Chapter S:III

III. Informed Search

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. evaluation of the information given by 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.

Remarks:

- ❑ Node n represents a solution base. Therefore, n gives access to information about a path from s to n.
- ❑ The remaining problem is the problem of determining a solution path as a continuation of the solution base given by n. The complexity estimation of this problem is beyond the information encoded in nodes and edges.
- ❑ Knowing that G is Euclidean is an example for domain specific problem solving knowledge. Euclidean distances can be used for estimating remaining path length.
- ❑ Depending on an evaluations of the explored part G of the search space graph, the emphasis on specific knowledge in the computation of f can be changed.

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.
   (b) Select a node in that solution base, which has not been expanded yet.
   (c) Expand the respective node in the solution base and process the newly generated solution bases (test / evaluate / add / discard / substitute).

- ❑ Node expansion is used as basic step.
- ❑ Best-First Algorithms are informed versions of `Basic_OR_Search`.
- ❑ General Best-First Algorithms are informed versions of `Basic_AND-OR_Search`.

Remarks:

- The schema is further extended by termination tests for failure and success.
- The job of the evaluation function 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

Principles of Algorithm BF [GBF principles]

When searching state-space graphs with the Algorithm BF,

1. Evaluation function f evaluates solution bases P_{s-n},
2. f estimates the optimal cost for a solution path that extends P_{s-n},
3. a most promising solution base minimizes f,
4. a most promising solution base is searched *among all solution bases* currently maintained by BF,
5. *at most one solution base* P_{s-n} is maintained for each node n in G,
6. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
7. 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. Evaluation function f evaluates solution bases P_{s-n},
2. f estimates the optimal cost for a solution path that extends P_{s-n},
3. a most promising solution base minimizes f,
4. a most promising solution base is searched *among all solution bases* currently maintained by BF,
5. *at most one solution base P_{s-n}* is maintained for each node n in G,
6. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
7. 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. Evaluation function f evaluates solution bases P_{s-n},
2. f estimates the optimal cost for a solution path that extends P_{s-n},
3. a most promising solution base minimizes f,
4. a most promising solution base is searched *among all solution bases* currently maintained by BF,
5. *at most one solution base* P_{s-n} is maintained for each node n in G,
6. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
7. 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. Evaluation function f evaluates solution bases P_{s-n},
2. f estimates the optimal cost for a solution path that extends P_{s-n},
3. a most promising solution base minimizes f,
4. a most promising solution base is searched *among all solution bases* currently maintained by BF,
5. *at most one solution base* P_{s-n} is maintained for each node n in G,
6. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
7. 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. Evaluation function f evaluates solution bases P_{s-n},
2. f estimates the optimal cost for a solution path that extends P_{s-n},
3. a most promising solution base minimizes f,
4. a most promising solution base is searched *among all solution bases* currently maintained by BF,
5. *at most one solution base* P_{s-n} is maintained for each node n in G,
6. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
7. 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. Evaluation function f evaluates solution bases P_{s-n},
2. f estimates the optimal cost for a solution path that extends P_{s-n},
3. a most promising solution base minimizes f,
4. a most promising solution base is searched *among all solution bases* currently maintained by BF,
5. *at most one solution base* P_{s-n} is maintained for each node n in G,
6. for two paths leading to the same node, the one with the higher f-value is discarded (Keyword: *Path discarding*),
7. 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:

- When searching state-space graphs with the algorithm BF, we assume that f values are available for s and the nodes newly generated by *successors*(n).
- When searching state-space graphs with the algorithm BF, the solution bases P_{s-n} currently under consideration are defined by the backpointers stored with nodes in `OPEN`.
- If a dead end recognition $\perp(n)$ is available, no solution base will be considered that contains a node labeled "unsolvable" using $\perp(n)$. A dead end recognition $\perp(.)$ can be integrated in f by setting $f(n) = \infty$ if $\perp(n)$ is true.
- 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:III Specialized Cost Measures]
- 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 represents a solution path.
$f(n)$. Evaluation function for solution base represented by n.

Output: A node γ representing a solution path or the symbol *Fail*.

Best-First Search for State-Space Graphs

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

1.  $insert(s, \text{OPEN})$;
2.  **LOOP**
3.    IF ($\text{OPEN} = \emptyset$) 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 $= \emptyset$) 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.  $insert(s, \texttt{OPEN})$;
  2.  LOOP
  3.    IF ($\texttt{OPEN} = \emptyset$) THEN RETURN($Fail$);
  4.    $n = min(\texttt{OPEN}, f)$;    // Find most promising solution base.
        $remove(n, \texttt{OPEN})$;  $push(n, \texttt{CLOSED})$;
  5.    FOREACH $n'$ IN $successors(n)$ DO
          $add\_backpointer(n', n)$;
          IF $\star(n')$ THEN RETURN($n'$);

            $insert(n', \texttt{OPEN})$;

        ENDDO
  6.  ENDLOOP
```

Best-First Search for State-Space Graphs

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

 1.  insert($s$, OPEN);
 2.  LOOP
 3.    IF (OPEN $= \emptyset$) 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 $n$.

         add_backpointer($n'$, $n$);

         IF $\star(n')$ THEN RETURN($n'$);

         IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)    // Instance of $n'$ in OPEN $\cup$ CLOSED ?
         THEN   // $n'$ encodes an instance of a new state.
           insert($n'$, OPEN);
         ELSE   // $n'$ encodes an instance of an already visited state.







         ENDIF
       ENDDO
 6.  ENDLOOP
```

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

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

  1.  insert($s$, OPEN);
  2.  LOOP
  3.    IF (OPEN $= \emptyset$) 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 $n$.

          add_backpointer($n'$, $n$);

          IF $\star(n')$ THEN RETURN($n'$);

          IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)   // Instance of $n'$ in OPEN $\cup$ CLOSED ?
          THEN   // $n'$ encodes an instance of a new state.
            insert($n'$, OPEN);
          ELSE   // $n'$ encodes an instance of an already visited state.

            $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
            IF ($f(n') < f(n'_{old})$)
            THEN   // The state of $n'$ is reached via a cheaper path.
              update_backpointer($n'_{old}$, $n$);
              IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
            ENDIF

          ENDIF
        ENDDO
  6.  ENDLOOP
```

Remarks:

- The first complete version of algorithm BF on page S:III-21 can be seen as a generalization of uniform-cost search. Both algorithms maintain multiple instances of nodes if generated by *successors*(n). Whereas uniform-cost search can only make use of the knowledge from the explored part of the search space graph, the evaluation function f can use domain knowledge. The algorithm on page S:III-21 could be named `Basic-BF`, since it is a specialization of `Basic_OR_Search` with respect to node selection from OPEN.

- 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.

- The function *retrieve*$(n', \texttt{OPEN} \cup \texttt{CLOSED})$ retrieves a previously stored instance of n' from OPEN resp. CLOSED.

- The updating of backpointers done by BF algorithms preserves the tree structure the graph $G_{BF,t}$ (maintained by BF via nodes stored in OPEN and CLOSED and backpointers) at any point in time t. So, BF algorithms store traversal trees.

Best-First Search for State-Space Graphs

Path Discarding for a Node n'

```
5.    FOREACH $n'$ IN successors($n$)
        ...
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
          $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
          IF ($f(n') < f(n'_{old})$)
          THEN   // The state of $n'$ is reached via a cheaper path.
            update_backpointer($n'_{old}$, $n$);
            IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
          ENDIF
```

- $f(n')$ is computed using the new instance of n' and the backpointer path from s to n' via its parent n.
- $f(n'_{old})$ is computed using the old instance of n' and the backpointer path from s to n'_{old}.
- Path discarding is performed implicitly by maintaining at most one instantiation per node and, therefore, one backpointer per node.
- Algorithm BF cannot recover paths that were discarded, i.e., path discarding is irrevocable.
- For increased performance, f-values are stored with the nodes.

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 ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
         ...
           $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
           IF ($f(n') < f(n'_{old})$)
           THEN   // The state of $n'$ is reached via a cheaper path.
             update_backpointer($n'_{old}$, $n$);
             IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
           ENDIF
```

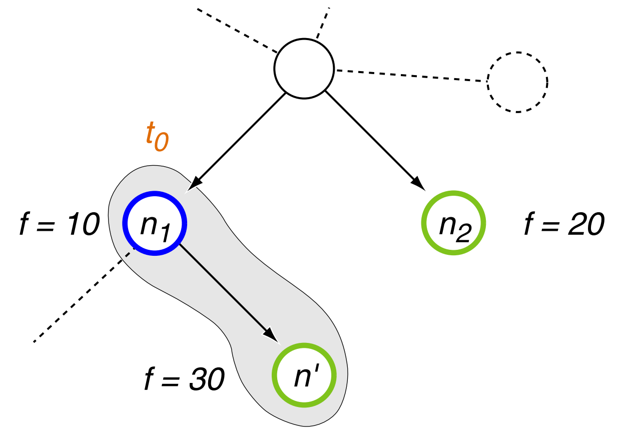
Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

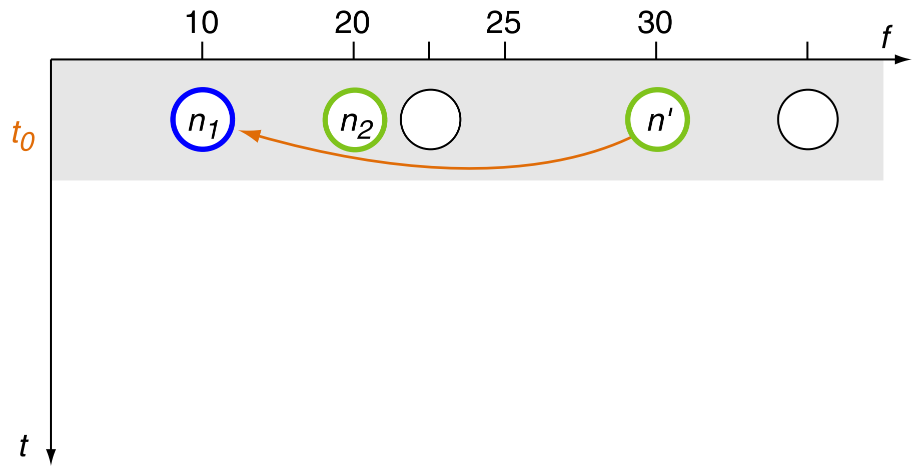
Case 1: n' is still on OPEN.

```
5.    FOREACH $n'$ IN $successors(n)$
        ...
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
          $n'_{old} = retrieve(n', \text{OPEN} \cup \text{CLOSED})$;
          IF ($f(n') < f(n'_{old})$)
          THEN   // The state of $n'$ is reached via a cheaper path.
            $update\_backpointer(n'_{old}, n)$;
            IF $n'_{old} \in$ CLOSED THEN $remove(n'_{old}, \text{CLOSED})$; $insert(n'_{old}, \text{OPEN})$; ENDIF
          ENDIF
```

State-space:



OPEN ∪ CLOSED list:



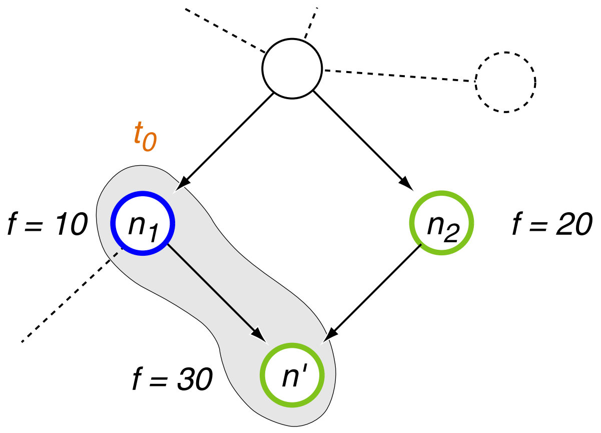
Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

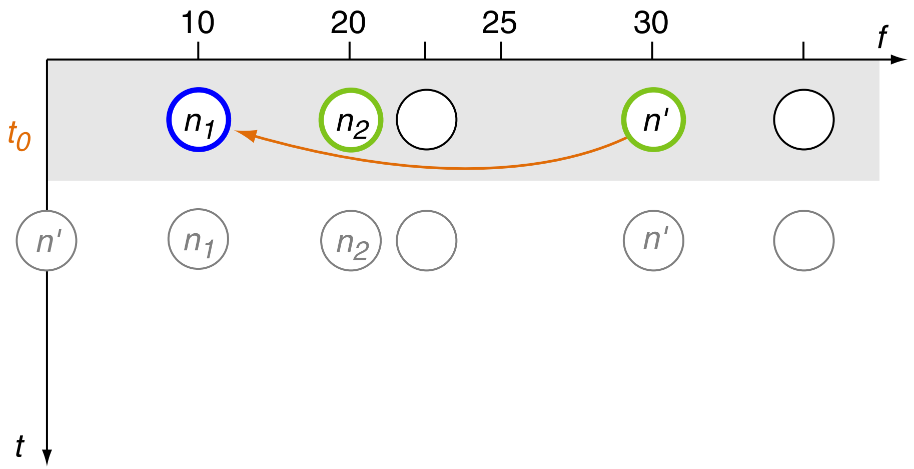
Case 1: n' is still on OPEN.

```
5.    FOREACH $n'$ IN successors($n$)
        ...
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
          $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
          IF ($f(n') < f(n'_{old})$)
          THEN   // The state of $n'$ is reached via a cheaper path.
            update_backpointer($n'_{old}$, $n$);
            IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
          ENDIF
```

State-space:



OPEN $\cup$ CLOSED list:



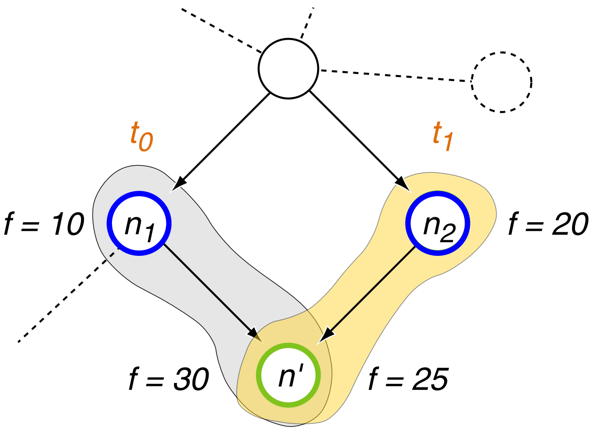
Best-First Search for State-Space Graphs

Re-evaluation of a Node n'

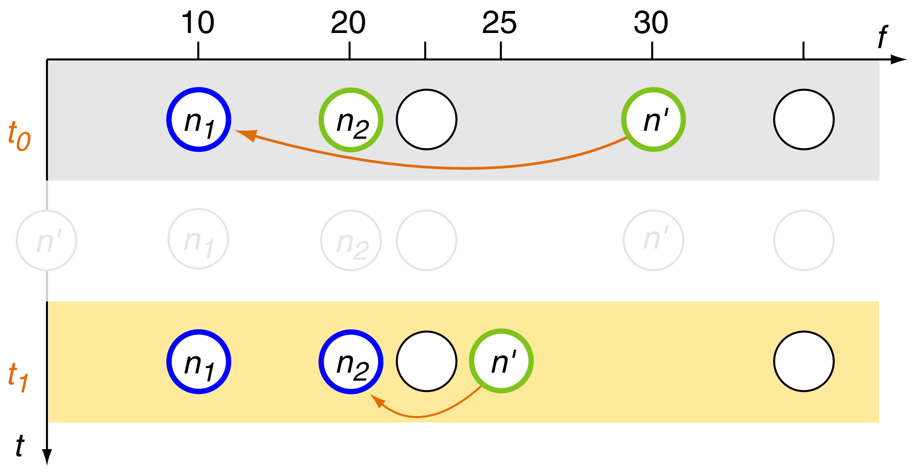
Case 1: n' is still on OPEN.

```
5.    FOREACH $n'$ IN $successors(n)$
        ...
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
          $n'_{old} = retrieve(n', \text{OPEN} \cup \text{CLOSED})$;
          IF ($f(n') < f(n'_{old})$)
          THEN  // The state of $n'$ is reached via a cheaper path.
            $update\_backpointer(n'_{old}, n)$;
            IF $n'_{old} \in$ CLOSED THEN $remove(n'_{old}, \text{CLOSED})$; $insert(n'_{old}, \text{OPEN})$; ENDIF
          ENDIF
```

State-space:



OPEN ∪ CLOSED 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 ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
          $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
          IF ($f(n') < f(n'_{old})$)
          THEN   // The state of $n'$ is reached via a cheaper path.
            update_backpointer($n'_{old}$, $n$);
            IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
          ENDIF
```

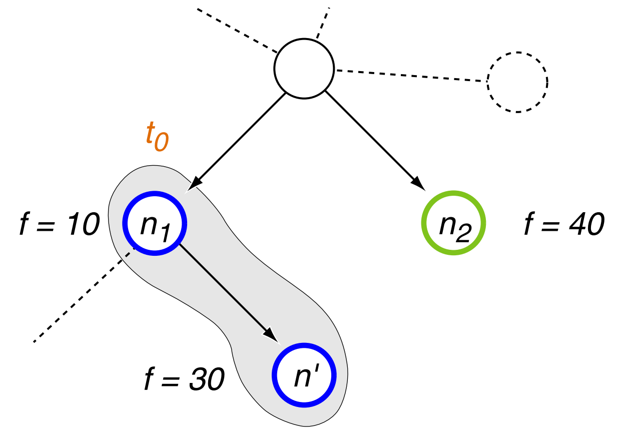
Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

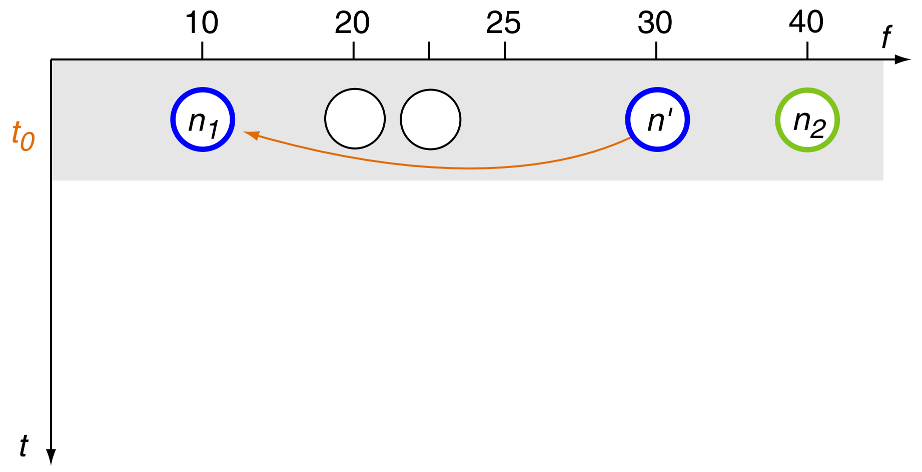
Case 2: n' is already on CLOSED.

```
5.    FOREACH $n'$ IN successors$(n)$
        ...
        IF $(n' \notin$ OPEN AND $n' \notin$ CLOSED$)$
        ...
          $n'_{old}$ = retrieve$(n'$, OPEN $\cup$ CLOSED$)$;
          IF $(f(n') < f(n'_{old}))$
          THEN   // The state of $n'$ is reached via a cheaper path.
            update_backpointer$(n'_{old}, n)$;
            IF $n'_{old} \in$ CLOSED THEN remove$(n'_{old}$, CLOSED$)$; insert$(n'_{old}$, OPEN$)$; ENDIF
          ENDIF
```

State-space:



OPEN $\cup$ CLOSED list:



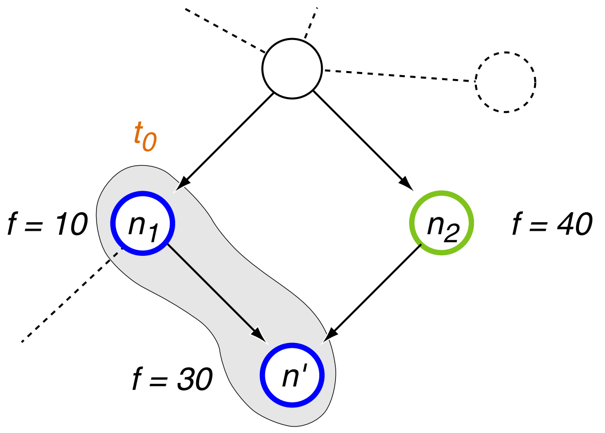
Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

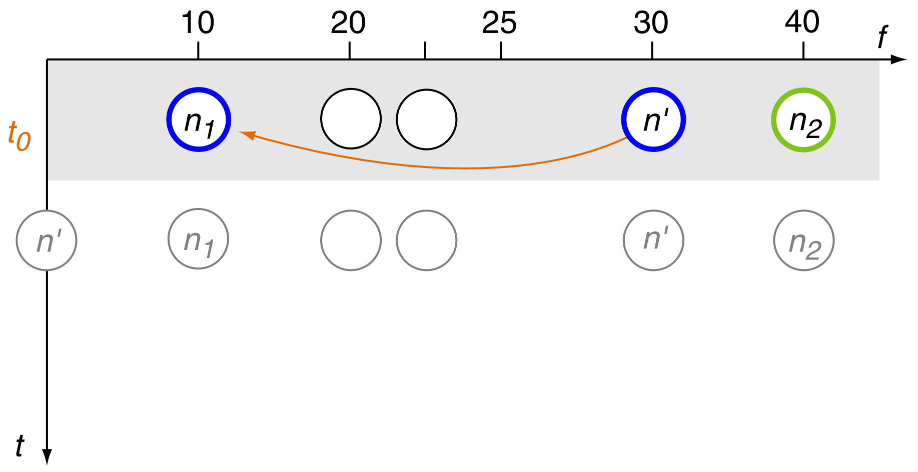
Case 2: n' is already on CLOSED.

```
5.    FOREACH $n'$ IN successors($n$)
        ...
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
         $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
         IF ($f(n') < f(n'_{old})$)
         THEN  // The state of $n'$ is reached via a cheaper path.
           update_backpointer($n'_{old}$, $n$);
           IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
         ENDIF
```

State-space:



OPEN $\cup$ CLOSED list:



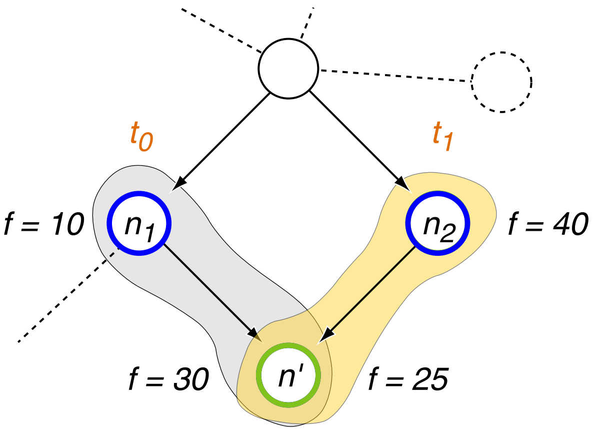
Best-First Search for State-Space Graphs

Re-evaluation of a Node n' (continued)

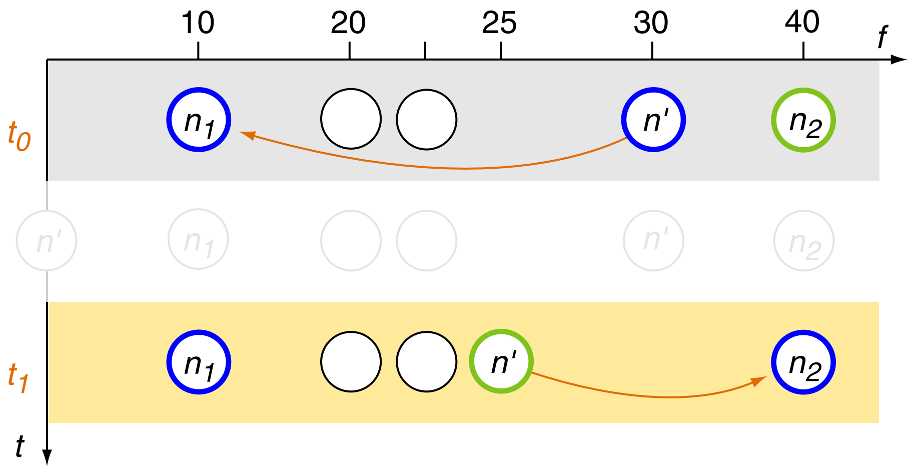
Case 2: n' is already on CLOSED.

```
5.    FOREACH $n'$ IN successors($n$)
        ...
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        ...
          $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
          IF ($f(n') < f(n'_{old})$)
          THEN  // The state of $n'$ is reached via a cheaper path.
            update_backpointer($n'_{old}$, $n$);
            IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
          ENDIF
```

State-space:



OPEN ∪ CLOSED 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 done with care as 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 backpointer 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 ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
➜       THEN   // $n'$ encodes a new state.
          insert($n'$, OPEN)
        ...
```

Possible reasons:

1. There is no occurrence check. (State-space graph G is modeled as a tree.)
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 ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
➜       THEN   // $n'$ encodes a new state.
          insert($n'$, OPEN)
        ...
```

Possible reasons:

1. There is no occurrence check. (State-space graph G is modeled as a tree.)
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 ($\star(n) =$ *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 ($\star(n) =$ *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*.

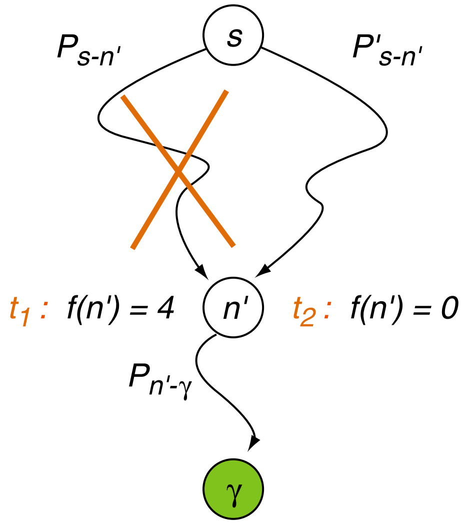
Best-First Search for State-Space Graphs [BF, GBF, GBF$\star$, BFS]

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

1.  insert($s$, OPEN);
2.  LOOP
3.    IF (OPEN $= \emptyset$) THEN RETURN(Fail);
4.    $n$ = min(OPEN, $f$);
➜     IF $\star(n)$ THEN RETURN($n$);    // Delayed termination.
      remove($n$, OPEN); push($n$, CLOSED);
5.    FOREACH $n'$ IN successors($n$) DO
        add_backpointer($n'$, $n$);
➜       ~~IF $\star(n')$ THEN RETURN($n'$);~~
        IF ($n' \notin$ OPEN AND $n' \notin$ CLOSED)
        THEN
          insert($n'$, OPEN)
        ELSE
          $n'_{old}$ = retrieve($n'$, OPEN $\cup$ CLOSED);
          IF ($f(n') < f(n'_{old})$)
          THEN   // The state of $n'$ is reached via a cheaper path.
            update_backpointer($n'_{old}$, $n$);
            IF $n'_{old} \in$ CLOSED THEN remove($n'_{old}$, CLOSED); insert($n'_{old}$, OPEN); ENDIF
          ENDIF
        ENDIF
      ENDDO
6.  ENDLOOP
```

Best-First Search for State-Space Graphs

Irrevocable Path Discarding in Algorithm BF



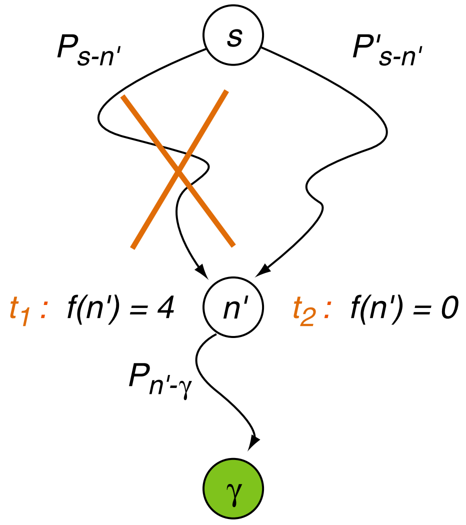
Irrevocability is crucial (solutions missed) if constraints on solution paths take into account *global properties* of the path.

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 constraint satisfaction problems, if the following equivalence holds:

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

   $\Leftrightarrow$

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

2. For optimization problems, if, additionally, for alternative solution bases the order w.r.t. cost estimations is independent of their shared continuation.