Chapter S:IV

IV. Informed Search

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

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

- Hybrid Strategies

Best-First Search

"To enhance the performance of AI's programs, knowledge [about the problem domain, which enables us to guide search into promising directions] is the power."

[Feigenbaum 1980]

Best-First Search

"To enhance the performance of AI's programs, knowledge [about the problem domain, which enables us to guide search into promising directions] *is the power."*

[Feigenbaum 1980]

Best-First Search

> *"To enhance the performance of AI's programs, knowledge* [about the problem domain, which enables us to guide search into promising directions] *is the power."*
>
> [Feigenbaum 1980]

Examples for heuristic functions [S:I Examples for Search Problems]:

- 8-queens problem. Maximize h_1, the number of unattacked cells.
- 8-puzzle problem. Minimize h_1, the number of non-matching tiles.

Knowledge on how to achieve this (Maximize..., Minimize...) is beyond that which is built into the state and operator definitions.

Best-First Search

> *"To enhance the performance of AI's programs, knowledge* [about the problem domain, which enables us to guide search into promising directions] *is the power."*
>
> [Feigenbaum 1980]

Examples for heuristic functions [S:I Examples for Search Problems]:

- 8-queens problem. Maximize h_1, the number of unattacked cells.
- 8-puzzle problem. Minimize h_1, the number of non-matching tiles.

Knowledge on how to achieve this (Maximize..., Minimize...) is beyond that which is built into the state and operator definitions.

Where is heuristic knowledge employed in the formalism of systematic search?

- Hill-climbing. Move into the direction of a most promising successor n' of the current node.
- Best-First Search. Move into the direction of a most promising node n, where n is chosen among all nodes encountered so far.

Best-First Search

"The promise of a node is estimated numerically by a heuristic evaluation function $f(n)$ which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

[Pearl 1984]

Best-First Search

> *"The promise of a node is estimated numerically by a heuristic evaluation function $f(n)$ which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."*
>
> [Pearl 1984]

The evaluation function f may depend on

1. estimates of the amount of information gained by expanding n,
2. estimates of the complexity of the remaining problem at n in relation to Γ,
3. evaluations of the explored part G of the search space graph,
4. domain specific problem solving knowledge K.

$$f = f(n, \Gamma, G, K)$$

Objective is to quantify for a generated, but yet unexpanded node n its potential of guiding the search into a desired direction.

Best-First Search

Schema for Best-First Algorithms

1. Initialize a set of solution bases.
2. Loop.
   (a) Select a most promising solution base using an evaluation function f (state-space graph) or f_1 (problem-reduction graphs).
   (b) Select a most informative node in that solution base using an evaluation function f_2 (problem-reduction graphs).
   (c) Expand the respective node in the solution base and process the newly generated solution bases (add / discard / substitute).

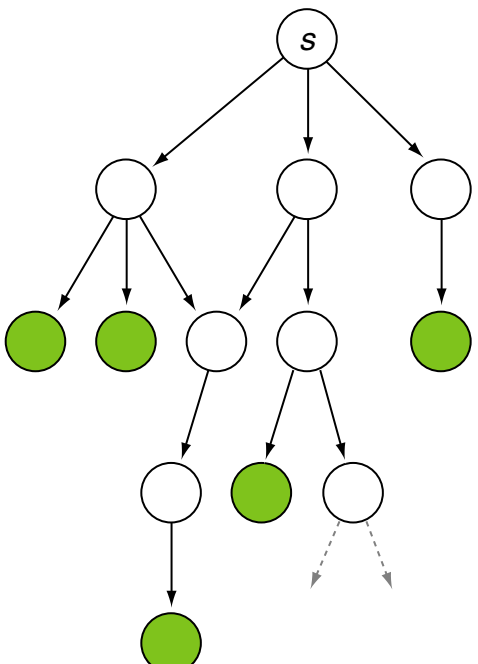
Remarks:

- The schema is further extended by termination tests for failure and success.
- The job of the evaluation function f is to make two solution bases comparable and hence to provide an order on them.
- Best-first strategies differ in the evaluation functions they use. Placing restrictions on the computation of these functions will establish a taxonomy of best-first algorithms.

Best-First Search for State-Space Graphs

Illustration of Solution Paths and Solution Bases

State-space graph:

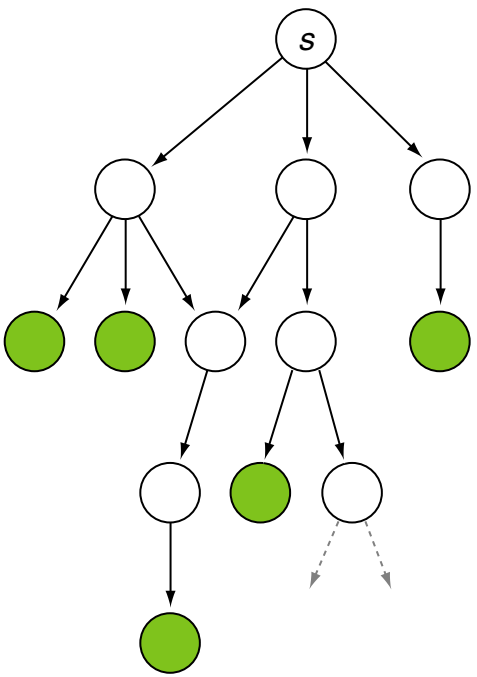


Solved rest problem

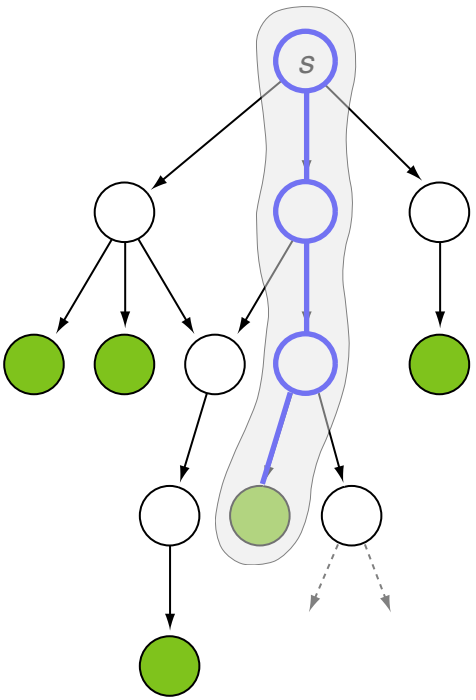
Best-First Search for State-Space Graphs

Illustration of Solution Paths and Solution Bases

State-space graph:



Solution path for node *s*:

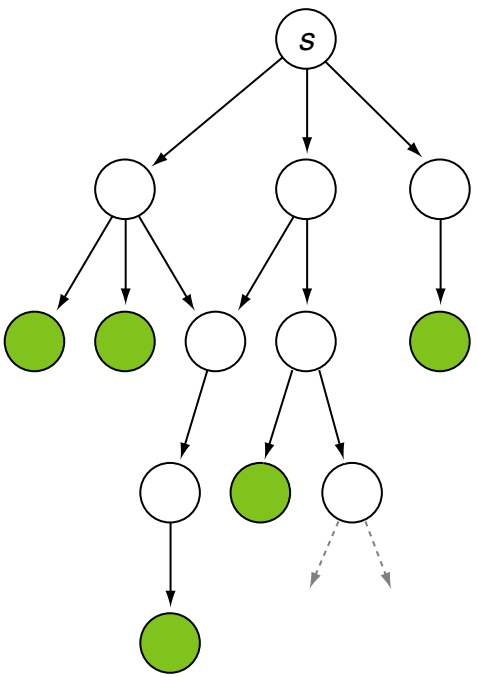


Solved rest problem

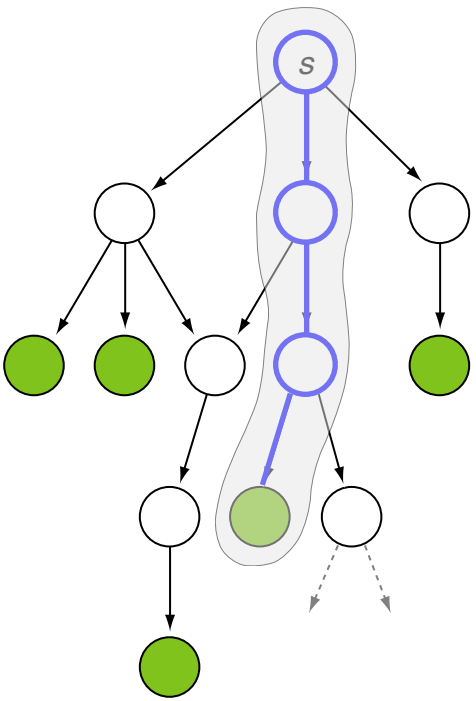
Best-First Search for State-Space Graphs

Illustration of Solution Paths and Solution Bases

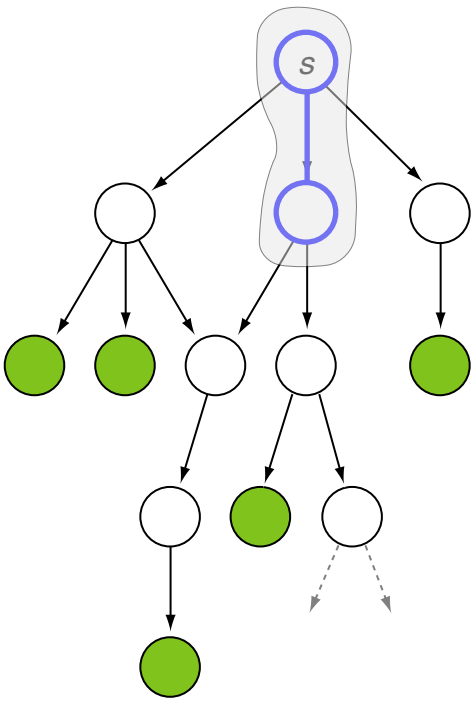
State-space graph:



Solution path for node *s*:



Solution base for *s*:



Solved rest problem

Best-First Search for State-Space Graphs

Definition 11 (Solution Base in a State-Space Graph)

Let G be an explored subgraph of a state-space graph with root node s, let n_1, n_2 be nodes in G, and let n_2 be a terminal node in G. Then a path P from n_1 to n_2 in G is called *solution base for* n_1 *in* G.

Compare:

- Solution base in a problem-reduction graph.
- Solution path in a state-space graph. [S:II State-Space Representation]

Usage of P :

- Usually, we are interested in finding a solution base P for the root node s in G.

Remarks:

- Obviously, all solution paths contained in G are also solution bases.
- If P is a solution base for a node n in G and if n' is some node in P, then the subpath P' of P starting in n' and ending in the endnode of P is a solution base for a node n' in G. This subpath P' is sometimes called the *solution base in P induced by n'*.

Best-First Search for State-Space Graphs

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. f evaluates solution bases P_{s-n},
2. a most promising solution base minimizes f,
3. *only one solution base* P_{s-n} is maintained for each terminal node n,
4. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
5. especially, if a node is reached a second time because of a cycle, the f-value of the cyclefree path (the first path found) must not be higher the f-value for the cyclic path.

Best-First Search for State-Space Graphs

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. f evaluates solution bases P_{s-n},
2. a most promising solution base minimizes f,
3. *only one solution base* P_{s-n} is maintained for each terminal node n,
4. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
5. especially, if a node is reached a second time because of a cycle, the f-value of the cyclefree path (the first path found) must not be higher the f-value for the cyclic path.

Best-First Search for State-Space Graphs

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. f evaluates solution bases P_{s-n},
2. a most promising solution base minimizes f,
3. *only one solution base* P_{s-n} is maintained for each terminal node n,
4. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
5. especially, if a node is reached a second time because of a cycle, the f-value of the cyclefree path (the first path found) must not be higher the f-value for the cyclic path.

Best-First Search for State-Space Graphs

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. f evaluates solution bases P_{s-n},
2. a most promising solution base minimizes f,
3. *only one solution base* P_{s-n} is maintained for each terminal node n,
4. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
5. especially, if a node is reached a second time because of a cycle, the f-value of the cyclefree path (the first path found) must not be higher the f-value for the cyclic path.

Best-First Search for State-Space Graphs

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. f evaluates solution bases P_{s-n},
2. a most promising solution base minimizes f,
3. *only one solution base* P_{s-n} is maintained for each terminal node n,
4. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
5. especially, if a node is reached a second time because of a cycle, the f-value of the cyclefree path (the first path found) must not be higher the f-value for the cyclic path.

Best-First Search for State-Space Graphs

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. f evaluates solution bases P_{s-n},
2. a most promising solution base minimizes f,
3. *only one solution base* P_{s-n} is maintained for each terminal node n,
4. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
5. especially, if a node is reached a second time because of a cycle, the f-value of the cyclefree path (the first path found) must not be higher the f-value for the cyclic path.

Remarks:

- G denotes the explored part of the search space graph.
- When searching state-space graphs with the algorithm BF, only solution bases P_{s-n} need to be considered that have not been considered so far. I.e., only solution bases P_{s-n} with $n \in$ `OPEN` are considered.
- If a dead end recognition $\perp(n)$ is available, no solution base will be considered that contains a node labeled "unsolvable" using $\perp(n)$.
- Since only one solution base P_{s-n} is maintained for each terminal node n, this node is a representative of the respective solution base. This solution base can be recovered following the backpointers starting in n.
- Usually, the evaluation function $f(n)$ is based on a heuristic $h(n)$.
- $h(n)$ estimates the optimum cost 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.
- Path discarding entails the risk of not finding desired solutions. The risk can be eliminated by restricting to evaluation functions f that fulfill particular properties. Keyword: *Order preserving property* [S:IV Specialized Cost Measures]
- When equipped with both path discarding and backpointer installation, search algorithms for state-space graphs can be considered as algorithms that maintain for each node $n \in G$ a path from s to n.
- If cyclic paths have lesser f-values than corresponding cyclefree paths, the backpointer structure will be corrupted when a cycle is found.

Best-First Search for State-Space Graphs

Algorithm: BF

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.
$f(n)$. Evaluation function for solution bases (= nodes on OPEN).

Output: A goal node or the symbol *Fail*.

Best-First Search for State-Space Graphs

BF$(s, \mathit{successors}, \star, f)$

1. *insert*$(s,$ OPEN$)$;
2. **LOOP**
3. IF $($OPEN $= \emptyset)$ THEN RETURN$($*Fail*$)$;
4.
5.
6. **ENDLOOP**

Best-First Search for State-Space Graphs

BF$(s,$ *successors*$,\star,f)$

```
1.  insert(s, OPEN);
2.  LOOP
3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.    n = min(OPEN, f);    // Find most promising solution base.
      remove(n, OPEN); push(n, CLOSED);
5.
6.  ENDLOOP
```

Best-First Search for State-Space Graphs

BF(s, *successors*, $\star$, f)

```
1.  insert(s, OPEN);
2.  LOOP
3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.    n = min(OPEN, f);    // Find most promising solution base.
      remove(n, OPEN); push(n, CLOSED);
5.    FOREACH n' IN successors(n) DO
        IF ⋆(n') THEN add_backpointer(n', n); RETURN(n'); ENDIF


          add_backpointer(n', n);
          insert(n', OPEN);









      ENDDO
6.  ENDLOOP
```

Best-First Search for State-Space Graphs

BF$(s, \mathit{successors}, \star, f)$

```
1.  insert(s, OPEN);
2.  LOOP
3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.    n = min(OPEN, f);    // Find most promising solution base.
      remove(n, OPEN); push(n, CLOSED);
5.    FOREACH n' IN successors(n) DO    // Expand n.
        IF ⋆(n') THEN add_backpointer(n', n); RETURN(n'); ENDIF
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        THEN   // n' encodes a new state.
          add_backpointer(n', n);
          insert(n', OPEN);
        ELSE   // n' encodes an already visited state.
        ENDIF
      ENDDO
6.  ENDLOOP
```

Best-First Search for State-Space Graphs [BF⋆, GBF, GBF⋆]

BF$(s, \mathit{successors}, \star, f)$

```
1.  insert(s, OPEN);
2.  LOOP
3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.    n = min(OPEN, f);    // Find most promising solution base.
      remove(n, OPEN); push(n, CLOSED);
5.    FOREACH n' IN successors(n) DO    // Expand n.
        IF ⋆(n') THEN add_backpointer(n', n); RETURN(n'); ENDIF
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        THEN   // n' encodes a new state.
          add_backpointer(n', n);
          insert(n', OPEN);
        ELSE   // n' encodes an already visited state.
          IF (f_n(n') < f(n'))   // f_n(n'): compute f for n' via n.
          THEN   // The state of n' is reached via a cheaper path.
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
        ENDIF
      ENDDO
6.  ENDLOOP
```

Remarks:

- Algorithm BF maintains a finite subgraph G of the search space graph. This graph G can be seen as implicitly defined via the backpointers.
- The function *insert*$(n, \texttt{OPEN})$ stores a node n according to $f(n)$ in the underlying data structure of the OPEN list. Given a sorted tree (a heap), a node with the minimum f-value is found in logarithmic (constant) time.

Best-First Search for State-Space Graphs

Path Discarding for a Node n'

```
5.   FOREACH n' IN successors(n)
       ...
       IF (n' ∉ OPEN AND n' ∉ CLOSED)
       ...
         IF (f_n(n') < f(n'))   // f_n(n'): compute f for n' via n.
         THEN
           update_backpointer(n', n);
           IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
         ENDIF
```

- Backpointers maintained by Algorithm BF define unique paths, the pointer-paths, from s to any node in OPEN and CLOSED in the explored subgraph of the search space graph. The union of these paths is called traversal tree.
- $f(n')$ is computed using the currently stored pointer-path from s to n'.
- $f_n(n')$ is computed using edge (n, n') and the currently stored pointer-path from s to n.
- Path discarding is performed implicitly by maintaining at most one backpointer per node.
- Algorithm BF cannot recover paths that were discarded, i.e., path discarding is irrevocable.

Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

Case 1: n' is still on OPEN.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

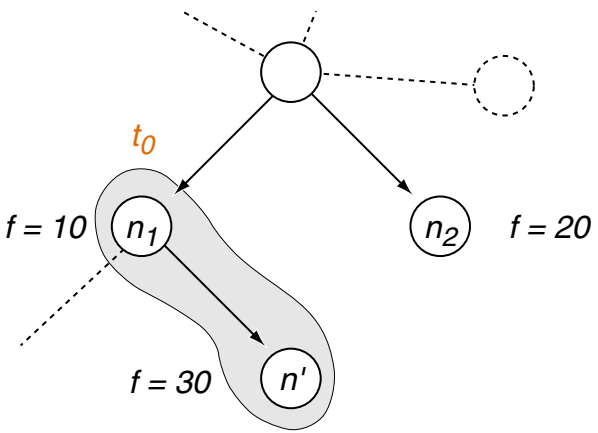
Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

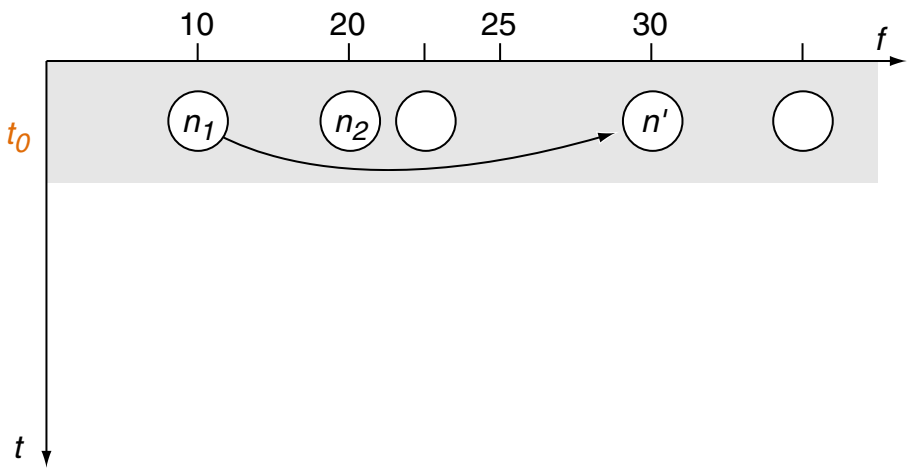
Case 1: n' is still on OPEN.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

State-space:



OPEN list:



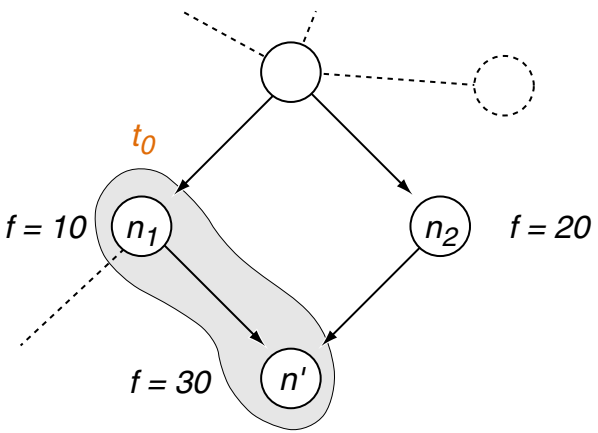
Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

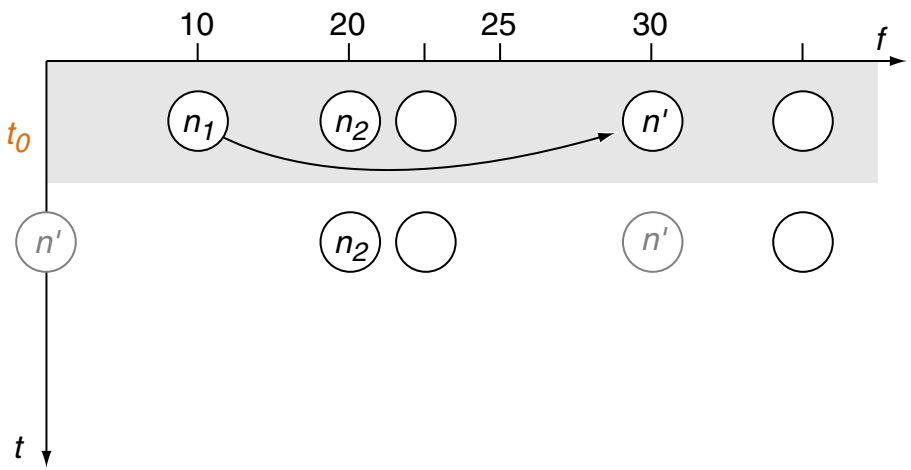
Case 1: n' is still on OPEN.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

State-space:



OPEN list:



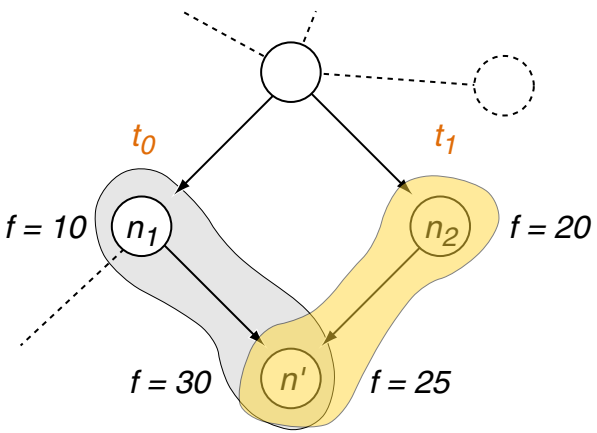
Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

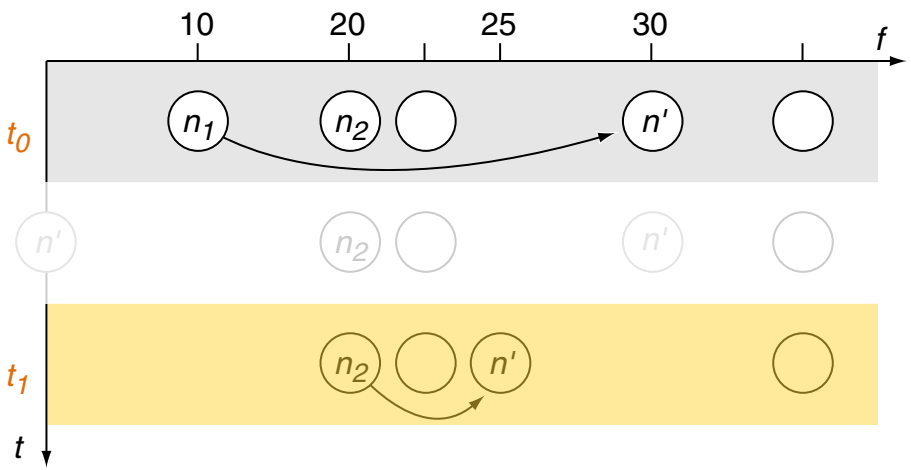
Case 1: n' is still on OPEN.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

State-space:



OPEN list:



Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

Case 2: n' is already on CLOSED.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

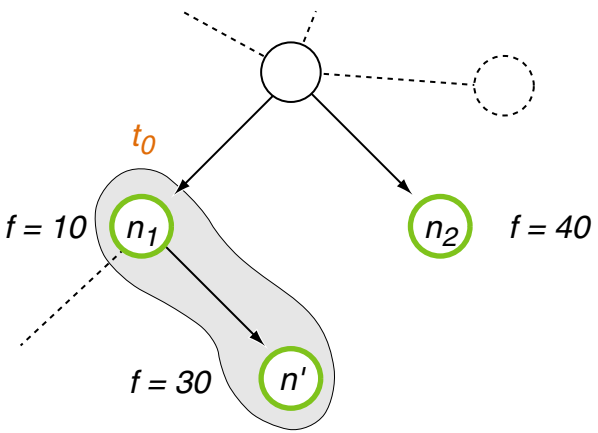
Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

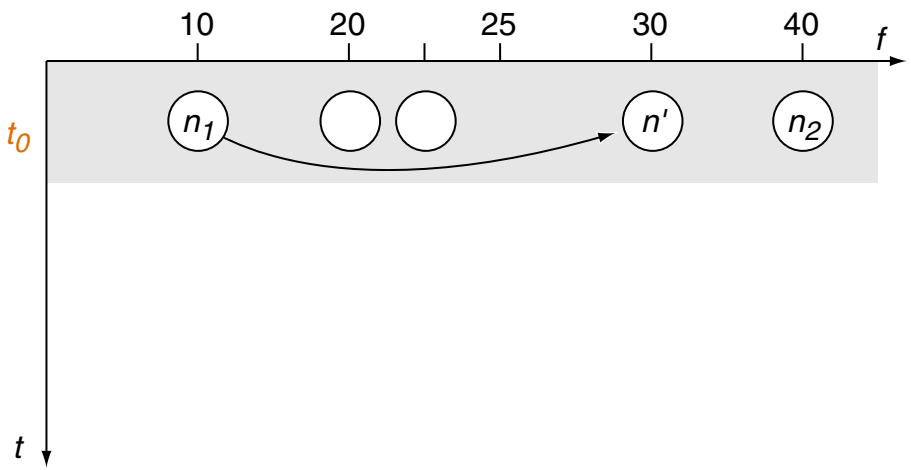
Case 2: n' is already on CLOSED.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

State-space:



OPEN list:



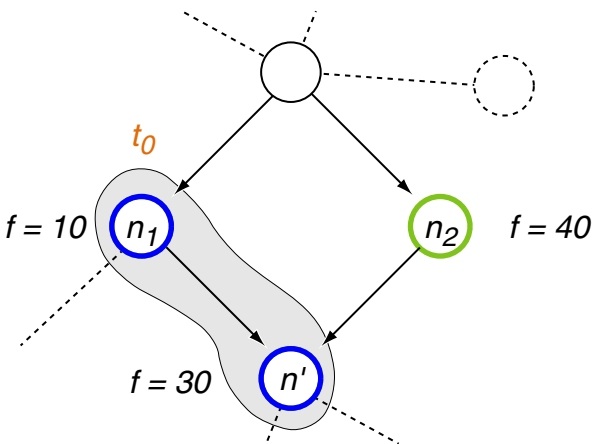
Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

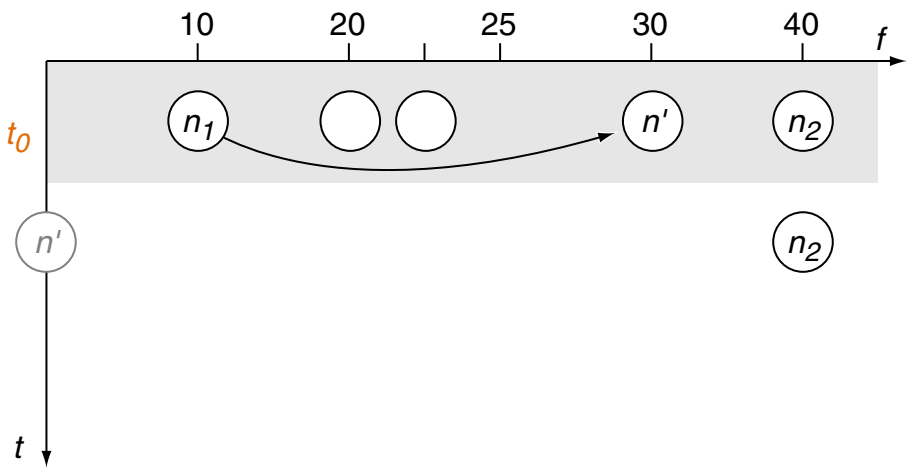
Case 2: n' is already on CLOSED.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

State-space:



OPEN list:



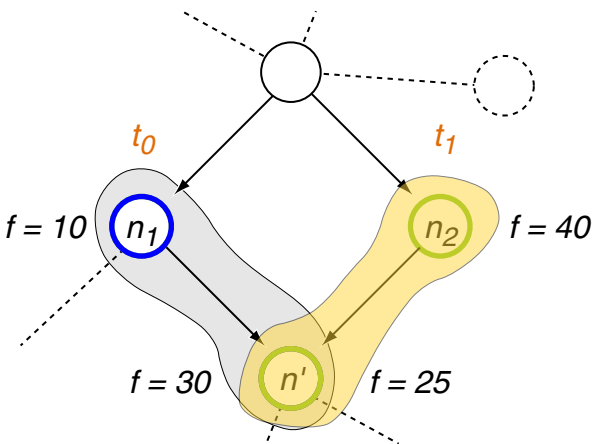
Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

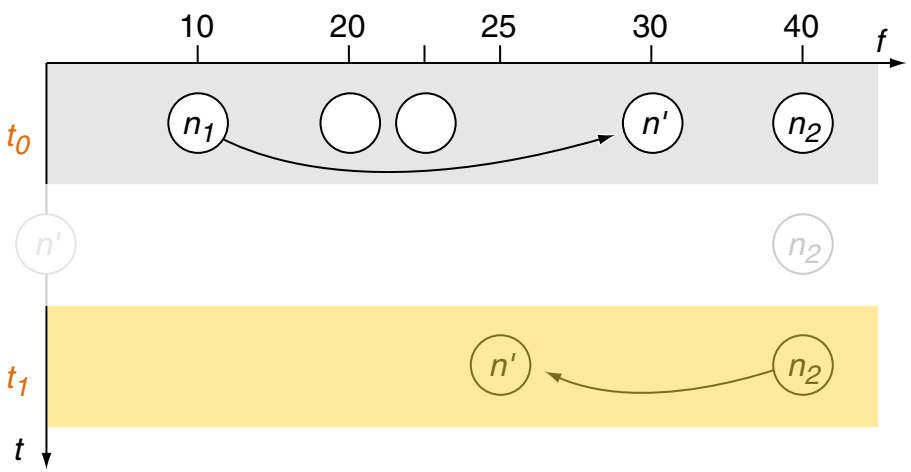
Case 2: n' is already on CLOSED.

```
5.    FOREACH n' IN successors(n)
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        ...
          IF (f_n(n') < f(n'))    // f_n(n'): compute f for n' via n.
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
```

State-space:



OPEN list:



Remarks:

- Given an occurrence of Case 2, it follows that f is not a monotonically increasing function in the solution base size (path length): $f(n') < f(n_2)$.
- Q. Given Case 2, and given the additional information that n_2 is a descendant of n'. What does this mean?
- Case 1 and Case 2 illustrate the path discarding behavior of Algorithm BF, it follows that f is not a monotonically increasing function in the solution base size (path length): $f(n') < f(n_2)$.
- Implementation / efficiency issue: Instead of reopening a node n', i.e., instead of moving n' from CLOSED to OPEN, a recursive update of the f-values and the backpointers of its successors can be done. This is highly efficient but should only be applied by experts since it can easily lead to inconsistent traversal trees (wrong backpointers).

  After reopening a node n', all the nodes n'' from which n' is reachable using only backpointers are still available. Since the f-values stored with such nodes n'' are not updated, subsequent node expansions may use f-values not matching pointer-paths. This can cause additional search efforts. Performing node expansion for nodes with invalid f-values can be avoided by using order-preserving functions f. Reopening nodes can be avoided by using monotonically increasing functions f, i.e., $f(n) \leq f(n')$ for successors n' of n.

Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

Case 3: n' has been on OPEN but is not found on OPEN or CLOSED.

```
5.    FOREACH n' IN successors(n) DO
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
➜       THEN   // n' encodes a new state.
          add_backpointer(n', n);
          insert(n', OPEN)
        ...
```

Possible reasons:

1. There is no occurrence check.
2. The occurrence check does not work properly. Note that state recognition can be a very hard (even undecidable) problem.
3. Explored parts of the state-space graph that seemed to be no longer required have been deleted by *cleanup_closed*.

Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

Case 3: n' has been on OPEN but is not found on OPEN or CLOSED.

```
5.    FOREACH n' IN successors(n) DO
        ...
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
➜       THEN   // n' encodes a new state.
          add_backpointer(n', n);
          insert(n', OPEN)
        ...
```

Possible reasons:

1. There is no occurrence check.
2. The occurrence check does not work properly. Note that state recognition can be a very hard (even undecidable) problem.
3. Explored parts of the state-space graph that seemed to be no longer required have been deleted by *cleanup_closed*.

Remarks:

- Q. What is the effect of the occurrence check in Case 1 and Case 2?
- Q. Should each visited node be stored in order to recognize the fact that its associated problem is encountered again?
- Q. Does a missing occurrence check affect the correctness of Algorithm BF?
- The shown version of the Algorithm BF has no call to *cleanup_closed*. However, such a call can be easily integrated, similar to the algorithms DFS or BFS.

Best-First Search for State-Space Graphs

Optimality of Algorithm BF [GBF optimality]

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

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

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

   In particular, the true cost of a cheapest solution path $P_{s-\gamma}$ extending a solution base P_{s-n} exceeds its f-value: $C_{P_{s-\gamma}}(s) \geq f(n)$.

2. The termination in case of success ($\star(n) = \mathit{True}$) must be delayed.

   In particular, there is no termination test when reaching a node the first time, but each time when choosing a node from the OPEN list.

   Algorithm BF with delayed termination is called Algorithm BF*.

Best-First Search for State-Space Graphs

Optimality of Algorithm BF [GBF optimality]

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

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

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

   In particular, the true cost of a cheapest solution path $P_{s-\gamma}$ extending a solution base P_{s-n} exceeds its f-value: $C_{P_{s-\gamma}}(s) \geq f(n)$.

2. The termination in case of success ($\star(n) = \mathit{True}$) must be delayed.

   In particular, there is no termination test when reaching a node the first time, but each time when choosing a node from the OPEN list.

   Algorithm BF with delayed termination is called Algorithm BF*.

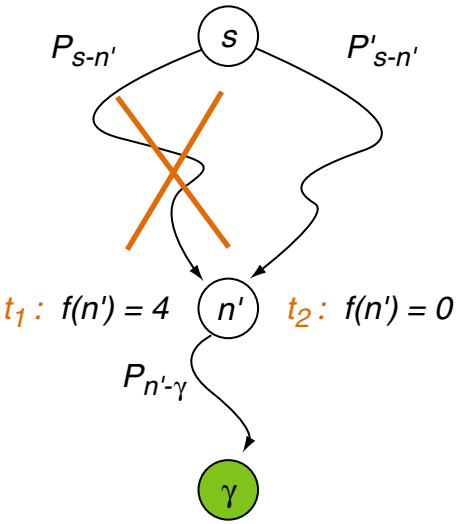
Best-First Search for State-Space Graphs [BF, GBF, GBF$\star$]

BF$^*(s, \mathit{successors}, \star, f)$

```
1.  insert(s, OPEN);
2.  LOOP
3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.    n = min(OPEN, f);
➜     IF ⋆(n) THEN RETURN(n);   // Delayed termination.
      remove(n, OPEN); push(n, CLOSED);
5.    FOREACH n' IN successors(n) DO
➜       IF ⋆(n') THEN add_backpointer(n', n); RETURN(n'); ENDIF
        IF (n' ∉ OPEN AND n' ∉ CLOSED)
        THEN
          add_backpointer(n', n);
          insert(n', OPEN)
        ELSE
          IF (f_n(n') < f(n'))
          THEN
            update_backpointer(n', n);
            IF n' ∈ CLOSED THEN remove(n', CLOSED); insert(n', OPEN); ENDIF
          ENDIF
        ENDIF
      ENDDO
6.  ENDLOOP
```

Best-First Search for State-Space Graphs

Irrevocable Path Discarding in Algorithm BF



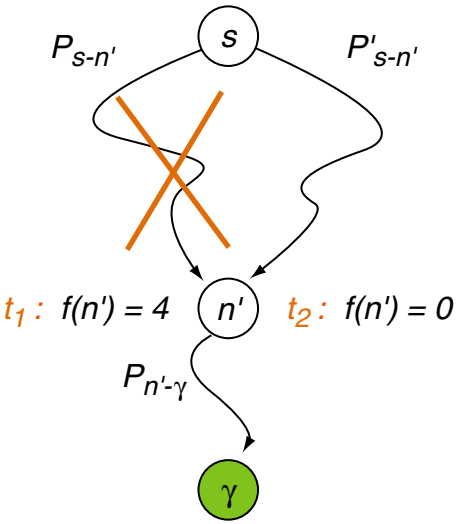
Irrevocability is crucial and will probably lead to suboptimum solutions if the evaluation function f depends on *global properties* of a solution.

Examples:

1. "Determine the shortest path (cheapest solution) that has two edges (operators) of equal costs."
2. "Determine a path (a solution) that minimizes the maximum edge cost (operator cost) difference."

Best-First Search for State-Space Graphs

Irrevocable Path Discarding in Algorithm BF (continued)



Irrevocability is reasonable:

1. For optimization problems, if the cost estimations for alternative solution bases are independent of their shared continuation.

2. For constraint satisfaction problems, if the following equivalence holds:

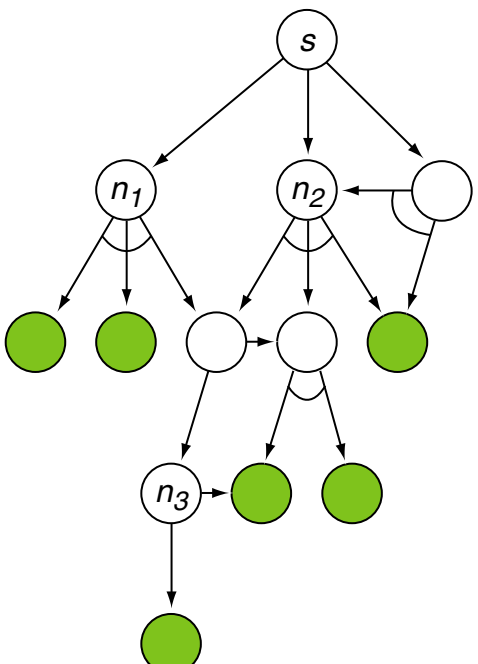
   $\Leftrightarrow$ “Solution base $P_{s-n'}$ can be completed to a solution”

   “Solution base $P'_{s-n'}$ can be completed to a solution.”

Best-First Search for AND-OR Graphs

Illustration of Solution Graphs and Solution Bases

Problem-reduction graph:

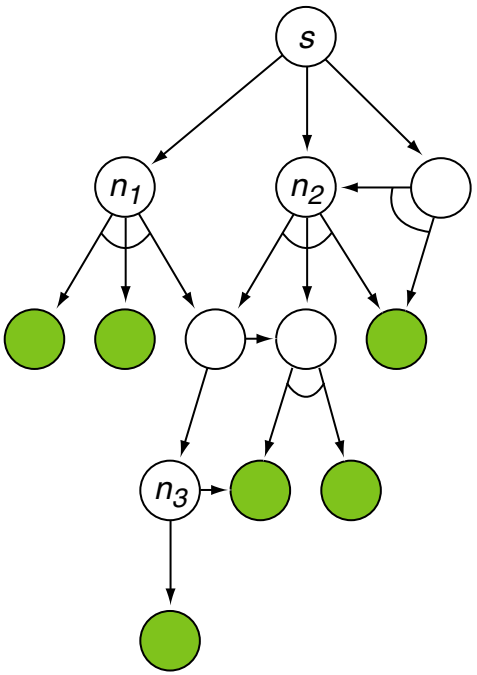


Solved rest problem

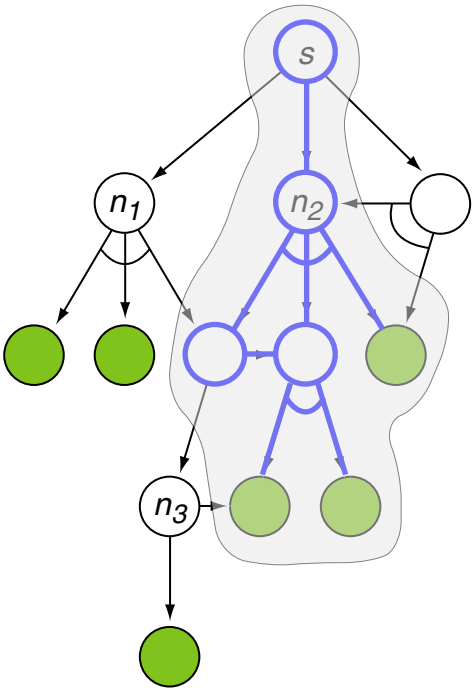
Best-First Search for AND-OR Graphs

Illustration of Solution Graphs and Solution Bases

Problem-reduction graph:



Solution graph for node *s*:

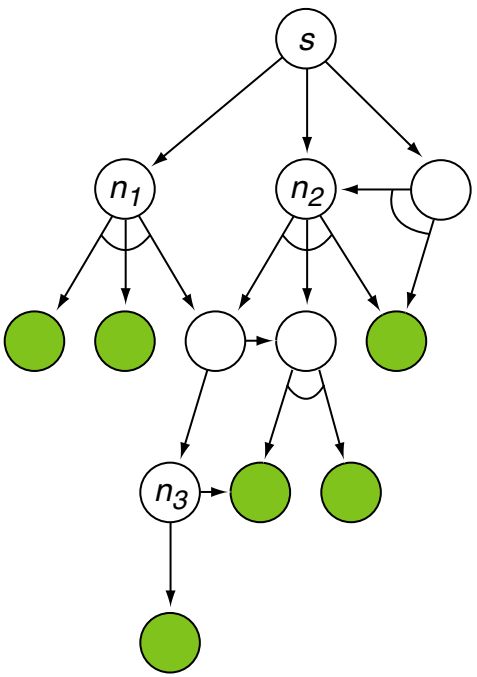


Solved rest problem

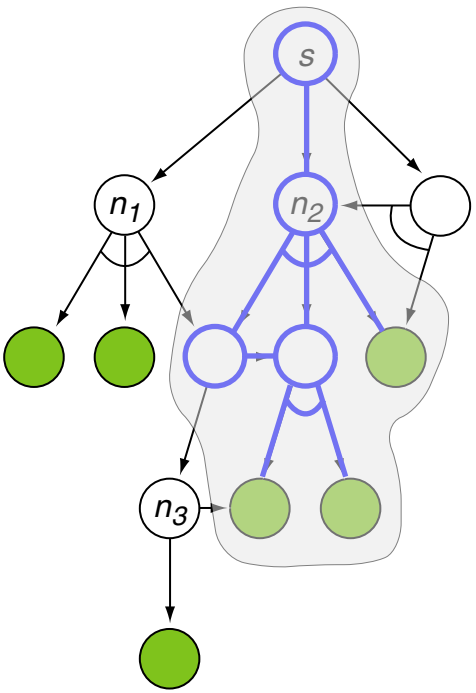
Best-First Search for AND-OR Graphs

Illustration of Solution Graphs and Solution Bases

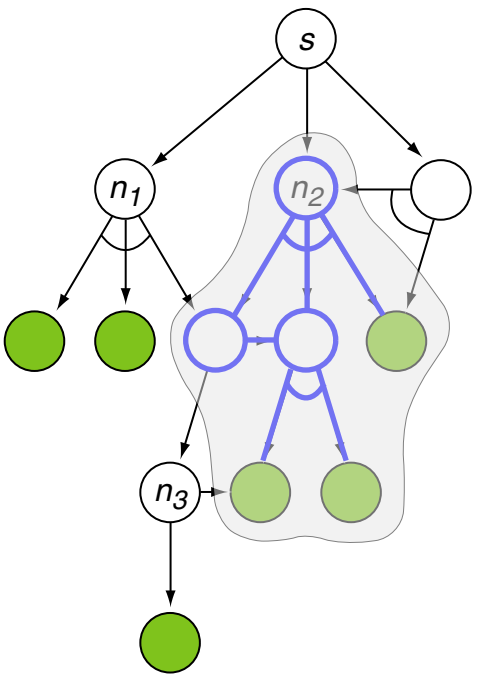
Problem-reduction graph:



Solution graph for node s:



Solution graph for node n_2:

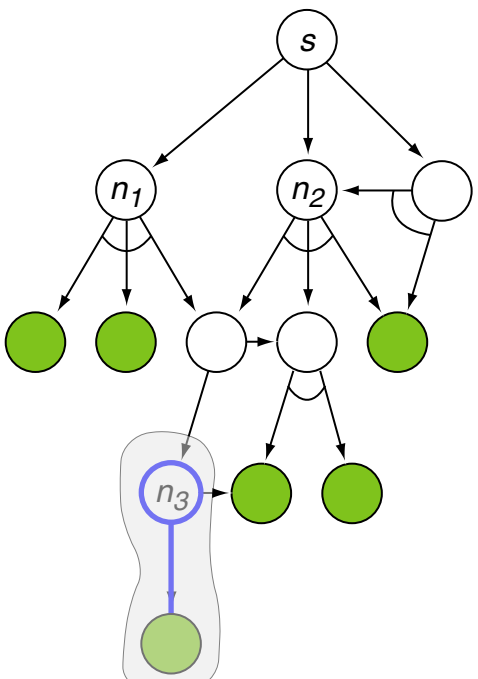


Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of Solution Graphs and Solution Bases (continued)

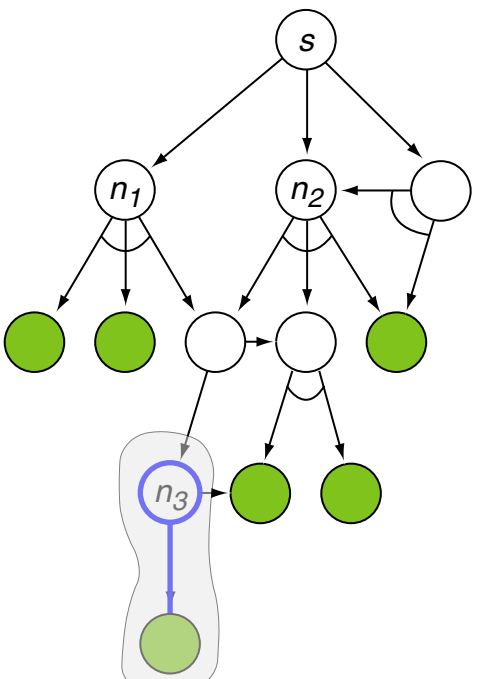
Solution graph for n_3:



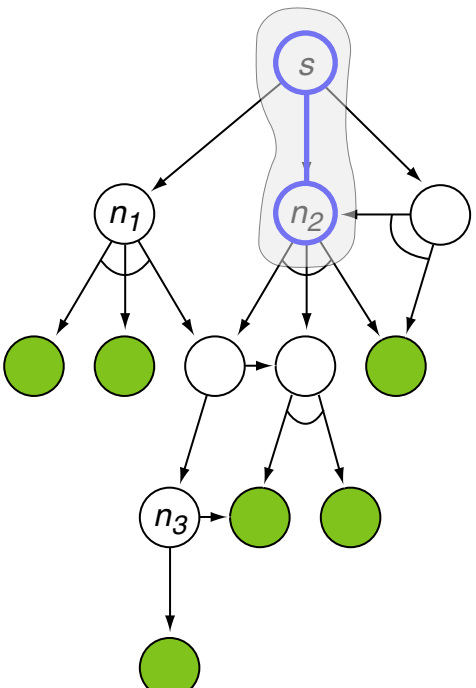
Best-First Search for AND-OR Graphs

Illustration of Solution Graphs and Solution Bases (continued)

Solution graph for n_3:



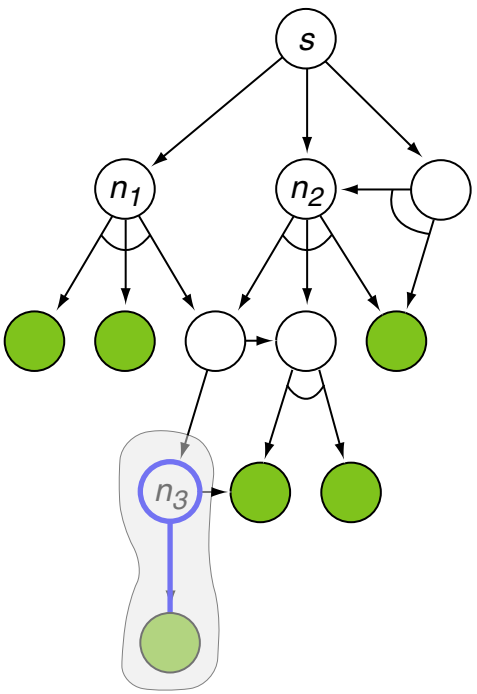
Solution base for s:



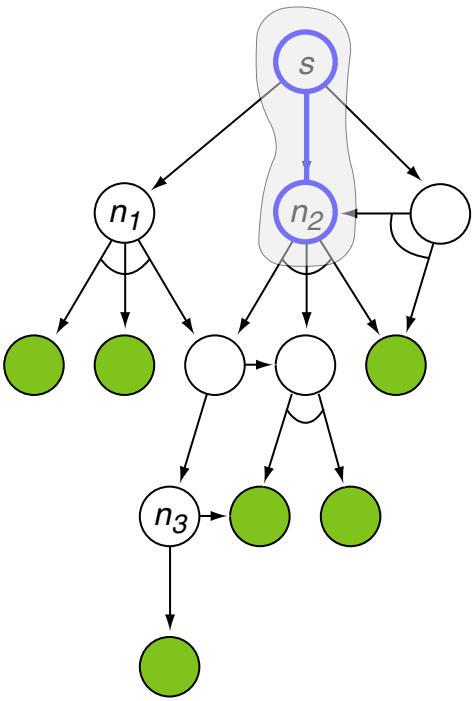
Best-First Search for AND-OR Graphs

Illustration of Solution Graphs and Solution Bases (continued)

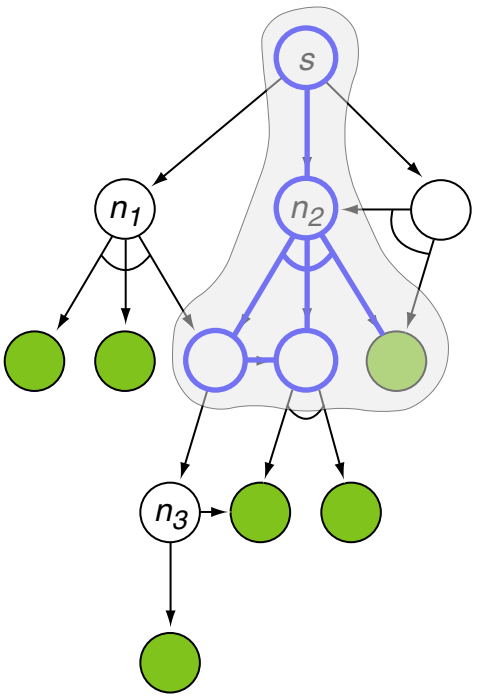
Solution graph for n_3:



Solution base for s:



Solution base for s:



Best-First Search for AND-OR Graphs

Definition 12 (Solution Base in a Problem-Reduction Graph)

Let G be an explored subgraph of an acyclic AND-OR graph, and let n be a node in G. A finite subgraph H of G is called a *solution base for* n *in* G iff ($\leftrightarrow$) the following conditions hold:

1. H contains the node n.
2. If H contains an inner OR node, then H contains exactly one link to a direct successor in G and this successor node.
3. If H contains an inner AND node, then H contains all links to its direct successors in G and all these successor nodes.
4. The leaf nodes in H are terminal nodes in G, not known to be dead ends.
5. H is minimal: it contains no additional nodes and edges.

Compare:

- Solution base in a state-space graph.
- Solution graph in a problem-reduction graph. [S:II Problem-Reduction Representation]

Remarks:

- Usually, we are interested in finding a solution base H for the root node s in G.
- Simply put, a subgraph of a search space graph is called solution base if it can be extended towards a solution (graph).
- Obviously, all solution graphs contained in G are also solution bases.
- If H is a solution base for a node n in G and if n' is some node in H, then the subgraph of H rooted at n', H', is a solution base for a node n' in G. This subgraph is sometimes called the *solution base in H induced by n'*.
- The fact whether a problem-reduction graph G contains a solution base H for the start node s can be checked by recursively applying the propagation rules of the unsolvable-labeling procedure. [S:II Problem-Reduction Representation]

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

When searching problem-reduction graphs with the Algorithm GBF,

- 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* in G,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

When searching problem-reduction graphs with the Algorithm GBF,

- 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* in G,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

When searching problem-reduction graphs with the Algorithm GBF,

- 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* in G,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

When searching problem-reduction graphs with the Algorithm GBF,

- 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* in G,
- f_2 returns a most informative node in a solution base.

Best-First Search for AND-OR Graphs

Principles of Algorithm GBF [BF principles]

When searching problem-reduction graphs with the Algorithm GBF,

- 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* in G,
- 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.
$\star(n)$. Predicate that is *True* if n is a goal node.
$\perp(n)$. Predicate that is *True* if n is a dead end.
$f_1(H)$. 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*.

Best-First Search for AND-OR Graphs

GBF$(s,$ *successors*$, \bot, \star, f_1, f_2)$

```
 1.  insert(s, OPEN);
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.a
4.b


 5.




 6.  ENDLOOP
```

Best-First Search for AND-OR Graphs

GBF$(s, \textit{successors}, \bot, \star, f_1, f_2)$

```
 1.  insert(s, OPEN);
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.a    H = min_solution_base(s, f_1);    // Find most promising solution base.
4.b    n = f_2(H);    // Find most informative node in H.
       remove(n, OPEN); push(n, CLOSED);
 5.
 6.  ENDLOOP
```

Best-First Search for AND-OR Graphs

GBF$(s, \textit{successors}, \bot, \star, f_1, f_2)$

```
 1.  insert(s, OPEN);
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.a    H = min_solution_base(s, f_1);    // Find most promising solution base.
4.b    n = f_2(H);    // Find most informative node in H.
       remove(n, OPEN); push(n, CLOSED);
 5.    FOREACG n' IN successors(n) DO
         add_backpointer(n', n);
         IF ⋆(n') THEN solved_labeling(n');    // Check for solution graphs.
         IF ⋆(s) THEN RETURN(solution_graph(s));
         insert(n', OPEN);
       ENDDO
 6.  ENDLOOP
```

Best-First Search for AND-OR Graphs

GBF$(s, \mathit{successors}, \bot, \star, f_1, f_2)$

```
 1.  insert(s, OPEN);
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.a    H = min_solution_base(s, f_1);    // Find most promising solution base.
4.b    n = f_2(H);    // Find most informative node in H.
       remove(n, OPEN); push(n, CLOSED);
 5.    FOREACG n' IN successors(n) DO
         add_backpointer(n', n);
         IF ⋆(n') THEN solved_labeling(n');    // Check for solution graphs.
         IF ⋆(s) THEN RETURN(solution_graph(s));



         IF (n' ∉ OPEN AND n' ∉ CLOSED)
         THEN   // n' encodes a new state.
           insert(n', OPEN);
         ENDIF
       ENDDO
 6.  ENDLOOP
```

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

GBF$(s, \textit{successors}, \bot, \star, f_1, f_2)$

```
 1.  insert(s, OPEN);
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.a    H = min_solution_base(s, f1);    // Find most promising solution base.
4.b    n = f2(H);    // Find most informative node in H.
       remove(n, OPEN); push(n, CLOSED);
 5.    FOREACG n' IN successors(n) DO
         add_backpointer(n', n);

         IF ⋆(n') THEN solved_labeling(n');    // Check for solution graphs.
         IF ⋆(s) THEN RETURN(solution_graph(s));

         IF ⊥(n') THEN unsolvable_labeling(n');    // Check for unsolvability.
         IF ⊥(s) THEN RETURN(Fail);

         IF (n' ∉ OPEN AND n' ∉ CLOSED)
         THEN    // n' encodes a new state.
           insert(n', OPEN);
         ENDIF
       ENDDO
 6.  ENDLOOP
```

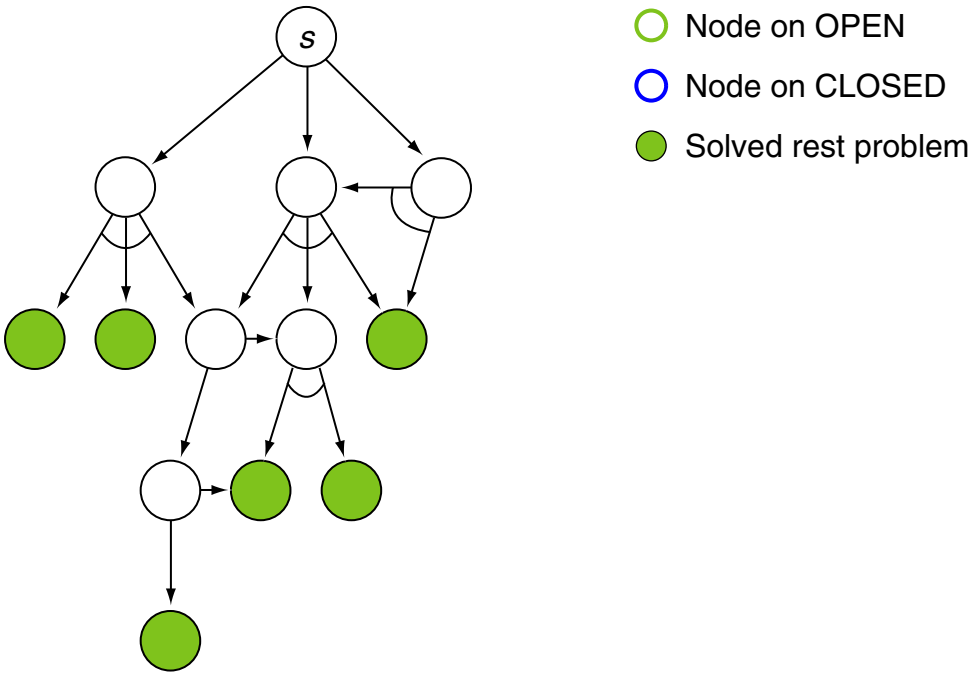
Remarks:

- 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. As a consequence, G is valid subgraph of the true search space graph. Though this compact representation avoids the re-solving of an already solved problem, it requires a more involved analysis of G, e.g. to deal with cycles.
  (b) Without occurrence check. Each successor becomes a new node, irrespective of duplicate rest problems. As a consequence, G corresponds to an AND-OR tree that can be considered as an unfolding of the true search space graph.

  An occurrence check is used in the *successors*(n) function. The test "$(n' \notin$ `OPEN` AND $n' \notin$ `CLOSED`$)$" simply checks whether given references are contained in the lists.
  Therefor, 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, f_1 computation and solution base construction can be implemented very efficiently in a combined fashion.

Best-First Search for AND-OR Graphs

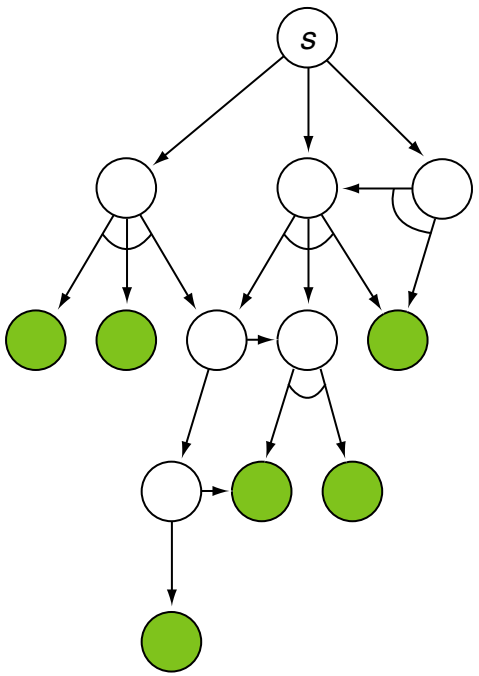
Illustration of GBF



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.

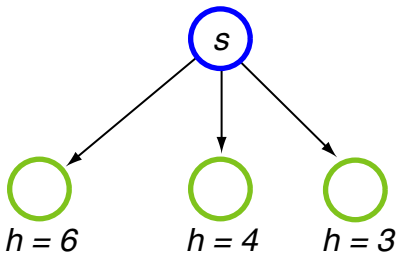
Remarks:

- The evaluation function $f_1(H)$ is 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

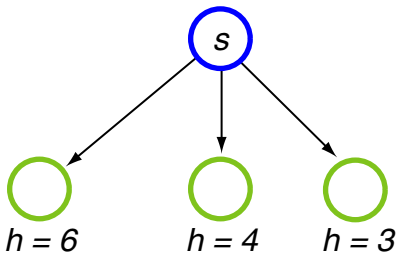
Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

1. s is expanded and moved to CLOSED.

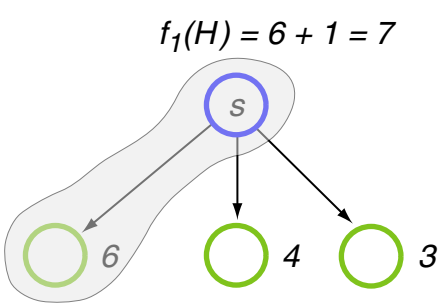


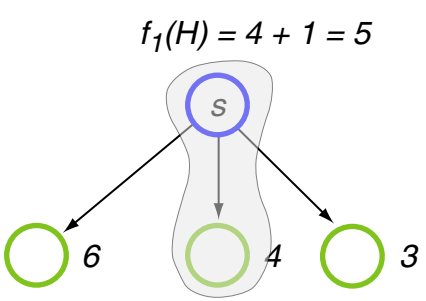
Node on OPEN

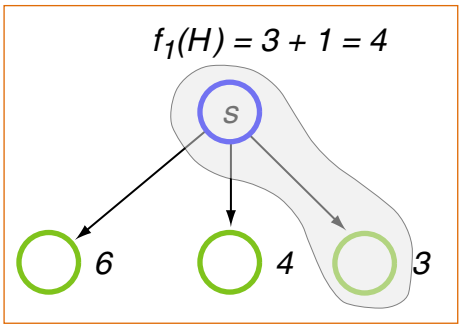
Node on CLOSED

Solved rest problem

Possible solution bases:





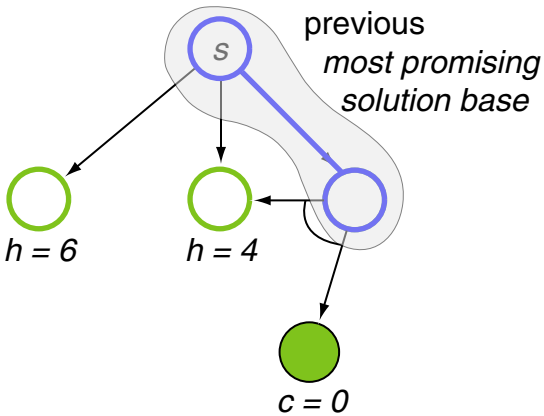


Most promising solution base H

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

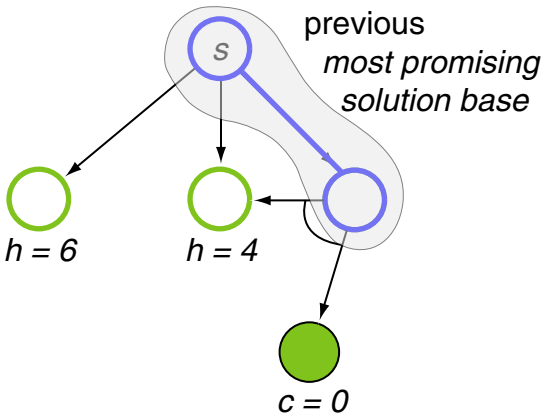
Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

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

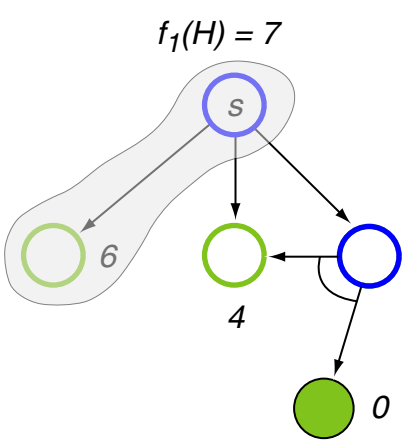


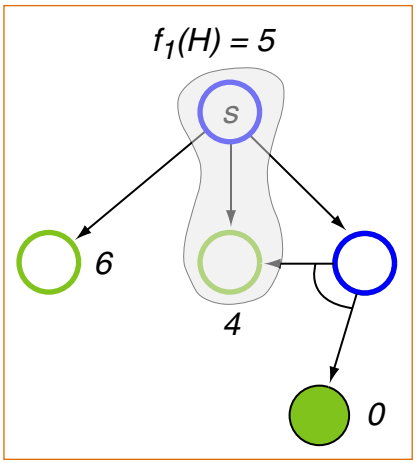
Node on OPEN

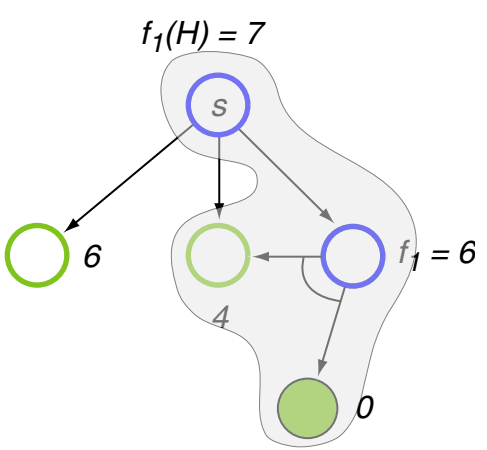
Node on CLOSED

Solved rest problem

Possible solution bases:





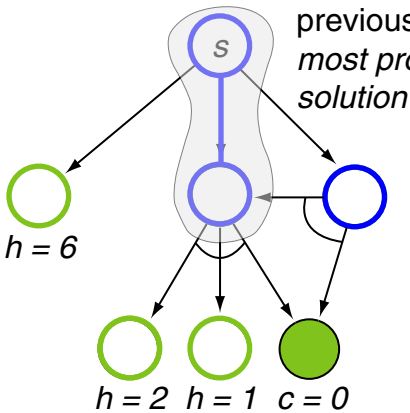


Most promising solution base H

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

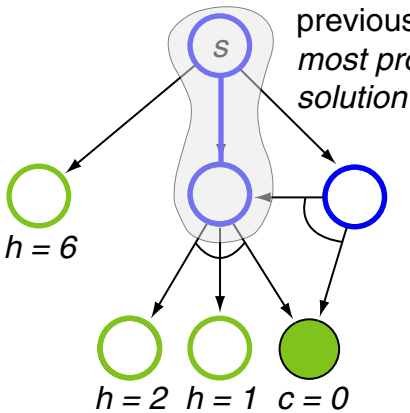
Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

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

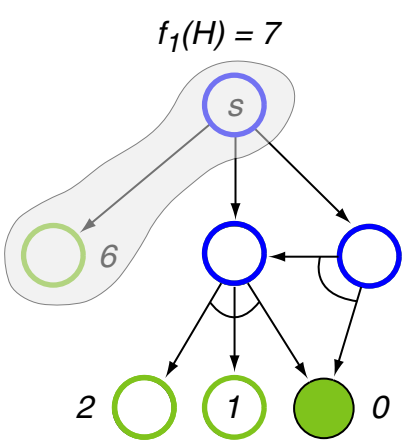


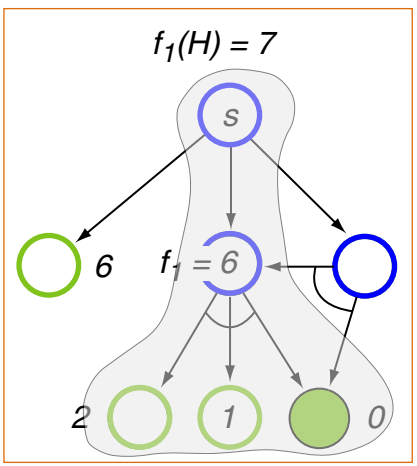
Node on OPEN

Node on CLOSED

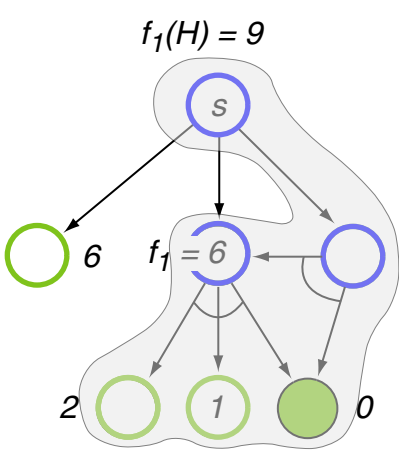
Solved rest problem

Possible solution bases:





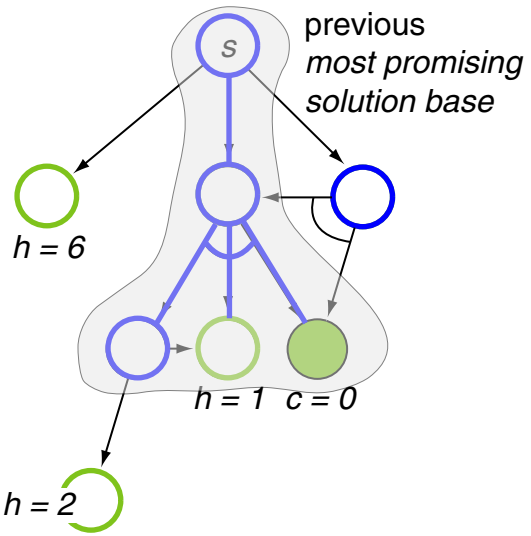
Most promising solution base H



Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

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

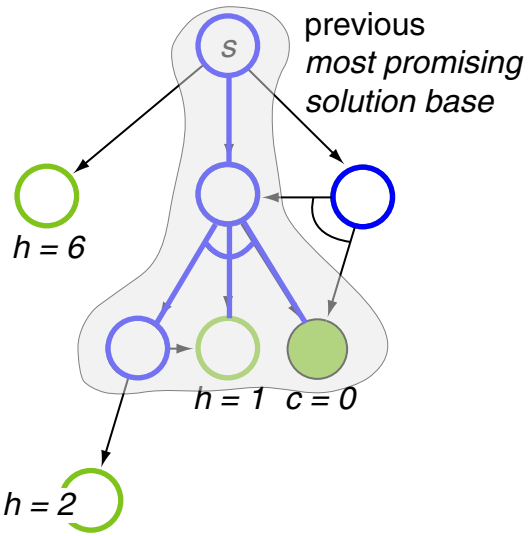


- Node on OPEN
- Node on CLOSED
- Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued)

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

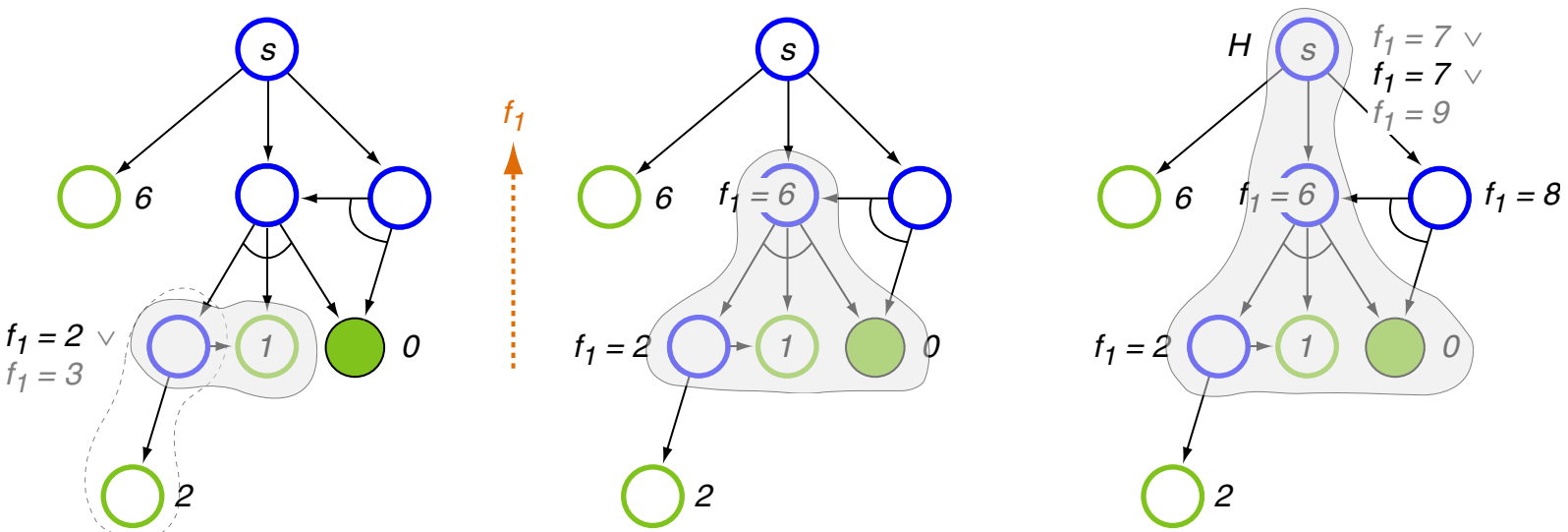


Node on OPEN

Node on CLOSED

Solved rest problem

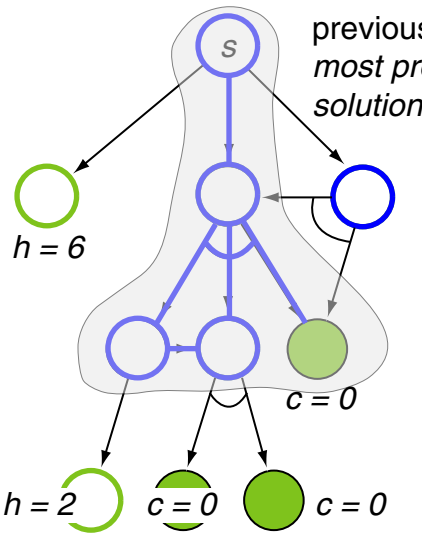
Solution bases:



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

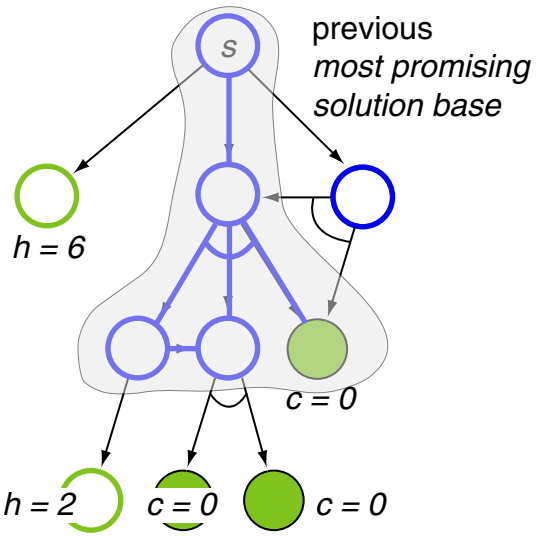
Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF (continued) [next step]

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

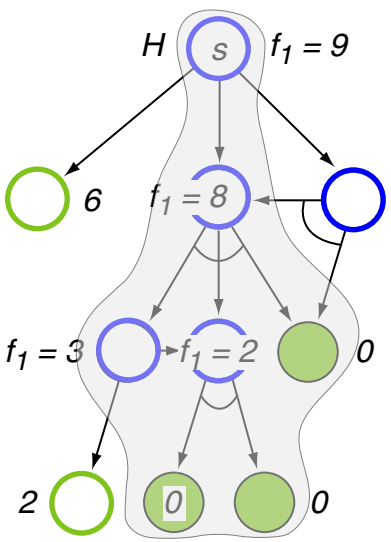


Node on OPEN

Node on CLOSED

Solved rest problem

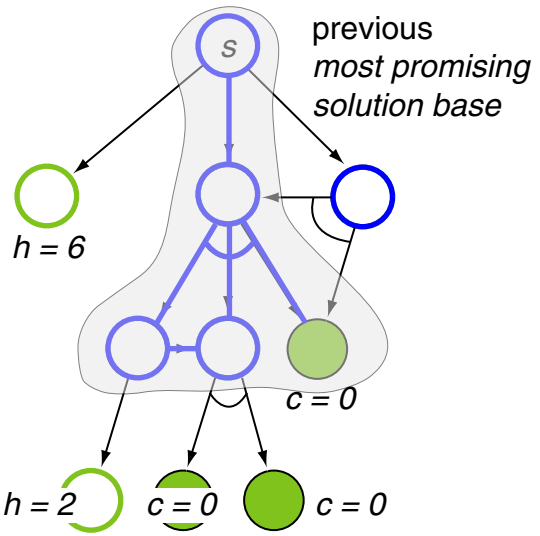
A solution graph H:



Best-First Search for AND-OR Graphs

Illustration of GBF (continued) [next step]

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

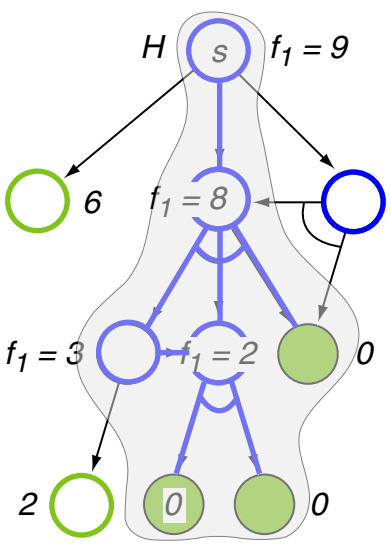


Node on OPEN

Node on CLOSED

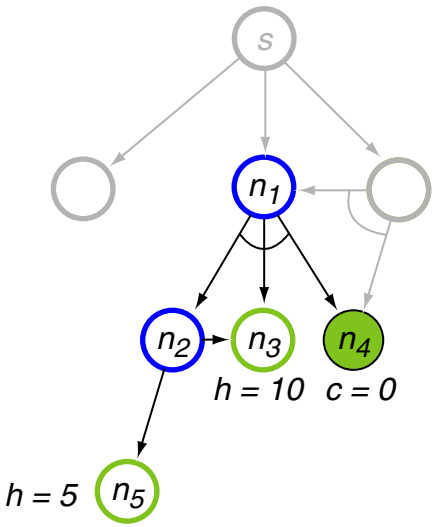
Solved rest problem

A solution graph H:



Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



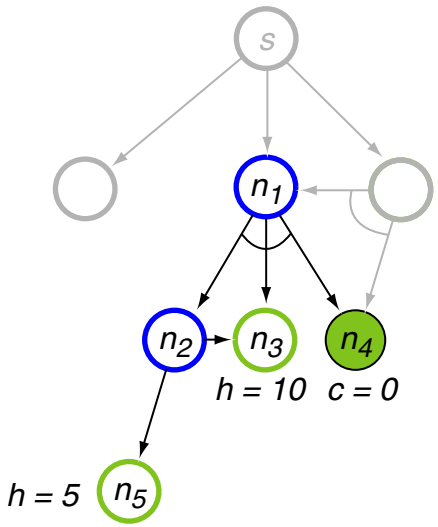
Node on OPEN

Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



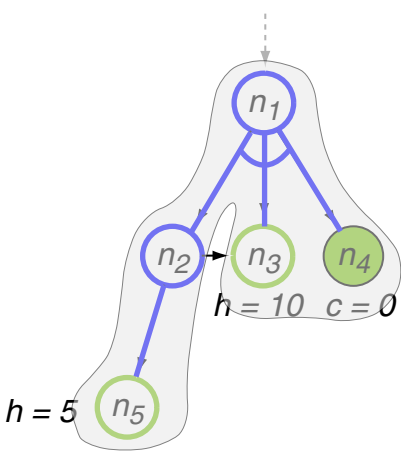
Node on OPEN

Node on CLOSED

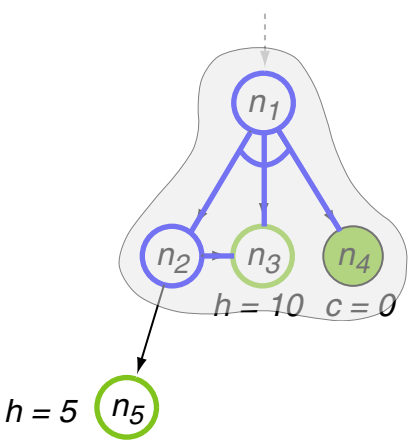
Solved rest problem

n_1 start node

(a)



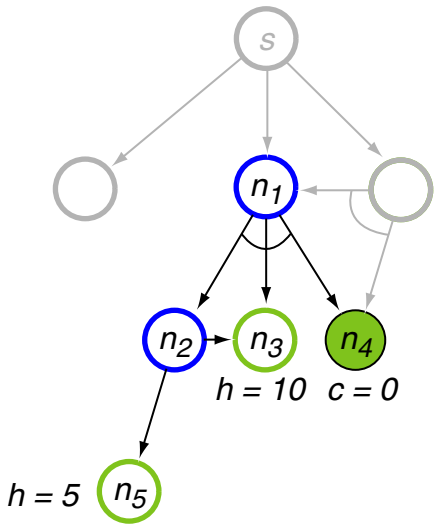
(b)



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

Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



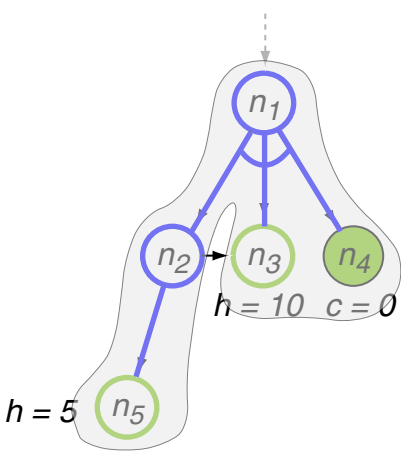
Node on OPEN

Node on CLOSED

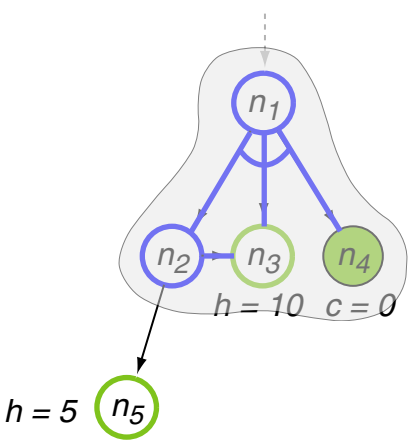
Solved rest problem

n_1 start node

(a)



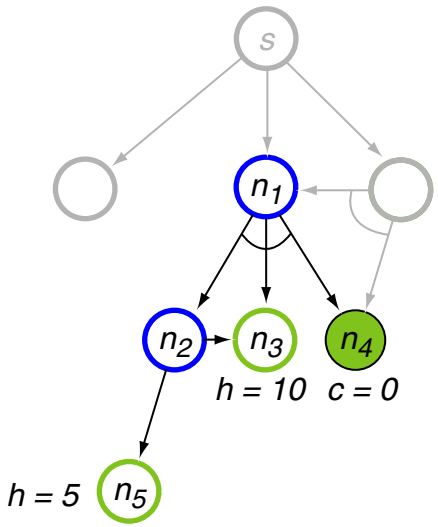
(b)



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

Best-First Search for AND-OR Graphs

Multiple Accounting of a Node



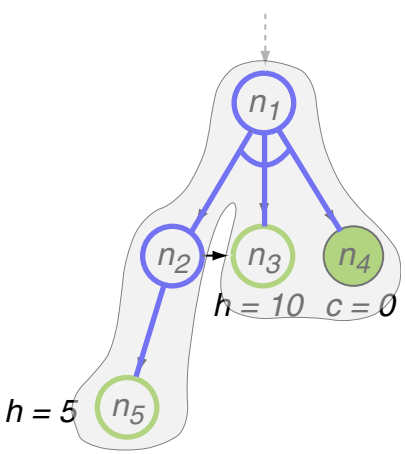
Node on OPEN

Node on CLOSED

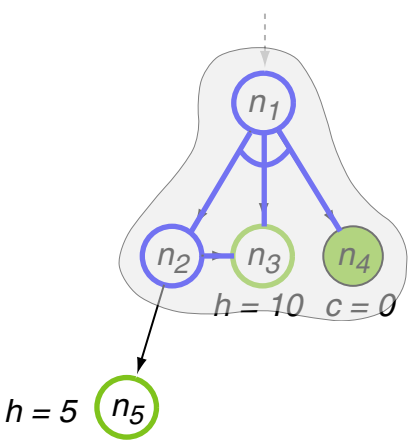
Solved rest problem

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 ($\star(s) = \mathit{True}$) 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 ($\star(s) = \mathit{True}$) 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⋆, GBF]

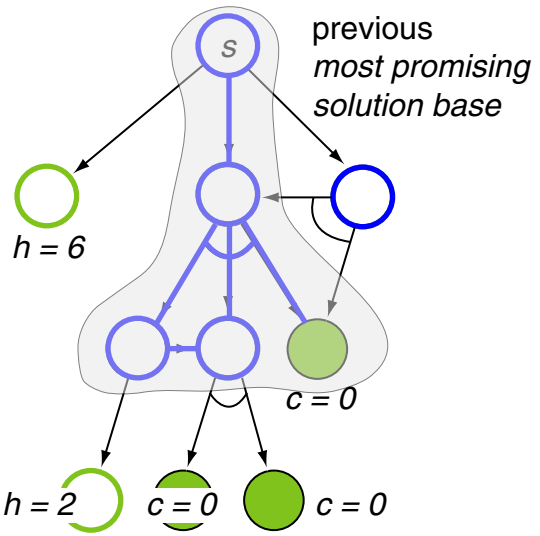
GBF*$(s, \mathit{successors}, \bot, \star, f_1, f_2)$

```
 1.  insert(s, OPEN);
 2.  LOOP
 3.    IF (OPEN = ∅) THEN RETURN(Fail);
4.a    H = min_solution_base(s, f_1);
 ➜     solved_labeling(H);     // Check if H is a solution graph.
 ➜     IF ⋆(s) THEN RETURN(H);     // Delayed termination.
4.b    n = f_2(H);
       remove(n, OPEN); push(n, CLOSED);
 5.    FOREACH n' IN successors(n) DO
         add_backpointer(n', n);
 ➜       IF ⋆(n') THEN solved_labeling(n');
 ➜       IF ⋆(s) THEN RETURN(solution_graph(s));
         IF ⊥(n') THEN unsolvable_labeling(n');
         IF ⊥(s) THEN RETURN(Fail);
         IF (n' ∉ OPEN AND n' ∉ CLOSED)
         THEN
           insert(n', OPEN);
         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

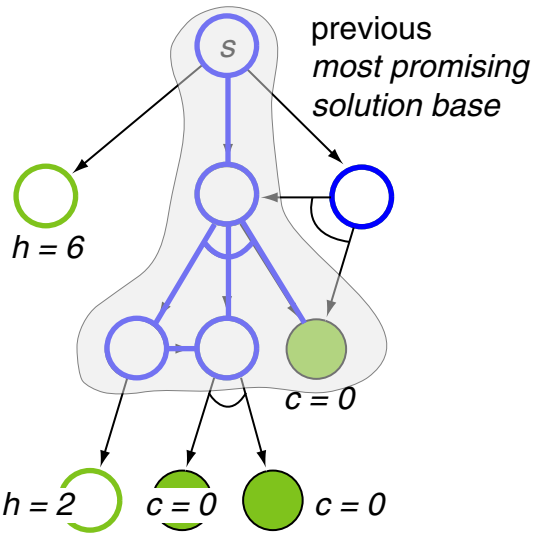
Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

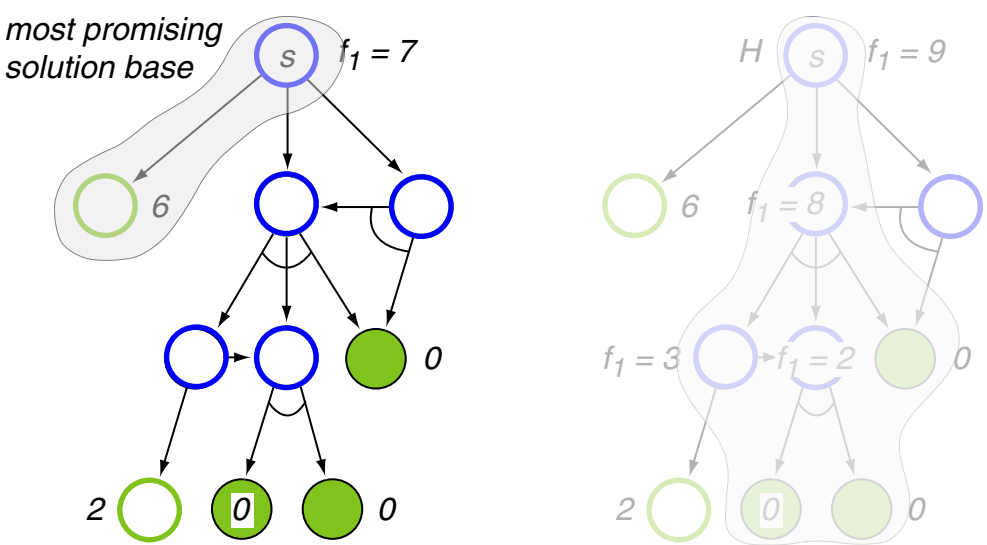
Illustration of GBF* (continued) [previous step]

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



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

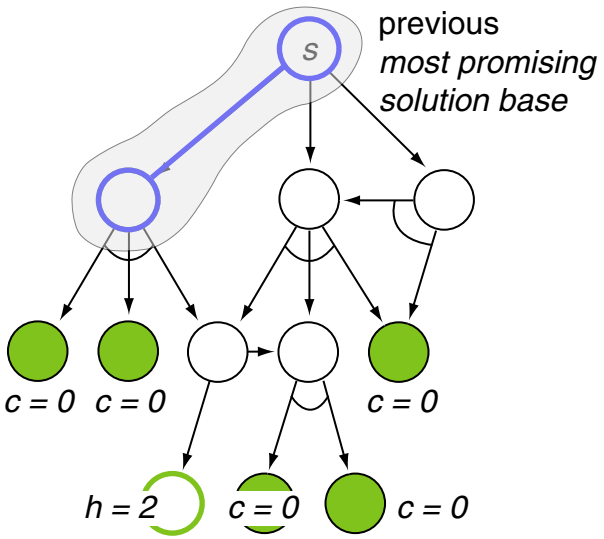
Solution base / graph:



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

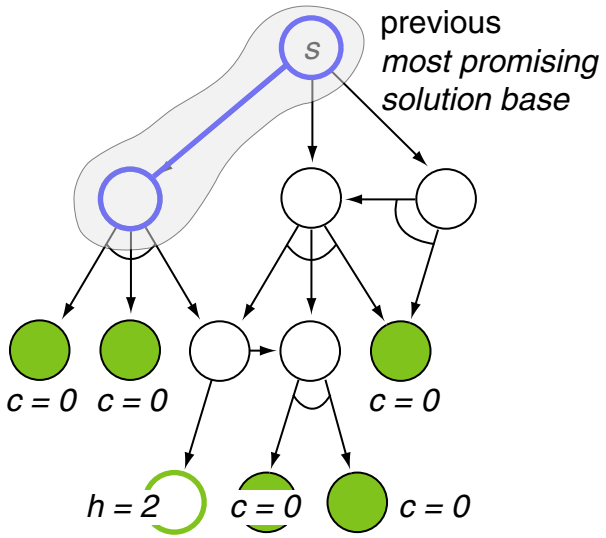
Node on CLOSED

Solved rest problem

Best-First Search for AND-OR Graphs

Illustration of GBF* (continued)

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

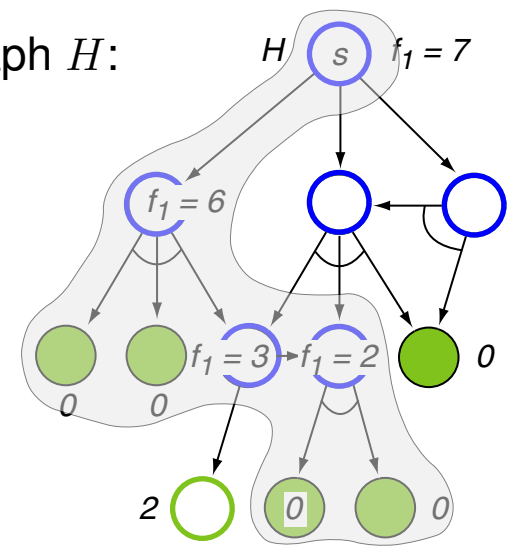


Node on OPEN

Node on CLOSED

Solved rest problem

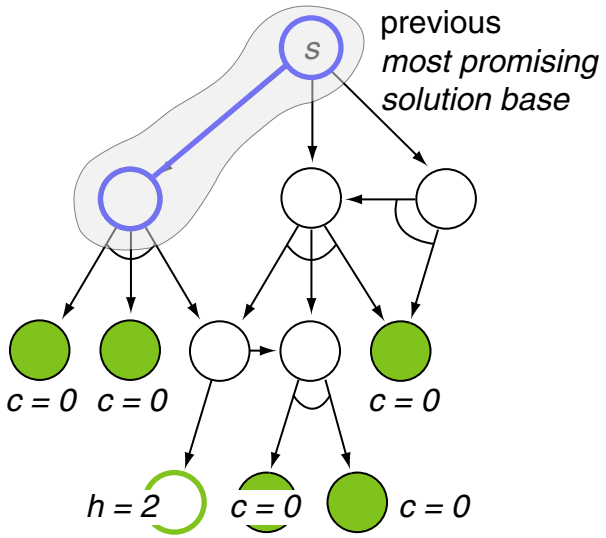
Returned solution graph H:



Best-First Search for AND-OR Graphs

Illustration of GBF* (continued)

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

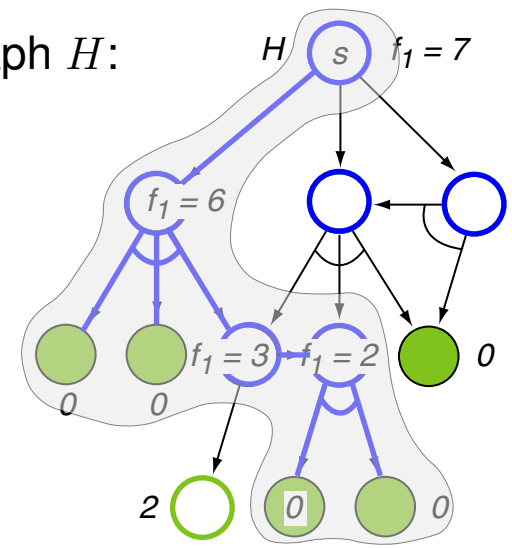


Node on OPEN

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:IV 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:IV 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:IV 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, 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:IV 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, 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:IV 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, 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*.

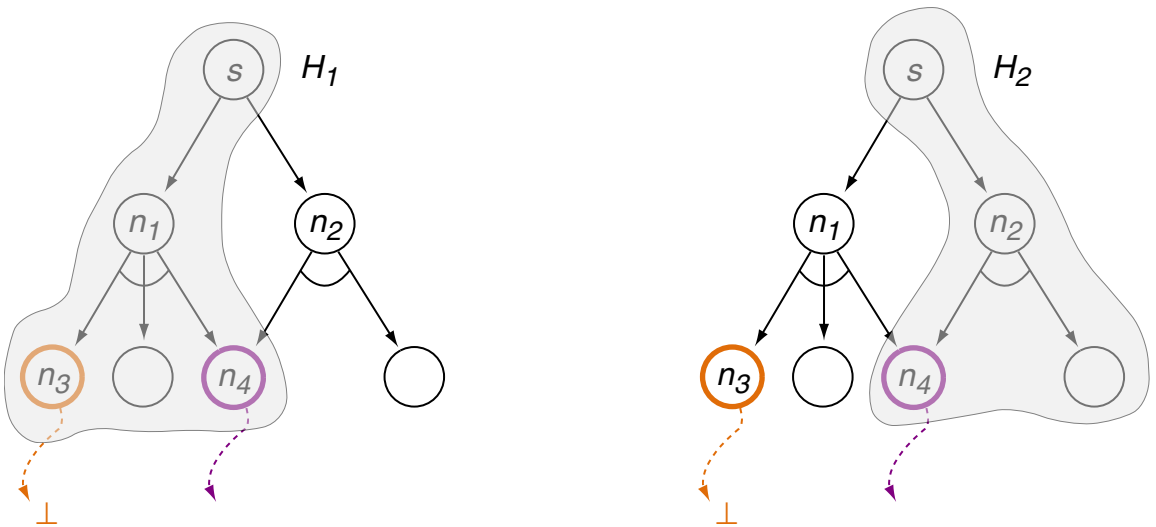
Remarks:

- (Under BF) the tree formed by the union of all solution bases is also called *traversal tree*. It is defined by the backpointers 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:

⇔ “Solution base H_1 can be completed to a solution”

“Solution base H_2 can be completed to a solution.”

Remarks:

- The illustrated solution bases H_1, H_2, fulfill not 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.