Chapter S:III

III. Informed Search

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

- BF* Variants
- Hybrid Strategies

- Best-First Search for AND-OR Graphs
- Relation between GBF and BF
- Cost Functions for AND-OR Graphs
- Evaluation of AND-OR Graphs

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 part of a 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 part of a 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 part of a 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 part of a 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 nodes in paths)
2. Solution cost (for nodes in a given solution path)
3. Optimum solution cost (for nodes in a complete search space graph)
4. Estimated solution cost (for nodes in a given solution base)
5. Estimated optimum solution cost (for nodes in a part of a 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 nodes in paths)
2. Solution cost (for nodes in a given solution path)
3. Optimum solution cost (for nodes in a complete search space graph)
4. Estimated solution cost (for nodes in a given solution base)
5. Estimated optimum solution cost (for nodes in a part of a 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

For a known solution path, the solution cost must be determined.

Definition 26 (Cost Function C_P)

Let G be an OR-graph and let M be an ordered set.

A *cost function* C_P is a (partial) function that assigns a cost value $C_P(n)$ in M to nodes n and paths P in G. $C_P(n)$ is cost of P starting in n to the end of P.

Usage and notation of C_P :

- As ordered set M usually $\mathbf{R} \cup \{-\infty, +\infty\}$ is chosen.
- C_P is at least defined for solution paths P and nodes in path 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 :

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

- No provisions are made how to compute $C_P(n)$ for a path P that is no solution path or a node n that is not in P. $C_P(n) := \infty$ is reasonable in this case.

Remarks:

- $C_P(n)$ should be seen as a two-argument 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 aggregates 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.
  For instance, greedy algorithms often employ merit functions.
- A cost function can be a complex accounting rule, considering 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 26 catches up for the missing notation and semantics.
- Please note that in the above definition of $C_P(n)$ the path P is not required to be a solution path for n. At least for solution paths P and their nodes n, $C_P(n)$ should be defined. If $C_P(n)$ is undefined, we assume $C_P(n) := \infty$.

Cost Functions for State-Space Graphs

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

Definition 27 (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* $C^*(n)$ for a node n in G is defined as

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

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

Usage of $C^*(n)$:

- If G contains no solution path for n, let $C^*(n) := \infty$.
- $\inf$ denotes the operator that gives the greatest lower bound of a set.
  For finite sets, the infimum is the minimum.
- A task should be modeled in such a way that the $\inf$ can be replaced by min.

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 28 (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* $C^*_P(n)$ *for a node* n *in a* solution base P for s is defined as

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

Usage of $C^*_P(n)$:

- An algorithm $\mathcal{A}$ maintaining a set of solution bases can find a solution path with cost

  $$C^*_{\mathcal{A}}(s) = \min\{C^*_P(s) \mid P \text{ is solution base currently maintained in OPEN by } \mathcal{A}\}$$

  Therefore, it is essential for search algorithms to keep available enough solution bases such that we have

  $$C^*(s) = C^*_{\mathcal{A}}(s)$$

  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 29 (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 is an estimate of $C_P^*(n)$.

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

For BF we define $f(n) := \widehat{C}_P(s)$ with P being the back-pointer path of n.

Usage of $\widehat{C}_P$:

- $f(n)$ is optimistic, if $f(n) \leq C_P^*(s)$ with P being the back-pointer path of n.
- $f(n)$ is an estimate of the optimum solution path cost for s when extending the solution base identified by n, i.e., when extending the back-pointer path of n.

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. [Overview]

Definition 30 (Estimated Optimum Solution Cost $\widehat{C}$)

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.

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

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

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

Usage of $\widehat{C}_P$:

- In our algorithms, the finite set $\mathcal{T}$ is defined by the traversal tree available at a point in time. The traversal tree of solution bases is defined by the nodes in OPEN.
- For solving optimization problems, we want to have $\widehat{C}(s) = C^*(s)$, i.e. an optimum solution path is in the reach of the algorithm.

Cost Functions for State-Space Graphs

Cost Concept used 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 31 (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:

- C_P is a recursively defined function. Hence, $C_{P_{s-\gamma}}(s)$ can be computed bottom-up, from the end node γ to the start node s along path $P_{s-\gamma}$.
- Function E and F should be given in such away that computation of $C_P(n)$ is possible for solution paths P and nodes n in such P.
- The brackets $[\ldots]$ indicate a list notation. For AND/OR graphs, there is usually not only one successor node n' that is taken into account.
- Observe that for each node n in a solution path P the subpath of P starting in n is a solution path for n.
- Local properties have to be seen with respect to the successor node in the path P. An example is cost of the operation that was applied at n to get to n'.
- If merits, quality, or other positive aspects are measured for a solution base, 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 32 (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 $C^*(n)$ (i.e., the optimum cost of a solution path for n), is called *heuristic function* (for G).

Usage of $h(n)$:

- In order to emphasize the dualism of estimated and real values again, 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:

$$h(n) := \infty \quad \Leftrightarrow \quad \perp(n) = \text{true}$$

Cost Functions for State-Space Graphs

Recursive Cost Functions (continued)

Definition 33 (Recursive Estimated Solution Cost Function $\widehat{C}_P$ 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, recursively defined using functions F and E. Further, let h be a heuristic function.

The *recursively defined* *estimated solution cost $\widehat{C}_P(n)$ for solution base P* in G based on $C_P(n)$ is defined as

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

Usage of $h(n)$:

- The computation of an estimated solution cost is based on the the face-value principle: Estimated cost values $C_P(n')$ are used as if they were the real values.
- As a shorthand we use $c(n) := F[E(n)]$ for the remaining cost at a (nontrivial) goal node. Often we have $c(n) = 0$.
- $\widehat{C}_P$ is a recursive 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

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 the optimum solution cost $C^*_P(s)$ for a solution base P.

Evaluation of State-Space Graphs

Recursive Cost Functions and Efficiency (continued)

Corollary 34 (Optimum Solution Cost C^* using Recursive Cost Functions)

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

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

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

Usage of the recursive computation of $C^*(n)$:

- Optimum solution cost can be computed this way, only if there are no further solution constraints that have to be considered.

Evaluation of State-Space Graphs

Recursive Cost Functions and Efficiency (continued)

Corollary 35 (Estimated Optimum Solution Cost $\widehat{C}$ using Recursive Cost Estimations)

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

The estimated optimum solution cost $\widehat{C}(n)$ for a node n occurring in $\mathcal{T}$ can be computed by

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

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

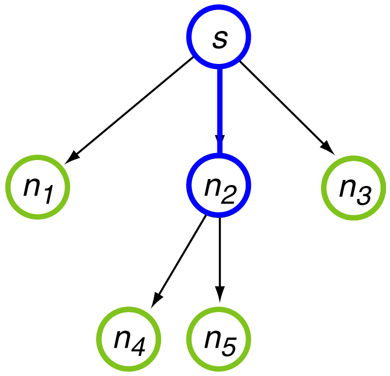
Remarks:

- In the computation of $\widehat{C}(n)$ we again make use the face value principle.
- In our algorithms, $\mathcal{T}$ is defined by the traversal tree rooted in s and the OPEN nodes available at some point in time.
- $\widehat{C}(n)$ 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 $\mathcal{T}$.

  Observe that in algorithms Basic_BF and Basic_BF* we might have multiple occurrences of nodes referring to the same state in a traversal tree. Then, the minimum is computed using all solution bases sharing the same occurrence of a state.

Evaluation of State-Space Graphs

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



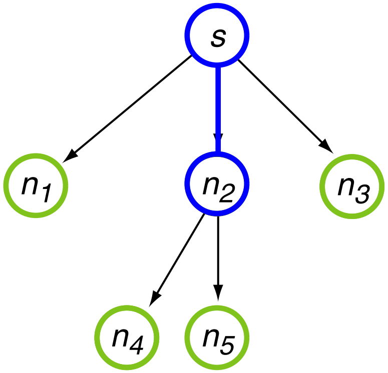
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Evaluation of State-Space Graphs

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

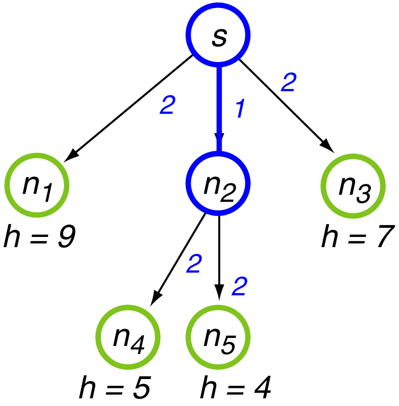


○ Node on OPEN

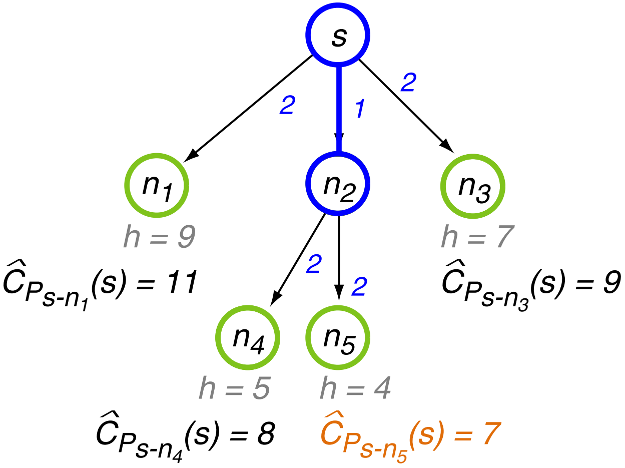
○ Node on CLOSED

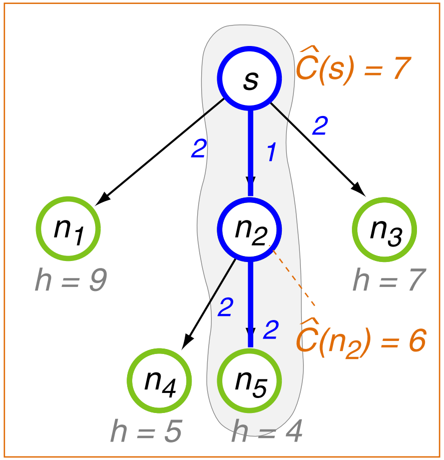
● Solved rest problem

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



Path cost computation based on summation of edge costs.


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 36 (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 ($\leftrightarrow$) it is of the following form:

$$F[e, c] = e + c$$

Remarks:

- $g_{P_{s-n}}(n)$ is a shorthand for 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

```
BF*($s$, $\mathit{successors}$, $\star$, $f$)   // BF*: The delayed termination variant of BF.
    ...
    LOOP
      IF (OPEN $= \emptyset$) THEN RETURN($\mathit{Fail}$);
      $n = \mathit{min}$(OPEN, $f$);   // Find most promising (cheapest) solution base.
      IF $\star(n)$ THEN RETURN($n$);   // Delayed termination.
```

Define $f(n)$ as $\widehat{C}_P(s)$ with P back-pointer path of n using Definition 33:

➜ $f(n)$ is defined by a recursive cost function.

➜ Algorithm BF becomes Algorithm Z.

Make use of delayed termination:

➜ Algorithm BF becomes Algorithm BF*.

➜ Algorithm Z becomes Algorithm Z*.

Make use of an additive cost measure ($f = g + h$ for short):

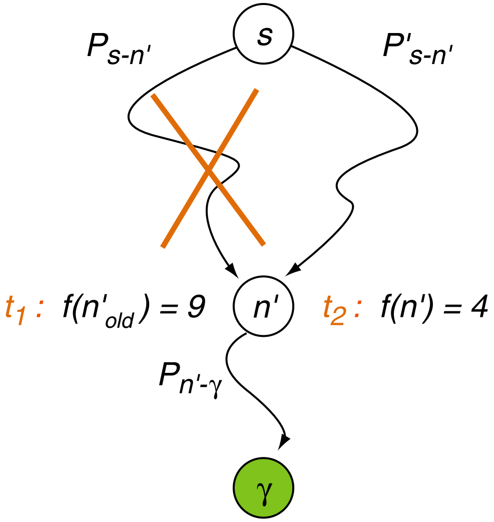
➜ Algorithm Z* becomes Algorithm A*.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation

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
          $n'_{old}$ = $retrieve(n', \text{OPEN} \cup \text{CLOSED})$;
          IF ($f(n') < f(n'_{old})$)
          THEN
            $update\_backpointer(n'_{old}, n)$;
            IF $n'_{old} \in$ CLOSED THEN ...ENDIF
          ENDIF
```

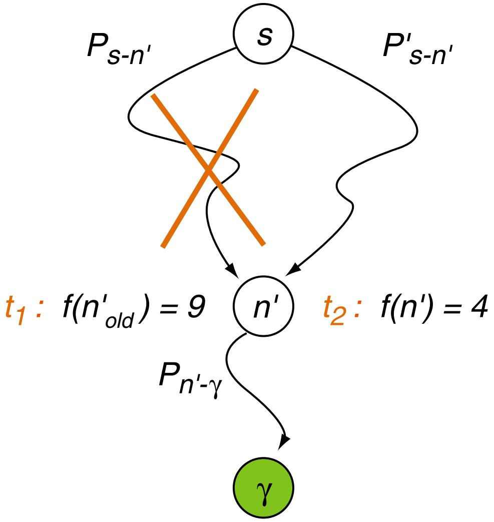


Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation

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 37 (Order-Preserving Cost Estimation)

A cost function $\widehat{C}_P(n)$ is called *order-preserving* if for all nodes n_1, n_2, n_3 and for all paths $P_{n_1-n_2}$, $P'_{n_1-n_2}$ from n_1 to n_2, and all paths $P_{n_2-n_3}$ from n_2 to n_3 holds:

$$\widehat{C}_{P_{n_1-n_2}}(n_1) \leq \widehat{C}_{P'_{n_1-n_2}}(n_1) \quad \Rightarrow \quad \widehat{C}_{P_{n_1-n_3}}(n_1) \leq \widehat{C}_{P'_{n_1-n_3}}(n_1)$$

$P_{n_1-n_3}$ resp. $P'_{n_1-n_3}$ denote the paths from n_1 to n_3 that result from concatenating the paths $P_{n_1-n_2}$ resp. $P'_{n_1-n_2}$ with $P_{n_2-n_3}$.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 37 (Order-Preserving Cost Estimation)

A cost function $\widehat{C}_P(n)$ is called *order-preserving* if for all nodes n_1, n_2, n_3 and for all paths $P_{n_1-n_2}$, $P'_{n_1-n_2}$ from n_1 to n_2, and all paths $P_{n_2-n_3}$ from n_2 to n_3 holds:

$$\widehat{C}_{P_{n_1-n_2}}(n_1) \leq \widehat{C}_{P'_{n_1-n_2}}(n_1) \quad \Rightarrow \quad \widehat{C}_{P_{n_1-n_3}}(n_1) \leq \widehat{C}_{P'_{n_1-n_3}}(n_1)$$

$P_{n_1-n_3}$ resp. $P'_{n_1-n_3}$ denote the paths from n_1 to n_3 that result from concatenating the paths $P_{n_1-n_2}$ resp. $P'_{n_1-n_2}$ with $P_{n_2-n_3}$.

Corollary 38 (Order-Preserving Cost Estimation)

If a cost function $\widehat{C}_P(n)$ is order-preserving, then for all nodes n_1, n_2, n_3 and for all paths $P_{n_1-n_2}$, $P'_{n_1-n_2}$ from n_1 to n_2, and all paths $P_{n_2-n_1,n_2,n_3}$ from n_2 to n_1, n_2, n_3 holds:

$$\widehat{C}_{P_{n_1-n_3}}(n_1) > \widehat{C}_{P'_{n_1-n_3}}(n_1) \quad \Rightarrow \quad \widehat{C}_{P_{n_1-n_2}}(n_1) > \widehat{C}_{P'_{n_1-n_2}}(n_1)$$

and

$$\widehat{C}_{P_{n_1-n_2}}(n_1) = \widehat{C}_{P'_{n_1-n_2}}(n_1) \quad \Rightarrow \quad \widehat{C}_{P_{n_1-n_3}}(n_1) = \widehat{C}_{P'_{n_1-n_3}}(n_1)$$

Again, $P_{n_1-n_3}$ and $P'_{n_1-n_3}$ denote the paths from n_1 to n_3 that result from concatenating the paths $P_{n_1-n_2}$ and $P'_{n_1-n_2}$ with $P_{n_2-n_3}$.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 39 (Order-Preserving Cost Estimation for Solution Paths)

A cost function $\widehat{C}_P(n)$ is called *order-preserving for solution paths* if for all nodes n_2, γ and for all paths P_{s-n_2}, P'_{s-n_2} from s to n_2, and all paths $P_{n_2-\gamma}$ from n_2 to γ holds:

$$\widehat{C}_{P_{s-n_2}}(s) \leq \widehat{C}_{P'_{s-n_2}}(s) \quad \Rightarrow \quad \widehat{C}_{P_{s-\gamma}}(s) \leq \widehat{C}_{P'_{s-\gamma}}(s)$$

$P_{s-\gamma}$ resp. $P'_{s-\gamma}$ denote the paths from s to γ that result from concatenating the paths P_{s-n_2} resp. P'_{s-n_2} with $P_{n_2-\gamma}$.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Definition 39 (Order-Preserving Cost Estimation for Solution Paths)

A cost function $\widehat{C}_P(n)$ is called *order-preserving for solution paths* if for all nodes n_2, γ and for all paths P_{s-n_2}, P'_{s-n_2} from s to n_2, and all paths $P_{n_2-\gamma}$ from n_2 to γ holds:

$$\widehat{C}_{P_{s-n_2}}(s) \leq \widehat{C}_{P'_{s-n_2}}(s) \quad \Rightarrow \quad \widehat{C}_{P_{s-\gamma}}(s) \leq \widehat{C}_{P'_{s-\gamma}}(s)$$

$P_{s-\gamma}$ resp. $P'_{s-\gamma}$ denote the paths from s to γ that result from concatenating the paths P_{s-n_2} resp. P'_{s-n_2} with $P_{n_2-\gamma}$.

Corollary 40 (Order-Preserving Cost Estimation for Solution Paths)

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

Corollary 41 (Order-Preserving)

An evaluation functions f that is defined by an order preserving cost function $\widehat{C}_P(n)$ is order-preserving.

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Lemma 42 (Order-Preserving)

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

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Lemma 42 (Order-Preserving)

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

Proof (of Lemma, sketch)

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 (by induction on length of $P_{n'-n}$):

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

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

Lemma 42 (Order-Preserving)

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

Proof (of Lemma, sketch)

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 (by induction on length of $P_{n'-n}$):

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

Remarks:

- $g(n)$ denotes the sum of the edge cost values along the back-pointer path from s to n. Since A* as BF* variant maintains for each node generated at each point in time a unique back-pointer, 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 only depends on n.

- For clarity, the proof of the above lemma was given for the special case that the node properties used in the recursive computation are the edge cost values to its successor in the considered path. In a general proof use $E(n_i)$ instead of $c(n_i, n_j)$ for edges (n_i, n_j).

Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

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

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

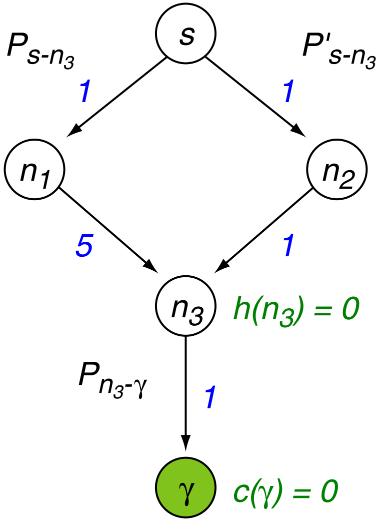
Evaluation of State-Space Graphs

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

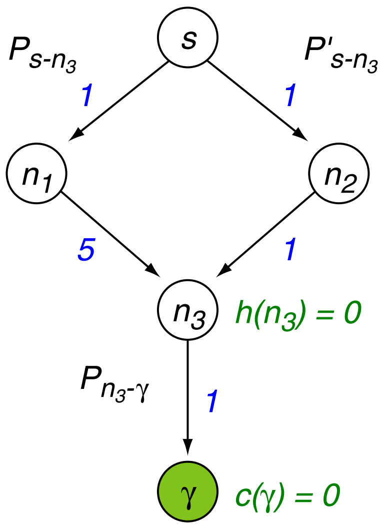
Optimum Solution Cost and Order Preservation (continued)

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

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

$$P_{s-n_3} = (s, n_1, n_3)$$
$$P'_{s-n_3} = (s, n_2, n_3)$$
$$P_{n_3-\gamma} = (n_3, \gamma)$$

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



Evaluation of State-Space Graphs

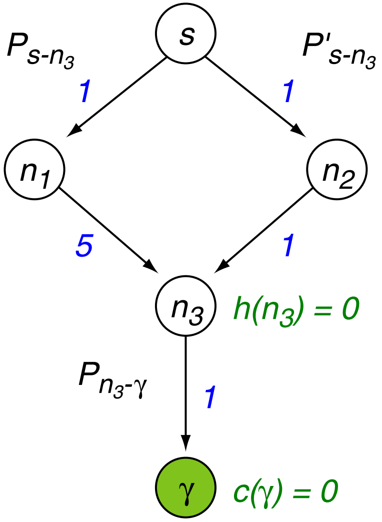
Optimum Solution Cost and Order Preservation (continued)

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)

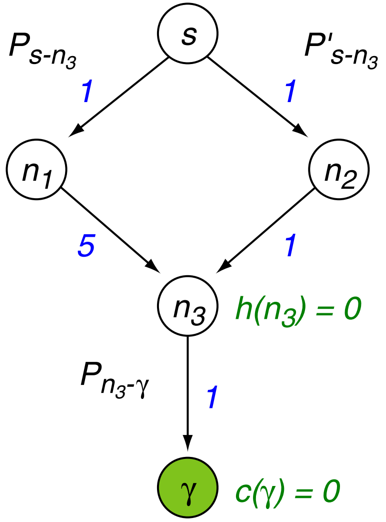
Example for a cost function that is recursive but *not* order-preserving:

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

$$P_{s-n_3} = (s, n_1, n_3)$$
$$P'_{s-n_3} = (s, n_2, n_3)$$
$$P_{n_3-\gamma} = (n_3, \gamma)$$

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

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



Evaluation of State-Space Graphs

Optimum Solution Cost and Order Preservation (continued)

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

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

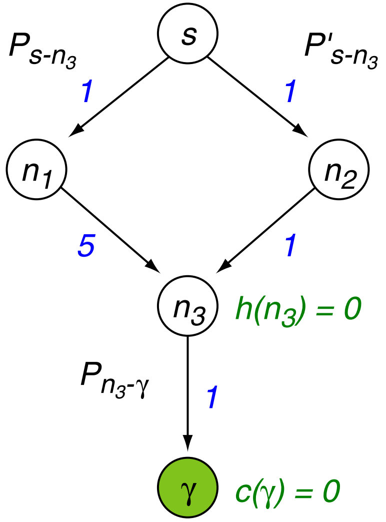
$$P_{s-n_3} = (s, n_1, n_3)$$
$$P'_{s-n_3} = (s, n_2, n_3)$$
$$P_{n_3-\gamma} = (n_3, \gamma)$$

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

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

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

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



Evaluation of State-Space Graphs

Examples of simple path cost functions based on non-negative edge cost values

- Maximum Edge-Cost: smaller is better, recursively definable

  $$C_P = \max_{e \in P} c(e)$$

  $\widehat{C}_P$ using $h = 0$: order-preserving, optimistic.

- Minimum Edge-Cost: smaller is better, recursively definable

  $$C_P = \min_{e \in P} c(e)$$

  $\widehat{C}_P$ using $h = 0$: not order-preserving, but optimistic.
  (Observe: $\widehat{C}_P(n) = 0$ for non-solution paths P.)
  $\widehat{C}_P := C_P$: order-preserving, not optimistic.

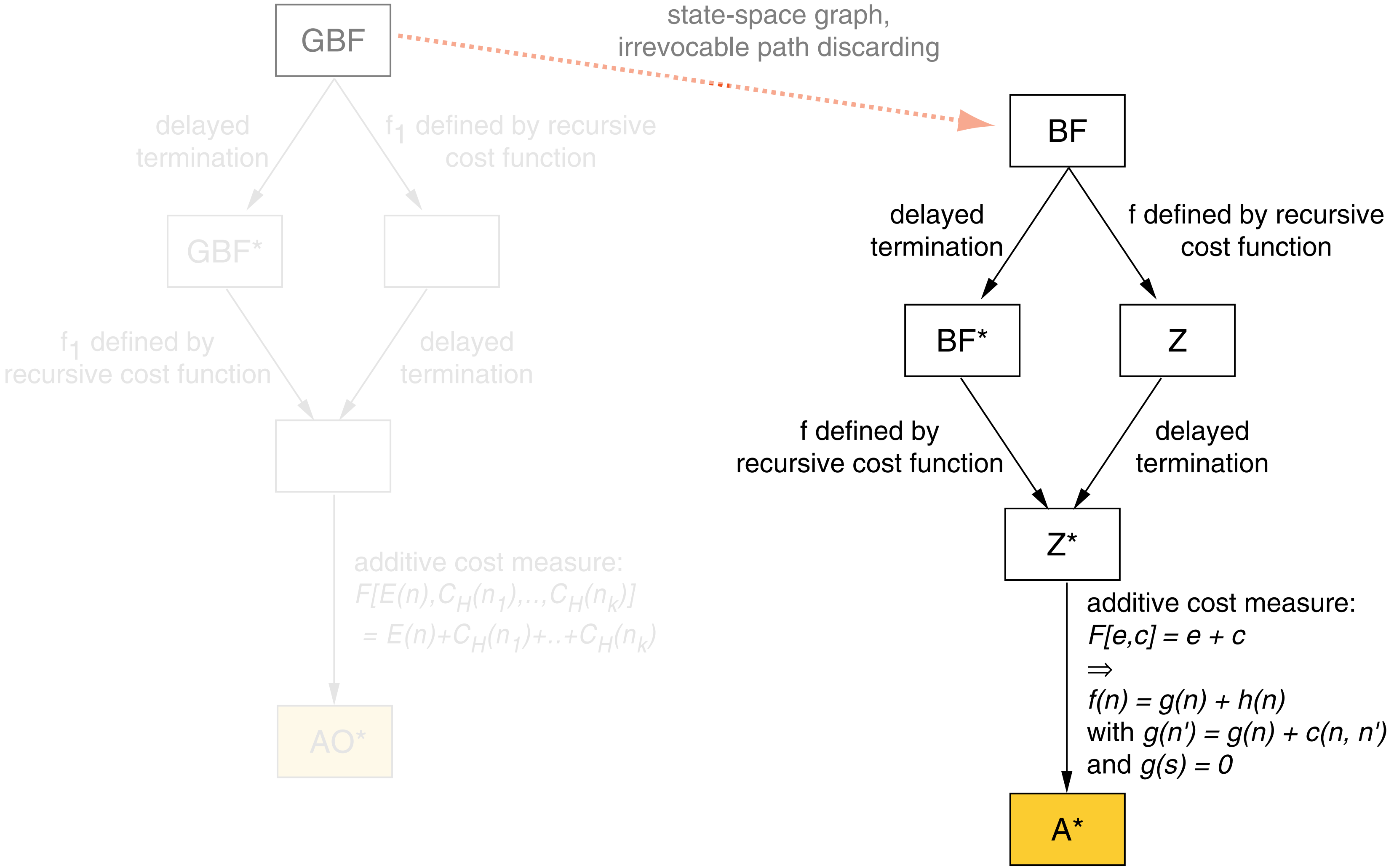
- Edge-Cost Bandwidth: smaller is better, not recursively definable

  $$C_P = \max_{e \in P} c(e) - \min_{e \in P} c(e)$$

  $\widehat{C}_P := C_P$: not order-preserving, not optimistic

Evaluation of State-Space Graphs

Taxonomy of Best-First Algorithms



GBF

state-space graph,
irrevocable path discarding

delayed
termination

f_1 defined by recursive
cost function

GBF*

f_1 defined by
recursive cost function

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*

BF

delayed
termination

f defined by recursive
cost function

BF*

Z

f defined by
recursive cost function

delayed
termination

Z*

additive cost measure:
$F[e,c] = e + c$
$\Rightarrow$
$f(n) = g(n) + h(n)$
with $g(n') = g(n) + c(n, n')$
and $g(s) = 0$

A*

Algorithm A*

Algorithm: A* (Compare Basic-BF*, BF*.)

Input: s. Start node representing the initial state (problem) in G.
successors(n). Returns *new instances of* nodes for the successor states in G.
$\star(n)$. Predicate that is *True* if n represents a goal state in G.
$c(n, n')$. Cost of the edge in G represented by (n, n').
$h(n)$. Heuristic cost estimation for the state in G represented by n.

Output: A node γ representing an (optimum) solution path for s in G or the symbol *Fail*.

Definition of f :

- ❑ The evaluation function in A* is $f = g + h$.
- ❑ f is order-preserving.
- ❑ $g(n)$ is the sum of edge cost values $c(n', n'')$ in the current back-pointer path PP_{s-n} of n.
- ❑ f can be defined by $f(n) := \widehat{C}_{PP_{s-n}}(s)$ with $\widehat{C}_P$ recursively defined using addition "$+$" as cost measure F, local property $c(n', n'')$, and estimation $h(n)$ in non-goal nodes n.

Algorithm A*

```
A*($s$, successors, $\star$, $c$, $h$)    // A special case of BF*.

1.  $s$.parent = null;  $g(s) = 0$;  $f(s) = g(s) + h(s)$;  add($s$, OPEN, $f(s)$);
2.  LOOP
3.    IF (OPEN == $\emptyset$) THEN RETURN(Fail);
4.    $n$ = min(OPEN, $g + h$);    // Most promising sol. base minimizes $f = g + h$.
      IF $\star(n)$ THEN RETURN($n$);    // Delayed termination.
      remove($n$, OPEN);  add($n$, CLOSED);
5.    FOREACH $n'$ IN successors($n$) DO    // Expand $n$.
        $n'$.parent = $n$;
        $g(n') = g(n) + c(n, n')$;  $f(n') = g(n') + h(n')$;
        $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);    // State of $n'$ already visited?
        IF ( $n'_{old}$ == null )
        THEN    // $n'$ refers to a new state.
          add($n'$, OPEN, $f(n')$);
        ELSE    // $n'$ refers to an already visited state.
          IF ( $g(n') < g(n'_{old})$ )    // Compare cost of backpointer paths.
          THEN    // Solution base of $n'$ is cheaper:  path discarding.
            $n'_{old}$.parent = $n'$.parent;  $g(n'_{old}) = g(n')$;
            IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED);  add($n'_{old}$, OPEN, $f(n'_{old})$);  ENDIF
          ENDIF
        ENDIF
      ENDDO
6.  ENDLOOP
```

Remarks:

- $h(n)$ estimates the rest problem cost at node n (optimum cost of a solution path starting in 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 proven, we still have to assume for the case of additional solution constraints that the following equivalence holds:

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

  $\Leftrightarrow$

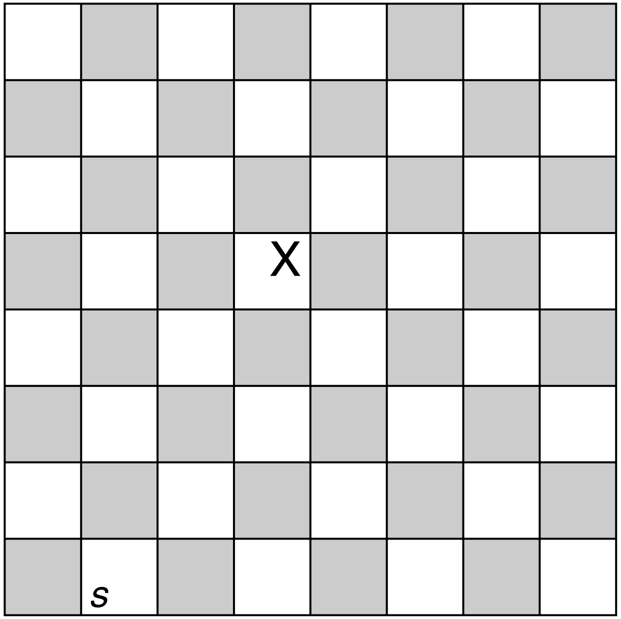
  "Solution base $P'_{s-n'}$ can be completed by $P_{n'-\gamma}$ to a solution path."

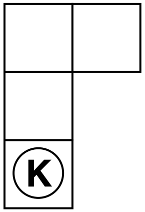
  This equivalence is trivially satisfied, if solution constraints restrict only local properties of γ but not properties of the back-pointer path of γ.

Algorithm A*

Example: Knight Moves

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





Knight move

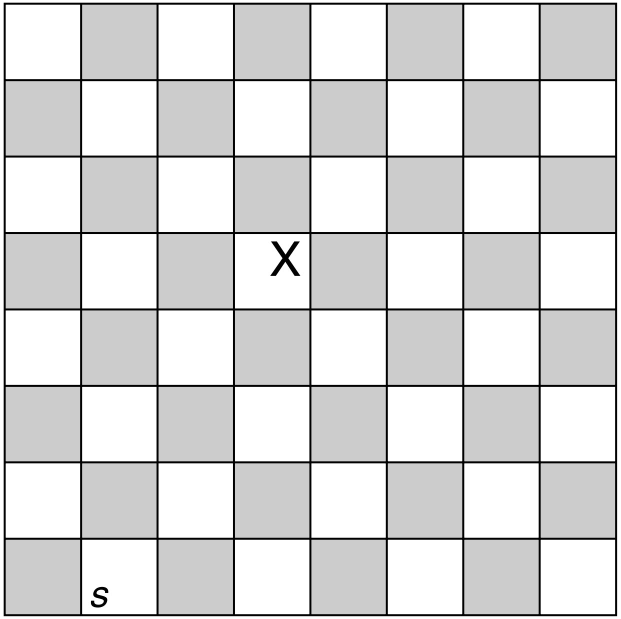
Let n' be a direct successor of n.

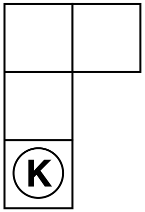
- $f(n') = g(n') + h(n')$
- $g(n') = g(n) + c(n, n')$
- $g(s) = 0$
- $c(n, n') = 1$

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.

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

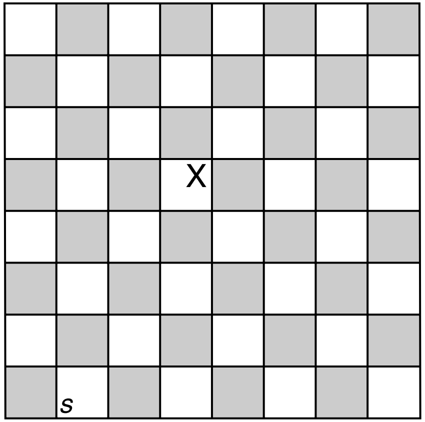
$$h_1 = \left\lceil \frac{\#rows}{2} \right\rceil$$

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

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

Algorithm A*

Example: Knight Moves (continued)



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

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

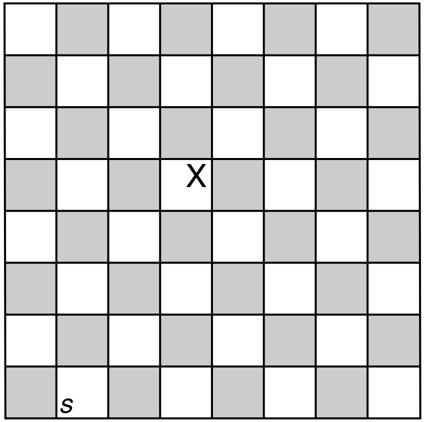
$$h = h_1 = \lceil \frac{\#rows}{2} \rceil$$

Notation:

- OPEN and CLOSED contain nodes, not states. Therefore, a listed state has to be seen as a node structure that refers to this state. If a state is listed more than once, these occurrences have to be understood as different node structures that refer to the same state. Therefore, diffferent values can be assigned by f and g. A better but more cumbersome notation would be $n_i(s)$, where i uniquely identifies a node.

Algorithm A*

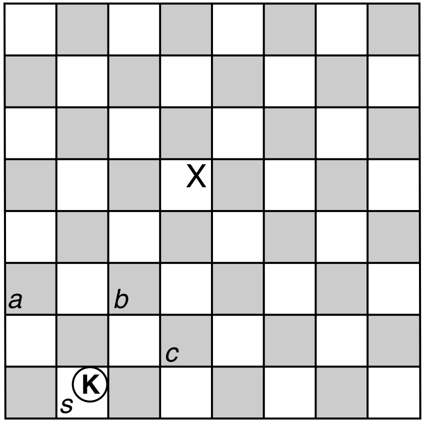
Example: Knight Moves (continued)



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

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

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



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

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

Remarks:

- In this example, successor nodes in node expansions are enumerated clockwise, starting from top left.
- In this example, nodes in OPEN are sorted by the criteria

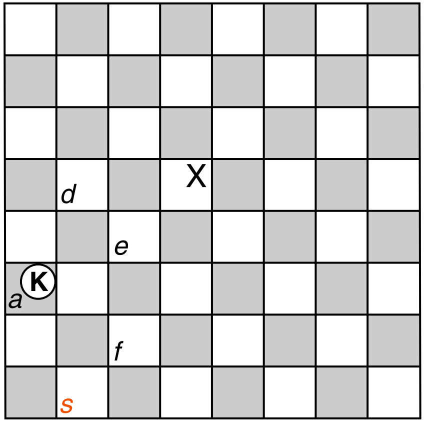
  1. f-value (major, smallest first),
  2. h-value (minor, smallest first) with preference to goal nodes.

  Ties are broken arbitrarily. (E.g., nodes found earlier last.)

  Therefore, we can assume that A* will select the first node from OPEN.

Algorithm A*

Example: Knight Moves (continued)



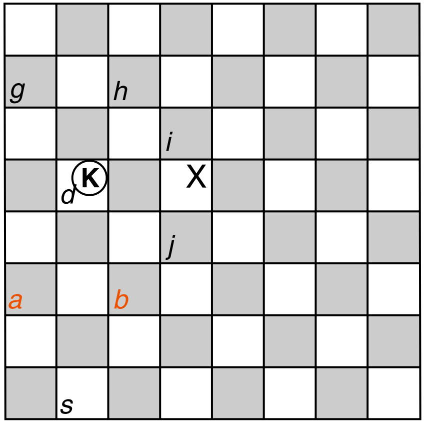
OPEN	CLOSED
$\{\underline{s}\}$	$\{\}$
$\{\underline{a}, b, c\}$	$\{s\}$
$\{\underline{d}, b, e, c, f\}$	$\{a, s\}$

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
s	2	2	4

Position s was reached again with $f(s) = 4$, no reopening.

Algorithm A*

Example: Knight Moves (continued)

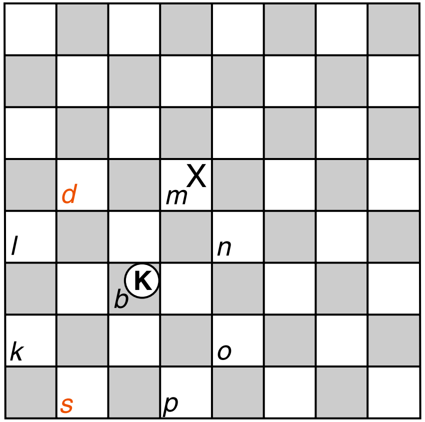


OPEN	CLOSED
$\{\underline{s}\}$	$\{\}$
$\{\underline{a}, b, c\}$	$\{s\}$
$\{\underline{d}, b, e, c, f\}$	$\{a, s\}$
$\{\underline{b}, e, c, g, h, i, j, f\}$	$\{d, a, s\}$

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
s	2	2	4
g	3	1	4
h	3	1	4
i	3	1	4
j	3	1	4
b	3	1	4
a	3	1	4

Algorithm A*

Example: Knight Moves (continued)

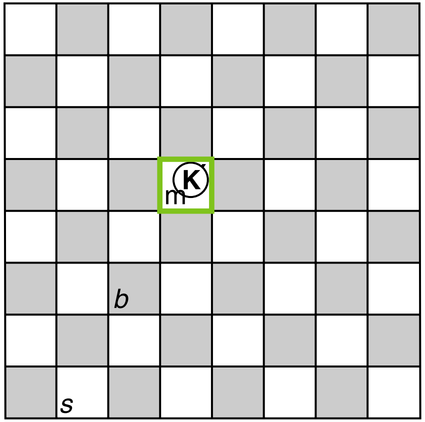


OPEN	CLOSED
$\{\underline{s}\}$	$\{\}$
$\{\underline{a}, b, c\}$	$\{s\}$
$\{\underline{d}, b, e, c, f\}$	$\{a, s\}$
$\{\underline{b}, e, c, f, g, h, i, j\}$	$\{d, a, s\}$
$\{\underline{m}, n, l, e, c,$ $o, p, k, g, h, i, j, f\}$	$\{b, d, a, s\}$

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
s	2	2	4
g	3	1	4
h	3	1	4
i	3	1	4
j	3	1	4
b	3	1	4
a	3	1	4
d	2	0	2
m	2	0	2
n	2	1	3
o	2	2	4
p	2	2	4
s	2	2	4
k	2	2	4
l	2	1	3

Algorithm A*

Example: Knight Moves (continued)



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

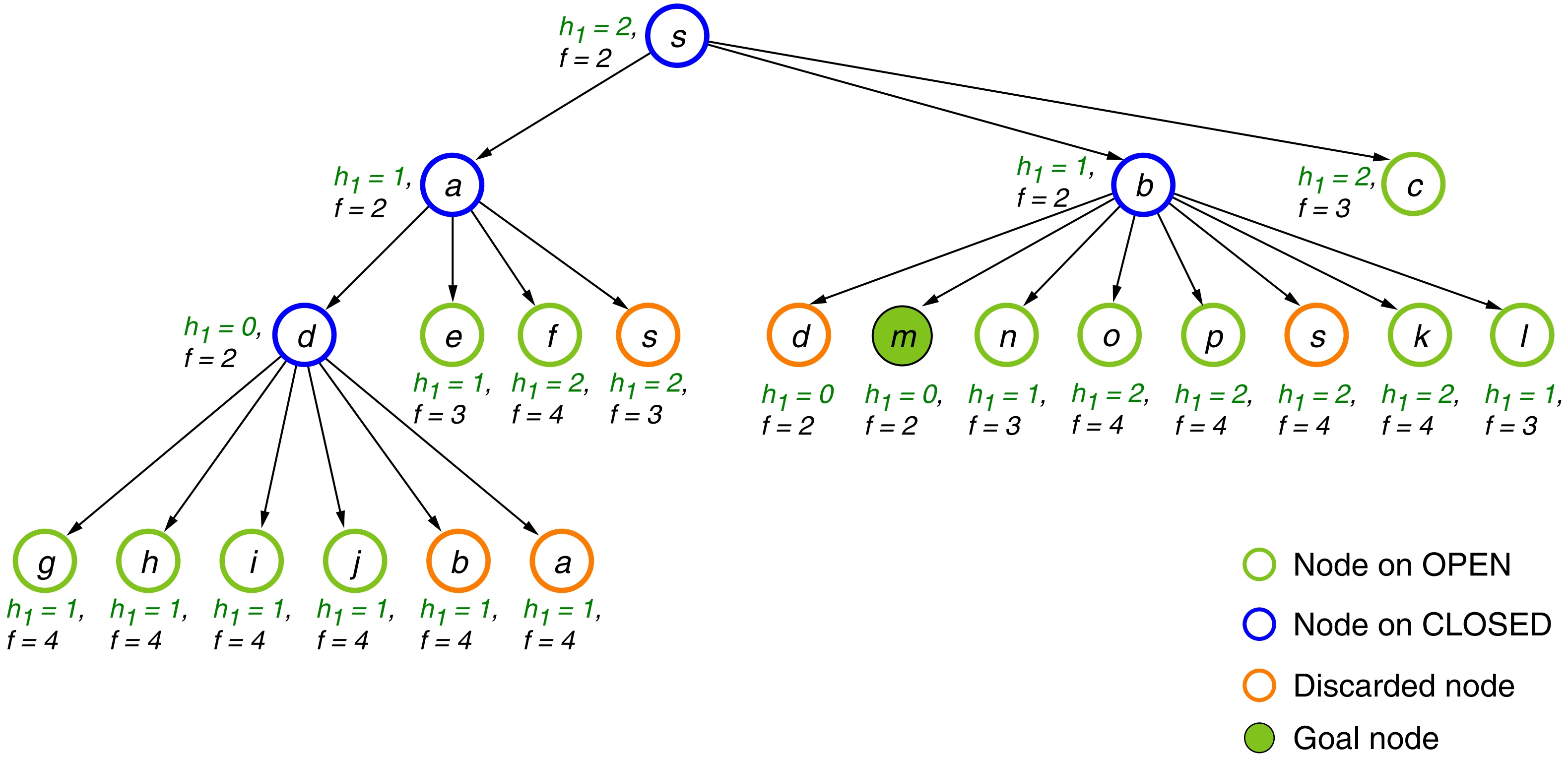
n	$g(n)$	$h_1(n)$	$f(n)$
s	0	2	2
a	1	1	2
b	1	1	2
c	1	2	3
d	2	0	2
e	2	1	3
f	2	2	4
s	2	2	4
g	3	1	4
h	3	1	4
i	3	1	4
j	3	1	4
b	3	1	4
a	3	1	4
d	2	0	2
m	2	0	2
n	2	1	3
o	2	2	4
p	2	2	4
s	2	2	4
k	2	2	4
l	2	1	3

Goal node found, termination.

Algorithm A*

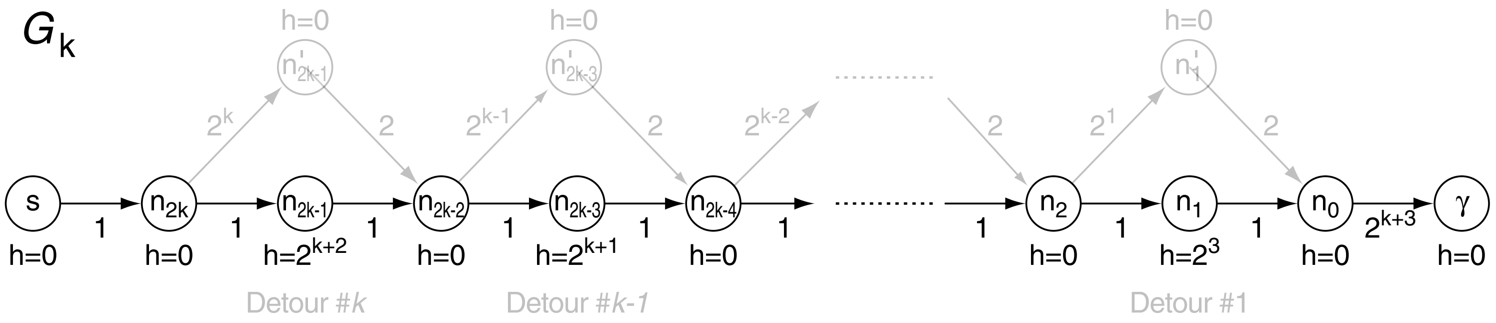
Example: Knight Moves (continued)

Analyzed part of the search space graph at termination (tree unfolding):



Algorithm A*

Exponential Runtime Example

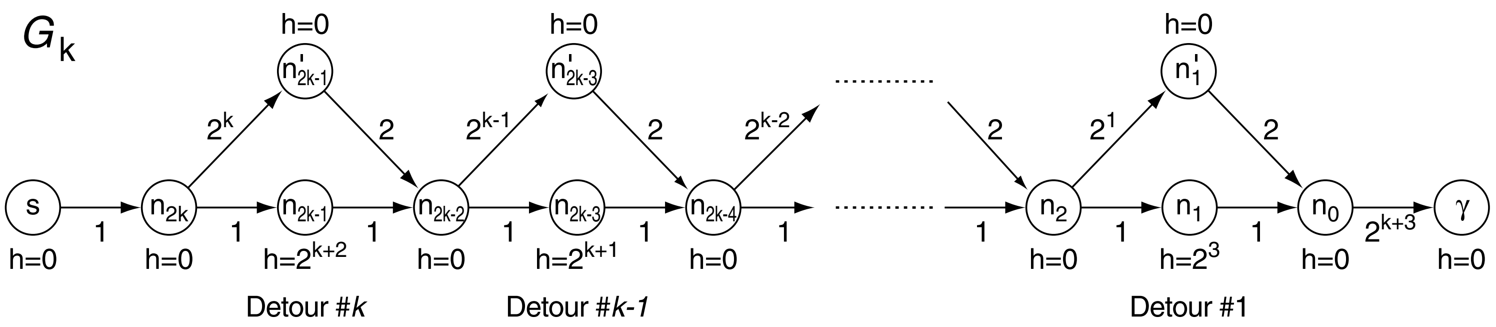


- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial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

➜ G_k has $3k + 3$ nodes and $4k + 2$ edges. h is optimistic.
A* requires more than 2^k node expansions before termination.

Algorithm A*

Exponential Runtime Example (continued)

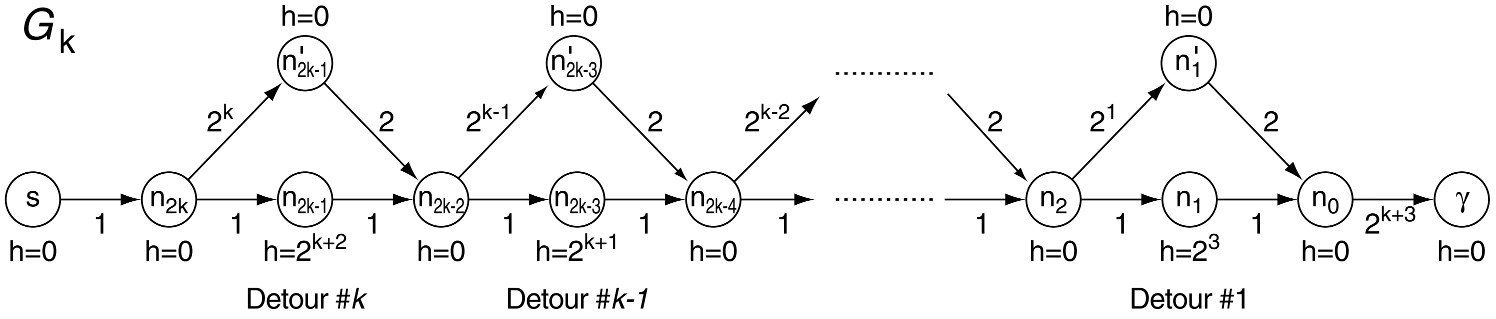


- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - – at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - – any two nodes on OPEN share the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

➜ G_k has $3k + 3$ nodes and $4k + 2$ edges. h is optimistic.
A* requires more than 2^k node expansions before termination.

Algorithm A*

Exponential Runtime Example (continued)

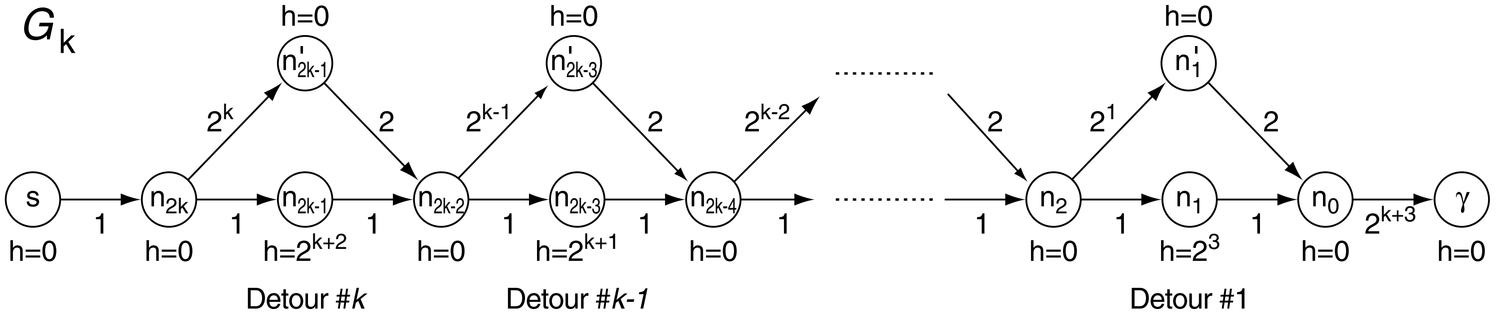


- ❑ Optimum cost path with path cost $2^{k+3} + 2k + 1$.
- ❑ Additional cost for using detours starting from $n_{2j}, 1 \leq j \leq k$ less than 2^{j+1}.
- ❑ At each point in time before A* terminates,
  - at most one node $n_{2j}, 1 \leq j \leq k$ is on OPEN,
  - any two nodes on OPEN share the initial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

➜ G_k has $3k + 3$ nodes and $4k + 2$ edges. h is optimistic.
A* requires more than 2^k node expansions before termination.

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 back-pointer 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.

➜ G_k has $3k + 3$ nodes and $4k + 2$ edges. h is optimistic.
A* requires more than 2^k node expansions before termination.