Chapter S:III

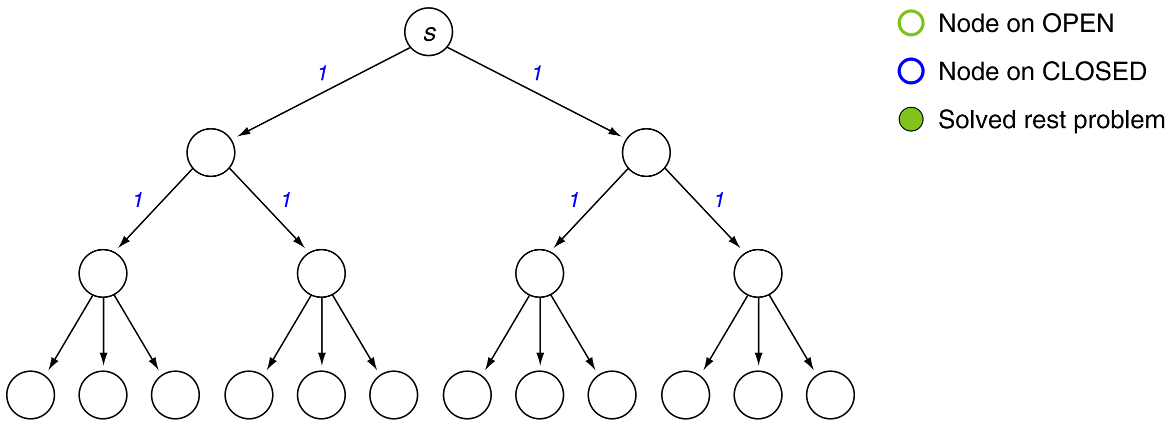
III. Informed Search

BF* Variants

For trees G: Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.

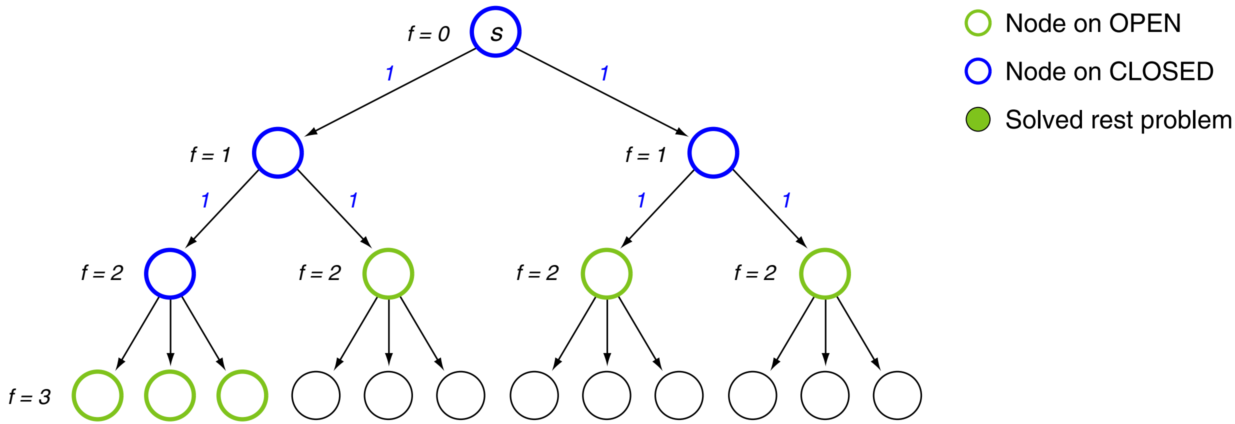
BF* Variants

For trees G: Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.



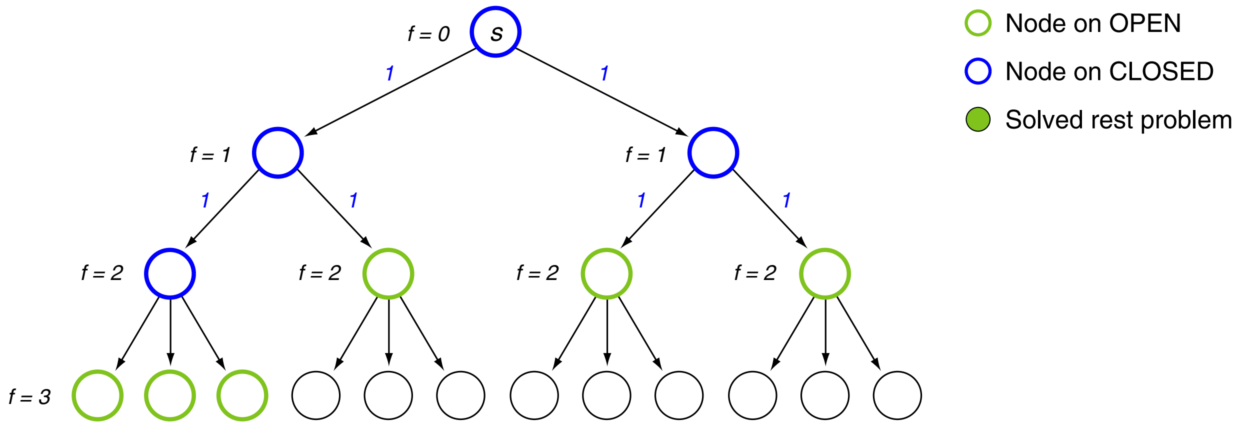
BF* Variants

For trees G: Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.



BF* Variants

For trees G: Breadth-first search is a special case of A*, where $h = 0$ and $c(n, n') = 1$ for all successors n' of n.



Proof (sketch)

1. $g(n)$ defines the depth of n (consider path from n to s).
2. $f(n) = g(n)$.
3. Breadth-first search $\equiv$ the depth difference of nodes on OPEN is ≤ 1.
4. Assumption: Let n_1, n_2 be on OPEN, having a larger depth difference: $f(n_2) - f(n_1) > 1$.
5. $\Rightarrow$ For the direct predecessor n_0 of n_2 holds: $f(n_0) = f(n_2) - 1 > f(n_1)$.
6. $\Rightarrow$ n_1 must have been expanded before n_0 (consider minimization of f under A*).
7. $\Rightarrow$ n_1 must have been deleted from OPEN. Contradiction to 4.

BF* Variants

For trees G: Uniform-cost search is a special case of A*, where $h = 0$.

Proof (sketch)

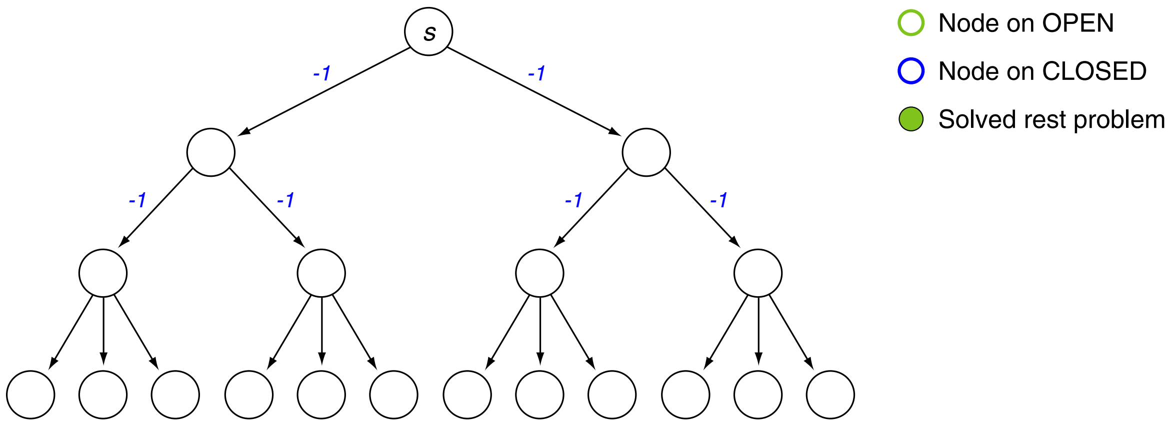
See lab class.

BF* Variants

For trees G: Depth-first search is a special case of A*, where $h = 0$ and $c(n, n') = -1$ for all successors n' of n.

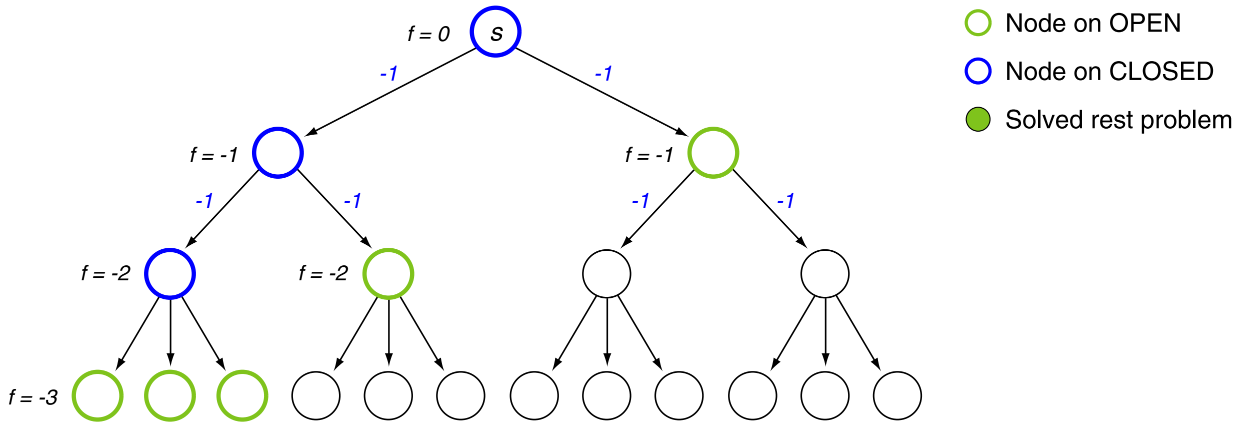
BF* Variants

For trees G: Depth-first search is a special case of A*, where $h = 0$ and $c(n, n') = -1$ for all successors n' of n.



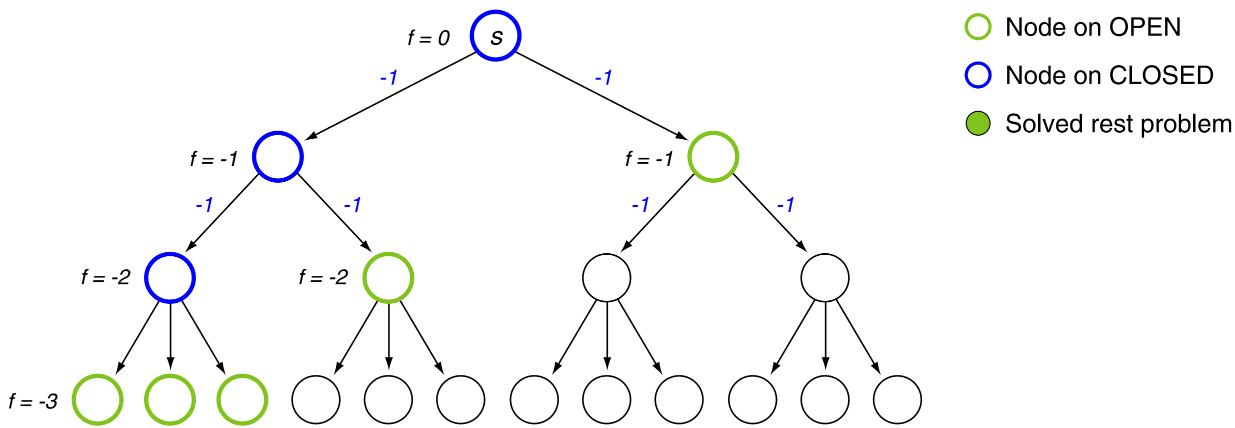
BF* Variants

For trees G: Depth-first search is a special case of A*, where $h = 0$ and $c(n, n') = -1$ for all successors n' of n.



BF* Variants

For trees G: Depth-first search is a special case of A*, where $h = 0$ and $c(n, n') = -1$ for all successors n' of n.



Proof (sketch)

1. $f(n') < f(n) \;\Rightarrow n'$ was inserted on OPEN after n.
   $f(n') \leq f(n) \;\Leftrightarrow n'$ was inserted on OPEN after n.
2. Depth-first search $\equiv$ the most recently inserted node on OPEN is expanded.
3. Let n_2 be the most recently inserted node on OPEN.
4. Assumption: Let n_1 have been expanded before $n_2 \;\wedge\; f(n_1) \neq f(n_2)$.
5. $\Rightarrow f(n_1) < f(n_2)$ (consider minimization of f under Z*).
6. $\Rightarrow n_1$ was inserted on OPEN after n_2.
7. $\Rightarrow n_2$ is not the most recently inserted node on OPEN. Contradiction to 3.

Remarks:

- Of course, depth-first search can also be seen as a special case of A*, where all edge cost values are -1. The recursive cost function $C_P(n)$ defined by

$$C_P(n) = \begin{cases} 0 & n \text{ is leaf in } P \\ -1 + C_P(n') & n \text{ is inner node in } P \text{ and } n' \text{ direct successor of } n \text{ in } P \end{cases}$$

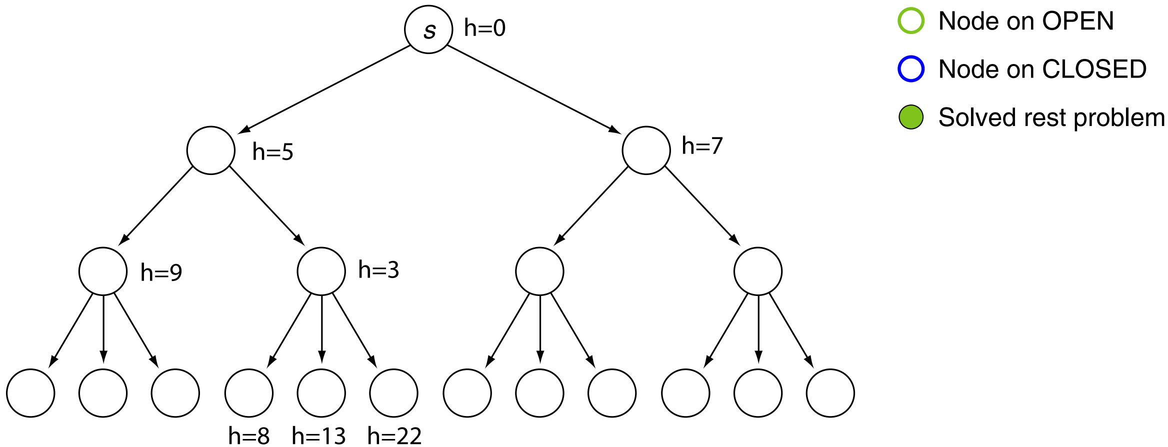
  allows the computation of f directly, i.e., without using local properties like edge cost values. The equivalent recursive definition of f by $f(n') = f(n) - 1$, $f(s) = 0$ shows that f values can be propagated downwards resulting in a local computation of f values in Z* analogously to the computation in A*.

BF* Variants

Greedy best-first search is a special case of BF*, where $f(n) = h(n)$, for all nodes n.

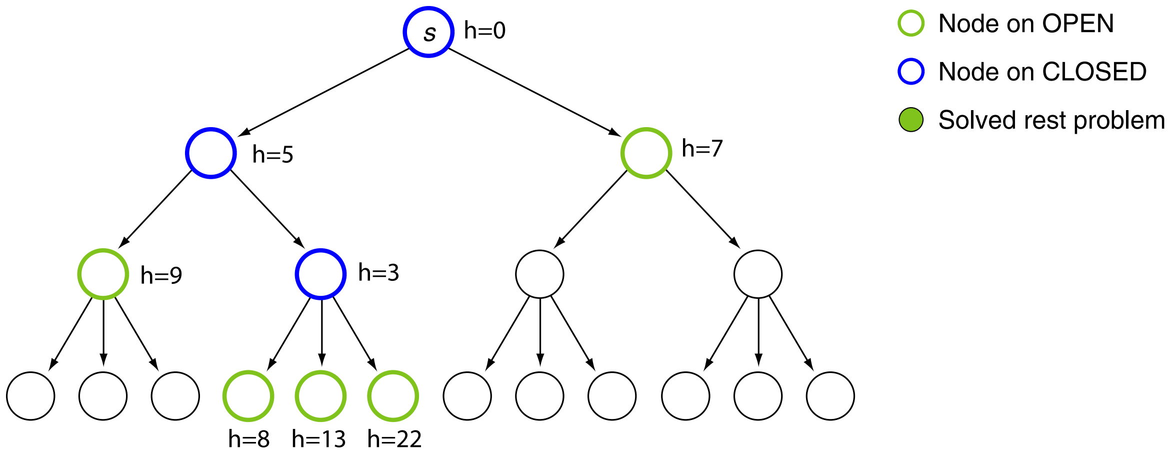
BF* Variants

Greedy best-first search is a special case of BF*, where $f(n) = h(n)$, for all nodes n.



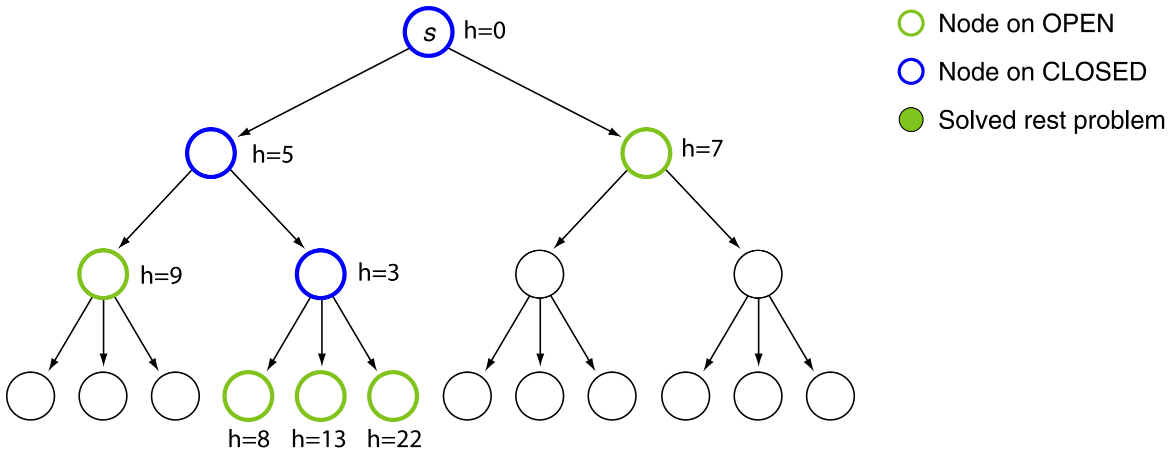
BF* Variants

Greedy best-first search is a special case of BF*, where $f(n) = h(n)$, for all nodes n.



BF* Variants

Greedy best-first search is a special case of BF*, where $f(n) = h(n)$, for all nodes n.



- Greedy best-first search is *greedily* going for early termination (assuming small h-values indicate small remaining problem, i.e., trusting in the *Small-is-Quick Principle*).
- Greedy best-first search represents an abuse of the evaluation function: although it is easy to define f by a recursive cost function, the path cost concept used in computation of h is not the path cost estimated by f.
- Greedy best-first search can take early found alternatives in OPEN into account, if h-values were misleading.
- The name "Hill-Climbing" is often used as synonym for "Greedy (Best-First) Search", if no alternatives are stored in OPEN.

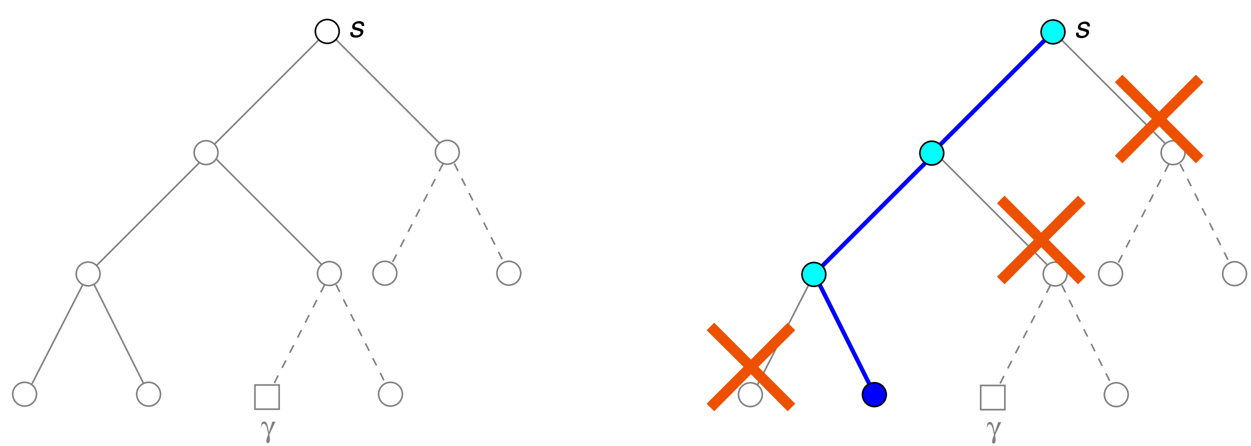
BF* Variants

OPEN List Size Restriction:: Hill-Climbing (HC)

Hill-climbing best-first search is an informed, irrevocable search strategy.

HC characteristics:

- local or greedy optimization:
  take the direction of steepest ascend (merit) / steepest descend (cost)
- "never look back" :
  alternatives are not remembered ➜ no OPEN/CLOSED lists needed
- usually low computational effort
- a strategy that is often applied by humans: *online* search



Remarks:

- Originally, hill-climbing was formulated as a local search strategy (i.e., hill-climbing is working on solution candidates, whereas best-first algorithms are working on solution bases that are partial solution candidates which represent sets of solution candidates, any continuation of the initial sequence of solution steps).

  Restarts with random starting points, which are often used to avoid local optima in hill-climbing, are not applicable to the best-first approach, since, except for the treatment of ties, the exploration of the search space graph starting from the starting node s is fixed. For the solution of optimization problems, the evaluation of solution bases must guarantee that an optimal solution can be found by expanding the most promising solution basis.

- Simulating hill-climbing in best-first search:

  For the 8-Queens problem, a solution candidate is a positioning of eight queens on the board. The cost of a positioning is computed from the number of attacks between queens. By moving single queens on the board to adjacent fields, the positioning can be changed.

  Hill-climbing will evaluate the neighborhood of a solution candidate and switch to the most promising one.

  Obviously, this neighborhood could be computed by a *successors*$(.)$ function and quality/cost values can be assigned by an evaluation function f. (Applied to solution candidates, f does not employ estimations.) Restarts can be simulated by using different initial problems.

- Implementing hill-climbing as best-first search:

  A size limited priority list can be used as OPEN in order to implement the hill-climbing strategy. Only the solution bases that performed best with respect to f are stored in OPEN. The "never look back" idea is realized when using a size limit of 1.

BF* Variants

OPEN List Size Restriction: Best-First Beam Search [Rich & Knight 1991]

Characteristics:

- Best-first search is used with an OPEN list of limited size k (the beam-width).
- If OPEN exceeds its size limit, nodes with worst f-values are discarded until size limit is adhered to.
- Hill-climbing best-first search is best-first beam search with $k = 1$ and checking $f(n') < f(n)$.

OPEN List Size Restriction: Breadth-First Beam Search [Lowerre 1976, Bisiani 1981]

Characteristics:

- All nodes of the current level (initially only s) are expanded.
- Only the best of all these successors are kept and used for the next level.

  (For the selection of these nodes "a threshold of acceptability" can be defined [Lowerre 1976]. A simpler option is to restrict the next level to at most k nodes with best f-values.)

Operationalization for both variants:

- A *cleanup_closed* function is needed to prevent CLOSED from growing uncontrollably.

BF* Variants

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

Input: s. Start node representing the initial state (problem) in G.

...

k. Limit for the size of OPEN.

Output: A node γ representing a solution path for s in G or the symbol *Fail*.

```
Beam-BF*(s, successors, ⋆, f, k)   // A OPEN size limited variant of Basic-BF*.
  1.  s.parent = null;  add(s, OPEN, f(s));    // Store s on f-sorted OPEN.
  2.  LOOP
  3.    IF (OPEN == ∅) THEN RETURN(Fail);
  4.    n = min(OPEN, f);    // Find most promising (cheapest) solution base.
        IF ⋆(n) THEN RETURN(n);
        remove(n, OPEN);  add(n, CLOSED);
  5.    FOREACH n' IN successors(n) DO    // Expand n.
          n'.parent = n;
          IF ( size(OPEN) == k )
          THEN
            n_0 = max(OPEN, f);    // Find least promising solution base.
            IF ( f(n') < f(n_0) ) THEN remove(n_0, OPEN);
          ENDIF
          IF ( size(OPEN) < k ) THEN add(n', OPEN, f(n'));
        ENDDO
  6.  ENDLOOP
```

Remarks:

- In hill-climbing best-first search, it is often sufficient to use the heuristic function h instead of f. In best-first beam search, this is not meaningful, since in this case the guiding function would no longer be directly related to solution path cost.

- In the introduction, heuristic functions are given for some of the example problems. Informally a hill-climbing approach was described for using these heuristic functions in a search for solutions. The above version of hill-climbing can be directly applied to these problems.
  - For the 8-Queens problem, the heuristic functions can be seen as describing a merit (i.e., the potential of positioning further queens): the higher, the better. Use $f := -h$
  - For the 8-Puzzle problem the heuristic functions describe a cost (i.e., the distance to the target configuration on the board): the lower, the better. Use $f := h$
  - For the map problem and for TSP, the heuristic functions compute the Euclidean distance to the target position, resp. the sum cost of a cost-minimal spanning tree / degree-2 graph. Again, these values can be seen as cost values. Use $f := -h$
    However, the distance of the edge traversed last should also be taken into account, i.e., for a successor n' the value $h(n')$ is compared to $h(n)$ for the parent node n, but comparisons between successors n', n'' use $c(n,n') + h(n')$ and $c(n,n'') + h(n'')$.

Remarks:

- For $k = 1$, best-first beam search is similar to the above described version of hill-climbing search. However, there is an important difference:

  Since f-values of successors are not compared to the f-value of the parent, best-first beam search will continue search even in case a local optimum is found.

  Therefore, in hill-climbing best-first search we would use

```
5.   FOREACH n' IN successors(n) DO     // Expand n.
       n'.parent = n;
       IF ( ( size(OPEN) == 1 ) AND ( f(n') < f(n) ) )
       THEN
         n_0 = max(OPEN, f);     // Find least promising solution base.
         IF ( f(n') < f(n_0) ) THEN remove(n_0, OPEN);
       ENDIF
       IF ( ( size(OPEN) < 1 ) AND ( f(n') < f(n) ) ) THEN add(n', OPEN, f(n'));
     ENDDO
```

- For $k = 1$, best-first beam search is an informed, irrevocable search strategy.

- Similar to hill-climbing, a low-quality evaluation function f may lead the search into parts of the search space graph that do not contain solution paths. Completeness is endangered.

Hybrid Strategies

Spectrum of Search Strategies

The search strategies

- Hill-climbing best-first
- Informed backtracking
- Best-first search

form the extremal points within the spectrum of search strategies, based on the following dimensions:

R **Recovery.**
How many previously suspended alternatives (nodes) are reconsidered after finding a dead end?

S **Scope.**
How many alternatives (nodes) are considered for each expansion?

Hybrid Strategies

Spectrum of Search Strategies

The search strategies

- ❑ Hill-climbing best-first — irrevocable decisions, consideration of newest alternatives
- ❑ Informed backtracking — tentative decisions, consideration of newest alternatives
- ❑ Best-first search — tentative decisions, consideration of all alternatives

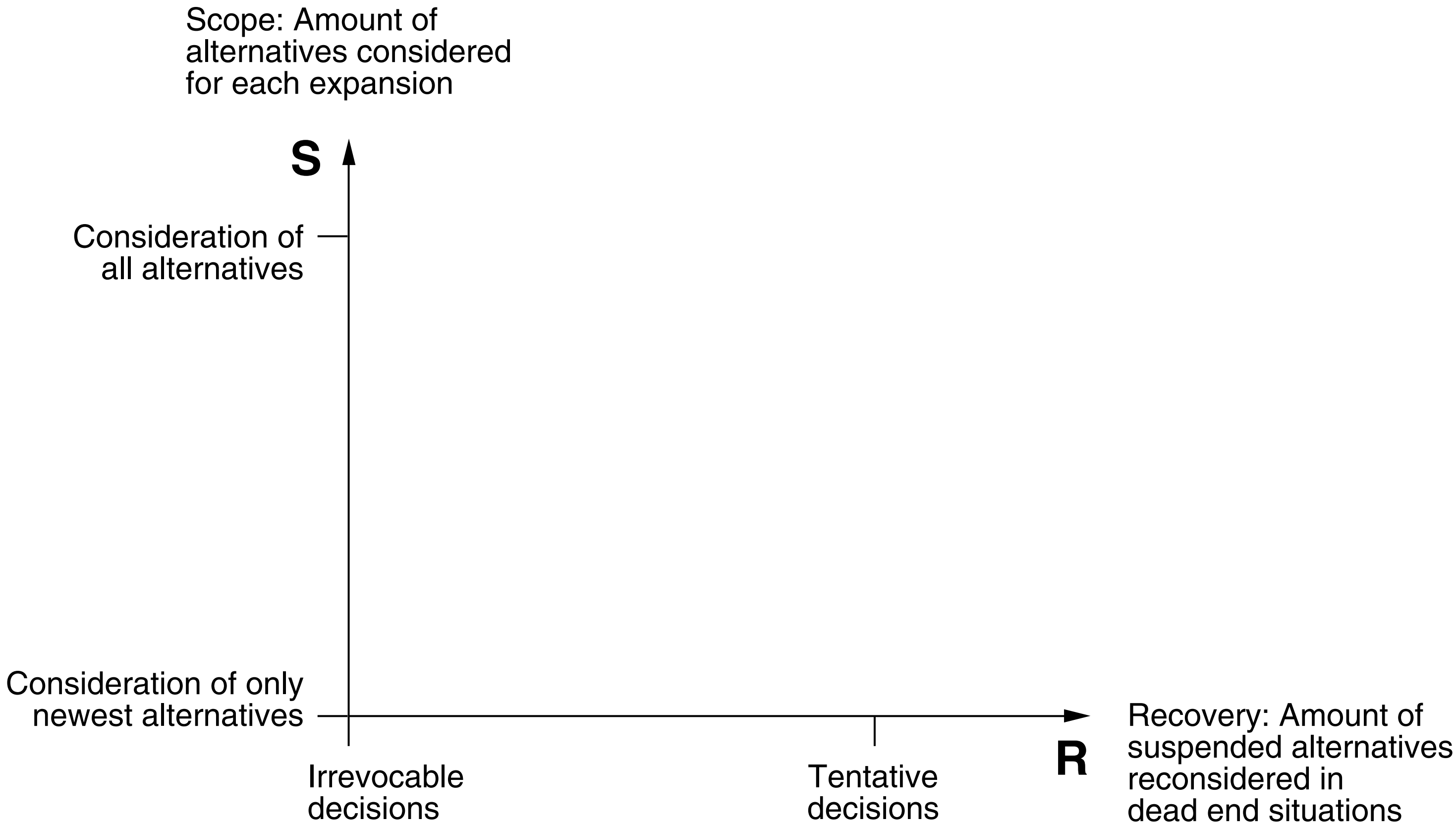
form the extremal points within the spectrum of search strategies, based on the following dimensions:

R Recovery.
How many previously suspended alternatives (nodes) are reconsidered after finding a dead end?

S Scope.
How many alternatives (nodes) are considered for each expansion?

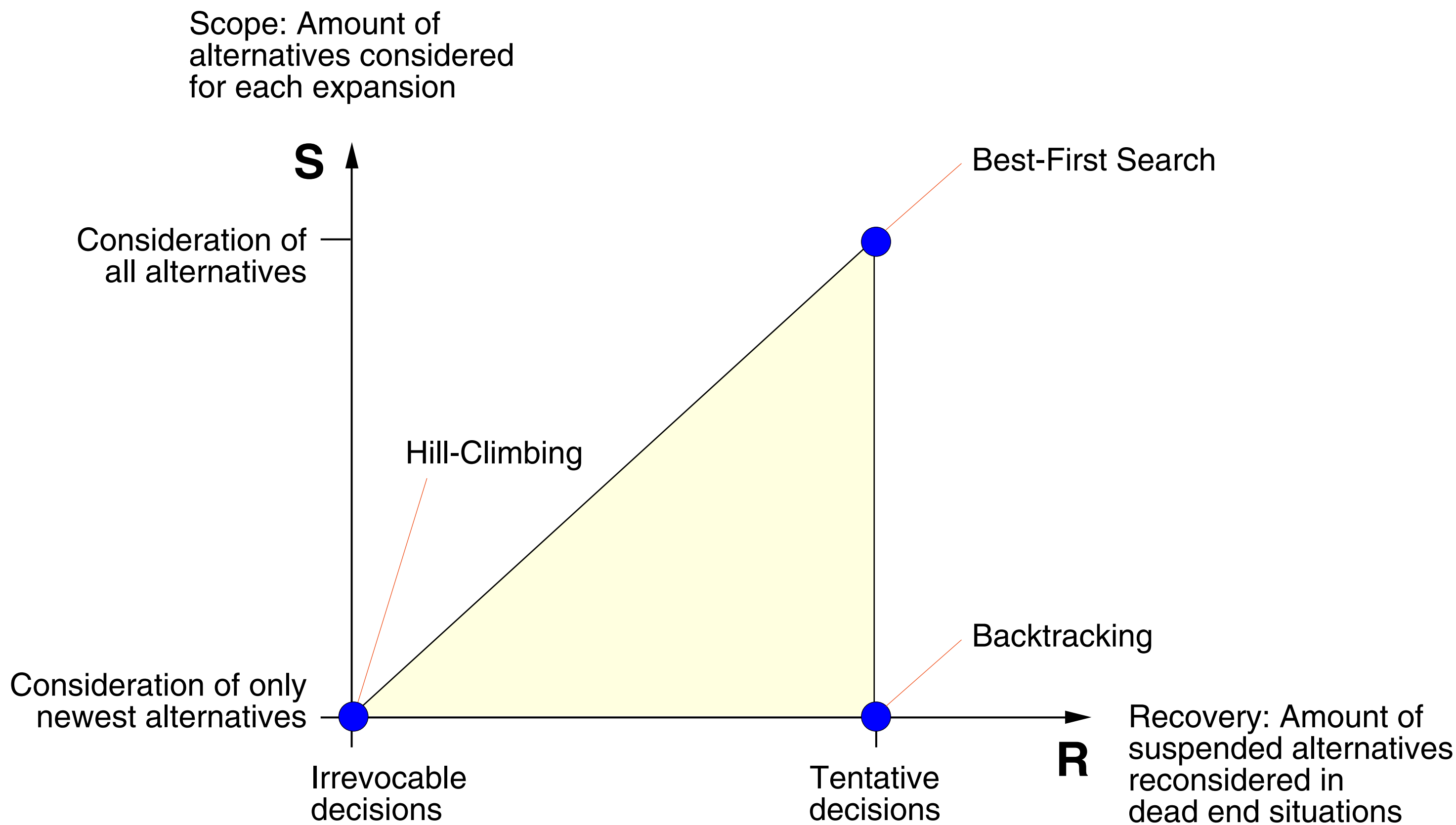
Hybrid Strategies

Spectrum of Search Strategies



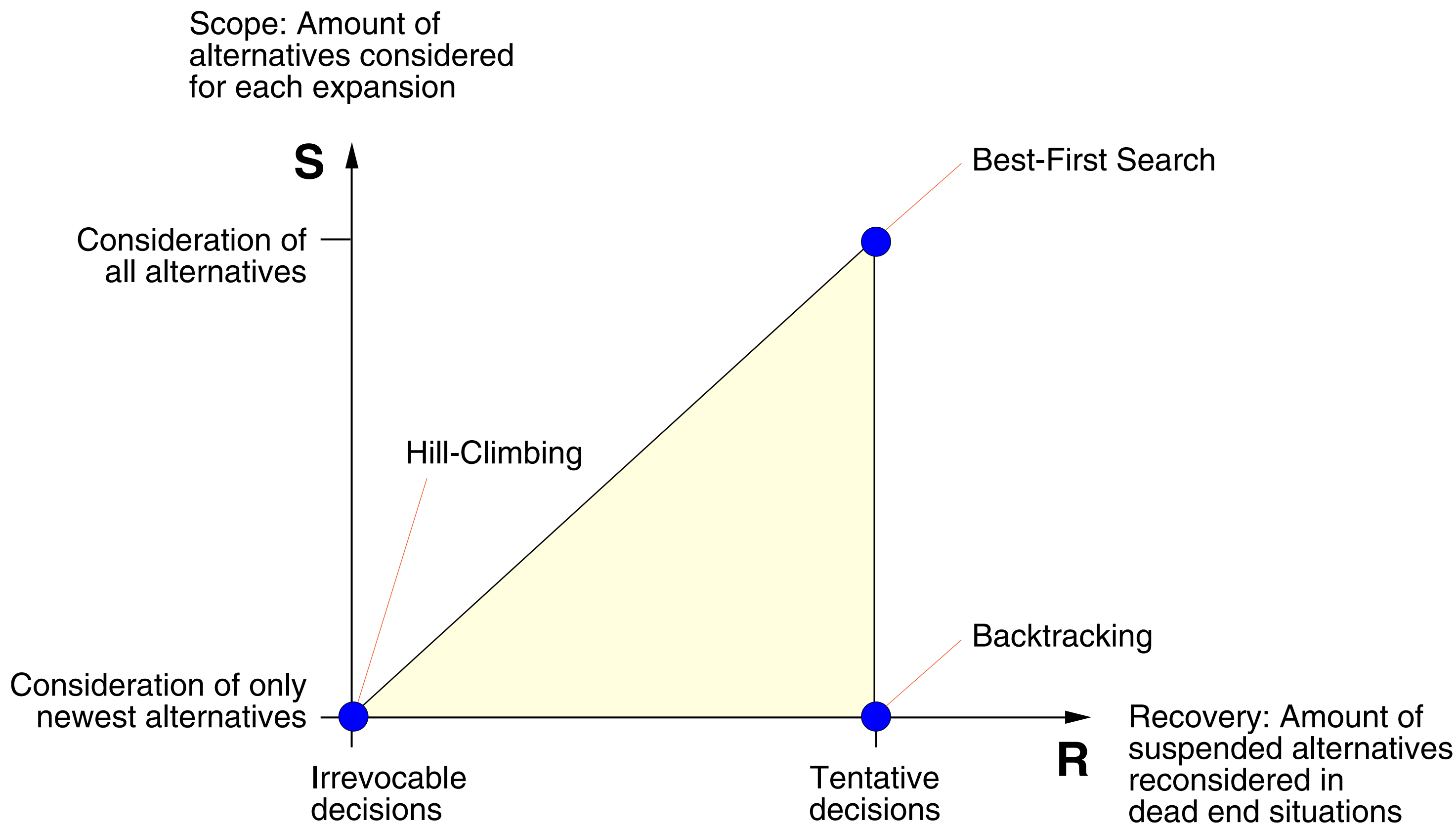
Hybrid Strategies

Spectrum of Search Strategies



Hybrid Strategies

Spectrum of Search Strategies



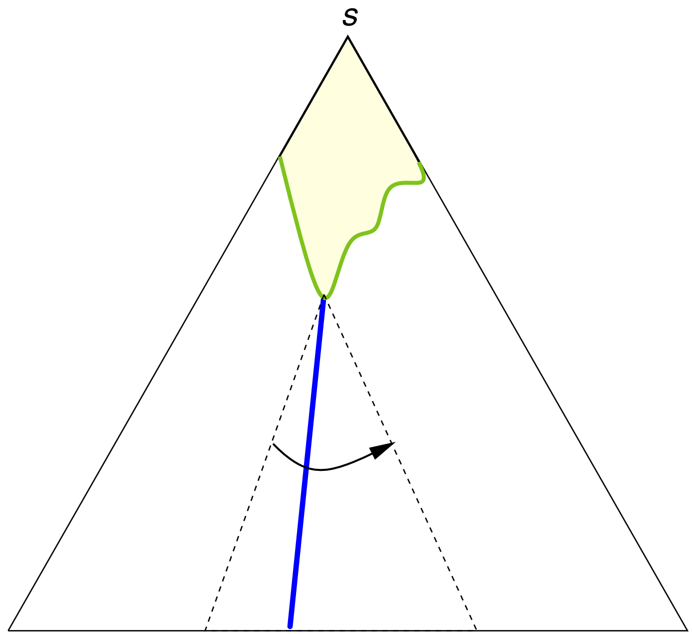
- The large scope of best-first search requires a high memory load.
- This load can be reduced by mixing it with backtracking.

Remarks:

- ❑ Recall that the memory consumption of best-first search is an (asymptotically) exponential function of the search depth.
- ❑ Hill-climbing is the most efficient strategy, but its effectiveness (solution quality) can only be guaranteed for problems that can be solved with a greedy approach.
- ❑ Informed backtracking (i.e., generate successors with some quality ordering) requires not as much memory as best-first search, but usually needs more time as its scope is limited.
- ❑ Without a highly informed heuristic h, the degeneration of best-first strategies down to a uniform-cost search is typical and should be expected as the normal case.

Hybrid Strategies

Strategy 1: BF at Top



Characteristics:

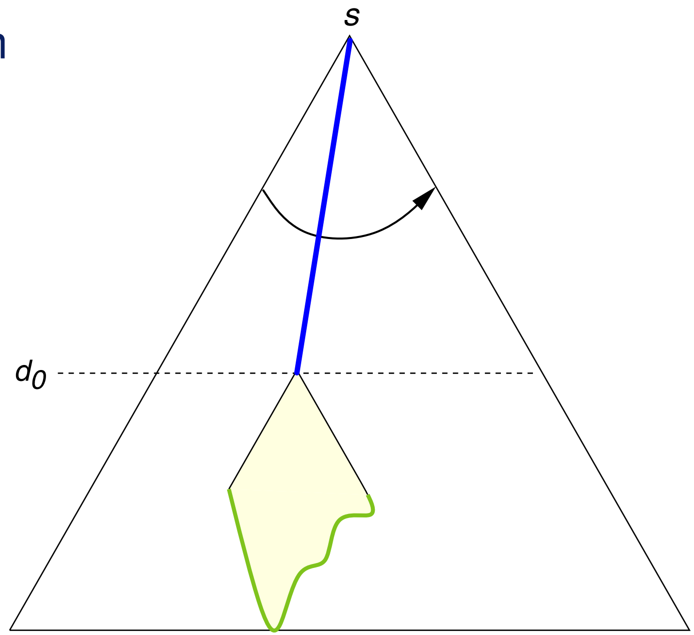
- Best-first search is applied at the top of the search space graph.
- Backtracking is applied at the bottom of the search space graph.

Operationalization:

1. Best-first search is applied until a memory allotment of size M_0 is exhausted.
2. Then backtracking starts with a most promising node n' on OPEN.
3. If backtracking fails, it restarts with the next most promising OPEN node.

Hybrid Strategies

Strategy 2: BF at Bottom



Characteristics:

- Backtracking is applied at the top of the search space graph.
- Best-first search is applied at the bottom of the search space graph.

Operationalization:

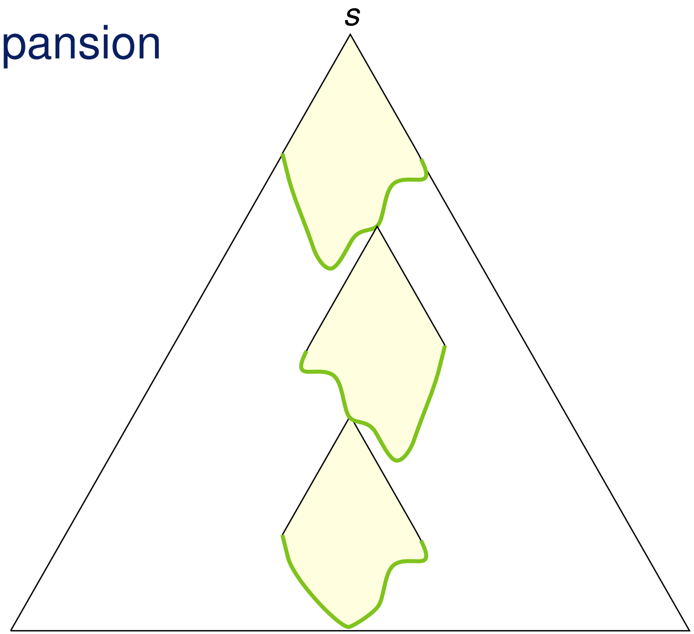
1. Backtracking is applied until the search depth bound d_0 is reached.
2. Then best-first search starts with the node at depth d_0.
3. If best-first search fails, it restarts with the next node at depth d_0 found by backtracking.

Remarks:

- The depth bound d_0 in Strategy 2 must be chosen carefully to avoid best-first search running out of memory. Hence, this strategy is more involved than Strategy 1 where the switch between best-first search and backtracking is triggered by the exhausted memory.
- If a sound depth bound d_0 is available, Strategy 2 (best-first search at bottom) is usually superior to Strategy 1 (best-first search at top). Q. Why?

Hybrid Strategies

Strategy 3: Extended Expansion



Characteristics:

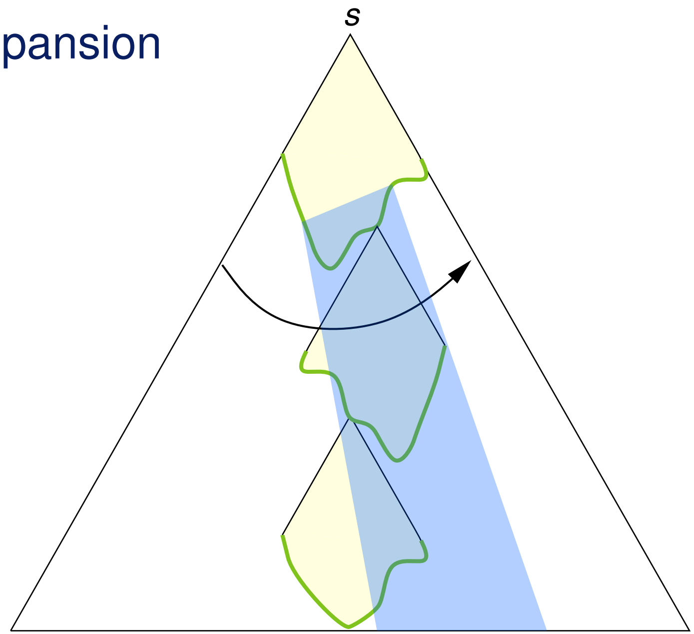
- Best-first search acts locally to generate a restricted number of promising nodes.
- Informed depth-first search acts globally, using best-first as an “extended node expansion”.

Operationalization:

1. An informed depth-first search selects the nodes n for expansion.
2. But a best-first search with a memory allotment of size M_0 is used to “expand” n.
3. The nodes on OPEN are returned to the depth-first search as “direct successors” of n.

Hybrid Strategies

Strategy 3: Extended Expansion



Characteristics:

- Best-first search acts locally to generate a restricted number of promising nodes.
- Informed depth-first search acts globally, using best-first as an "extended node expansion".

Operationalization:

1. An informed depth-first search selects the nodes n for expansion.
2. But a best-first search with a memory allotment of size M_0 is used to "expand" n.
3. The nodes on OPEN are returned to the depth-first search as "direct successors" of n.

Remarks:

- Strategy 3 is an informed depth-first search whose node expansion is operationalized via a memory-restricted best-first search.
- Q. What is the asymptotic memory consumption of Strategy 3 in relation to the search depth?

Hybrid Strategies

Strategy 4: Focal Search [Ibaraki 1978]

Characteristics:

- An informed depth-first search is used as basic strategy.
- Nodes are selected from newly generated nodes and the best nodes encountered so far.

Operationalization:

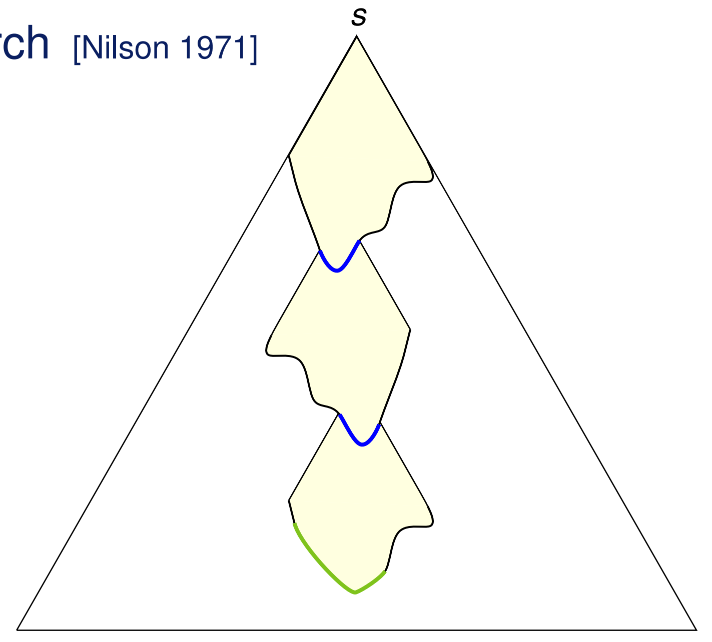
- The informed depth-first search expands the cheapest node n from its list of alternatives.
- For the next expansion, it chooses from the newly generated nodes and the k best nodes (without n) from the previous alternatives.

Remarks:

- For $k = 0$ this is identical to an informed depth-first search.
- For $k = \infty$ this is identical to a best-first search.
- Memory consumption (without proof): $O(b \cdot d^{k+1})$, where b denotes the branching degree and d the search depth.
- An advantage of Strategy 4 is that its memory consumption can be controlled via the single parameter k.
- Differences to beam search:
  - In focal search no nodes are discarded. Therefore, focal search will never miss a solution.
  - In best-first beam search the OPEN list is of limited size.

Hybrid Strategies

Strategy 5: Staged Search [Nilson 1971]



Characteristics:

- ❑ Best-first search acts locally to generate a restricted number of promising nodes.
- ❑ Hill-climbing acts globally, but by retaining a set of nodes.

Operationalization:

1. Best-first search is applied until a memory allotment of size M_0 is exhausted.
2. Then only the cheapest OPEN nodes (and their pointer-paths) are retained.
3. Best-first search continues until Step 1. is reached again.

Remarks:

- Staged search can be considered as a combination of best-first search and hill-climbing. While a pure hill-climbing discards all nodes except one, staged search discards all nodes except a small subset.
- Staged search addresses the needs of extreme memory restrictions and tight runtime bounds.
- Recall that the Strategies 1–4 are complete with regard to recovery, but that Strategy 5, Hill Climbing, and Best-First Beam Search are not.

Hybrid Strategies

Strategy 6: Iterative Deepening A* – IDA* [Korf 1985]

Characteristics:

- Depth-first search is used in combination with an iterative deepening approach for f-values.
- Nodes are considered only if their f-values do not exceed a given threshold.

Operationalization of IDA*:

1. f-*bound* is initialized with $f(s)$.
2. Calling f-limited-DFS:
   In depth-first search, only nodes are considered with $f(n) \leq f$-*bound*.
   The value *min*-f-*over* which is the minimum of all f-values that exceeded the current threshold is also returned (call by reference).
3. If depth-first search fails, f-*bound* is increased to *min*-f-*over* and f-limited-DFS is rerun.

Hybrid Strategies

f-value-limited DFS

An f-value limit f-*bound* helps to avoid infinite paths, a call by reference parameter allows the interpretation of negative results ((min-f-$over$ < f-*bound*) ⇒ no solution available).

```
f-limited-DFS(s, successors, ⋆, c, h, f-bound, min-f-over)     // DL-DFS variant.
 1. s.parent = null;  push(s, OPEN);  g(s) = 0;  f(s) = g(s) + h(s);  min-f-over = f(s);
 2. LOOP
 3.   IF (OPEN == ∅) THEN RETURN(Fail);
 4.   n = pop(OPEN);  add(n, CLOSED);
 5.   IF (f(n) > f-bound)   // Do not include nodes with higher f-values.
      THEN
        IF ( min-f-over > f-bound ) THEN min-f-over = min(f(n), min-f-over);
        ELSE min-f-over = f(n);
        ENDIF
        cleanup_closed();    // Remove unreferenced nodes from CLOSED.
      ELSE;
        FOREACH n' IN successors(n) DO    // Expand n.
          n'.parent = n;  g(n') = g(n) + c(n, n');  f(n') = g(n') + h(n');
          IF ⋆(n') THEN RETURN(n');
          push(n', OPEN);    // Add node at the front of OPEN.
        ENDDO
        IF (successors(n) == ∅) THEN cleanup_closed();    // Remove dead ends.
      ENDIF
 6. ENDLOOP
```

Hybrid Strategies

Algorithm: ID-A* (Iterative-Deepening-A*) (Compare f-L-DFS, ID-DFS.)

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

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

ID-A*(s, *successors*, $\star$, $\perp$, *initial-f-bound*, *f-bound*) // A variant of ID-DFS.

1. *current-f-bound* $=$ *initial-f-bound*; *min-f-over* $= 0$;
2. **LOOP**
3. IF (*current-f-bound* $>$ *f-bound*) THEN RETURN(*Fail*);
4. *result* $=$ f-L-DFS(s, *successors*, $\star$, *current-f-bound*, *min-f-over*);
   // *min-f-over* is a call by reference parameter.
   IF (*result* != *Fail*) THEN RETURN(*result*);
   IF (*min-f-over* $<$ *current-f-bound*) THEN RETURN(*Fail*); // *result* is reliable.
   current-f-bound $=$ *min-f-over*;
5. **ENDLOOP**

Remarks:

- IDA* always finds a cheapest solution path if the heuristic is admissible, or in other words the heuristic never overestimates the actual cost to a goal node. (In such a case, the evaluation function $f = g + h$ is optimistic.)
- IDA* uses space linear in the length of a cheapest solution.
- IDA* expands the same number of nodes, asymptotically, as A* if the state space graph G is a tree.