4 HEURISTIC SEARCH

4.0 Introduction

4.1 An Algorithm for Heuristic Search

4.2 Admissibility, Monotonicity, and Informedness

4.3 Using Heuristics in Games

4.4 Complexity Issues

4.5 Epilogue and References

4.6 Exercises

George F Luger

ARTIFICIAL INTELLIGENCE

Structure and Strategies for Complex Problem Solving

Fourth Edition

Figure 4.1: First three levels of the tic-tac-toe state space reduced by symmetry.

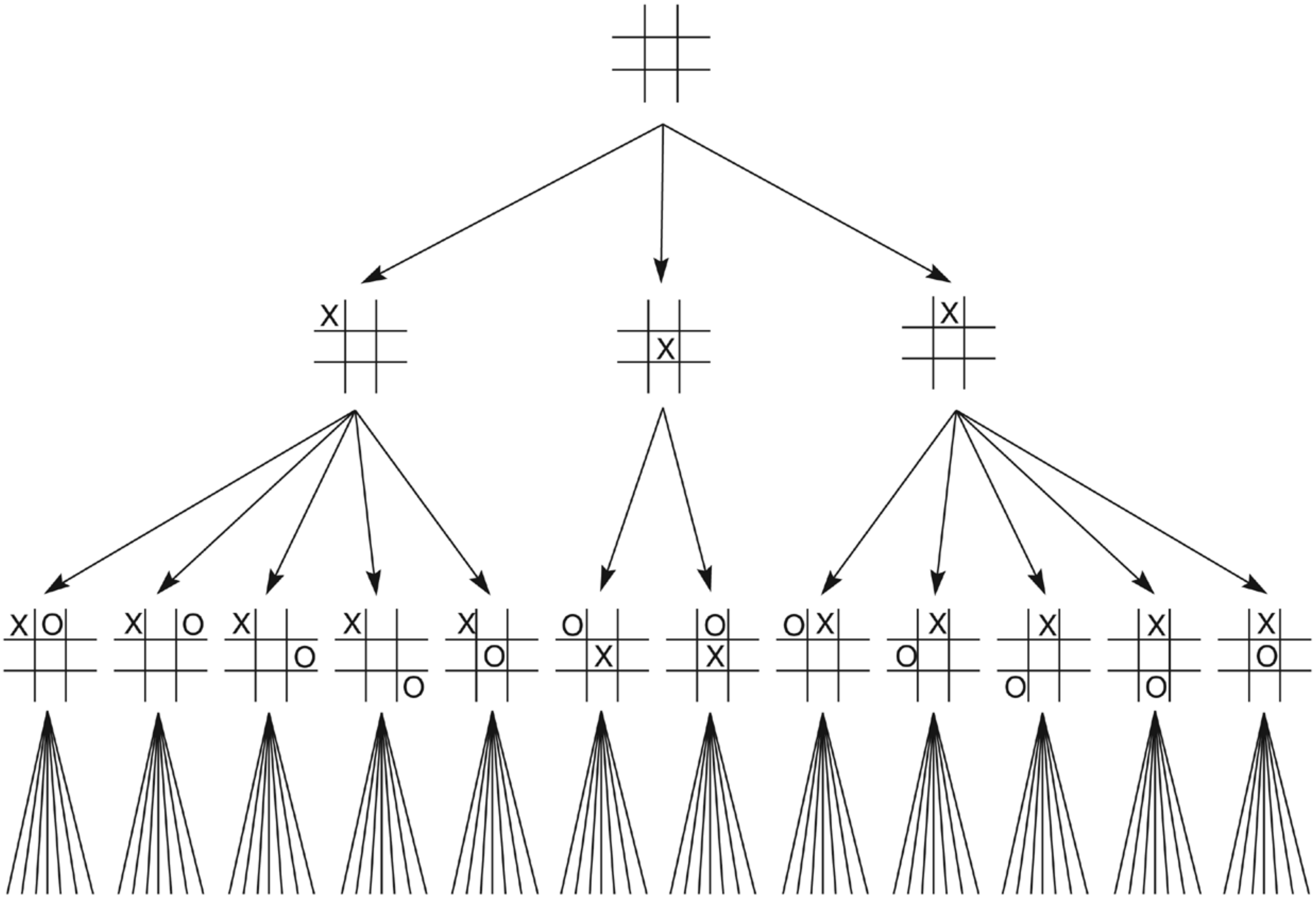
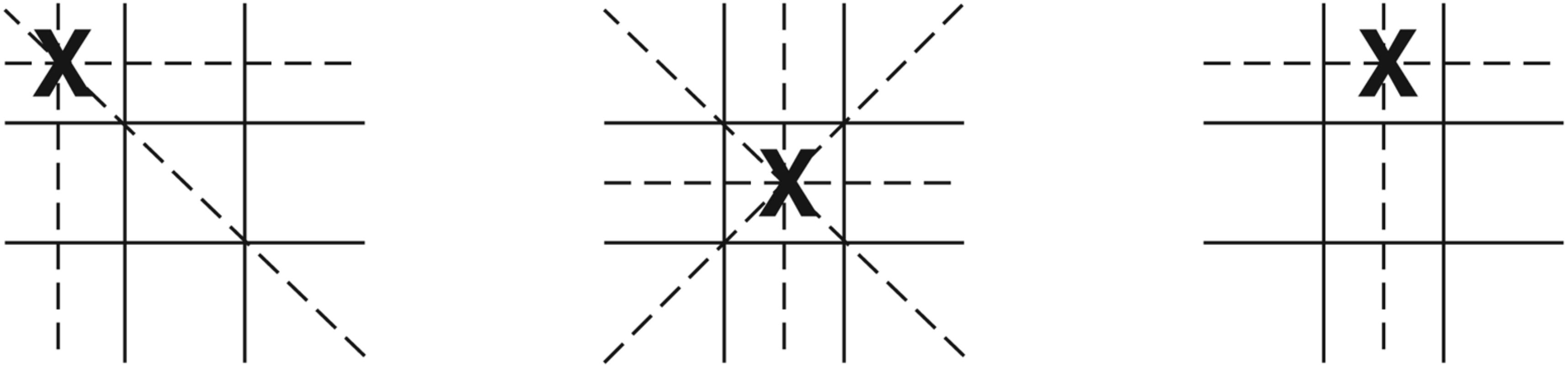


Figure 4.2: The “most wins” heuristic applied to the first children in tic-tac-toe.

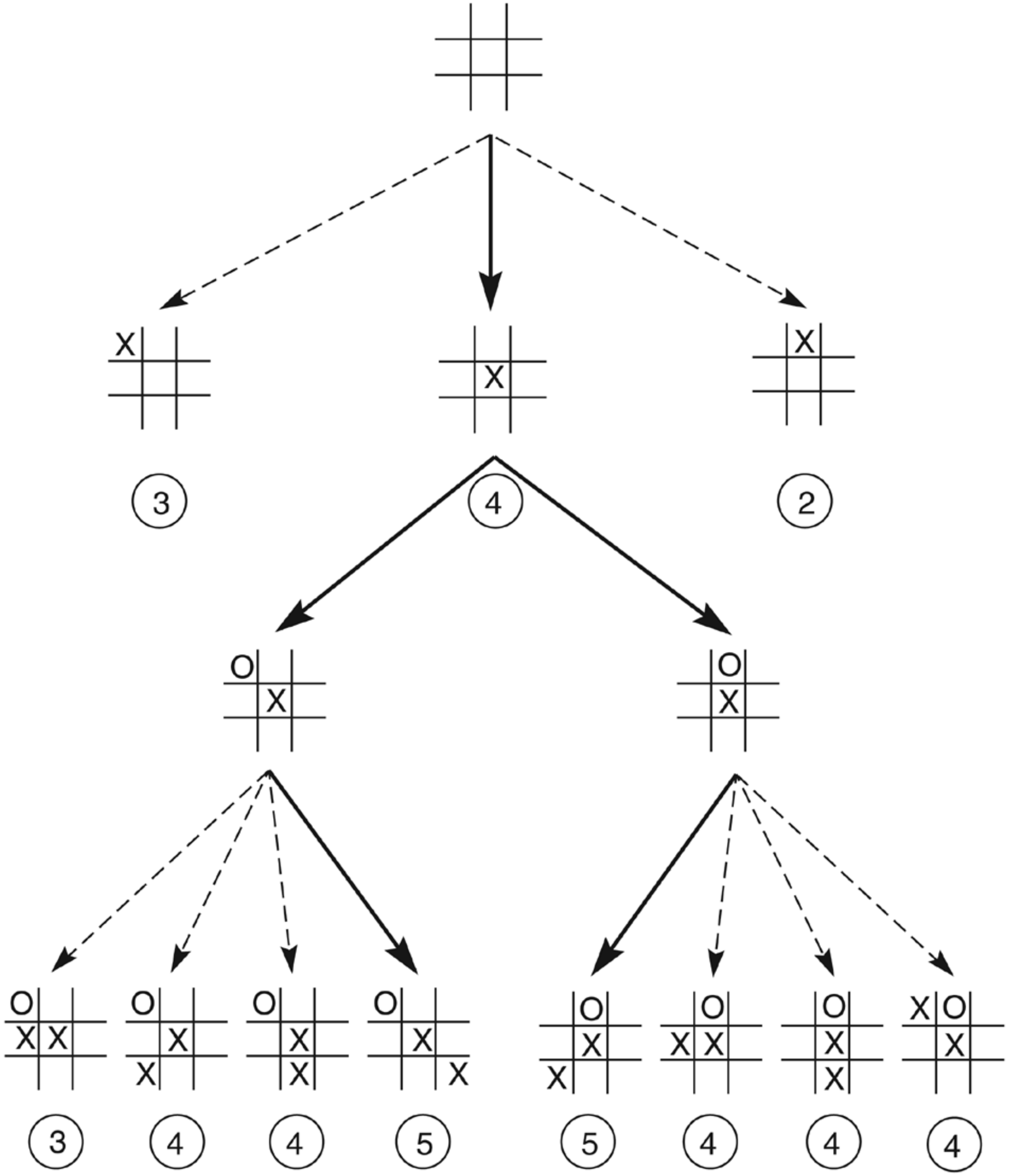


Three wins through a corner square

Four wins through the center square

Two wins through a side square

Figure 4.3: Heuristically reduced state space for tic-tac-toe.

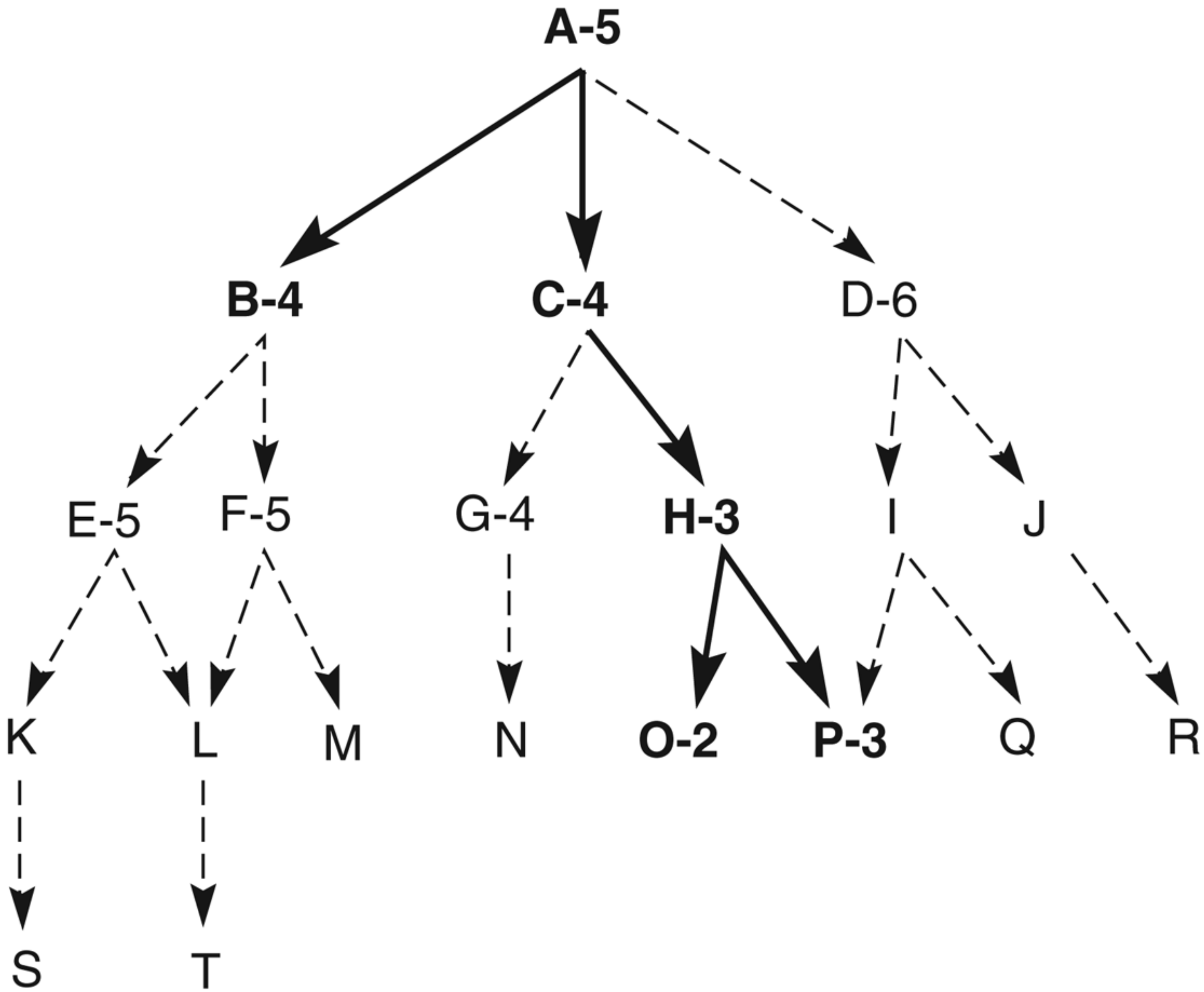


function best_first_search algorithm

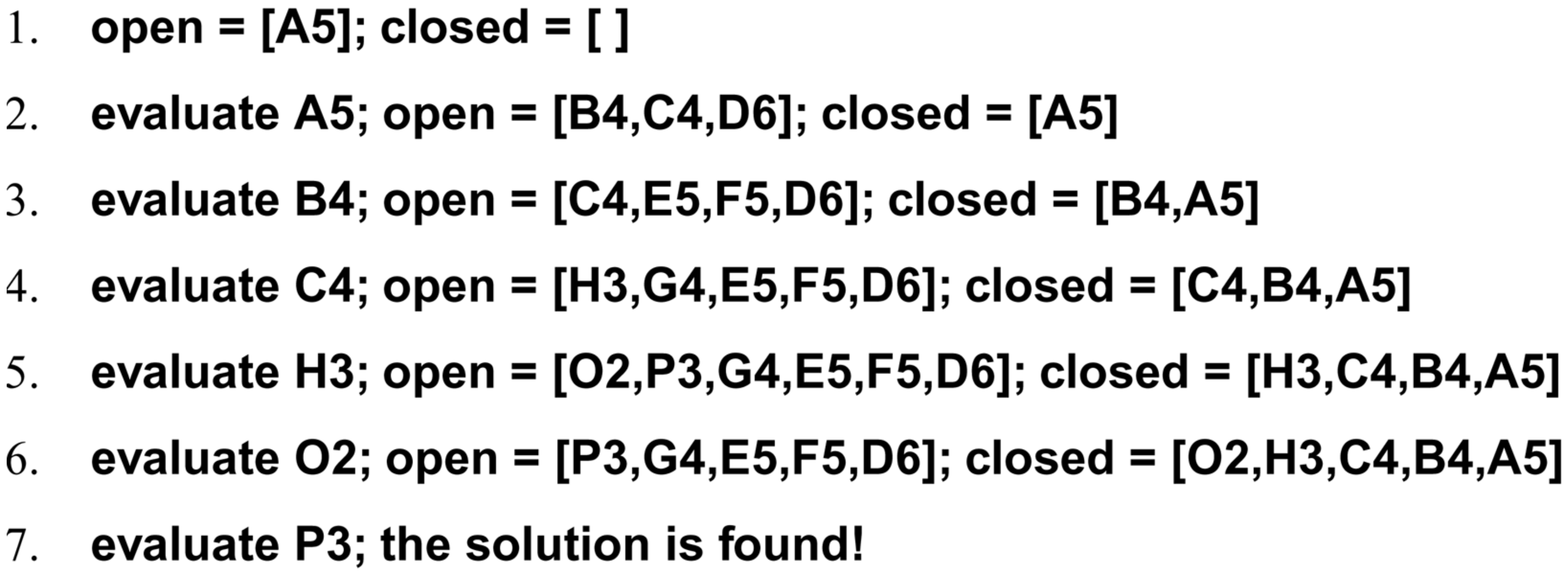
```
function best_first_search;

begin
   open := [Start];                                                % initialize
   closed := [ ];
   while open ≠ [ ] do                                             % states remain
      begin
         remove the leftmost state from open, call it X;
         if X = goal then return the path from Start to X
         else begin
               generate children of X;
               for each child of X do
               case
                  the child is not on open or closed:
                     begin
                        assign the child a heuristic value;
                        add the child to open
                     end;
                  the child is already on open:
                     if the child was reached by a shorter path
                     then give the state on open the shorter path
                  the child is already on closed:
                     if the child was reached by a shorter path then
                        begin
                           remove the state from closed;
                           add the child to open
                        end;
               end;                                                % case
               put X on closed;
               re-order states on open by heuristic merit (best leftmost)
            end;
return FAIL                                                        % open is empty
end.
```

Figure 4.4: Heuristic search of a hypothetical state space.



A trace of the execution of best_first_search for Figure 4.4



1. **open = [A5]; closed = [ ]**
2. **evaluate A5; open = [B4,C4,D6]; closed = [A5]**
3. **evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]**
4. **evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]**
5. **evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]**
6. **evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]**
7. **evaluate P3; the solution is found!**

Figure 4.5: Heuristic search of a hypothetical state space with open and closed states highlighted.

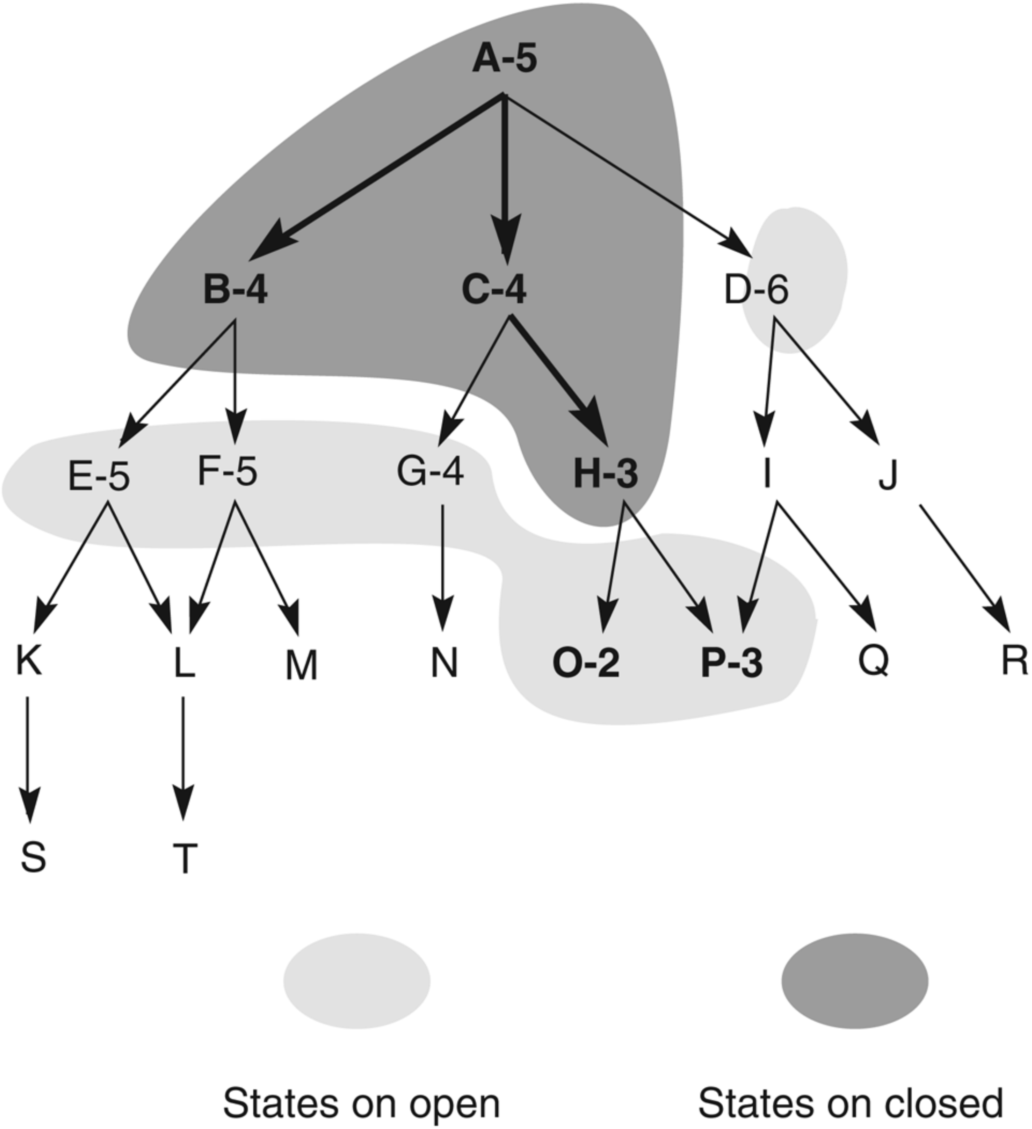


Figure 4.6: The start state, first set of moves, and goal state for an 8-puzzle instance.

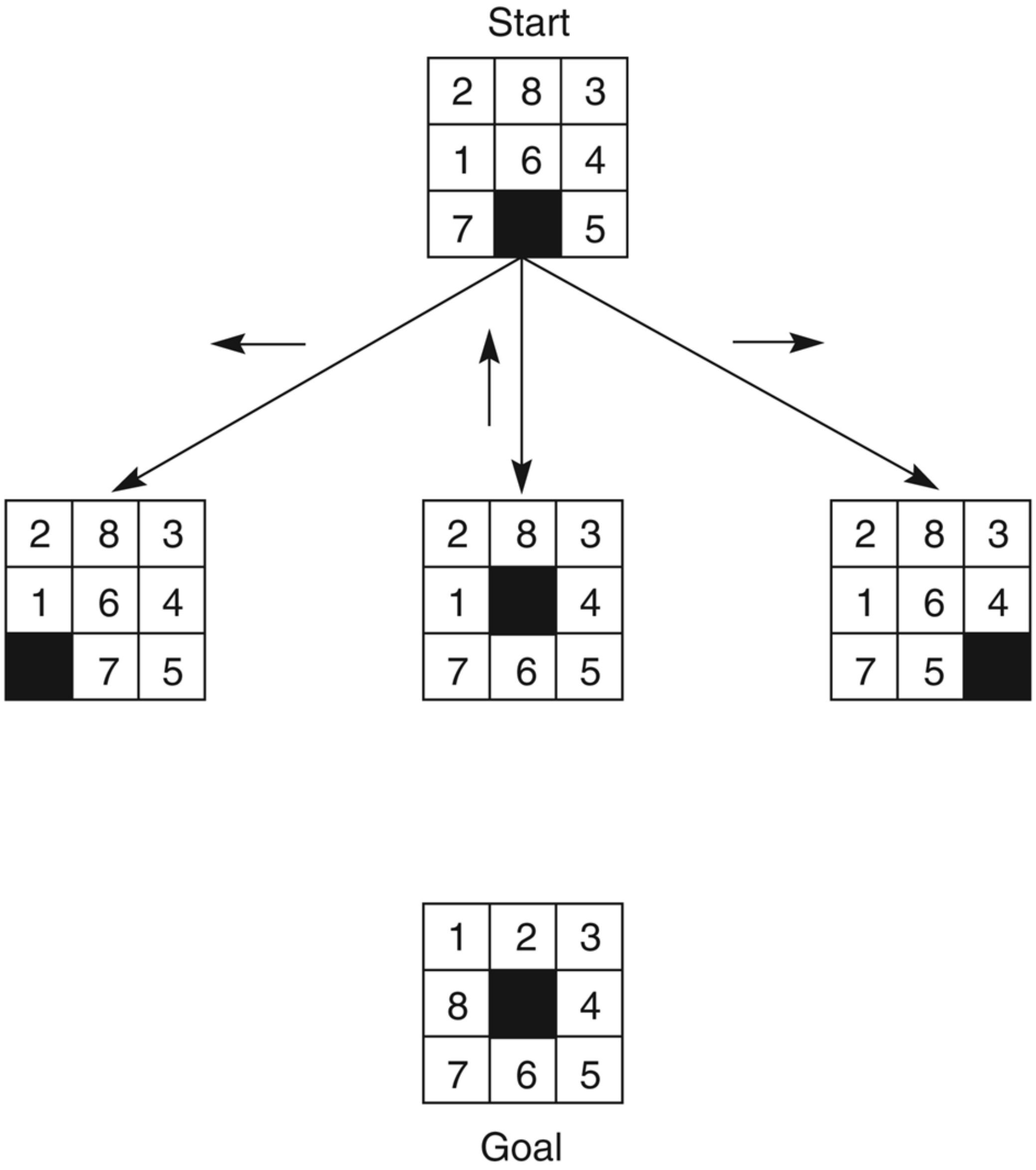


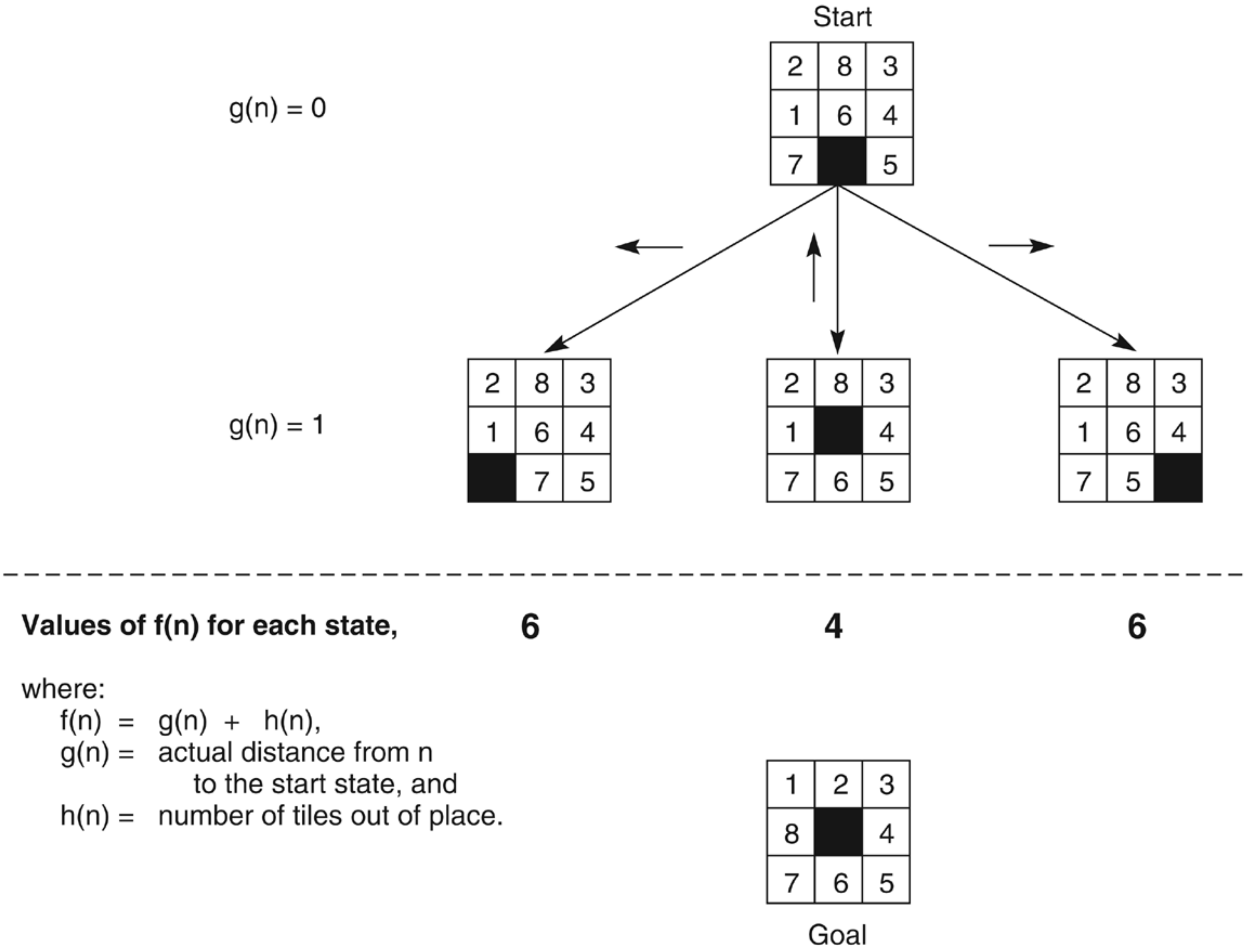
Figure 4.8: Three heuristics applied to states in the 8-puzzle.

2 8 3 1 6 4 ■ 7 5	**5**	**6**	**0**
2 8 3 1 ■ 4 7 6 5	**3**	**4**	**0**
2 8 3 1 6 4 7 5 ■	**5**	**6**	**0**
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

1 2 3
8 ■ 4
7 6 5

Goal

Figure 4.9: The heuristic **f** applied to states in the 8-puzzle.



Values of f(n) for each state, **6** **4** **6**

where:

f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.