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

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

Properties of Problem-Reduction Graphs

1. G is an implicitly defined directed AND-OR graph.
2. G is acyclic and locally finite.

When searching problem-reduction graphs with the Algorithm GBF,

- ❑ the underlying problem-reduction graph G is assumed to be acyclic,
- ❑ f_1 evaluates solution bases H,
- ❑ a most promising solution base minimizes f_1,
- ❑ a most promising solution base is searched *among all solution bases* maintained by GBF,
- ❑ solution bases maintained by GBF are solution bases for s in the explicit part G_e of G with tip nodes in OPEN,
- ❑ f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

Properties of Problem-Reduction Graphs

1. G is an implicitly defined directed AND-OR graph.
2. G is acyclic and locally finite.

When searching problem-reduction graphs with the Algorithm GBF,

- the underlying problem-reduction graph G is assumed to be acyclic,
- f_1 evaluates solution bases H,
- a most promising solution base minimizes f_1,
- a most promising solution base is searched *among all solution bases* maintained by GBF,
- solution bases maintained by GBF are solution bases for s in the explicit part G_e of G with tip nodes in OPEN,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

Properties of Problem-Reduction Graphs

1. G is an implicitly defined directed AND-OR graph.
2. G is acyclic and locally finite.

When searching problem-reduction graphs with the Algorithm GBF,

- the underlying problem-reduction graph G is assumed to be acyclic,
- f_1 evaluates solution bases H,
- a most promising solution base minimizes f_1,
- a most promising solution base is searched *among all solution bases* maintained by GBF,
- solution bases maintained by GBF are solution bases for s in the explicit part G_e of G with tip nodes in OPEN,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

Properties of Problem-Reduction Graphs

1. G is an implicitly defined directed AND-OR graph.
2. G is acyclic and locally finite.

When searching problem-reduction graphs with the Algorithm GBF,

- the underlying problem-reduction graph G is assumed to be acyclic,
- f_1 evaluates solution bases H,
- a most promising solution base minimizes f_1,
- a most promising solution base is searched *among all solution bases* maintained by GBF,
- solution bases maintained by GBF are solution bases for s in the explicit part G_e of G with tip nodes in OPEN,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

Properties of Problem-Reduction Graphs

1. G is an implicitly defined directed AND-OR graph.
2. G is acyclic and locally finite.

When searching problem-reduction graphs with the Algorithm GBF,

- the underlying problem-reduction graph G is assumed to be acyclic,
- f_1 evaluates solution bases H,
- a most promising solution base minimizes f_1,
- a most promising solution base is searched *among all solution bases* maintained by GBF,
- solution bases maintained by GBF are solution bases for s in the explicit part G_e of G with tip nodes in OPEN,
- f_2 returns a most informative node in a solution base.

Remarks:

- Again, G denotes the explored part of the search space graph.
- When searching AND-OR graphs with the algorithm GBF, only solution bases H need to be considered that have not been considered so far. I.e., only solution bases H with leaf nodes that are goal nodes or nodes in `OPEN` and inner nodes in `CLOSED` are considered.
- In an AND-OR graph, a node may be part of several solution bases.
- Usually, the evaluation function $f_1(H)$ is based on a heuristic $h(n)$.
- $h(n)$ estimates the optimum cost of solution graphs for the rest problem associated with a node n. Ideally, $h(n)$ should consider the probability of the solvability of the problem at node n.
- Compared to $f_2(H)$, the evaluation function $f_1(H)$ is usually more important.
- Searching AND-OR graphs with cycles is intricate. Note that the algorithms presented by Martelli and Montanari [1973, 1978], Nilsson [1980], or Pearl [1984] presume graphs without cycles. In the following, we will restrict to acyclic graphs as well.
- In the context of Markov Decision Processes, MDP, and advanced planning algorithms researchers have dealt with AND-OR graphs that contain cycles. [Bonet/Geffner 2005]

Best-First Search for AND-OR Graphs

Algorithm:	GBF
Input:	s. Start node representing the initial problem. *successors*(n). Returns the successors of node n. $f_1(H)$. Evaluation function for solution bases. $f_2(H)$. Selection function for OPEN-nodes in a solution base.
Output:	A solution graph or the symbol *Fail*.
Subroutines:	*is_solved*(n). Predicate that is *True* if n is labeled "solved" or n is a goal node. *propagate_label*(n). Function that propagates node label "solved" along backpointers.

Best-First Search for AND-OR Graphs

```
GBF($s$, $\mathit{successors}$, $\mathit{is\_solved}$, $f_1$, $f_2$)

  $\mathit{insert}(s, \text{OPEN})$;  $\mathit{add\_node}(s, G_e)$;    // $G_e$ is the explored part of $G$.
  LOOP
    IF (OPEN $= \emptyset$) THEN RETURN($\mathit{Fail}$);



  ENDLOOP
```

Best-First Search for AND-OR Graphs

```
GBF($s$, $successors$, $is\_solved$, $f_1$, $f_2$)

    $insert(s, \texttt{OPEN})$;  $add\_node(s, G_e)$;    // $G_e$ is the explored part of $G$.
    LOOP
        IF ($\texttt{OPEN} = \emptyset$) THEN RETURN($Fail$);
        $H = min\_solution\_base(s, G_e, f_1)$;    // Find most prom. sol. base in $G_e$.
        $n = min(\texttt{OPEN} \cap H, f_2)$;    // Find most informative non-goal node in $H$.
        $remove(n, \texttt{OPEN})$;  $push(n, \texttt{CLOSED})$;

    ENDLOOP
```

Best-First Search for AND-OR Graphs

```
GBF($s$, successors, is_solved, $f_1$, $f_2$)

 1.  insert($s$, OPEN);  add_node($s$, $G_e$);    // $G_e$ is the explored part of $G$.
 2.  LOOP
 3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
4.a    $H$ = min_solution_base($s$, $G_e$, $f_1$);    // Find most prom. sol. base in $G_e$.
4.b    $n$ = min(OPEN $\cap$ $H$, $f_2$);    // Find most informative non-goal node in $H$.
       remove($n$, OPEN);  push($n$, CLOSED);
 5.    FOREACH $n'$ IN successors($n$) DO



           insert($n'$, OPEN);  add_node($n'$, $G_e$);

         add_backpointer($n'$, $n$);  add_edge(($n$, $n'$), $G_e$);
         IF is_solved($n'$) // Is $n'$ goal node                    ?
         THEN
           propagate_label($n'$);
           IF is_solved($s$) THEN RETURN(compute_solution_graph($G_e$));
         ENDIF
       ENDDO
 6.  ENDLOOP
```

Best-First Search for AND-OR Graphs [Basic_AND-OR, BF, BF*, GBF*]

```
GBF($s$, successors, is_solved, $f_1$, $f_2$)

insert($s$, OPEN); add_node($s$, $G_e$);   // $G_e$ is the explored part of $G$.
LOOP
  IF (OPEN $= \emptyset$) THEN RETURN(Fail);
  $H$ = min_solution_base($s$, $G_e$, $f_1$);   // Find most prom. sol. base in $G_e$.
  $n$ = min(OPEN $\cap H$, $f_2$);   // Find most informative non-goal node in $H$.
  remove($n$, OPEN); push($n$, CLOSED);
  FOREACH $n'$ IN successors($n$) DO

    IF ( $n' \in$ OPEN OR $n' \in$ CLOSED )   // Instance of $n'$ seen before?
    THEN   // Use old instance of $n'$ instead.
      $n'$ = retrieve($n'$, OPEN $\cup$ CLOSED);
    ELSE   // $n'$ encodes an instance of a new state.
      insert($n'$, OPEN); add_node($n'$, $G_e$);
    ENDIF
    add_backpointer($n'$, $n$); add_edge(($n$, $n'$), $G_e$);

    IF is_solved($n'$) // Is $n'$ goal node or labeled solvable?
    THEN
      propagate_label($n'$);
      IF is_solved($s$) THEN RETURN(compute_solution_graph($G_e$));
    ENDIF

  ENDDO
ENDLOOP
```

Remarks:

- Finding dead ends can be handled in the same way as finding goal nodes.

```
IF ( $\perp(n')$ OR is_labeled$(n')$ )    // Is $n'$ unsolvable?
THEN
   propagate_label$(n')$; IF is_unsolvable$(s)$ THEN RETURN$(Fail)$;
ENDIF
```

- The expansion of nodes in G can be implemented with or without occurrence check:

  (a) With occurrence check. If an already existing rest problem is encountered, only an additional link is added to G_e. As a consequence, G_e is a subgraph of the underlying search space graph G.

  (b) Without occurrence check. Each successor becomes a new node, irrespective of duplicate rest problems. As a consequence, G_e corresponds to an AND-OR tree that can be considered as an unfolding of the underlying search space graph G.

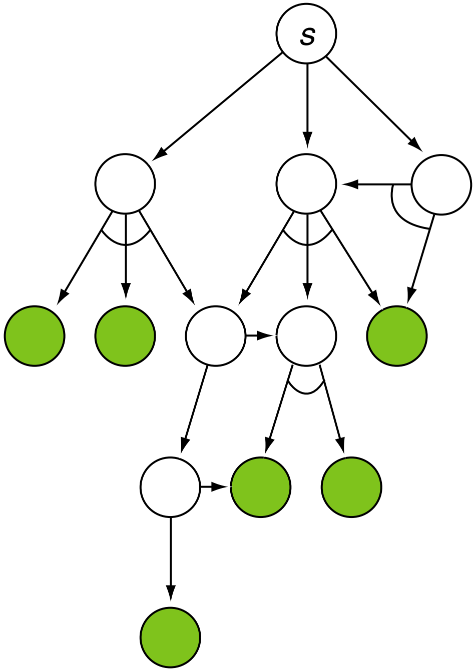
  Therefore, using an occurrence check or not can be seen as different ways of modeling a problem.

- The construction of solution bases depends strongly on the structure of the evaluation function f_1. In the general case, an individual construction of all possible solution bases plus an individual computation of the f_1-values cannot be avoided. However, if f_1 can be stated in a recursive manner using specific functions, computation of the minimum f_1 value and computation of a most promising solution base can be done very efficiently.

- General Best-First Algorithms are informed versions of Basic_AND-OR_Search.

Best-First Search for AND-OR Graphs

Illustration of GBF



○ Node on OPEN

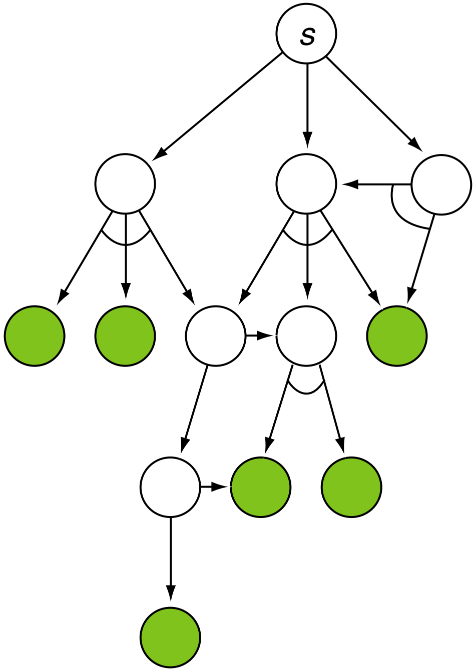
○ Node on CLOSED

● Solved rest problem

The underlying AND-OR graph of the following GBF illustration.

Best-First Search for AND-OR Graphs

Illustration of GBF



○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

$f_1(H)$ Evaluates solution bases H in regard to their edge count.
Answers the question: What does the solution associated with H cost?

$f_2(H)$ Returns the node with a most expensive rest problem in H.
Answers the question: What node to expand next?

$h(n)$ Estimates the number of edges for the rest problem at node n.

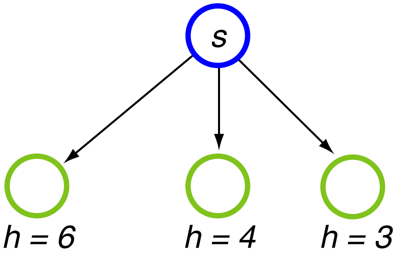
Remarks:

- The evaluation function $f_1(H)$ estimates the total number of edges of a solution graph extending solution base H. Obviously, $f_1(H)$ will be based on the heuristic $h(n)$.
- Rationale for f_1. Searching the smallest solution may quickly lead to a first solution graph. Keyword: *Small-is-Quick Principle*
- Rationale for f_2. A possible estimation error of h, which allegedly claims an unsolvable rest problem as a solvable one, should be detected earliest. Since it is reasonable to assume that the estimation error is proportional to the cost estimate, we pick a most expensive rest problem first.
- Recall that the complete AND-OR graph is neither given nor manageable for real-world problems.

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

1. s is expanded and moved to CLOSED.



○ Node on OPEN

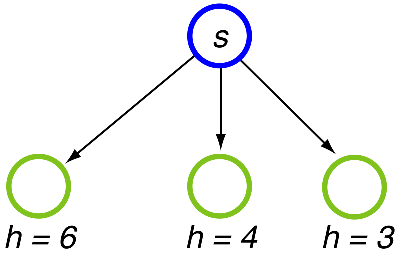
○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

1. s is expanded and moved to CLOSED.

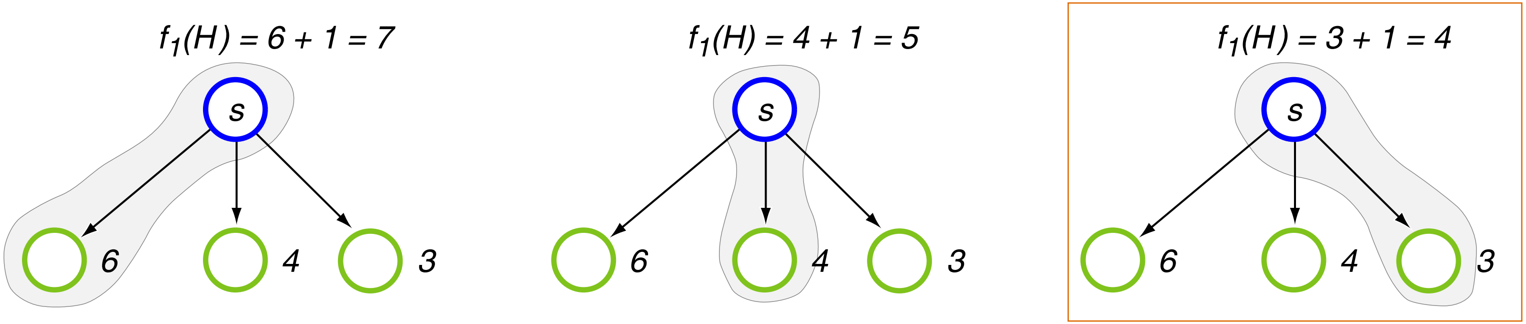


○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Possible solution bases:

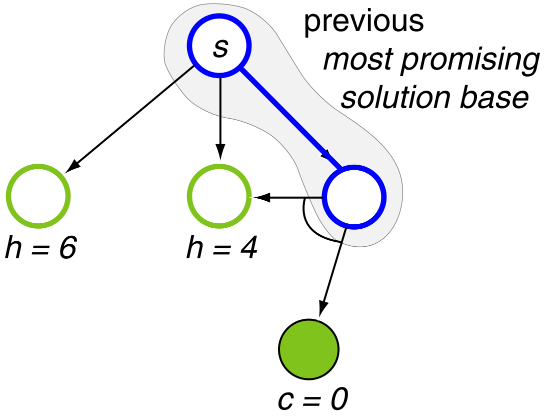


Most promising solution base H

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

2. Expansion of the node $n \in (\text{OPEN} \cap H)$.



○ Node on OPEN

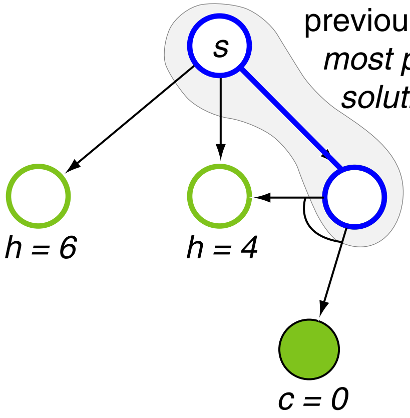
○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

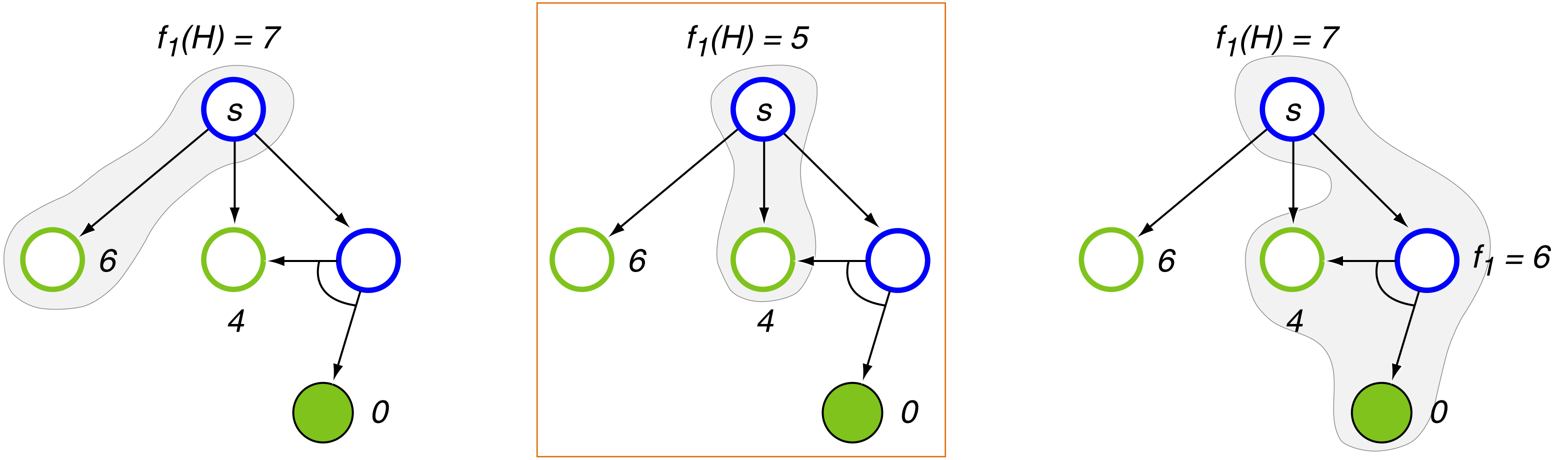
2. Expansion of the node $n \in (\text{OPEN} \cap H)$.



previous *most promising solution base*

- ○ Node on OPEN
- ○ Node on CLOSED
- ● Solved rest problem

Possible solution bases:

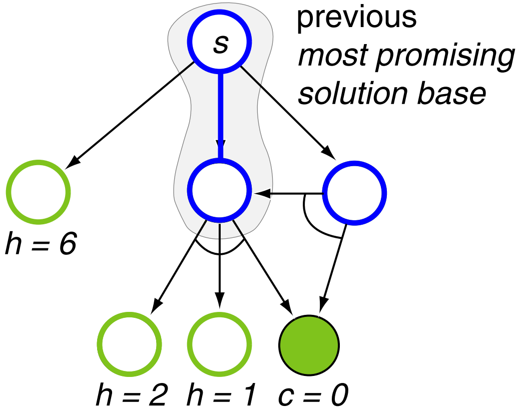


Most promising solution base H

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

3. Expansion of the node $n \in (\text{OPEN} \cap H)$.



○ Node on OPEN

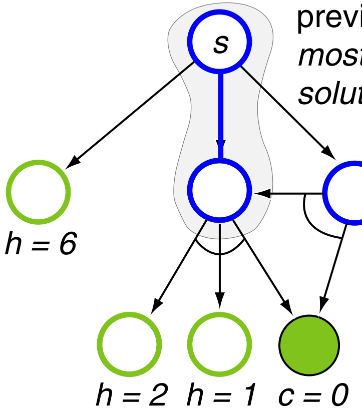
○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

3. Expansion of the node $n \in (\text{OPEN} \cap H)$.



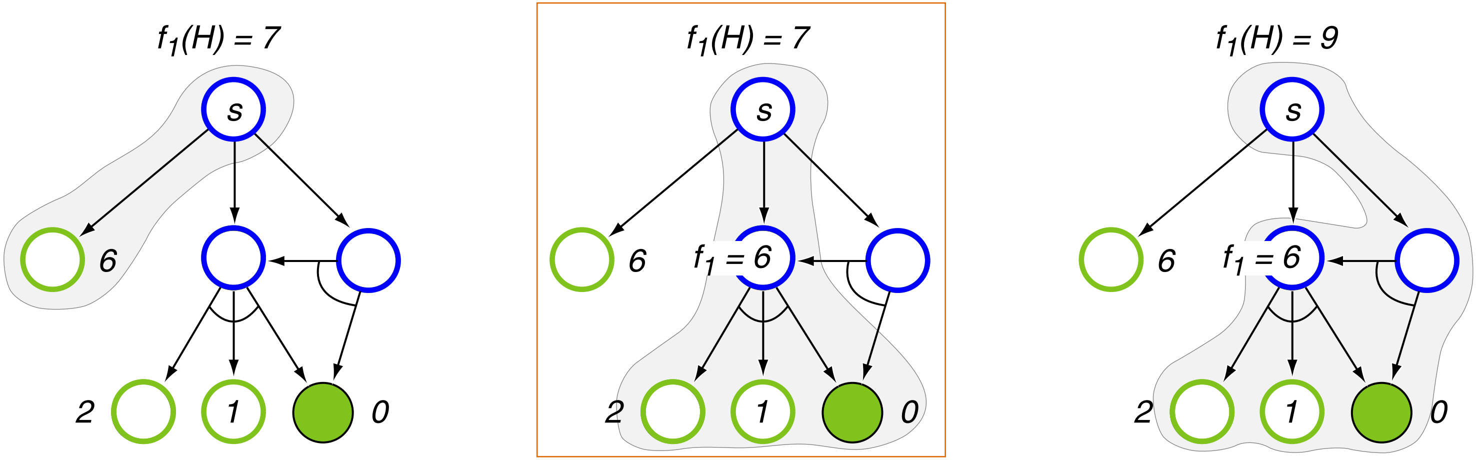
previous
most promising
solution base

○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Possible solution bases:



$f_1(H) = 7$

$f_1(H) = 7$

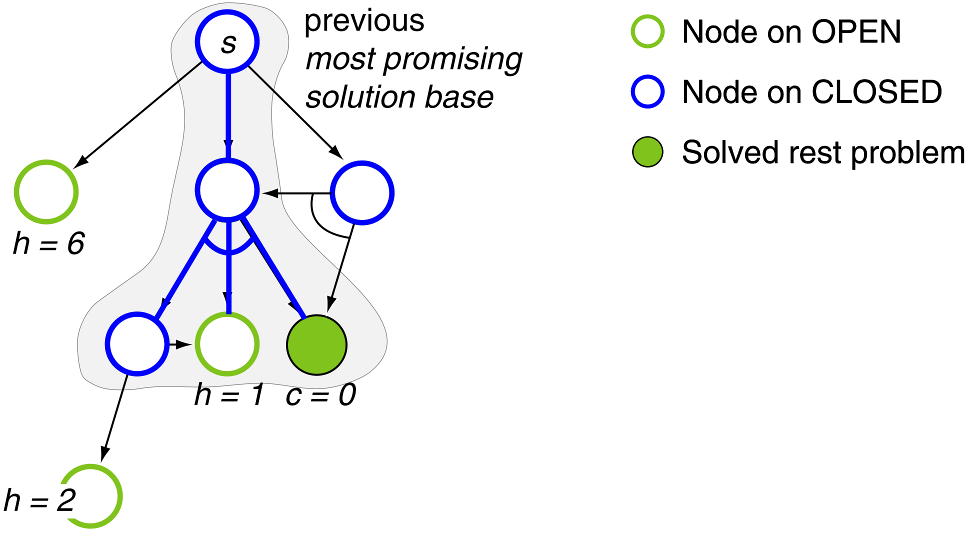
$f_1(H) = 9$

Most promising solution base H

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

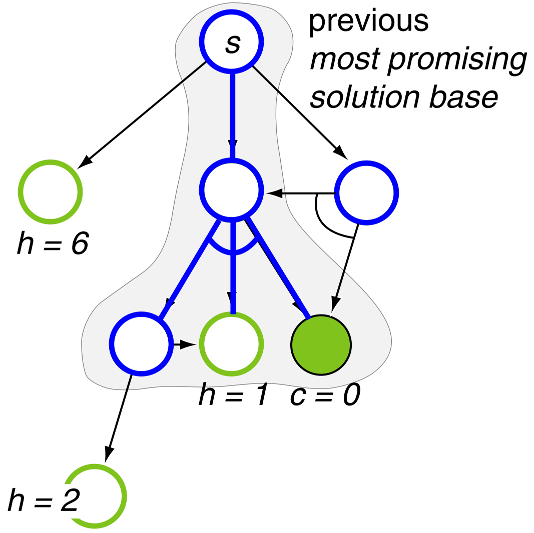
4. Based on $f_2(n)$: expansion of some node $n \in (\text{OPEN} \cap H)$.



Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

4. Based on $f_2(n)$: expansion of some node $n \in (\text{OPEN} \cap H)$.



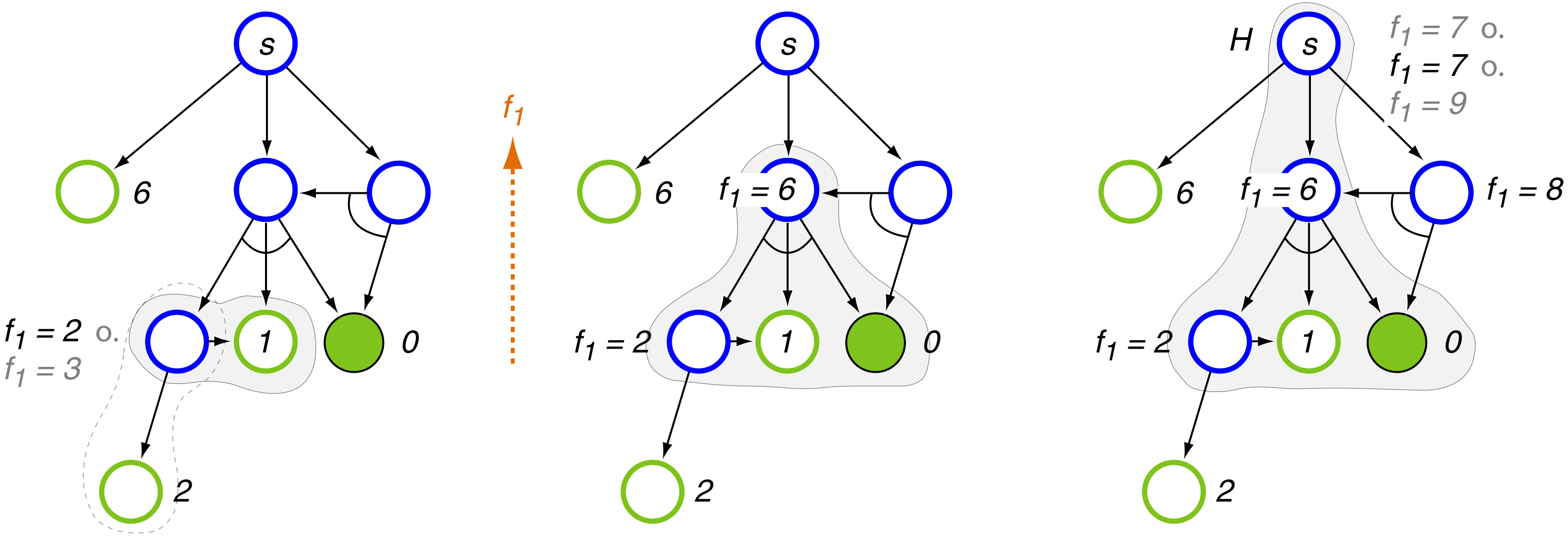
previous *most promising solution base*

○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

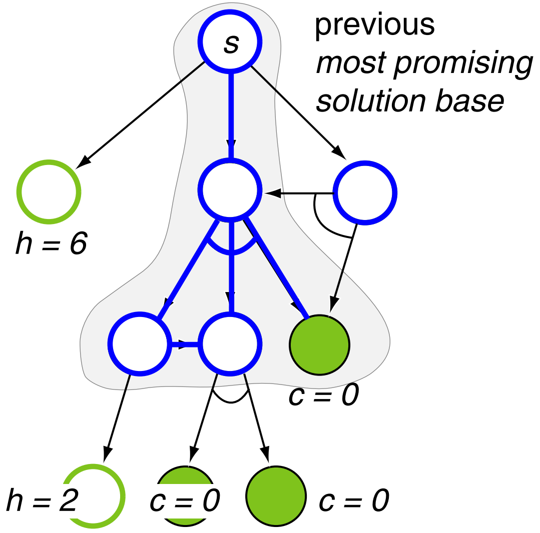
Solution bases:



Best-First Search for AND-OR Graphs

Illustration of GBF (continued) [next step]

5. Expansion of the node $n \in (\text{OPEN} \cap H)$.



○ Node on OPEN

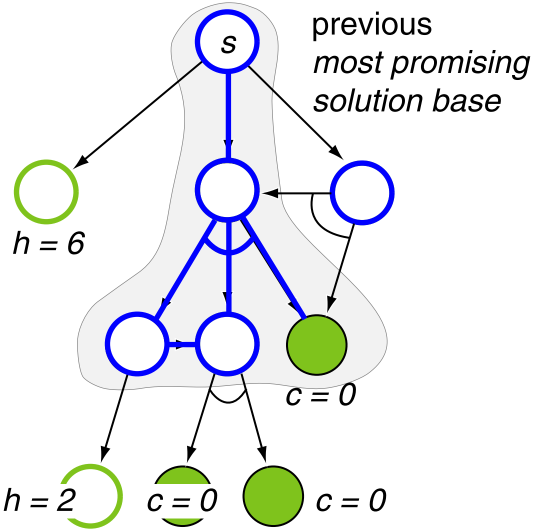
○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued) [next step]

5. Expansion of the node $n \in (\text{OPEN} \cap H)$.

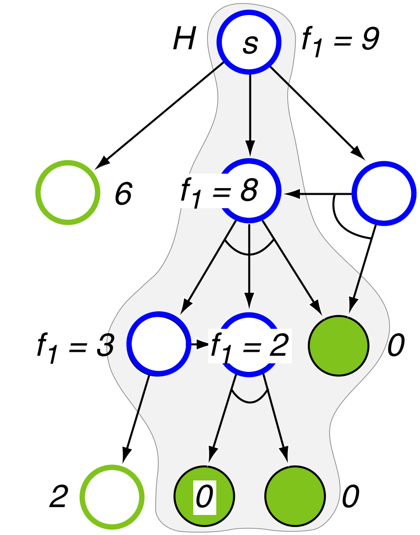


○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

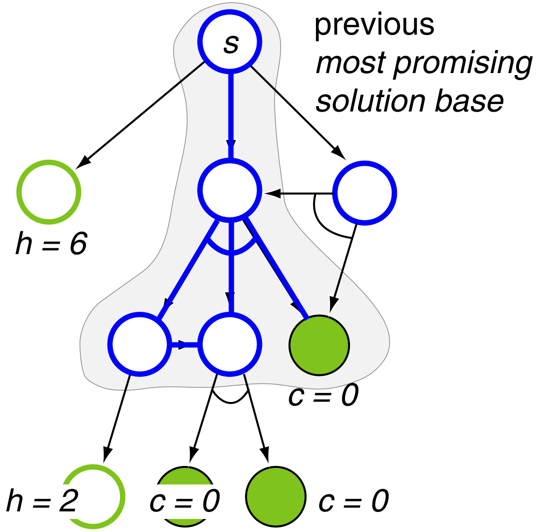
A solution graph H:



Best-First Search for AND-OR Graphs

Illustration of GBF (continued) [next step]

5. Expansion of the node $n \in (\text{OPEN} \cap H)$.

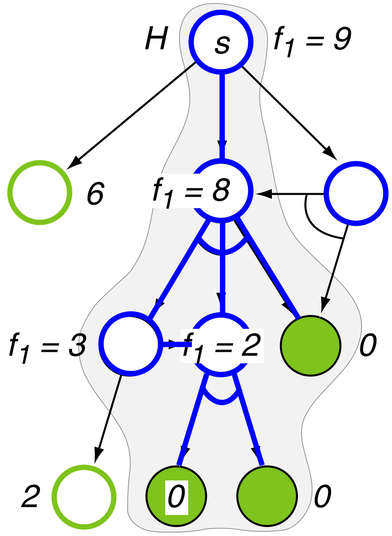


○ Node on OPEN

○ Node on CLOSED

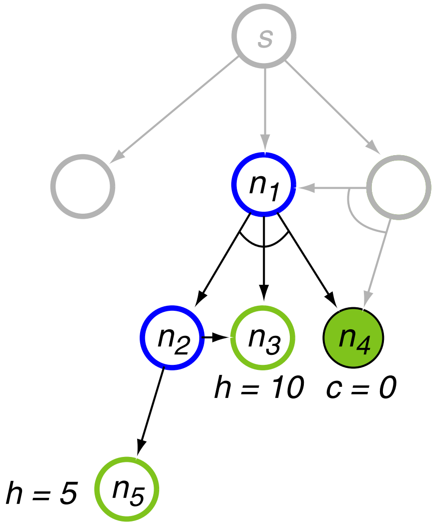
● Solved rest problem

A solution graph H:



Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



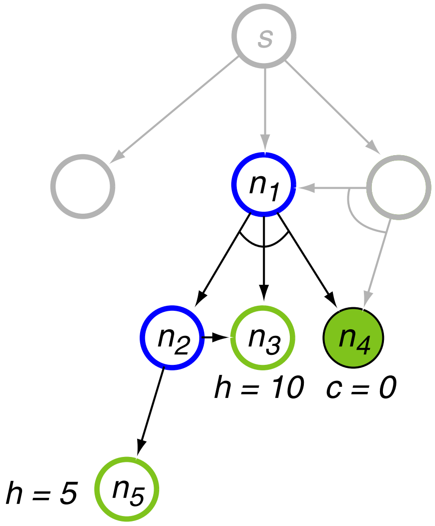
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



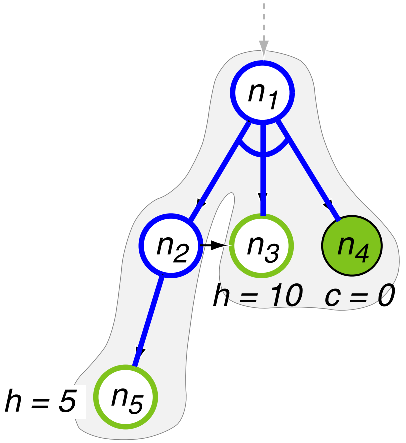
○ Node on OPEN

○ Node on CLOSED

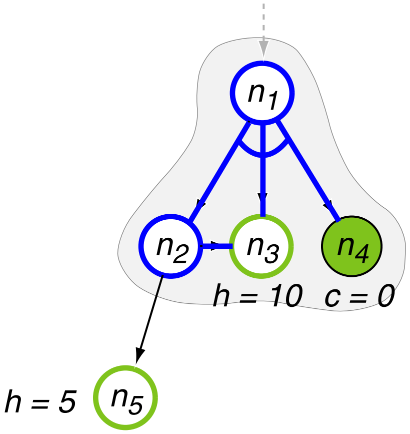
● Solved rest problem

n_1 start node

(a)



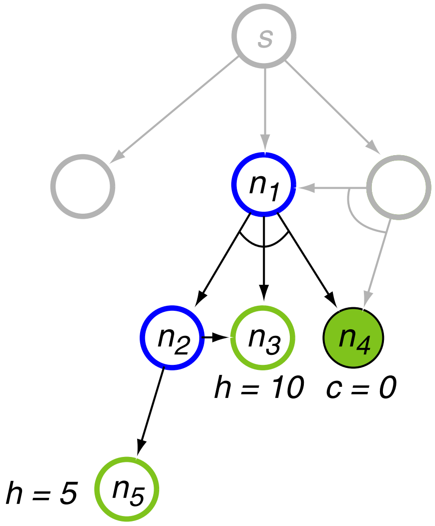
(b)



$C_G(n_1)$	(a)	(b)
duplicate count		
true count		

Best-First Search for AND-OR Graphs

Multiple Accounting of a Node

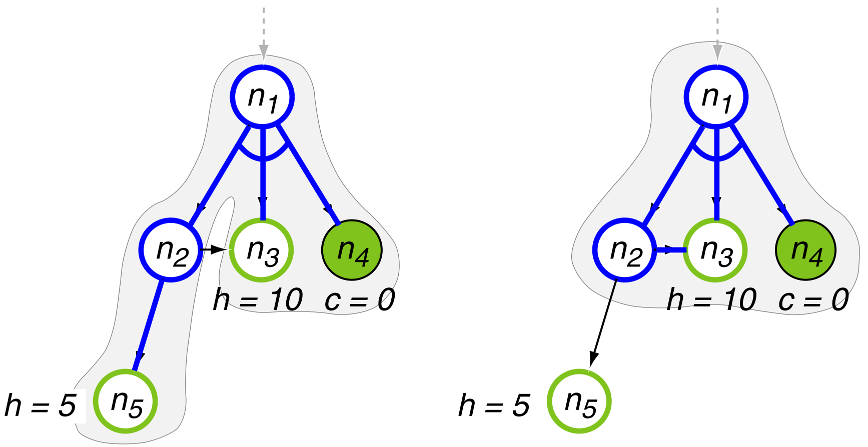


○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

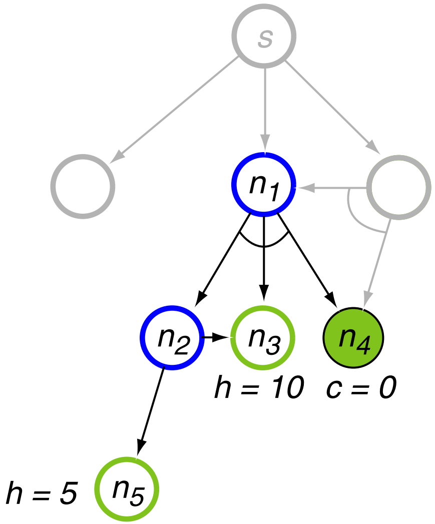
n_1 start node (a) (b)



$C_G(n_1)$	(a)	(b)
duplicate count	**19**	24
true count		

Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



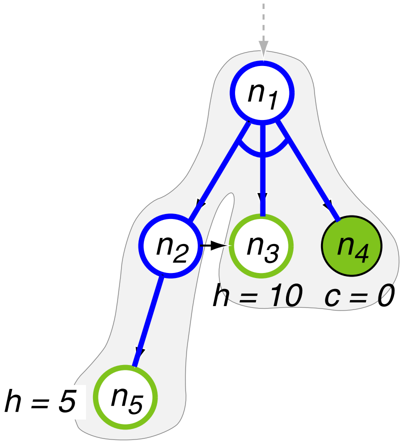
○ Node on OPEN

○ Node on CLOSED

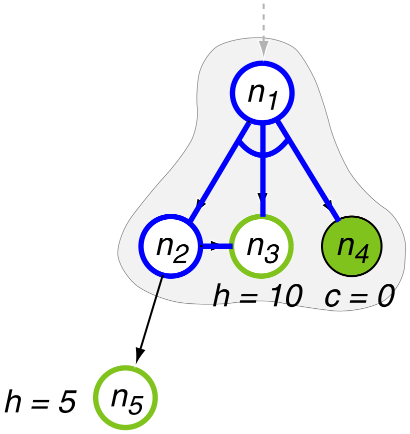
● Solved rest problem

n_1 start node

(a)



(b)



$C_G(n_1)$	(a)	(b)
duplicate count	**19**	24
true count	19	**14**

Remarks:

- The minimum cost contribution of node n_2 to a solution is not constant, depending on the multiple accounting of node n_3.
- Q. Which approach is to be preferred—simple computation of f_1 ($\sim$ multiple accounting) or true accounting?

Best-First Search for AND-OR Graphs

Optimality of Algorithm GBF [BF optimality]

In general, the first solution found by Algorithm GBF may not be optimum with respect to the evaluation function f_1.

Important preconditions for (provably) finding optimum solution graphs in AND-OR-graphs by general best-first algorithms:

1. The cost estimate underlying f_1 must be optimistic, i.e., underestimating costs or overestimating merits.

   In particular, the true cost of a cheapest solution graph H extending a solution base H exceeds its f_1-value: $C_H(s) \geq f_1(H)$.

2. The termination in case of success (*is_solved*$(s) =$ *True*) must be delayed.

   In particular, there is no call to *solved_labeling* after node expansion, but each time when determining a most promising solution base H.

   Algorithm GBF with delayed termination is called Algorithm GBF*.

Best-First Search for AND-OR Graphs

Optimality of Algorithm GBF [BF optimality]

In general, the first solution found by Algorithm GBF may not be optimum with respect to the evaluation function f_1.

Important preconditions for (provably) finding optimum solution graphs in AND-OR-graphs by general best-first algorithms:

1. The cost estimate underlying f_1 must be optimistic, i.e., underestimating costs or overestimating merits.

   In particular, the true cost of a cheapest solution graph H extending a solution base H exceeds its f_1-value: $C_H(s) \geq f_1(H)$.

2. The termination in case of success (*is_solved*$(s) =$ *True*) must be delayed.

   In particular, there is no call to *solved_labeling* after node expansion, but each time when determining a most promising solution base H.

   Algorithm GBF with delayed termination is called Algorithm GBF*.

Best-First Search for AND-OR Graphs [BF, BF$\star$, GBF]

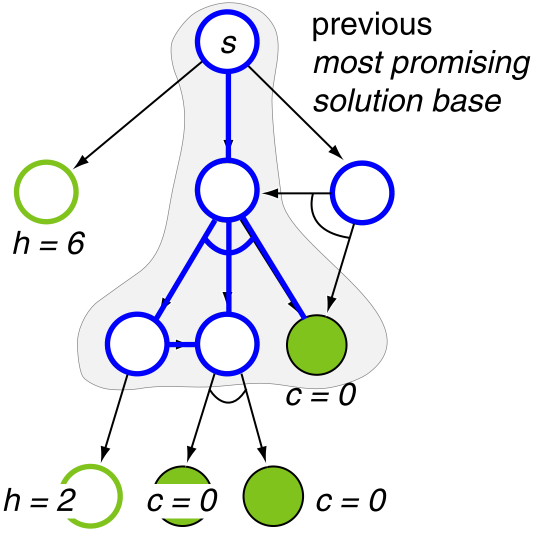
GBF*(s, *successors*, *is_solved*, f_1, f_2)

```
 1.  insert($s$, OPEN);  add_node($s$, $G_e$);    // $G_e$ is the explored part of $G$.
 2.  LOOP
 3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
4.a    $H$ = min_solution_base($s$, $G_e$, $f_1$);    // Find most prom. sol. base in $G_e$.
 ➜     solved_labeling($H$);    // Is $H$ a solution graph?
 ➜     IF is_solved($s$) THEN RETURN($H$);    // Delayed termination.
4.b    $n$ = min(OPEN $\cap$ $H$, $f_2$);    // Find most informative non-goal node in $H$.
       remove($n$, OPEN);  push($n$, CLOSED);
 5.    FOREACH $n'$ IN successors($n$) DO
         IF ( $n' \in$ OPEN OR $n' \in$ CLOSED )    // Instance of $n'$ seen before?
         THEN   // Use old instance of $n'$ instead.
           $n'$ = retrieve($n'$, OPEN $\cup$ CLOSED);
         ELSE   // $n'$ encodes an instance of a new state.
           insert($n'$, OPEN);  add_node($n'$, $G_e$);
         ENDIF
         add_backpointer($n'$, $n$);  add_edge(($n$, $n'$), $G_e$);

 ➜       ~~IF is_solved($n'$)~~
 ➜       ~~THEN~~
 ➜         ~~propagate_label($n'$); IF is_solved($s$) THEN RETURN(compute_solution_graph($G_e$));~~
 ➜       ~~ENDIF~~

       ENDDO
 6.  ENDLOOP
```

Best-First Search for AND-OR Graphs

Illustration of GBF* (continued) [previous step]

5. Expansion of the node $n \in (\text{OPEN} \cap H)$.



○ Node on OPEN

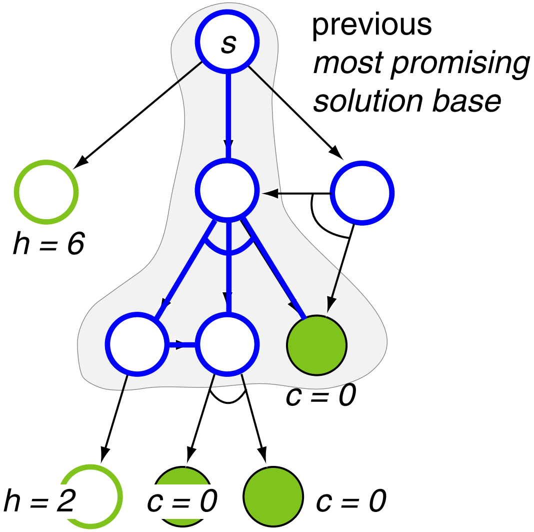
○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF* (continued) [previous step]

5. Expansion of the node $n \in (\text{OPEN} \cap H)$.

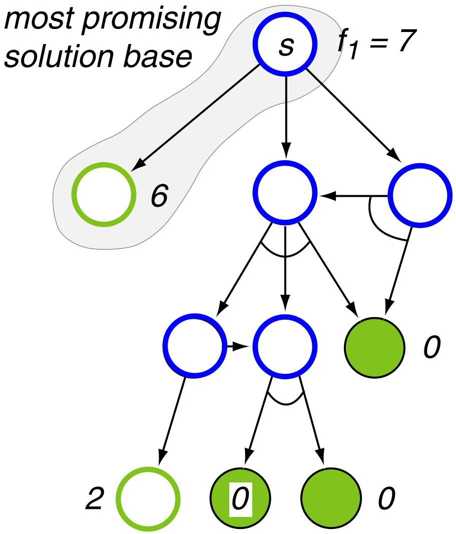


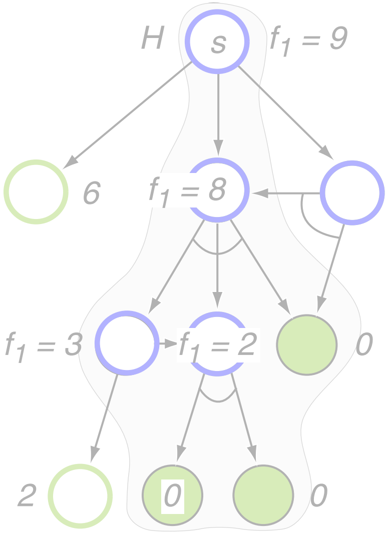
○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Solution base / graph:

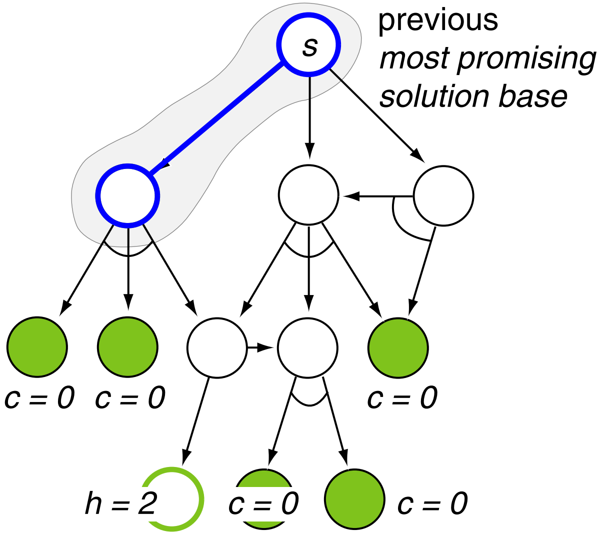




Best-First Search for AND-OR Graphs

Illustration of GBF* (continued)

6. Expansion of the node $n \in (\text{OPEN} \cap H)$.



○ Node on OPEN

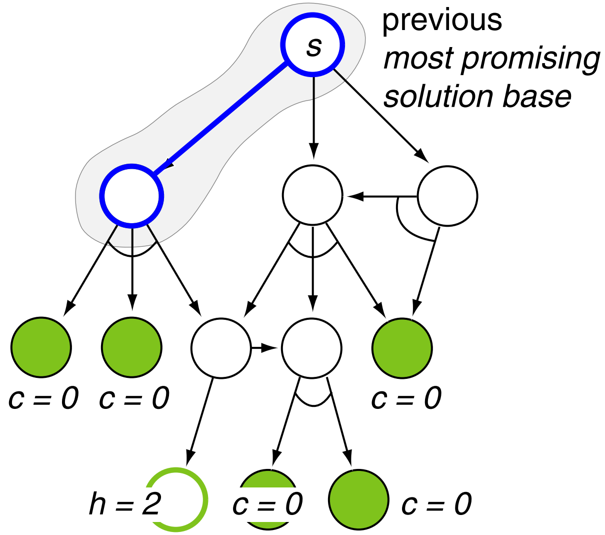
○ Node on CLOSED

● Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF* (continued)

6. Expansion of the node $n \in (\text{OPEN} \cap H)$.



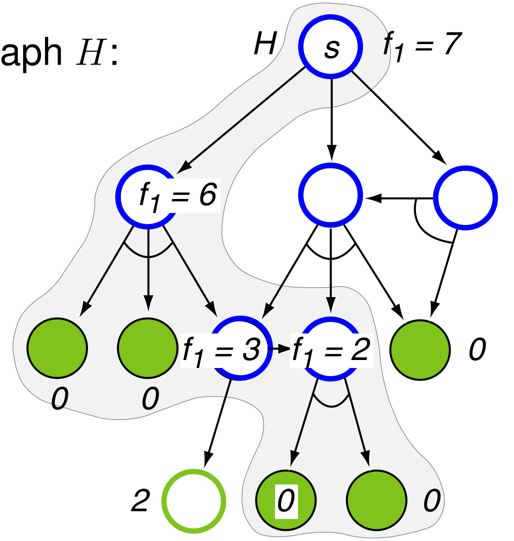
previous *most promising solution base*

○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

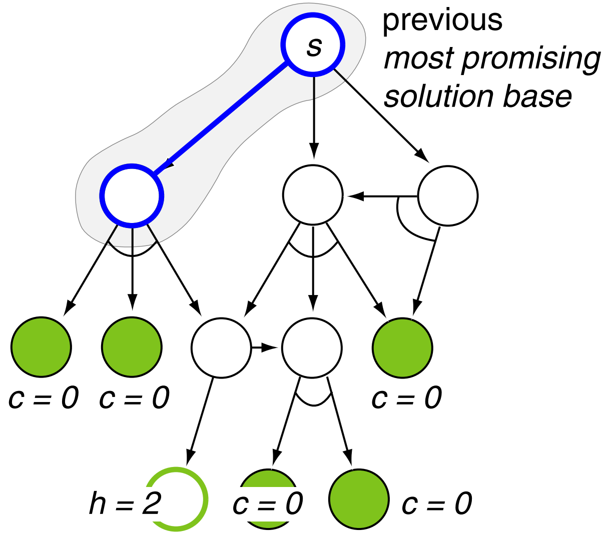
Returned solution graph H:



Best-First Search for AND-OR Graphs

Illustration of GBF* (continued)

6. Expansion of the node $n \in$ (OPEN $\cap$ H).

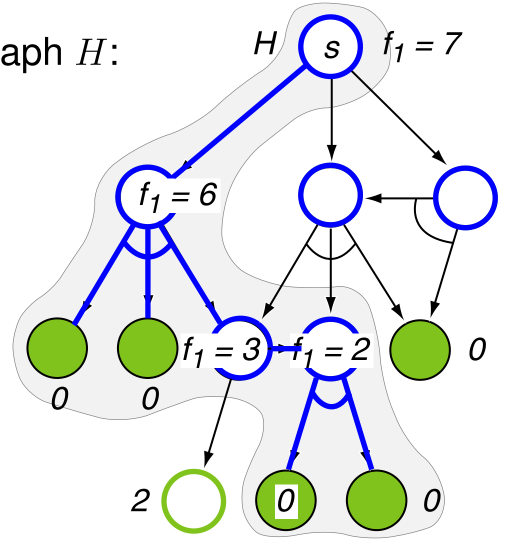


○ Node on OPEN

○ Node on CLOSED

● Solved rest problem

Returned solution graph H:



Relation between GBF and BF

Algorithm GBF applied to a state-space graph will simulate Algorithm BF.
(if the evaluation function f in BF is order preserving) [S:III Evaluation of State-Space Graphs]

Relation between GBF and BF

Algorithm GBF applied to a state-space graph will simulate Algorithm BF. (if the evaluation function f in BF is order preserving) [S:III Evaluation of State-Space Graphs]

The key difference between GBF and BF:

GBF Solution bases are graphs. (clear due to nature of the problem)

BF Solution bases are paths. (clear due to nature of the problem)

Relation between GBF and BF

Algorithm GBF applied to a state-space graph will simulate Algorithm BF.
(if the evaluation function f in BF is order preserving) [S:III Evaluation of State-Space Graphs]

The key difference between GBF and BF:

GBF Solution bases are graphs. (clear due to nature of the problem)

GBF The most promising solution base is searched among all possible solution bases in G_e, the explored part of the search space graph.

BF Solution bases are paths. (clear due to nature of the problem)

BF At each point in time the union of the considered solution bases forms a tree with root s. The tree structure is maintained by always discarding the inferior path of two paths leading to the same node. The most promising solution base is searched among all paths in this tree.

Relation between GBF and BF

Algorithm GBF applied to a state-space graph will simulate Algorithm BF.
(if the evaluation function f in BF is order preserving) [S:III Evaluation of State-Space Graphs]

The key difference between GBF and BF:

GBF Solution bases are graphs. (clear due to nature of the problem)

GBF The most promising solution base is searched among all possible solution bases in G_e, the explored part of the search space graph.

BF Solution bases are paths. (clear due to nature of the problem)

BF At each point in time the union of the considered solution bases forms a tree with root s. The tree structure is maintained by always discarding the inferior path of two paths leading to the same node. The most promising solution base is searched among all paths in this tree.

Relation between GBF and BF

Algorithm GBF applied to a state-space graph will simulate Algorithm BF.
(if the evaluation function f in BF is order preserving) [S:III Evaluation of State-Space Graphs]

The key difference between GBF and BF:

GBF Solution bases are graphs. (clear due to nature of the problem)

GBF The most promising solution base is searched among all possible solution bases in G_e, the explored part of the search space graph.

BF Solution bases are paths. (clear due to nature of the problem)

BF At each point in time the union of the considered solution bases forms a tree with root s. The tree structure is maintained by always discarding the inferior path of two paths leading to the same node. The most promising solution base is searched among all paths in this tree.

BF The discarding of a path (solution base) is *irrevocable*.

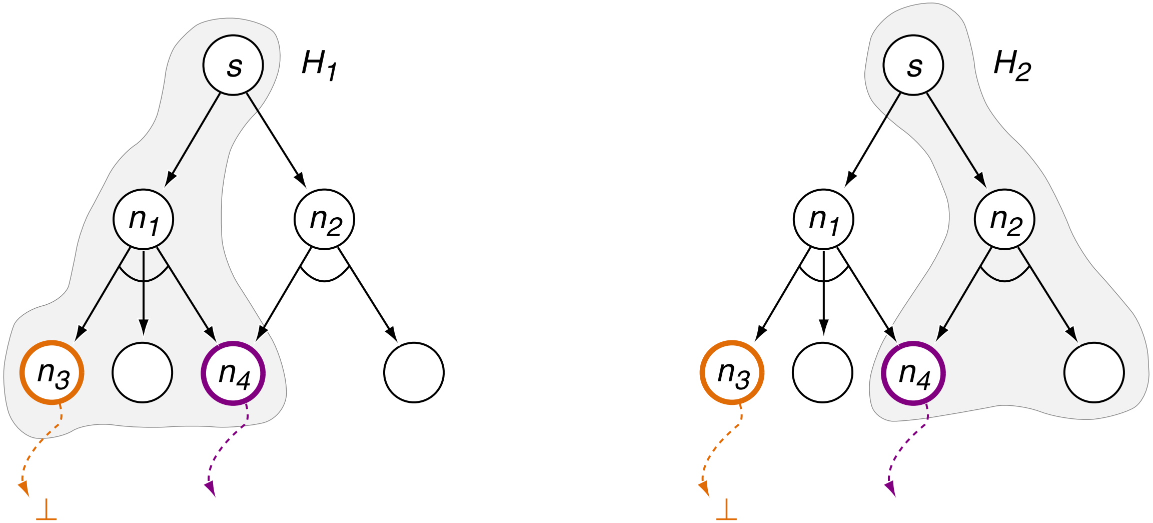
Remarks:

- (Under BF) the tree formed by the union of all solution bases is also called *traversal tree*. It is defined by the back-pointers that are set by the Algorithm BF.
- (Under BF) the decision to discard an inferior path (solution base) is based on a single parameter, usually a cost value, which is maintained at the respective OPEN node.
- Under BF the cost value assigned to a node is unique. Under GBF, however, a node can be used in different solution bases and may get assigned different cost values for different solution bases at the same time.
- Under BF there is a 1:1 relationship between expansion candidates (the nodes on OPEN) and solution bases (the paths that start at s and end at some node on OPEN). Put another way, each node on OPEN represents a solution base, and there is no solution base without a corresponding OPEN node.
- (Under BF) the term “irrevocable” means that an inferior path from s so some n will never become part of solution that also contains the node n. Irrevocability has wide-ranging consequences, which may be both crucial and reasonable, depending on the search problem at hand.

Relation between GBF and BF

Irrevocability is not Amenable to Problem Reduction Graphs

[Irrevocability in State-Space Graphs]



The following equivalence does not generally hold for problem reduction graphs:

$$
\begin{array}{rl}
 & \text{“Solution base } H_1 \text{ can be completed to a solution”} \\
\Leftrightarrow & \\
 & \text{“Solution base } H_2 \text{ can be completed to a solution.”}
\end{array}
$$

Remarks:

- The illustrated solution bases H_1, H_2, do not fulfill the equivalence. They share the continuation at node n_4, and let us assume that H_2 is inferior to H_1: $f_1(H_1) < f_1(H_2)$. However, H_2 cannot be discarded since at a later point in time the problem associated with node n_3 may turn out to be unsolvable. If we discarded H_2, GBF would miss the only possible solution.