satisfies ... ction is said to be **monotonic / monotone** if it

$$h(n) \le C(n, n') + h(n') \text{ for all } n, n'$$

such that n' is a successor of n.

Property V: *Every consistent heuristic is also admissible.*

Proof: We have

$$h(n) \le K(n, n') + h(n') \text{ [since h is consistent]}$$

Replacing γ against n', we have

$$h(n) \le K(n, \gamma) + h(\gamma)$$

$$\Rightarrow h(n) \le h^*(n),$$

which is the *condition for admissibility*.

The following example [7] illustrates that optimal solution will never be missed, if h (n) is admissible as presented below.

Example 4.3: Consider the search-space, given in fig. 4.9(a). Note that, here h > h*, in the case of overestimation, where we made node D so bad (by ... that we can never find the optimal path A-D-G.

$$= f^*(n')$$

$$= C^* \quad \square$$

Therefore, $f(n') \leq C^*$

Property IV: *A^* is admissible (returns optimal solution)[6]*

Proof: Suppose A^* terminates with a goal node t belonging to Γ for which $f(t) = g(t) > C^*$.

However, A^* tests nodes for compliance with termination criteria, only after it selects them for expansion. Hence, when t was chosen for expansion,

$$f(t) \leq f(n), \text{ for all open } n.$$

This means that immediately prior to termination, any node n on open satisfies:

$$f(n) > C^*,$$

which, however, contradicts property III, which claims that there exists at least one open node n with $f(n) \leq C^*$. Therefore, the terminating t must have $g(t) = C^*$, which means that A^* returns an optimal path. □

Monotonicity and Consistency of Heuristics

Informally speaking, by consistency, we mean that A^* never re-opens already closed nodes. From property I and II, we find that the cheapest path constrained to pass through n cannot be less costly than the cheapest path

which is the condition for consistency [6].

Definition 4.4: A heuristic function is said to b[...] satisfies

$$h(n) \leq C(n, n') + h(n') \text{ for all } n, n'$$

such that n' is a successor of n.

Property V: *Every consistent heuristic i[...]*

Proof: We have

$$h(n) \leq K(n, n') + h[...]$$

Replacing γ against n', we hav[...]

$$h(n) \leq K(n, \gamma[...]$$

$$\Rightarrow h(n) \leq h^*(n),$$

which is the condition for ad[...]

The following example [7[...] missed, if h (n) is admissi[...]

Example 4.3: Consid[...] here $h > h^*$, in the cas[...] making its h value too[...]

The generation cost $g(x)$ can be measured ... through a few state transitions. For instance, if node x was ... the starting node through m state transitions, the cost $g(x)$ will ... proportional to m (or simply m). But how does one evaluate the $h(x)$? It may be recollected that $h(x)$ is the cost yet to be spent to reach the goal from the current node x. Obviously, any cost we assign as $h(x)$ is through prediction. The predicted cost for $h(x)$ is generally denoted by $h'(x)$. Consequently, the predicted total cost is denoted by $f'(x)$, where

$$f'(x) = g(x) + h'(x).$$

Now, we shall present the A* algorithm formally.

Procedure A*

Begin

1. Put a new node n to the set of *open nodes (hereafter open)*; Measure its $f'(n) = g(n) + h'(n)$; Presume the set of closed nodes to be a null set initially;

2. **While** open is not empty **do**

   Begin

   If n is the goal, stop and return n and the path of n from the beginning node to n through back pointers;

   Else do

   Begin

   a) remove n from open and put it under closed;

   b) generate the children of n;

   c) **If** all of them are new (i.e., do not exist in the graph before generating them **Then** add them to open and label their f' and the path from the root node through back pointers;

   d) **If** one or more children of n already existed as open nodes in the graph before their generation **Then** those children must have multiple parents; Under this circumstance compute their f' through current path and compare it through their old paths, and keep them connected only through the shortest path from the starting node and label the back pointer from the children of n to their parent, if such pointers do not exist;

   e) If one or more children of n already existed as closed nodes before generation of them, then they too must *have multiple parents;* Under this circumstance, find *the shortest path from* the starting node, i.e., the path *(may be current or old)* through which f' of n is *minimum; If the current* path is selected, then the *nodes in the sub-tree rooted* at the corresponding child *of n should have revised* f' as the g' for many of the *nodes in that sub-tree changed;* Label the back pointer *from the children of n to* their parent, if such pointers *do not exist;*

End;

End While;

End.

computation of f' (x) at the nodes of a given tree