Heuristic Search:

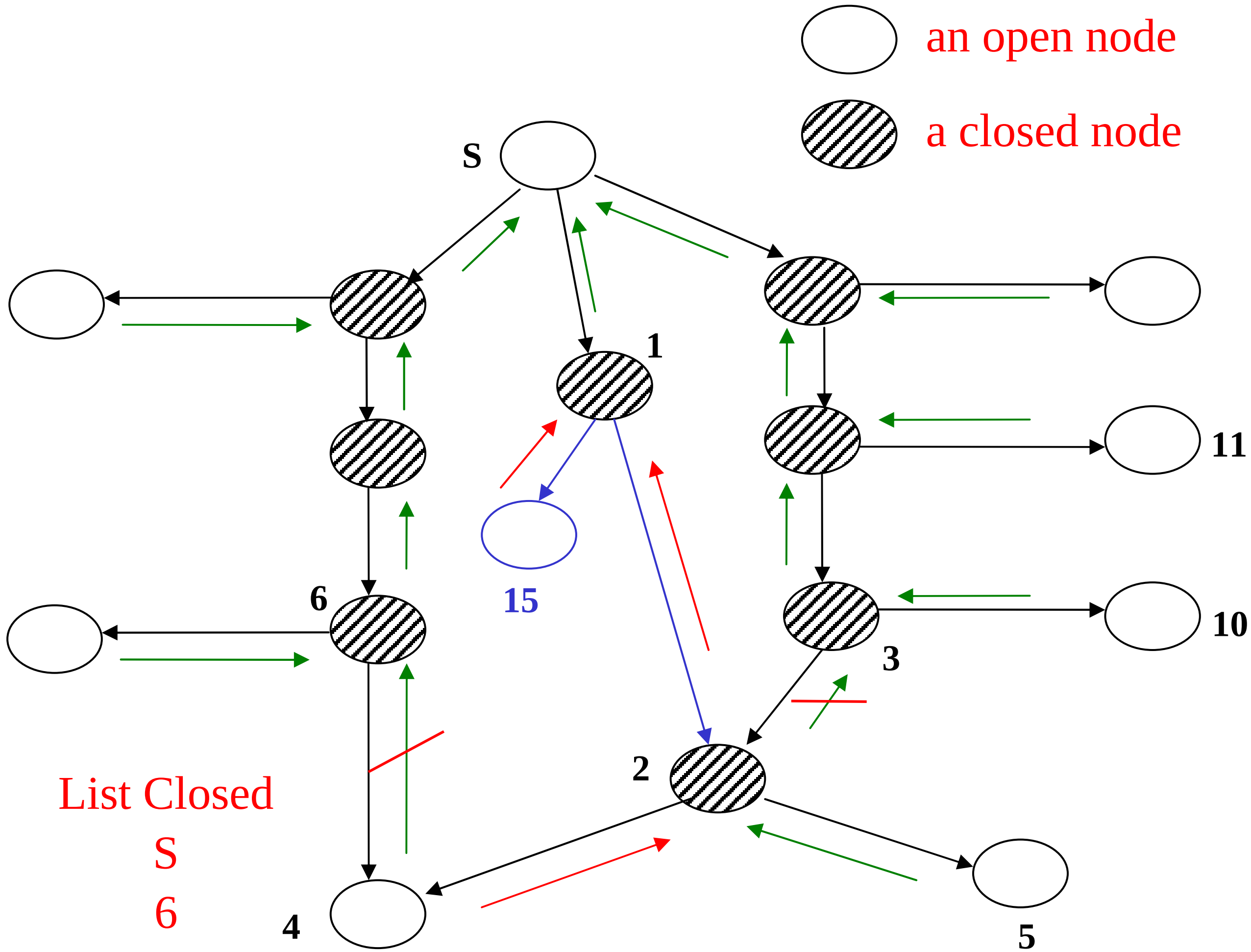
The problem of local maxima arises because Hill-Climbing makes irrevocable decisions at each point in search space.

A search that uses one or more items of domain-specific knowledge to traverse state-space is called heuristic search. A Heuristic is a rule of thumb, and may not be guaranteed to succeed, but it is useful in most cases.

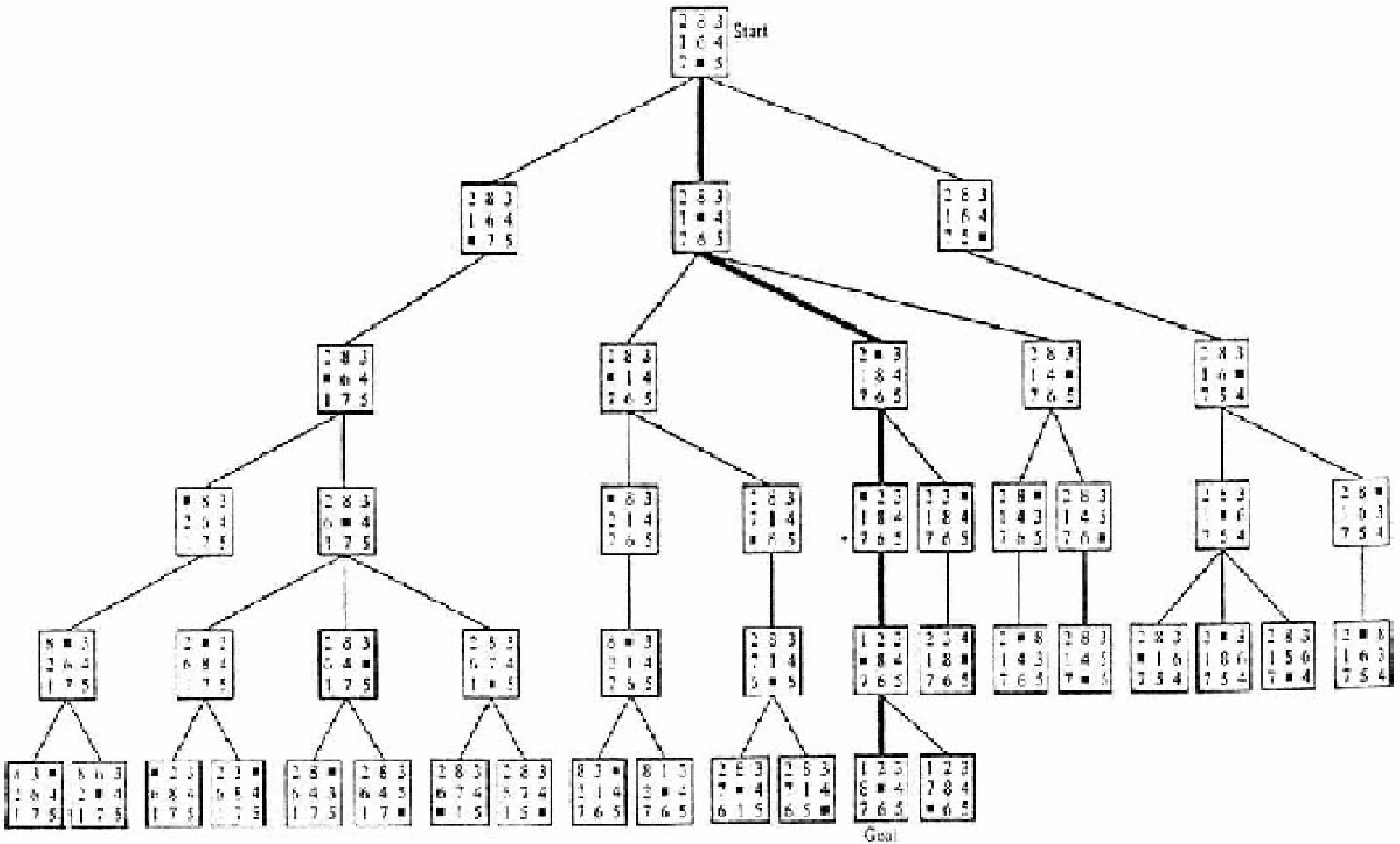
Heuristic search works the same way as hill-climbing except the next node to be expanded is selected among all possible nodes, not just the successors of the current state.

An Algorithm for Heuristic Search:

1. Create a search graph G, consisting solely of the start node S. Put S on a list called open.
2. Create a list called closed, which is initially empty.
3. Loop: If open is empty, exit with fail.
4. Select the first node from open, remove it from open and put it on closed. Call this node n.
5. If success(n) = true, then exit with success (to find the solution trace back from n to s.)
6. Expand node n, generating the set, M, of its successors, and put them in G as successors of n.
7. Establish a pointer to n from those members of M which are not already in G (that is not already in open or closed). Add these members to open. For those members of M which are on closed and their children, determine if their back pointers should be changed, and if so change.
8. Reorder the list open according to heuristic merit of each element.
9. Go to Loop.

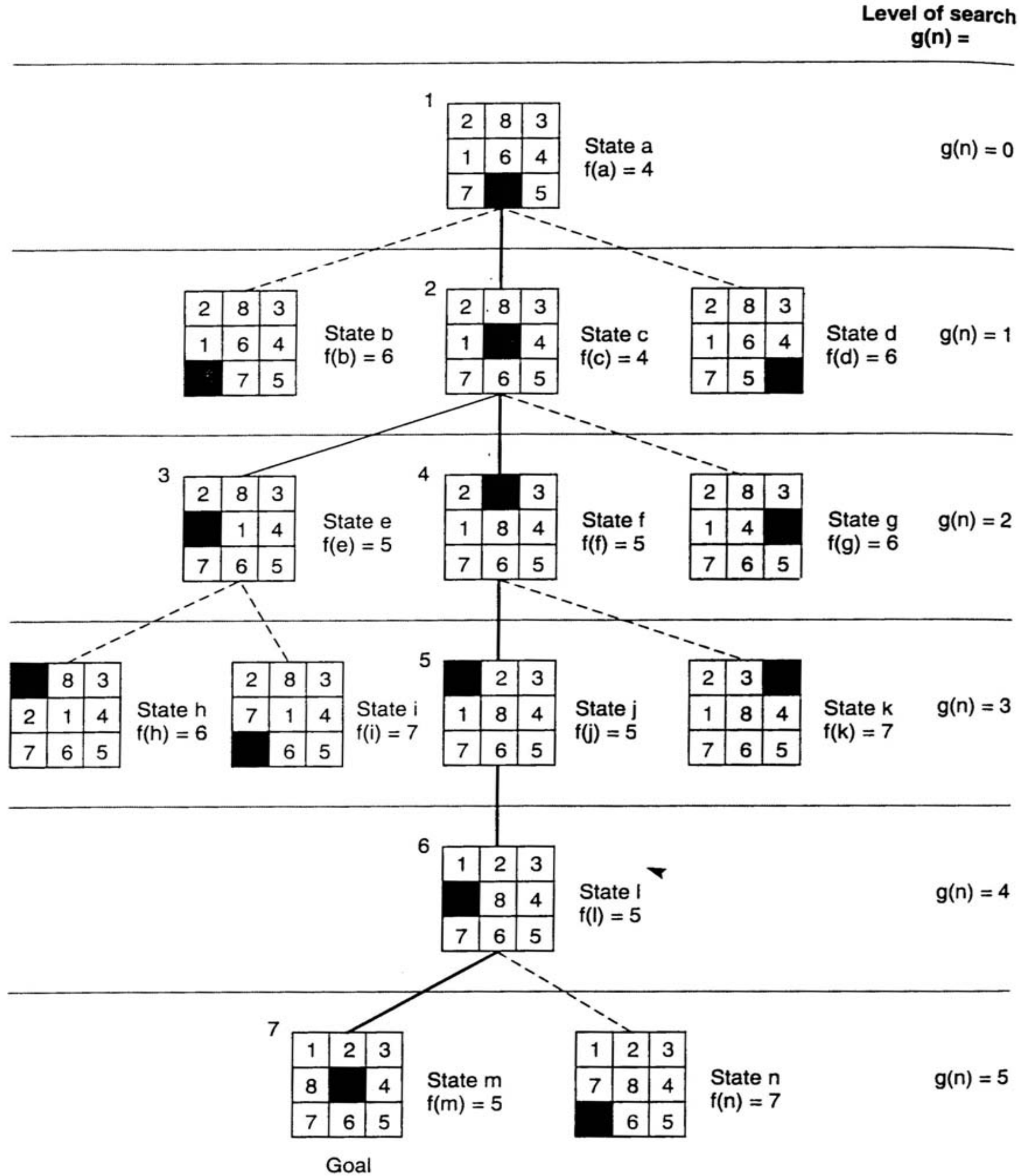


List Open	List Closed
4	S
5	6
:	3
:	2
10	:
11	:
1	1
15	

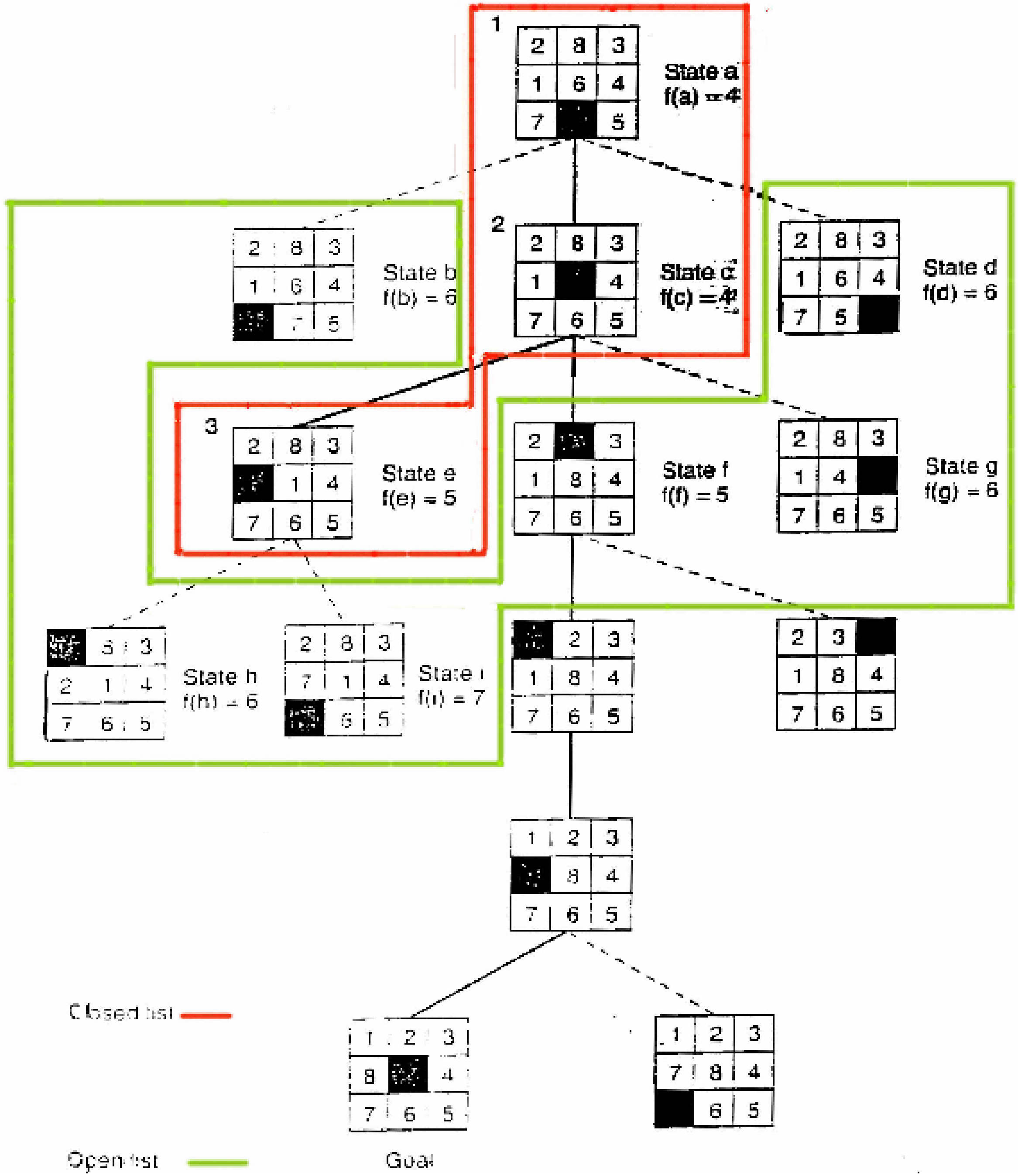


Start
Goal

State space generated in heuristic search of the 8-puzzle



open and **closed** as they appear after the third iteration of heuristic search



Evaluation Functions

We use the function f(n) to evaluate the promise of node n. f(n) must be the estimate of the cost of a minimal cost path from the start node to a goal node constrained to go through node n. f is then used to order the nodes in open in step 8 of the previous algorithm.

Designing Optimal Evaluation Functions:

Let $K(n_i, n_j)$ be the actual cost of a minimal cost path between two arbitrary nodes n_i and n_j. Then for a particular goal node t_n, $K(n, t_i)$ gives the minimal cost path from n to that goal node.

Let

$h^*(n) = \min K(n, t_i)$

thus $h^*(n)$ is an optimal path cost from n to a goal.

Similarly

$g^*(n) = K(s, n)$

cost from start node to node n.

then

$f^*(n) = g^*(n) + h^*(n)$

will be an evaluation function which at any node n gives us the cost of an optimal path from s to a goal node constrained to go through n.

To design an evaluation function, develop one which looks like

$$f(n) = g(n) + h(n)$$

and estimates the components of f* well.

An obvious choice of estimate for g*(n):

g(n) = Σ arc-costs while tracing from n to s on the best path found so far.

$$\Rightarrow \mathbf{g(n) >= g^*(n)}$$

Question:

For what value of f(n) will we produce breadth-first search?

Claim:

If h is a lower bound on h* (that is, if h(n) =< h*(n) for all nodes) then the heuristic search algorithm will be guaranteed to find an optimal path to a goal, if one exists. => Admissible

Recall:

A search is admissible if for any graph, it always terminates in an optimal path from s to a goal node whenever a path from s to a goal node exists.

Proof Steps:

1. Show that the algorithm terminates whenever a goal node is accessible.
2. Show that it terminates by finding a goal node.
3. Show that it terminates with an optimal path to a goal node.

An Important Result:

At anytime before A* terminates, there exists on open a node n' that is on an optimal path from S to a goal node with f(n') <= f*(S)

Another Important Result:

For any node n selected for expansion by heuristic search, f(n)<=f*(S)

Comparing HS Algorithms

Let us assume we have two versions of H. search, one with $f_1(n) = g_1(n) + h_1(n)$, and one with $f_2(n) = g_2(n) + h_2(n)$.

Now assume that h_1 and h_2 are both lower bounds on h^*, we say that HS2 is more informed than HS1, if for all non-goal nodes $h_2(n) > h_1(n)$.

$\Rightarrow$ You can show that if the implicit graph is searched by both algorithms, then at termination if node n was expanded by HS2, it was also expanded by HS1. Thus HS1 always expands as many or more nodes than HS2.