Chapter S:III

III. Informed Search

- ❑ Best-First Search
- ❑ Best-First Search for State-Space Graphs
- ❑ Cost Functions for State-Space Graphs
- ❑ Evaluation of State-Space Graphs
- ❑ Algorithm A*

- ❑ BF* Variants
- ❑ Hybrid Strategies

Cost Functions for State-Space Graphs

Overview

BF defines a schema for the design of search strategies for state-space graphs. Up to this point, the evaluation functions f remained unspecified.

Questions:

- How to compute f?
- How to evaluate a solution path?
- 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.

Cost Functions for State-Space Graphs

Overview (continued)

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

1. Recursive cost functions (for paths)
2. Solution cost (for a given solution path)
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 for State-Space Graphs

Overview (continued)

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

1. Recursive cost functions (for paths)
2. Solution cost (for a given solution path)
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_P	C^*
	partial	$\widehat{C}_P$	$\widehat{C}$

Cost Functions for State-Space Graphs

Overview (continued)

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

1. Recursive cost functions (for paths)
2. Solution cost (for a given solution path)
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_P	C^*
	partial	$\widehat{C}_P$	$\widehat{C}$

Cost Functions for State-Space Graphs

Overview (continued)

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

1. Recursive cost functions (for paths)
2. Solution cost (for a given solution path)
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_P	C^*
	partial	$\widehat{C}_P$	$\widehat{C}$

Cost Functions for State-Space Graphs

Overview (continued)

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

1. Recursive cost functions (for paths)
2. Solution cost (for a given solution path)
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	$C_P(n)$	$C^*(n)$
	partial	$\widehat{C}_P(n)$	$\widehat{C}(n)$

Cost Functions for State-Space Graphs

Overview (continued)

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

1. Recursive cost functions (for paths)
2. Solution cost (for a given solution path)
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	$C_P(s)$	$C^*(s)$
	partial	$\widehat{C}_P(s)$	$\widehat{C}(s) \rightsquigarrow n$

n represents a most promising solution base.

Cost Functions for State-Space Graphs

If solution paths are known, the solution cost for a solution path can be determined.

Definition 16 (Cost Function C_P)

For an OR-graph G and let M be an ordered set. A function C_P, which assigns each solution path P in G and each node n in P a cost value $C_P(n)$ in M is denoted by C_P.

Usage and notation of C_P :

- 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 :

  $$f(\gamma) = C_P(s) \text{ with } P \text{ backpointer path of } \gamma.$$

Remarks:

- As ordered set M usually $\mathbf{R} \cup \{\infty\}$ is chosen.
- $C_P(n)$ should be seen as binary function with arguments P and n.
- The cost value $C_P(n)$ is meaningful only if n is a node in P.
- Solution cost does not measure efforts for finding a solution. Solution cost aggregate properties of operations in a solution path 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_P for merits, and W_P for weights.
  E.g., hill-climbing algorithms often employ merit functions.
- A cost function can be a complex accounting rule, considering the properties of 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 $C_P(n)$ to denote the solution cost of a problem associated with node n. Definition 16 catches up for the missing notation and semantics.

Cost Functions for State-Space Graphs

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 17 (Optimum Solution Cost C^*, Optimum Solution)

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

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

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

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

Cost Functions for State-Space Graphs

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 17 (Optimum Solution Cost C^*, Optimum Solution)

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

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

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

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

Remarks:

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

Cost Functions for State-Space Graphs

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 18 (Optimum Solution Cost C_P^* for a Solution Base)

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

The optimum solution cost for a node n in a solution base P, $C_P^*(n)$, is defined as

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

Remarks:

- In the setting of Definition 18 we usually assume:

  $$C^*(s) = \text{min}\{C^*_P(s) \mid P \text{ is solution base currently maintained by an algorithm}\}$$

  The solution bases maintained by algorithm BF are the backpointer paths of the nodes in OPEN, i.e., the maximal solution bases for s in the traversal tree which is a subgraph of the search space graph G.

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

Cost Functions for State-Space Graphs

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 19 (Estimated Optimum Solution Cost $\widehat{C}_P$ for a Solution Base)

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

The estimated optimum solution cost, $\widehat{C}_P(n)$, for a node n in in a solution base P in G 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 $\widehat{C}_P$:

- In BF we use $f(n) = \widehat{C}_P(s)$ with P backpointer path of n.
- In order to emphasize the dualism of estimated and real values, we often write $f^*(n)$ instead of $C^*_P(n)$, i.e., $f(n)$ is an estimate of $f^*(n)$, the optimum solution path cost for s when extending P.

Cost Functions for State-Space Graphs

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 19 (Estimated Optimum Solution Cost $\widehat{C}_P$ for a Solution Base)

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

The estimated optimum solution cost, $\widehat{C}_P(n)$, for a node n in in a solution base P in G 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 $\widehat{C}_P$:

- In BF we use $f(n) = \widehat{C}_P(s)$ with P backpointer path of n.
- In order to emphasize the dualism of estimated and real values, we often write $f^*(n)$ instead of $C_P^*(n)$, i.e., $f(n)$ is an estimate of $f^*(n)$, the optimum solution path cost for s when extending P.

Cost Functions for State-Space Graphs

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 19 (Estimated Optimum Solution Cost $\widehat{C}_P$ for a Solution Base)

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

The estimated optimum solution cost, $\widehat{C}_P(n)$, for a node n in in a solution base P in G 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 $\widehat{C}_P$:

- In BF we use $f(n) = \widehat{C}_P(s)$ with P backpointer path of n.
- In order to emphasize the dualism of estimated and real values, we often write $f^*(n)$ instead of $C^*_P(n)$, i.e., $f(n)$ is an estimate of $f^*(n)$, the optimum solution path cost for s when extending P.

Cost Functions for State-Space Graphs

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

Definition 20 (Estimated Optimum Solution Cost $\widehat{C}$ [Overview])

Let G be an OR-graph with root node s and let $C_P(n)$ denote a cost function for G. Further, let $\mathcal{T}$ be a finite set of solution bases for s defined by a (traversal) tree rooted in s.

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

$$\widehat{C}(n) = \min\{\widehat{C}_P(n) \mid P \text{ is solution base in } \mathcal{T} \text{ containing } n\}$$

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

Cost Functions for State-Space Graphs

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

Definition 20 (Estimated Optimum Solution Cost $\widehat{C}$ [Overview])

Let G be an OR-graph with root node s and let $C_P(n)$ denote a cost function for G. Further, let $\mathcal{T}$ be a finite set of solution bases for s defined by a (traversal) tree rooted in s.

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

$$\widehat{C}(n) = \min\{\widehat{C}_P(n) \mid P \text{ is solution base in } \mathcal{T} \text{ containing } n\}$$

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

For an algorithm searching an OR-graph G, the traversal tree and thus the finite set $\mathcal{T}$ of solution bases is defined by the nodes in OPEN.

Cost Functions for State-Space Graphs

Cost Concept in Uniform-Cost Search

- Edge weight.

  Encode either cost values or merit values, which are accounted if the respective edges become part of the solution.

  $c(n, n')$ denotes the cost value of an edge from n to n'.

- Path cost.

  The cost of a path, C_P, results from applying a *cost measure* F, which specifies how cost of a continuing edge is combined with the cost of the rest of the path.

  Examples:

  Sum cost $:=$ the sum of all edge costs of a path P from s to n:

  $$C_P(s) = \sum_{i=0}^{k-1} c(n_i, n_{i+1}), \text{ with } n_0 = s \text{ and } n_k = n$$

  Maximum cost $:=$ the maximum of all edge costs of a path:

  $$C_P(s) = \max_{i \in \{0,\ldots,k-1\}} c(n_i, n_{i+1}), \text{ with } n_0 = s \text{ and } n_k = n$$

- Estimated optimum solution cost.

  The cost value for the solution base is taken as the estimate of optimum solution cost.

  $$\widehat{C}_P(n) := C_P(n)$$

Cost Functions for State-Space Graphs

Recursive Cost Functions

The computation of the evaluation functions f would be nearly impracticable if the cost of paths were based on complex global properties of the path.

Definition 21 (Recursive Cost Function, Cost Measure)

A cost function C_P for a solution path P is called recursive, if for each node n in P it holds:

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

- n' denotes the direct successor of n in P,
- $E(n) \in \mathbf{E}$ denotes a set of *local* properties of n with respect to P,
- F is a function that prescribes how local properties of n are accounted (better: combined) with properties of the direct successor of n:

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

  F is called cost measure.

Remarks:

- Observe that for each node n in a solution path P the subpath of P starting in n is a solution path for n (neglecting additional solution constraints).

- If for a solution base its merits, quality, or other positive aspects are measured, F is called merit measure.

Cost Functions for State-Space Graphs

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 22 (Heuristic Function h)

Let G be an 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 path for n, is called heuristic function (for G).

In order to emphasize the dualism of estimated and real values, we often write $h^*(n)$ instead of $C^*(n)$, i.e., $h(n)$ is an estimate of $h^*(n)$.

Remarks:

- 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.

Cost Functions for State-Space Graphs

Recursive Cost Functions (continued)

Corollary 23 (Estimated Solution Cost $\widehat{C}_P$ for a Solution Base)

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

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 base P in G is computed as follows:

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

Evaluation of State-Space Graphs

Recursive Cost Functions and Efficiency

If the search space graph is an 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, c' with $c \leq c'$ we have

$$F[e, c] \leq F[e, c']$$

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

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 $\widehat{C}_P(s)$ 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:

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

Compare to Bellman's equations.

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 OR graph rooted at s. Let h be a heuristic function and let $C_P(n)$ be a recursive cost function for G based on E and a monotone cost measure F. Further, let $\mathcal{T}$ be a finite set of solution bases for s defined by a (traversal) tree rooted in s.

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

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

Most promising solution base P for s in $\mathcal{T}$:
If P contains an inner node n, then the direct successor n' of n in P is a node n' in $\mathcal{T}$ with $F[E(n), \widehat{C}(n')] = \min_i\{F[E(n), \widehat{C}(n_i)]\}$.

Cost Functions for State-Space Graphs

Recursive Cost Functions and Efficiency (continued)

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

Let G be an OR graph rooted at s. Let h be a heuristic function and let $C_P(n)$ be a recursive cost function for G based on E and a monotone cost measure F. Further, let $\mathcal{T}$ be a finite set of solution bases for s defined by a (traversal) tree rooted in s.

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

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

Most promising solution base P for s in $\mathcal{T}$:
If P contains an inner node n, then the direct successor n' of n in P is a node n' in $\mathcal{T}$ with $F[E(n), \widehat{C}(n')] = \text{min}_i\{F[E(n), \widehat{C}(n_i)]\}$.

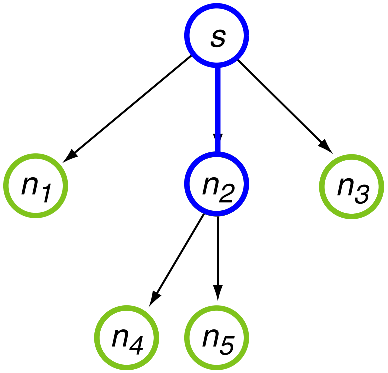
Remarks:

- $\widehat{C}(n)$ computes for a node n the minimum of the estimated costs among all solution bases in $\mathcal{T}$ containing n (paths from n to a leaf in the traversal tree defining $\mathcal{T}$). In particular, $\widehat{C}(s)$ computes the estimated optimum solution cost for the entire problem, and it hence defines a most promising solution base in $\mathcal{T}$.

- $\widehat{C}_P$, 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 $\widehat{C}(n)$, $\widehat{C}_P(n)$



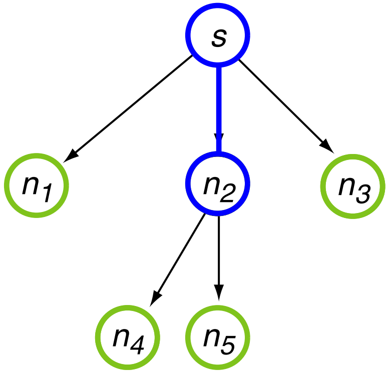
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Evaluation of State-Space Graphs

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

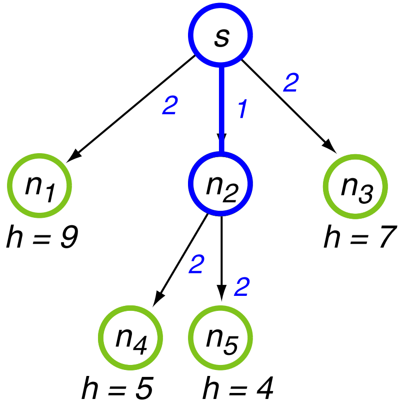


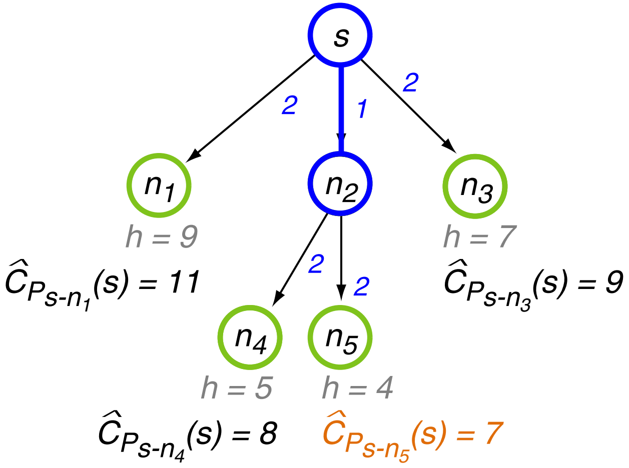
○ Node on OPEN

○ Node on CLOSED

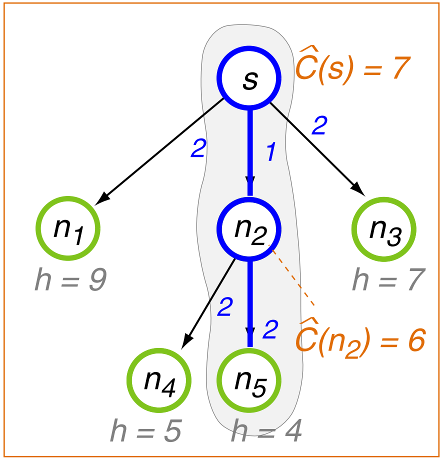
● Solved rest problem

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





s
2
1
2
n_1
n_2
n_3
$h = 9$
$h = 7$
$\widehat{C}_{P_{s\text{-}n_1}}(s) = 11$
$\widehat{C}_{P_{s\text{-}n_3}}(s) = 9$
2
2
n_4
n_5
$h = 5$
$h = 4$
$\widehat{C}_{P_{s\text{-}n_4}}(s) = 8$
$\widehat{C}_{P_{s\text{-}n_5}}(s) = 7$



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 G be an OR graph, n a node in G, n' a direct successor of n, and F a cost measure. F is called additive cost measure iff (↔) 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, ⋆, f)
     ...
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
 4.    n = min(OPEN, f);     // Find most promising solution base.
       IF ⋆(n) THEN RETURN(n);    // Delayed termination.
```

Define $f(n)$ as $\widehat{C}_P(s)$ with P backpointer path of n:

➜ $f(n)$ is a recursive evaluation function.

➜ Algorithm BF becomes Algorithm Z.

Delayed termination:

➜ Algorithm BF becomes Algorithm BF*.

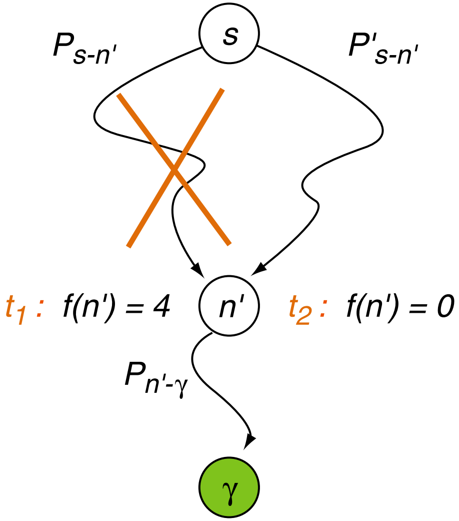
➜ Algorithm Z becomes Algorithm Z*.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation [S:III 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
          $n'_{old}$ = retrieve$(n'$, OPEN $\cup$ CLOSED$)$;
          IF $(f(n') < f(n'_{old}))$
          THEN
            update_backpointer$(n'_{old}, n)$;
            IF $n'_{old} \in$ CLOSED THEN ...ENDIF
          ENDIF
```

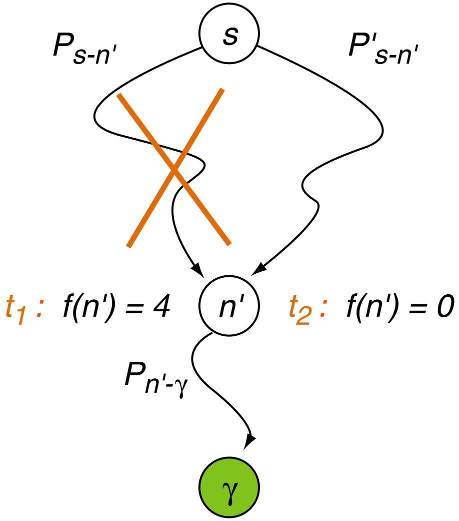


Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation [S:III 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
          $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
          IF ($f(n') < f(n'_{old})$)
          THEN
            update_backpointer($n'_{old}$, $n$);
            IF $n'_{old} \in$ 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-preserving if 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-preserving if 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

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

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

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

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:

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

Remarks:

- $g(n)$ denotes the sum of the edge cost values along the backpointer 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:

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

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

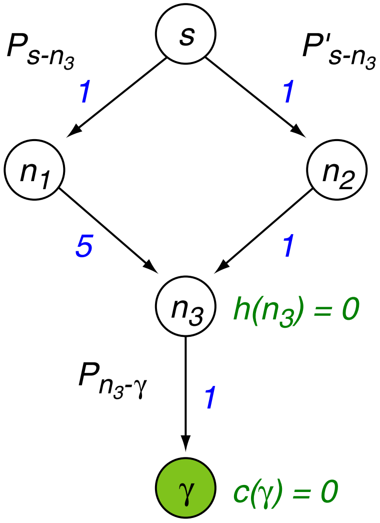
Example for a cost function that is recursive but *not* order-preserving:

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

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

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

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



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

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

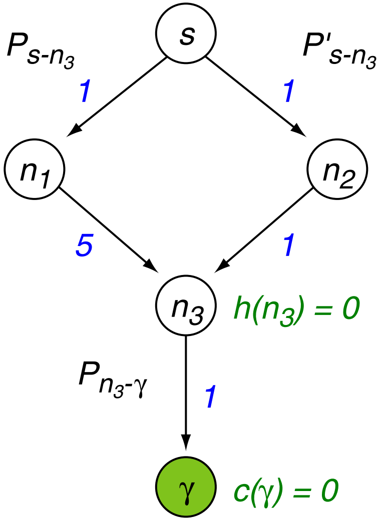
$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] & n \text{ is inner node in } P \text{ and} \\ \qquad = |c(n,n') + \widehat{C}_P(n') - 5| & n' \text{ is direct successor of } n \text{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

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

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

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

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



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

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

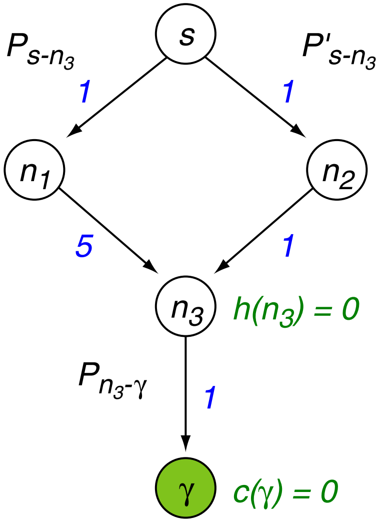
$$\widehat{C}_P(n) = \begin{cases} c(n) & n \text{ is goal node and leaf in } P \\ h(n) & n \text{ is leaf in } P \text{ but no goal node} \\ F[E(n), \widehat{C}_P(n')] & n \text{ is inner node in } P \text{ and} \\ \qquad = |c(n,n') + \widehat{C}_P(n') - 5| & n' \text{ is direct successor of } n \text{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

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

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

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

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



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

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

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

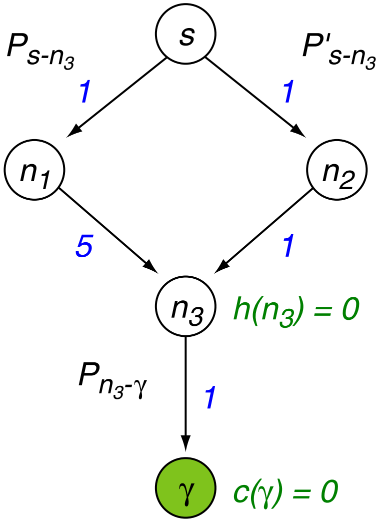
$$P_{s-n_3} = (s, n_1, n_3)$$

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

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

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

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



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

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

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

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

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

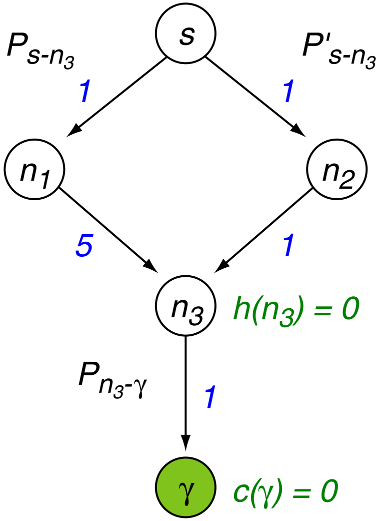
$$P_{n_3-\gamma} = (n_3, \gamma)$$

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

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

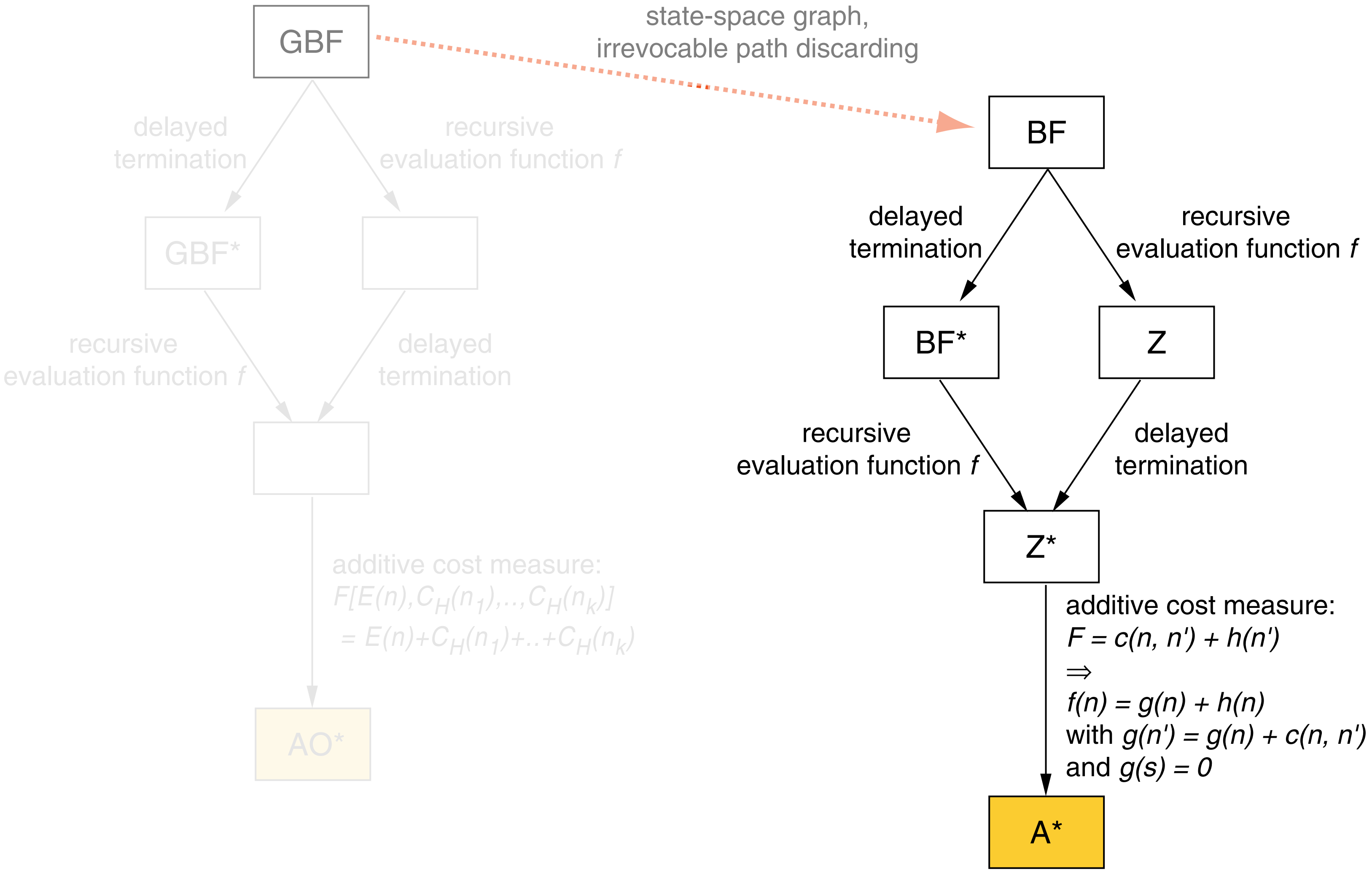
$$\widehat{C}_{P_{s-\gamma}}(s) = |1+|5+|1+0-5|-5|-5| = 0$$

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



Evaluation of State-Space Graphs

Taxonomy of Best-First Algorithms



GBF

delayed termination

recursive evaluation function *f*

GBF*

recursive evaluation function *f*

delayed termination

additive cost measure:

$F[E(n), C_H(n_1), .., C_H(n_k)]$

$= E(n) + C_H(n_1) + .. + C_H(n_k)$

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:

$F = c(n, n') + h(n')$

$\Rightarrow$

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

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, $\star$, $h$)

  1.  insert($s$, OPEN);
      $g(s) = 0$;
  2.  LOOP
  3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
  4.    $n$ = min(OPEN, $g + h$);    // Most promising solution base minimizes $f(n)$.
        IF $\star(n)$ THEN RETURN($n$);    // Delayed termination.
        remove($n$, OPEN);  push($n$, CLOSED);
  5.    FOREACH $n'$ IN successors($n$) DO    // Expand $n$.
          add_backpointer($n'$, $n$);
          $g(n') = g(n) + c(n, n')$;
          IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
          THEN   // $n'$ encodes a new state.
            insert($n'$, OPEN);
          ELSE   // $n'$ encodes an already visited state.
            $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
            IF ($g(n') < g(n'_{old})$)
            THEN   // The state of $n'$ is reached via a cheaper path.
              update_backpointer($n'_{old}$, $n$);  $g(n'_{old}) = g(n')$;
              IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED);  insert($n'_{old}$, 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.

- Although only the order-preserving property of $f = g + h$ was mentioned, we still assume that the following equivalence holds:

  "Solution base $P_{s-n'}$ can be completed by $P_{n'-\gamma}$ to a solution path."

  $\Leftrightarrow$

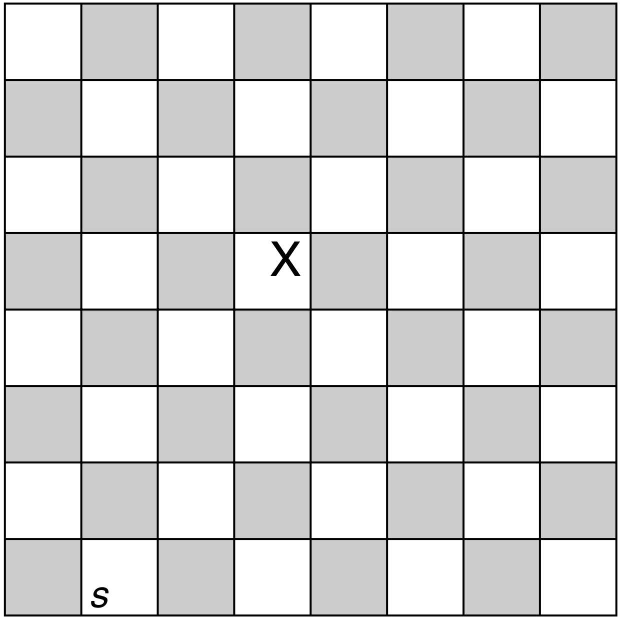
  "Solution base $P'_{s-n'}$ can be completed by $P_{n'-\gamma}$ to a solution path."

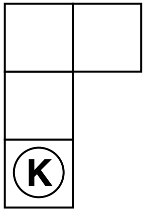
  This equivalence is trivially satisfied, if $\star(\gamma)$ checks only local properties of γ but not properties of the backpointer path of γ.

Algorithm A*

Example: Knight Moves

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





Knight move

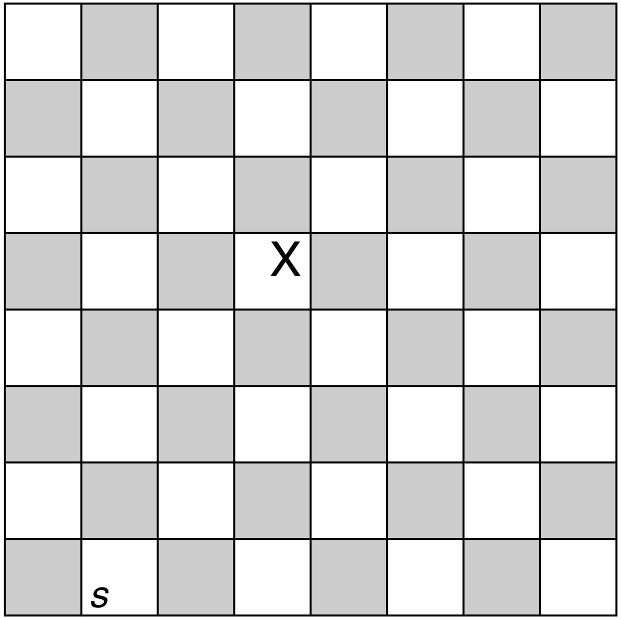
Let n' be a direct successor of n.

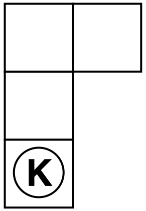
- $f(n') = g(n') + h(n')$
- $g(n') = g(n) + c(n, n')$
- $g(s) = 0$
- $c(n, n') = 1$

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.

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

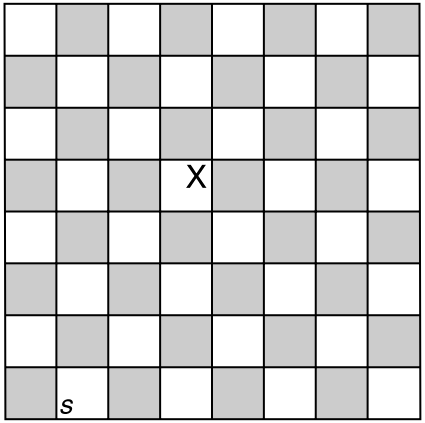
$$h_1 = \left\lceil \frac{\#rows}{2} \right\rceil$$

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

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

Algorithm A*

Example: Knight Moves (continued)



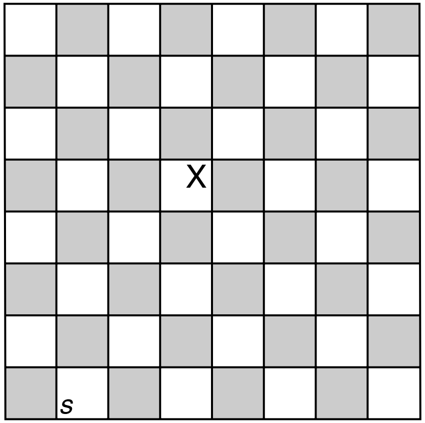
OPEN	CLOSED
$\{s\}$	$\{\}$

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

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

Algorithm A*

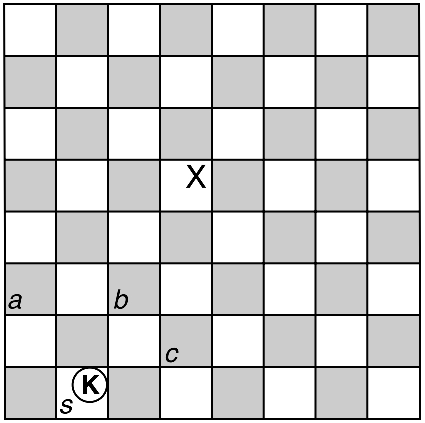
Example: Knight Moves (continued)



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

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

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

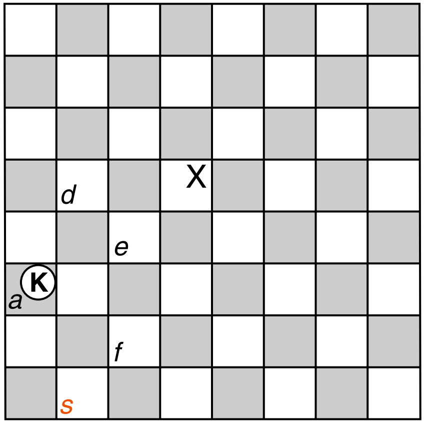


OPEN	CLOSED
$\{a, b, c\}$	$\{s\}$

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)

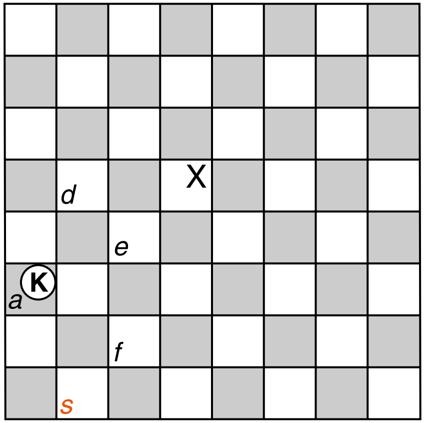


OPEN	CLOSED
$\{d, b, c, e, f\}$	$\{a, s\}$

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

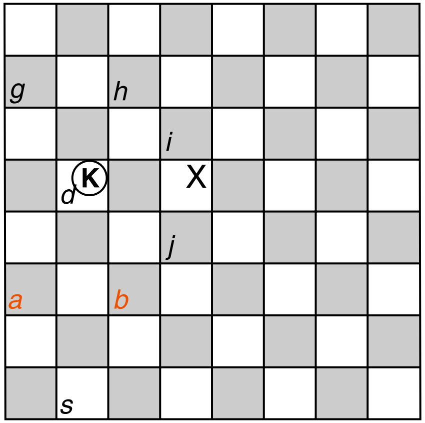
Algorithm A*

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

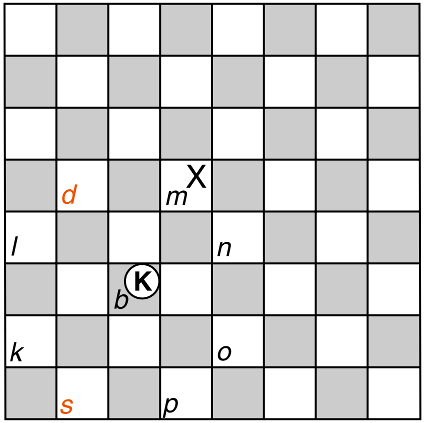


OPEN	CLOSED
$\{b, c, e, f, g, h, i, j\}$	$\{d, a, s\}$

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)

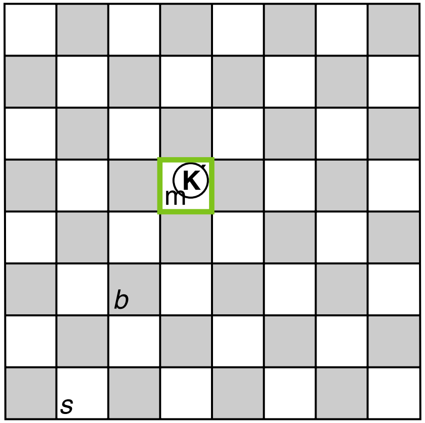


OPEN	CLOSED
$\{m, c, e, l, n,$ $f, g, h, i, j, k, o, p\}$	$\{b, d, a, s\}$

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)



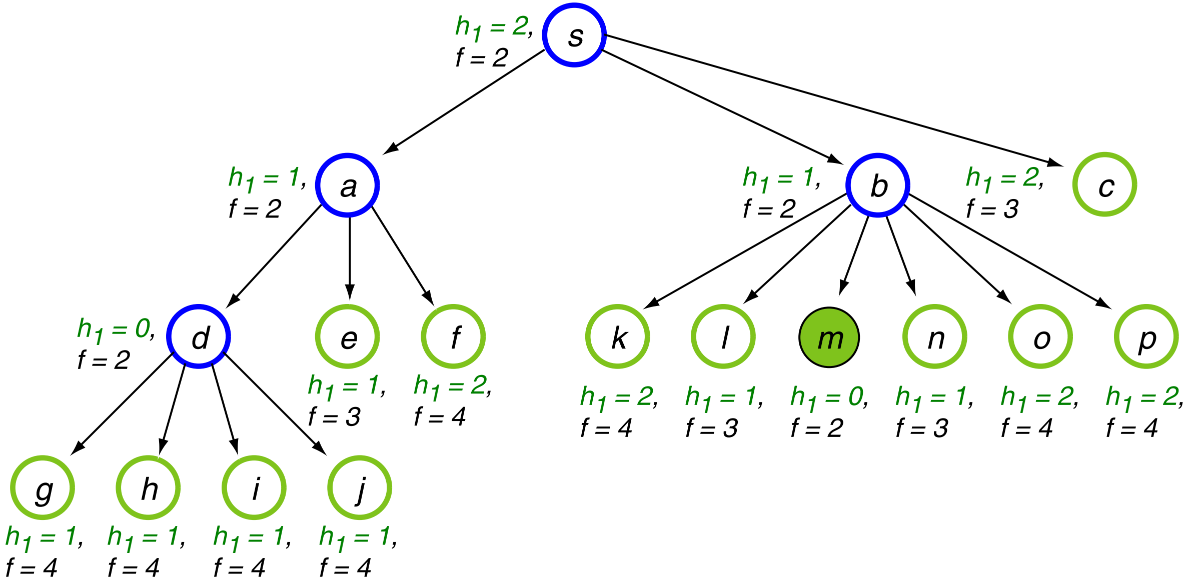
OPEN	CLOSED
$\{c, e, l, n, f, g, h, i, j, k, o, p\}$	$\{m, b, d, a, s\}$

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:



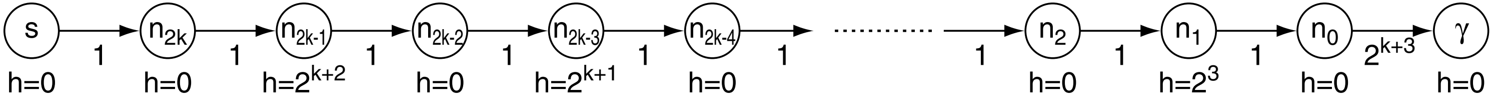
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Algorithm A*

Exponential Runtime Example

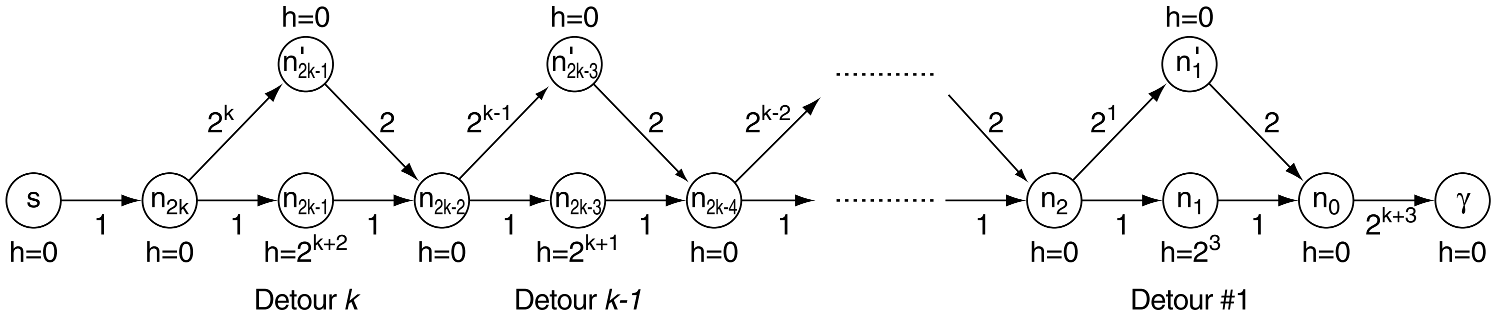


- Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their backpointer paths (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)

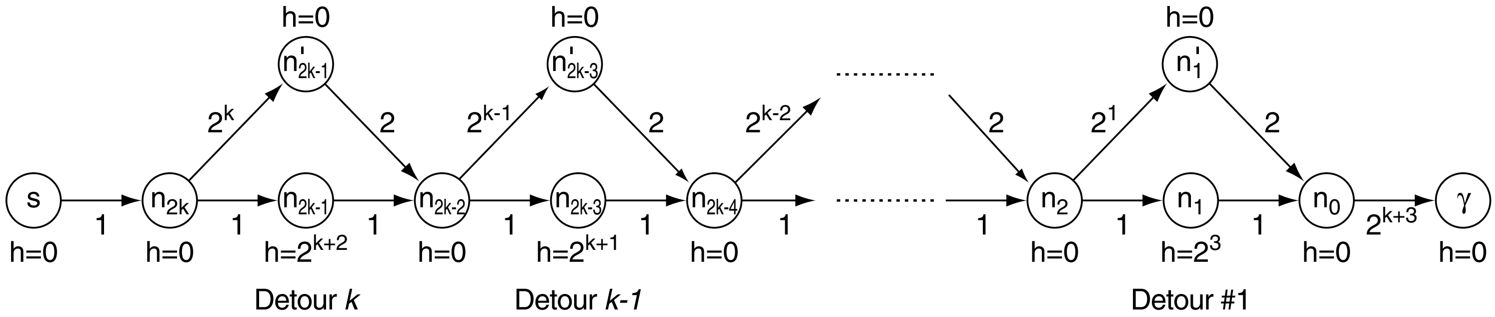


- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their backpointer paths (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)

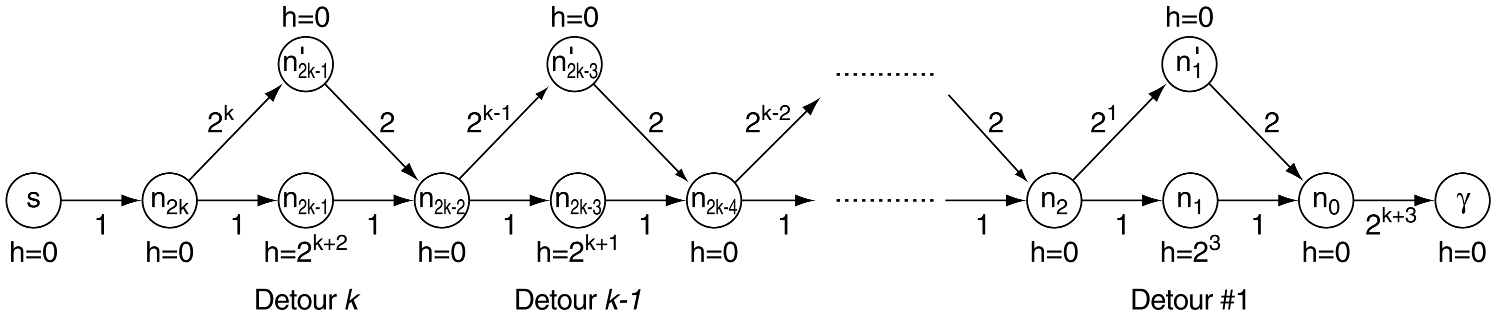


- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial part of their backpointer paths (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 backpointer paths (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.