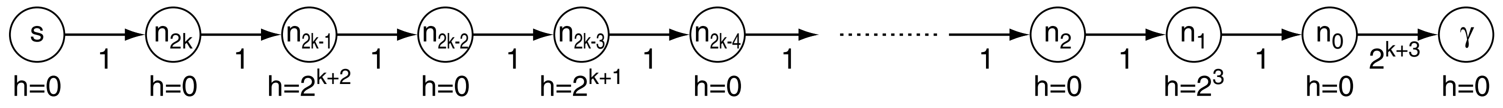
● Solved rest problem

Proof (sketch)

1. $f(n') < f(n) \;\Rightarrow n'$ was inserted on OPEN after n.
   $f(n') \leq f(n) \;\Leftrightarrow n'$ was inserted on OPEN after n.
2. Depth-first search $\equiv$ the most recently inserted node on OPEN is expanded.
3. Let n_2 be the most recently inserted node on OPEN.
4. Assumption: Let n_1 have been expanded before $n_2 \;\wedge\; f(n_1) \neq f(n_2)$.
5. $\Rightarrow f(n_1) < f(n_2)$ (consider minimization of f under Z*).
6. $\Rightarrow n_1$ was inserted on OPEN after n_2.
7. $\Rightarrow n_2$ is not the most recently inserted node on OPEN. Contradiction to 3.

Algorithm A*

Exponential Runtime Example

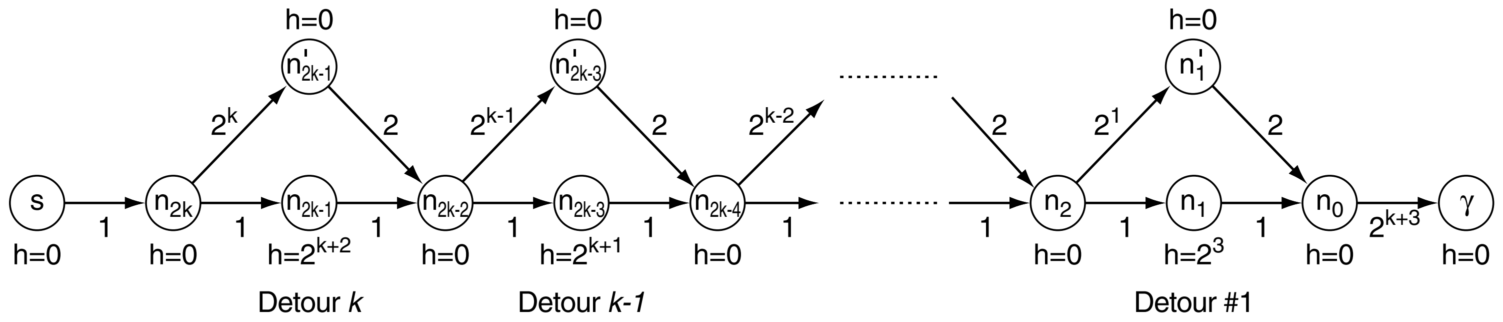


- Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their pointerpaths (starting from s to the predecessor of the leftmost of the two),
  - for any two non-goal nodes on OPEN with different position from left to right the leftmost node has a higher f-value,
  - for two nodes n_{2j+1} and $n'_{2j+1}, 1 \leq j < k$ on OPEN n_{2j+1} has a higher f-value,
  - for γ on OPEN the f-value is maximal wrt. OPEN.

➜ A* requires more than 2^k node expansions.

Algorithm A*

Exponential Runtime Example (continued)

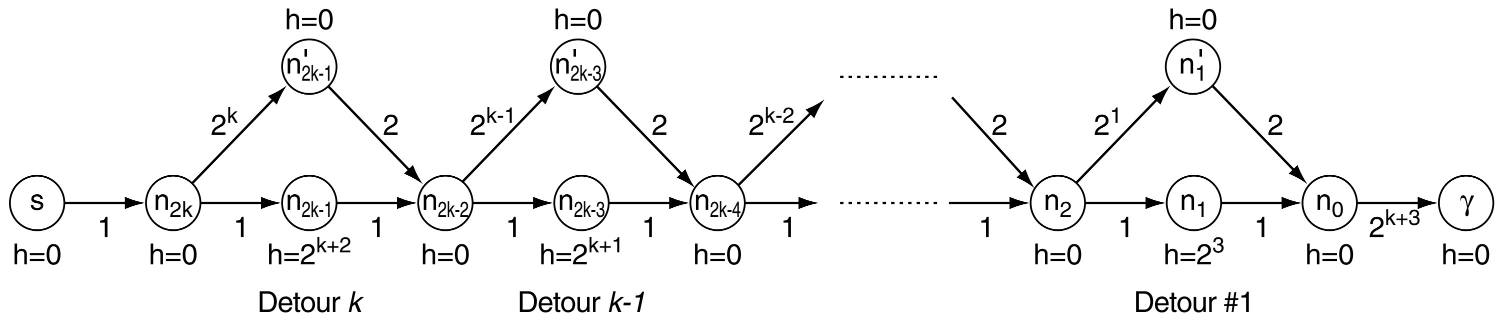


- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their pointerpaths (starting from s to the predecessor of the leftmost of the two),
  - for any two non-goal nodes on OPEN with different position from left to right the leftmost node has a higher f-value,
  - for two nodes n_{2j+1} and $n'_{2j+1}, 1 \leq j < k$ on OPEN n_{2j+1} has a higher f-value,
  - for γ on OPEN the f-value is maximal wrt. OPEN.

➜ A* requires more than 2^k node expansions.

Algorithm A*

Exponential Runtime Example (continued)

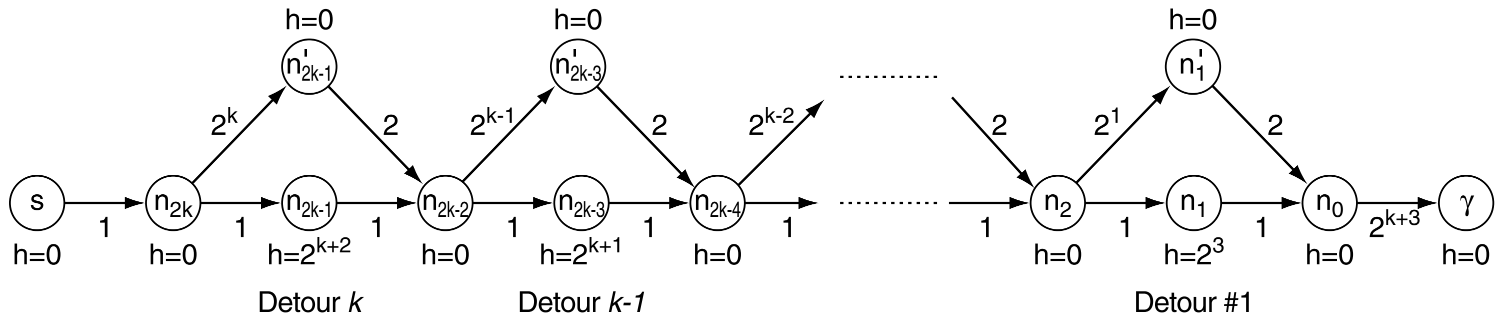


- Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their pointerpaths (starting from s to the predecessor of the leftmost of the two),
  - for any two non-goal nodes on OPEN with different position from left to right the leftmost node has a higher f-value,
  - for two nodes n_{2j+1} and $n'_{2j+1}, 1 \leq j < k$ on OPEN n_{2j+1} has a higher f-value,
  - for γ on OPEN the f-value is maximal wrt. OPEN.

➜ A* requires more than 2^k node expansions.

Algorithm A*

Exponential Runtime Example (continued)



- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their pointerpaths (starting from s to the predecessor of the leftmost of the two),
  - for any two non-goal nodes on OPEN with different position from left to right the leftmost node has a higher f-value,
  - for two nodes n_{2j+1} and $n'_{2j+1}, 1 \leq j < k$ on OPEN n_{2j+1} has a higher f-value,
  - for γ on OPEN the f-value is maximal wrt. OPEN.

➜ A* requires more than 2^k node expansions.

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.

Relation to Dynamic Programming

Optimum Solution Cost and Bellman's Principle of Optimality (continued)

Definition 32 (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.

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

Relation to Dynamic Programming

Optimum Solution Cost and Bellman's Principle of Optimality (continued)

Definition 32 (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.

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

If F is properly chosen ("If F obeys Bellman's principle of optimality"), the equations stated in Definition 32 form necessary and complete conditions for the optimality of the solution cost C^*.

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

"$\Leftarrow$" 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 32) 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 32.
  2. Then, operationalizing Definition 32 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 32, 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 $\widehat{C}(n)$.

Definition 33 (Bellman Equations for $\widehat{C}$ [Overview, monotone F])

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

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

Relation to Dynamic Programming

Optimum Solution Cost and Bellman's Principle of Optimality (continued)

Corollary 34 (Bellman Equations for $\widehat{C}$)

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 35 (Bellman Equations for $\widehat{C}$ [Overview, monotone F])

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.

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

Relation to Dynamic Programming

Partially Explored State Spaces

Definition 35 (Bellman Equations for $\widehat{C}$ [Overview, monotone F])

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.

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

Special case A*:

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

$$\widehat{C}(n) = \begin{cases} h(n) & n \text{ is on OPEN} \\ \min_i\{c(n,n_i) + \widehat{C}(n_i)]\} & n \text{ is on CLOSED, } n_i \text{ direct successors of } n \end{cases}$$

Do these two estimated optimum solution cost functions coincide?

Relation to Dynamic Programming

Partially Explored State Spaces (continued)

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. $\sim$ 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:

$$\widehat{C}_{P_{s-n'}}(s) \;\leq\; \widehat{C}_{P'_{s-n'}}(s) \quad\Rightarrow\quad \widehat{C}_{P_{s-n}}(s) \;\leq\; \widehat{C}_{P'_{s-n}}(s)$$

where P_{s-n} uses subpath $P_{s-n'}$ and P'_{s-n} uses $P'_{s-n'}$.

Compare to the example with a cost function that is not right-monotone.

Relation to Dynamic Programming

Partially Explored State Spaces (continued)

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 γ, and for all paths $P_{s-n'}$ from s to n' holds:

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