Universität des Saarlandes
Fachrichtung 6.2 – Informatik
J. Siekmann, S. Autexier, C. Benzmüller, C.E. Brown
C. Hahn

3rd Theoretical Assignment in Artificial Intelligence (SS 2005) **Solutions**

Exercise 3.1 **(10 P)**

Minsky claims to stem from Charles the Great. Which way is easier to check the claim: to show that Charles the Great is an ancestor of Minsky or to show that Minsky is a descendant of Charles the Great? Explain.

Solution:

Every person has only two parents, but can have arbitrarily many children. Therefore, backward search branches with exactly 2, but forward search may branch with much more than 2. On the other hand, forward search terminates for sure, the latest when we reach contemporary descendants of Charles the Great who don't have children yet. Backward search may not terminate at all (unless we all descend from Adam and Eve, if you see it from a religious perspective). Hence, the best search procedure would probably be backward with keeping track of the time when the person under scrutiny lived, such that the search in one branch is stopped as soon as the birthyear of Charles the Great has been passed. If it turned out Charles the Great had very few descendents, then forward search would be better.

Exercise 3.2 **(10 P)**

1. Which two advantages do breadth-first search have over depth-first Search? (5 P)
2. Why does one nevertheless use depth-first Search or similar procedures (like iterative deepening) in practical applications? (5 P)

Solution:

1. *Breadth-first search is* complete. *If there exists a solution, it will be found. Also, if all step costs are equal, then breadth-first search is* optimal. *If more than one solution exists, than the optimal solution will be found (assuming all step costs are equal). Depth-first search is neither complete nor optimal.*
2. *The space requirements of breath-first search increase exponentially with respect to the search-depth. In contrast, depth-search requires linear space. In practical applications, one might not find a solution on a low depth-level. Thus, space might not be sufficient to use breadth-first search. On the other hand, depth-first search can be made into a complete (and optimal, if all step costs are equal) search procedure by using iterative deepening.*

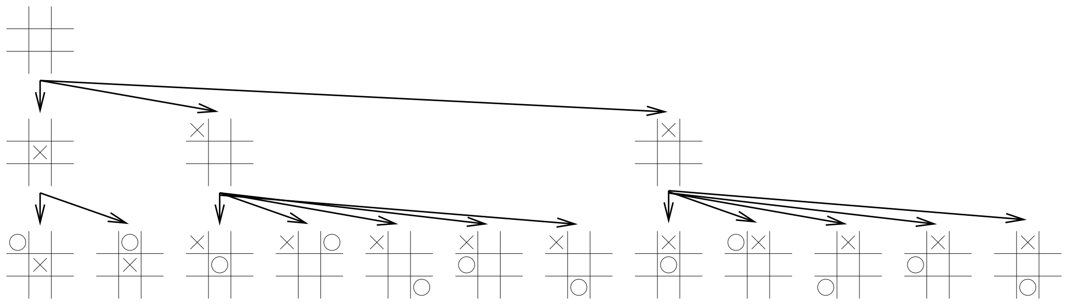
Exercise 3.3 **(25 P)**

Tic-Tac-Toe is a game for two players. The board is a square of $3 * 3$ fields. Each player is assigned a type of token ($\times$ or $\circ$). Initially the board is empty. The players play in turn and place a token on an empty field. A player wins, if she/he has first aligned three of her/his tokens either in a row, a column, or on one of the two diagonals. The game ends when a player wins or when there are no more empty fields.

1. Draw the search tree for Tic-Tac-Toe up to level 2. Take into account symmetric game states, i.e. those states that can be transformed into each other by rotation and mirroring. (5 P)
2. Give an estimation on the number of possible different Tic-Tac-Toe games. (5 P)
3. Develop an evaluation function for the Tic-Tac-Toe game. (5 P)
4. Indicate the values of your evaluation function for each node of the search tree at level 2. Use these values to compute the values of the nodes on level 1 and 0 using the Min-Max algorithm. (5 P)
5. Indicate all nodes in the search tree that would not have been considered when using Alpha-Beta pruning (see Russell/Norvig, 2nd edition, p. 167/191). (5 P)

Solution:

1. *Definition: Two states in this game are different iff for one state the Tic-Tac-Toe board cannot be transformed into the other state by using rotation and mirroring.*



2. *Definition: Two games are different if at least one move exists that prevent that both game-states can be transformed into the other.*
On depth-level 2, 12 different games can be identified (under the assumption that all other states can be obtained by rotating and mirroring these 12 states). For depth-level 3, 7 fields are empty; for level 3, 6 are empty etc. This consideration results in the following estimation:

$$\textit{differrent Tic-Tac-Toe games} \leq 12 * 7! = 60480$$

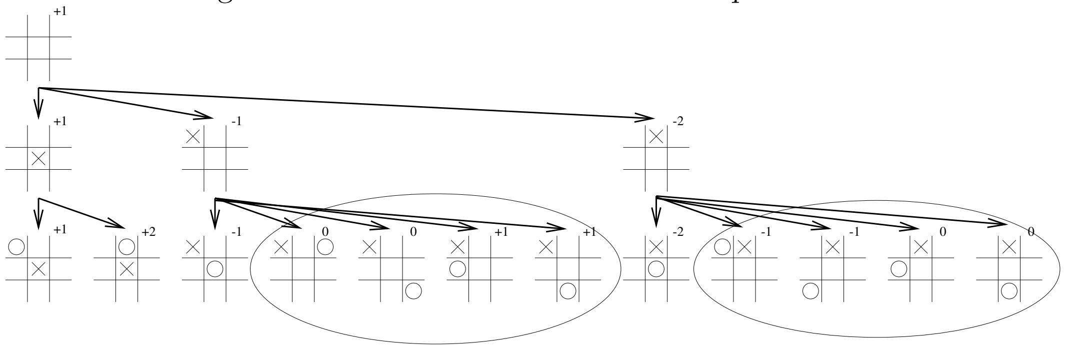
The number 60480 gives an upper bound for the number of different games. If we considered symmetric aspects of games beyond level 2, we could obtain a smaller upper bound.

3. *We define X_n ($n \in \{1,2,3\}$) as the number of rows, columns and diagonals in which exactly n $\times's$, but no $\bigcirc$ has been placed. Accordingly, O_n defines the number of rows, columns and diagonals in which in which exactly n $\bigcirc's$, but no $\times$ have been placed. As a player wins if $X_3 \geq 1$ (and loses if $O_3 \geq 1$), the function should consider these states. Situations in which X_2 is high and O_2 is low, should be positively evaluated. Accordingly,*

situations in which X_2 is low and O_2 is high, should negatively be rated. Therefore, we choose the following estimation function:

$$Eval = 10\ X_3 + 3\ X_2 + X_1 - (10\ O_3 + 3\ O_2 + O_1).$$

4. *The values on depth-level 2 were evaluated according to the function above. Then we use the Min-Max algorithm to evaluate the values of the parents.*



5. *All feasible states that are not considered using Alpha-Beta pruning are encircled above. Note, Alpha-Beta pruning starts at the left-most leaf node.*

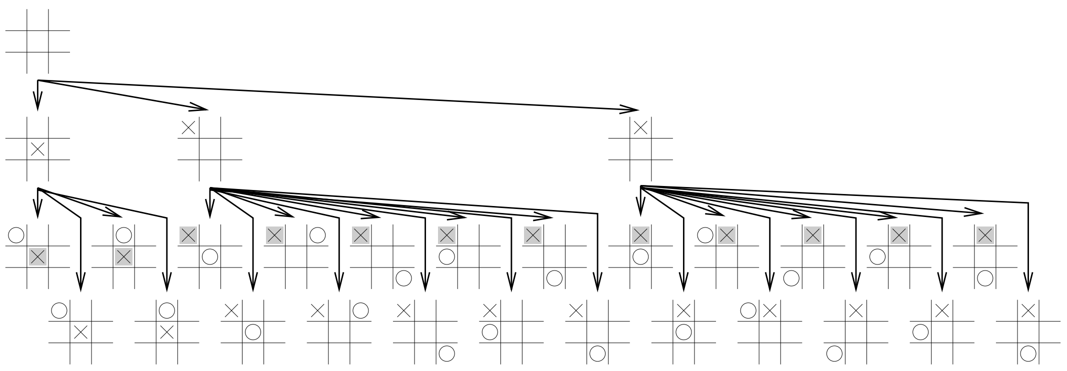
Exercise 3.4 **(25 P)**

We consider a variant of the Tic-Tac-Toe game, called *blind Tic-Tac-Toe*, where the players cannot see the board. When it is a player's turn, she/he proposes her/his move to a game master, who controls the board and notifies the players when to game is over and gives the result. If a player wants to place a token on a field that is empty, the respective token is set on that field. Otherwise the player has to chose a different move. Any kind of communication between one player and the game master (indicate move/accept/reject move) is hidden from the other player.

1. Draw the search tree for blind Tic-Tac-Toe up to level 2. Take into account symmetric game states, i.e. those states that can be transformed into each other by rotation and mirroring. (5 P)
2. Develop an evaluation function for blind Tic-Tac-Toe. (10 P)
3. Assume you want to develop an agent which is very good at playing that game. Do you think the Expectiminimax algorithm (see Russell/Norvig, 2nd edition, p. 177) is an adequate approach or would you prefer another variant of the Min-Max algorithm? Sketch how you would use that algorithm. (10 P)

Solution:

1. *Definition: Two states in this game are different iff for one state the Tic-Tac-Toe board cannot be transferred into the other state by using rotation and mirroring. But for this exercise, we have also to take into account the representation of both players. In the figure below, the $\times$-stones are shaded as those are unknown for the $\bigcirc$-player. Thus, this representation does not only illustrate the actual board but also the actual knowledge of the players.*



We define an upper bound for the number of games. Considering symmetric game states, there exist 24 different games for level 2. We separately regard row 4 and row 3 in this figure. In each state for the third row, player $\times$ has eight different possibilities to move next as he/she can only knows the positions of his/her own stones. If (s)he places on a free field, the stone is positioned. This results in $7 = \binom{1}{0} * 7$ feasible possibilities to continue the game. Otherwise, (s)he has to move again. This results in $7 = \binom{1}{1} * 7$ feasible possibilities. To sum up, there exists 14 $((\binom{1}{0} + \binom{1}{1}) * 7)$ possibilities for each of the different 12 states.

(a) $\binom{1}{0}$ *describes the fact: the player recognises* 0 *stone(s) out of one possible stone.*

(b) $\binom{1}{1}$ *describes the fact: the player recognises* 1 *stone(s) out of one possible stone.*

Analogically, we can describe the other levels and we get the following estimation:

$$\begin{array}{rll}
 & 12 & *((\binom{1}{0} + \binom{1}{1}) * 7) \\
 & *((\binom{2}{0} + \binom{2}{1} + \binom{2}{2}) * 6) & *((\binom{2}{0} + \binom{2}{1} + \binom{2}{2}) * 5) \\
 & *((\binom{3}{0} + \binom{3}{1} + \binom{3}{2} + \binom{3}{3})) * 4) & *((\binom{3}{0} + \binom{3}{1} + \binom{3}{2} + \binom{3}{3}) * 3) \\
 & *((\binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4}) * 2) & *((\binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4}) * 1) \\
= & \multicolumn{2}{l}{12 * (2 * 7) * (4 * 6) * (4 * 5) * (8 * 4) * (8 * 3) * (16 * 2) * (16 * 1)} \\
= & 31,708,938,240 &
\end{array}$$

For the fourth row we get the following estimation (assumption: one $\times$-stone has been detected):

$$\begin{array}{rl}
 & 12 * (2 * 7) * (2 * 6) * (4 * 5) * (4 * 4) * (8 * 3) * (8 * 2) * (16 * 1) \\
= & 3,963,617,280
\end{array}$$

Altogether $31,708,938,240 + 3,963,617,280 = 35,672,555,520$ *describes an upper bound of possible games.*

2. *One could modify the evaluation function for normal tic-tac-toe (as given in the previous problem):*

$$10X_3 + 3X_2 + X_1 - (10O_3 + 3O_2 + O_1)$$

where here X_n is the number of rows, columns and diagonals with exactly n $\times's$ and O_n is the number of rows, columns and diagonals which might *have n $\circ's$. In general, the values of O_n will be bigger than the values of X_n due to insufficient information. For this reason, coefficients giving more weight to the X values might be better. For example,*

$$50X_3 + 15X_2 + 5X_1 - (10O_3 + 3O_2 + O_1)$$

3. *Philosophically, using* Expectiminimax *is not adequate: The* Expectiminimax *algorithm is an approach for non-deterministic games (games with an element of chance). The blind tic-tac-toe game is* not *a non-deterministic game. Instead, blind tic-tac-toe*

is a game with imperfect information (such as a card game after the cards are dealt). One can simulate imperfect information as non-determinism (though, as explained in the slides from Chapter 6, this can be dangerous).

Even though we don't discuss EXPECTIMINIMAX *in this solution, it is a good idea to know how it works: Calculating "max" nodes by taking maximum of children, "min" nodes by taking minimum of children and "chance" nodes by taking a weighted sum (average). Look at Chapter 6 (e.g. Figure 6.12) for a reminder of this.*

The MINIMAX *algorithm could be modified to handle imperfect information by not requiring that every move by a player is followed by a move by the opponent. Also, instead of nodes corresponding to game states, nodes can correspond to collections of game states (considered equivalent if the positions of $\circ$'s are unknown to MAX). The game tree (for MAX, i.e. $\times$) using this modified alogrithm is formed as follows:*

- *Move 1: MAX places an $\times$ somewhere and there is a child for each possibility (up to symmetry). One example of a child corresponds to placing an $\times$ in the center (one game state for this node).*
- *Move 2: MIN tries to place an $\circ$ somewhere (in any of the slots) and there is a child for each possibility (up to symmetry). This will lead to two children. In one case, MIN has chosen a new slot leading to a node with a collection of states equivalent with respect to MAX (since MAX does not know the position of $\circ$). In this node, the next move is made by MAX. In the other case, MIN has chosen the slot with the $\times$. In this case the next turn belongs to MIN.*

  Consider the node where there is one $\times$ in the center slot. There are two children. In the first child MIN has chosen a slot other than the center and the corresponding set of game states is the set *of game states where there is a $\times$ in the center and $\circ$ elsewhere. In the second child the game state is still with a $\times$ in the center and MIN can go again.*
- *Move 3 (if it is MAX's turn): Create children based on where MAX tries to place an $\times$. Some children represent MAX finding a hidden $\circ$. In these cases, it remains MAX's turn. The set of game states at the node is partitioned among the children.*
- *Move 3 (if it is MIN's turn): Create children based on whether MIN chooses a slot with $\times$ (which should not happen at this level since MIN has already found the single $\times$, but later MIN might find several $\times$ marks in a row – also MIN might be forgetful).*
- *At some level, one can stop search and use an evaluation function as given above.*
- *Once one has values for the leaves (using an evaluation function), these values can be lifted to parent nodes by taking maximums and minimums. Note however, that there is not a clear alternation between taking maximums and minimums. One may take the maximum at two levels (in cases where maximum must go more than once since he chose where an $\circ$ is).*

Exercise 3.5 **(30 P)**

A set of cities and their connecting roads is given in the following table (Meaning of the entries: from city A there is a road to B of length 3km, a road to C of length 2km, and one to D of length 8km, etc.):

City	Connecting roads (to-city,length)
A	(B,3)(C,2)(D,8)
B	(A,3)(C,2)(F,2)
C	(A,2)(B,2)(E,9)
D	(A,8)(E,3)(F,2)
E	(C,9)(D,3)(F,6)(G,4)
F	(B,2)(D,2)(E,6)(G,10)
G	(E,4)(F,10)

We are looking for a shortest route to go from A to G. The evaluation function h should assume that the cities are aligned in alphabetical order at a respective distance of 1km, i.e. $h(A) = 6, h(B) = 5, \dots, h(F) = 1$.

1. What is the shape of the search tree after having expanded 4 nodes using the uniform-cost strategy? Indicate in which order the nodes are expanded. (5 P)

2. What could a hill-climbing search for that problem look like? Assume the initial state is the route A-B-C-E-F-G. The state transitions consists of either adding, removing or replacing exactly one city. Which route could the hill-climbing strategy find? Which problems can occur? (5 P)

3. Describe the (complete) search tree for the greedy search strategy based on the evaluation function h. Indicate in which order the nodes are expanded. (10 P)

4. Describe the (complete) search tree for the A*-algorithm using the evaluation function h. Indicate in which order the nodes are expanded. Indicate how the (complete) search tree would look like when using an evaluation function h' which assumes the cities are at 5km from one another, i.e. $h'(A) = 30, \dots, h'(F) = 5$. Compare the obtained solutions. How does this depend from h'?

   Hint: Note that the A*-algorithm terminates only when selecting a goal node for expansion, and not when generating that goal-node! (10 P)

Solution:

Note that an optimal route from A to G is A-B-F-D-E-G with path cost $3+2+2+3+4 = 14$. *(This is by no means obvious. Basically the only reason I believe it is because a search program said this is optimal. Checking it by hand is not impossible, but not realistic either.)*

1. *The search tree after 4 node expansions using uniform-cost search looks as follows:*

$$\begin{tikzcd}
 & & & A^1 \arrow[dll] \arrow[dr] \arrow[drr] & & \\
 & B^3_{(3)} \arrow[dl] \arrow[d] \arrow[dr] & & & C^2_{(2)} \arrow[dl] \arrow[d] \arrow[dr] & D_{(8)} \\
A_{(3+3=6)} & C_{(3+2=5)} & F_{(3+2=5)} & A^4_{(2+2=4)} \arrow[dl] \arrow[d] \arrow[dr] & B_{(2+2=4)} & E_{(2+9=11)} \\
 & & B_{2+2+3=7} & C_{2+2+2=6} & D_{2+2+8=12} &
\end{tikzcd}$$

   The order of the node expansions is indicate in the superscripts. The path cost associated with each node is indicated in the subscripts of each node. The fourth node we expanded could have been the node B with path cost 4 instead of the node A with path cost 3.

   (Note that the heuristic function h is not relevant for uniform-cost search since uniform-cost search is a form of "uninformed search".)

2. *The search space described in this part (Part 2) of the problem is different from the search space in the other parts of this problem. In parts 1, 3 and 4, the search space consists of states corresponding to cities and successors (transitions) indicated by roads between cities. On the other hand, in this part (Part 2), a state is a sequence of cities leading from A to G (a "route"), where each subsequent city in the sequence is connected by a road. A "successor" of a state (sequence of cities) is obtained by adding, removing or replacing one city in the route. (There is no step cost associated with making such a change.)*

   Instead of trying to achieve a goal (for example, reaching G), we are considering an optimization problem. In order to use hill-climbing search, we need an "objective function" to optimize. This objective function is given by summing the actual distances between the cities in a given route. This gives the actual distance of the route from A to G. "Optimizing" (in this case) means finding a route with minimum total distance.

   (Note that the heuristic function h is not relevant here.)

   (a) *We start with initial state A-B-C-E-F-G. The total distance is* $3 + 2 + 9 + 6 + 10 = 30$.

   (b) *Hill-climbing chooses a successor with minimum total distance (among possible successors). There nine possible successors (two by removing a city, two by changing a city, and five by adding a city). Among these possible successors, the new route A-B-C-E-G obtained by removing F has the least total distance* $3 + 2 + 9 + 4 = 18$. *Hill-climbing chooses this successor.*

   (c) *From A-B-C-E-G there are five possible successors (one by removing B, one by changing C to F, and three by adding a city). Among these, the two possible successors A-C-E-G (obtained by removing B) and A-B-F-E-G (obtained by changing C to F) both have total distance* 15. *Hill-climbing could choose either arbitrarily. We consider both possibilities.*

   (d) *If we choose A-C-E-G, there are three possible successors. All three have total distance of at least* 15, *so hill-climbing stops with route A-C-E-G.*

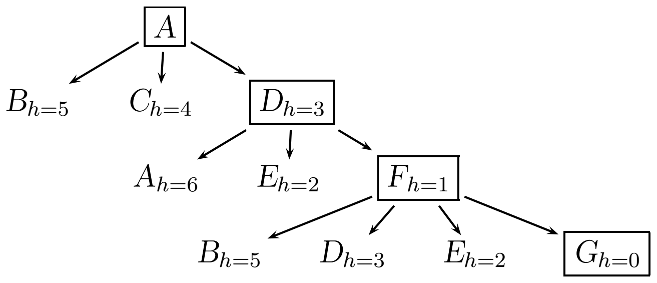
   (e) *Suppose we choose A-B-F-E-G instead of A-C-E-G. In this case there are seven possible successors. One of these successors (obtained by adding the city D between F and E) is an optimal solution A-B-F-D-E-G with path cost* 14. *Hill-climbing should choose this.*

   (f) *Finally, A-B-F-D-E-G has 9 possible successors, but all have total path cost at least* 15. *So hill-climbing terminates with A-B-F-D-E-G.*

   Note that, based on an arbitrary choice, hill-climbing will either terminate with the non-optimal solution A-C-E-G (with path cost of 15*) or optimal solution A-B-F-D-E-G (with path cost* 14*).*

   The problem of returning local (instead of global) optimums (e.g., A-C-E-G) is a general problem with hill-climbing algorithms.

3. *The greedy search strategy uses the heuristic function h to evaluate a node without considering the costs of that node. Thus, we obtain the following search tree.*



The nodes are expanded in the obvious order (A, then D, then F and finally G). Note that the found solution A-D-F-G (cost = 20) is not optimal.

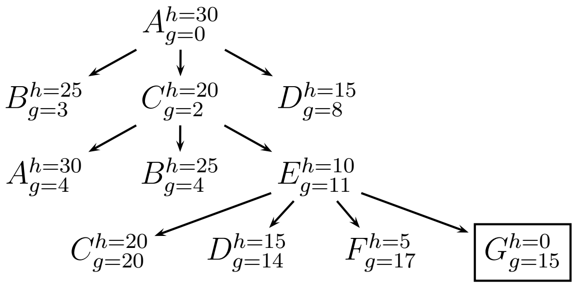
4. *The A⋆ algorithm uses the sum of the costs of a node (the path cost g) plus the heuristic function to determine the value of a node. It expands always the node that has the lowest value (choosing arbitrarily among nodes with the same lowest value).*

   (a) *Figure 1 shows the search tree for A^* using the heuristic function h and evaluation function $f = g + h$. The search tree is also described in a linearized form below. We describe the complete search tree by tracing the search below. In this case, A^* finds the optimal solution A-B-F-D-E-G with total path cost* 14.

   (b) *The trace of the search with h' is*

```
Selected node A with f(A) = g(A) + h(A) = 0 + 30 = 30
Expanding node (A, g(A)=0, h(A)=30) into ((B . 3) (C . 2) (D . 8))
Selected node C with f(C) = g(C) + h(C) = 2 + 20 = 22
Expanding node (C, g(C)=2, h(C)=20) into ((A . 4) (B . 4) (E . 11))
Selected node E with f(E) = g(E) + h(E) = 11 + 10 = 21
Expanding node (E, g(E)=11, h(E)=10) into ((C . 20) (D . 14) (F . 17)
                                           (G . 15))
Selected node G with f(G) = g(G) + h(G) = 15 + 0 = 15
Goal G reached with costs 15
Expanded 3 nodes
Generated 10 nodes
```

   The explored search tree using the heuristic function h' is



   In this case, A^* finds the suboptimal solution A-C-E-G with path cost 15. *The function h' overestimates the distance to G, for instance in the case of A. In particular, $h'(A) = 30$ even though the actual shortest distance to G is only* 15. *Thus the heuristic function h' is not "admissible". An important condition for A^* to be optimal is that the heuristic function be "admissible" (not overestimate actual path costs). That is, if h is admissible, then A^* must find the optimal solution. As we see in this example, A^* may find a solution which is not optimal if the heurisitic function is not admissible.*

A bit more discussion of this problem...

The original heuristic function h is admissible, but is not "consistent". (In class we said such heuristics exist, but did not give one.) Consequently, in the trace of the search using h, something strange happens when we expand the first node for state F (corresponding to the node in position `[1 1 3]` *in the search tree, representing the path A-B-F) after expanding the first node for state B (corresponding to the node in position* `[1 1]`*, representing the path A-B). (These are the third and fourth nodes expanded in the search.) The evaluation f actually decreases. In particular, the f value for* `[1 1]` *is $3 + 5 = 8$ and the f value for* `[1 1 3]` *is $5 + 1 = 6$. An intuitive idea underlying A^* search is that we search within "contours" (visiting all nodes with f values $< C$ before visiting a node with value $\geq C$). But this idea only works if h is "consistent" – as defined in the slides. (Questions: Why are we visiting the F state with f value 6 *after* the B state with f value 8? Does that not contradict the idea of searching within contours? Answers: The heuristic h is not consistent, so A^* does not search within contours. A^* is usually used with a consistent heuristic function.) Also, a graph search version of A^* with this h would not necessarily be optimal.*

- *Here is the trace for the original function h*

```
Selected node A with f(A) = g(A) + h(A) = 0 + 6 = 6
Expanding node (A, g(A)=0, h(A)=6) into ((B . 3) (C . 2) (D . 8))
Selected node C with f(C) = g(C) + h(C) = 2 + 4 = 6
Expanding node (C, g(C)=2, h(C)=4) into ((A . 4) (B . 4) (E . 11))
Selected node B with f(B) = g(B) + h(B) = 3 + 5 = 8
Expanding node (B, g(B)=3, h(B)=5) into ((A . 6) (C . 5) (F . 5))
Selected node F with f(F) = g(F) + h(F) = 5 + 1 = 6
Expanding node (F, g(F)=5, h(F)=1) into ((B . 7) (D . 7) (E . 11) (G . 15))
Selected node B with f(B) = g(B) + h(B) = 4 + 5 = 9
Expanding node (B, g(B)=4, h(B)=5) into ((A . 7) (C . 6) (F . 6))
Selected node F with f(F) = g(F) + h(F) = 6 + 1 = 7
Expanding node (F, g(F)=6, h(F)=1) into ((B . 8) (D . 8) (E . 12) (G . 16))
Selected node C with f(C) = g(C) + h(C) = 5 + 4 = 9
Expanding node (C, g(C)=5, h(C)=4) into ((A . 7) (B . 7) (E . 14))
Selected node A with f(A) = g(A) + h(A) = 4 + 6 = 10
Expanding node (A, g(A)=4, h(A)=6) into ((B . 7) (C . 6) (D . 12))
Selected node D with f(D) = g(D) + h(D) = 7 + 3 = 10
Expanding node (D, g(D)=7, h(D)=3) into ((A . 15) (E . 10) (F . 9))
Selected node C with f(C) = g(C) + h(C) = 6 + 4 = 10
Expanding node (C, g(C)=6, h(C)=4) into ((A . 8) (B . 8) (E . 15))
Selected node C with f(C) = g(C) + h(C) = 6 + 4 = 10
Expanding node (C, g(C)=6, h(C)=4) into ((A . 8) (B . 8) (E . 15))
Selected node F with f(F) = g(F) + h(F) = 9 + 1 = 10
Expanding node (F, g(F)=9, h(F)=1) into ((B . 11) (D . 11) (E . 15) (G . 19))
Selected node D with f(D) = g(D) + h(D) = 8 + 3 = 11
Expanding node (D, g(D)=8, h(D)=3) into ((A . 16) (E . 11) (F . 10))
Selected node D with f(D) = g(D) + h(D) = 8 + 3 = 11
Expanding node (D, g(D)=8, h(D)=3) into ((A . 16) (E . 11) (F . 10))
Selected node F with f(F) = g(F) + h(F) = 10 + 1 = 11
Expanding node (F, g(F)=10, h(F)=1) into ((B . 12) (D . 12) (E . 16)
                                          (G . 20))
Selected node F with f(F) = g(F) + h(F) = 10 + 1 = 11
Expanding node (F, g(F)=10, h(F)=1) into ((B . 12) (D . 12) (E . 16)
                                          (G . 20))
Selected node A with f(A) = g(A) + h(A) = 6 + 6 = 12
Expanding node (A, g(A)=6, h(A)=6) into ((B . 9) (C . 8) (D . 14))
```

```
Selected node B with f(B) = g(B) + h(B) = 7 + 5 = 12
Expanding node (B, g(B)=7, h(B)=5) into ((A . 10) (C . 9) (F . 9))
Selected node F with f(F) = g(F) + h(F) = 9 + 1 = 10
Expanding node (F, g(F)=9, h(F)=1) into ((B . 11) (D . 11) (E . 15) (G . 19))
Selected node B with f(B) = g(B) + h(B) = 7 + 5 = 12
Expanding node (B, g(B)=7, h(B)=5) into ((A . 10) (C . 9) (F . 9))
Selected node F with f(F) = g(F) + h(F) = 9 + 1 = 10
Expanding node (F, g(F)=9, h(F)=1) into ((B . 11) (D . 11) (E . 15) (G . 19))
Selected node B with f(B) = g(B) + h(B) = 7 + 5 = 12
Expanding node (B, g(B)=7, h(B)=5) into ((A . 10) (C . 9) (F . 9))
Selected node F with f(F) = g(F) + h(F) = 9 + 1 = 10
Expanding node (F, g(F)=9, h(F)=1) into ((B . 11) (D . 11) (E . 15) (G . 19))
Selected node E with f(E) = g(E) + h(E) = 10 + 2 = 12
Expanding node (E, g(E)=10, h(E)=2) into ((C . 19) (D . 13) (F . 16)
                                          (G . 14))
Selected node C with f(C) = g(C) + h(C) = 8 + 4 = 12
Expanding node (C, g(C)=8, h(C)=4) into ((A . 10) (B . 10) (E . 17))
Selected node E with f(E) = g(E) + h(E) = 11 + 2 = 13
Expanding node (E, g(E)=11, h(E)=2) into ((C . 20) (D . 14) (F . 17)
                                          (G . 15))
Selected node E with f(E) = g(E) + h(E) = 11 + 2 = 13
Expanding node (E, g(E)=11, h(E)=2) into ((C . 20) (D . 14) (F . 17)
                                          (G . 15))
Selected node A with f(A) = g(A) + h(A) = 7 + 6 = 13
Expanding node (A, g(A)=7, h(A)=6) into ((B . 10) (C . 9) (D . 15))
Selected node B with f(B) = g(B) + h(B) = 8 + 5 = 13
Expanding node (B, g(B)=8, h(B)=5) into ((A . 11) (C . 10) (F . 10))
Selected node F with f(F) = g(F) + h(F) = 10 + 1 = 11
Expanding node (F, g(F)=10, h(F)=1) into ((B . 12) (D . 12) (E . 16)
                                          (G . 20))
Selected node A with f(A) = g(A) + h(A) = 7 + 6 = 13
Expanding node (A, g(A)=7, h(A)=6) into ((B . 10) (C . 9) (D . 15))
Selected node B with f(B) = g(B) + h(B) = 8 + 5 = 13
Expanding node (B, g(B)=8, h(B)=5) into ((A . 11) (C . 10) (F . 10))
Selected node F with f(F) = g(F) + h(F) = 10 + 1 = 11
Expanding node (F, g(F)=10, h(F)=1) into ((B . 12) (D . 12) (E . 16)
                                          (G . 20))
Selected node B with f(B) = g(B) + h(B) = 8 + 5 = 13
Expanding node (B, g(B)=8, h(B)=5) into ((A . 11) (C . 10) (F . 10))
Selected node F with f(F) = g(F) + h(F) = 10 + 1 = 11
Expanding node (F, g(F)=10, h(F)=1) into ((B . 12) (D . 12) (E . 16)
                                          (G . 20))
Selected node E with f(E) = g(E) + h(E) = 11 + 2 = 13
Expanding node (E, g(E)=11, h(E)=2) into ((C . 20) (D . 14) (F . 17)
                                          (G . 15))
Selected node E with f(E) = g(E) + h(E) = 11 + 2 = 13
Expanding node (E, g(E)=11, h(E)=2) into ((C . 20) (D . 14) (F . 17)
                                          (G . 15))
Selected node C with f(C) = g(C) + h(C) = 9 + 4 = 13
Expanding node (C, g(C)=9, h(C)=4) into ((A . 11) (B . 11) (E . 18))
Selected node C with f(C) = g(C) + h(C) = 9 + 4 = 13
Expanding node (C, g(C)=9, h(C)=4) into ((A . 11) (B . 11) (E . 18))
Selected node C with f(C) = g(C) + h(C) = 9 + 4 = 13
Expanding node (C, g(C)=9, h(C)=4) into ((A . 11) (B . 11) (E . 18))
```

```
Selected node C with f(C) = g(C) + h(C) = 9 + 4 = 13
Expanding node (C, g(C)=9, h(C)=4) into ((A . 11) (B . 11) (E . 18))
Selected node C with f(C) = g(C) + h(C) = 9 + 4 = 13
Expanding node (C, g(C)=9, h(C)=4) into ((A . 11) (B . 11) (E . 18))
Selected node E with f(E) = g(E) + h(E) = 12 + 2 = 14
Expanding node (E, g(E)=12, h(E)=2) into ((C . 21) (D . 15) (F . 18)
                                           (G . 16))
Selected node A with f(A) = g(A) + h(A) = 8 + 6 = 14
Expanding node (A, g(A)=8, h(A)=6) into ((B . 11) (C . 10) (D . 16))
Selected node A with f(A) = g(A) + h(A) = 8 + 6 = 14
Expanding node (A, g(A)=8, h(A)=6) into ((B . 11) (C . 10) (D . 16))
Selected node D with f(D) = g(D) + h(D) = 11 + 3 = 14
Expanding node (D, g(D)=11, h(D)=3) into ((A . 19) (E . 14) (F . 13))
Selected node B with f(B) = g(B) + h(B) = 9 + 5 = 14
Expanding node (B, g(B)=9, h(B)=5) into ((A . 12) (C . 11) (F . 11))
Selected node F with f(F) = g(F) + h(F) = 11 + 1 = 12
Expanding node (F, g(F)=11, h(F)=1) into ((B . 13) (D . 13) (E . 17)
                                           (G . 21))
Selected node D with f(D) = g(D) + h(D) = 11 + 3 = 14
Expanding node (D, g(D)=11, h(D)=3) into ((A . 19) (E . 14) (F . 13))
Selected node D with f(D) = g(D) + h(D) = 11 + 3 = 14
Expanding node (D, g(D)=11, h(D)=3) into ((A . 19) (E . 14) (F . 13))
Selected node D with f(D) = g(D) + h(D) = 11 + 3 = 14
Expanding node (D, g(D)=11, h(D)=3) into ((A . 19) (E . 14) (F . 13))
Selected node G with f(G) = g(G) + h(G) = 14 + 0 = 14
Goal G reached with costs 14
Expanded 51 nodes
Generated 171 nodes
```

- *Here is the explored search tree in linearized form*

```
[1] A <g(A) = 0, h(A) = 6>:
[1 1] B <g(B) = 3, h(B) = 5>:
[1 1 1] A <g(A) = 6, h(A) = 6>:
[1 1 1 1] B <g(B) = 9, h(B) = 5>:
[1 1 1 1 1] A <g(A) = 12, h(A) = 6>:
[1 1 1 1 2] C <g(C) = 11, h(C) = 4>:
[1 1 1 1 3] F <g(F) = 11, h(F) = 1>:
[1 1 1 1 3 1] B <g(B) = 13, h(B) = 5>:
[1 1 1 1 3 2] D <g(D) = 13, h(D) = 3>:
[1 1 1 1 3 3] E <g(E) = 17, h(E) = 2>:
[1 1 1 1 3 4] G <g(G) = 21, h(G) = 0>:
[1 1 1 2] C <g(C) = 8, h(C) = 4>:
[1 1 1 2 1] A <g(A) = 10, h(A) = 6>:
[1 1 1 2 2] B <g(B) = 10, h(B) = 5>:
[1 1 1 2 3] E <g(E) = 17, h(E) = 2>:
[1 1 1 3] D <g(D) = 14, h(D) = 3>:
[1 1 2] C <g(C) = 5, h(C) = 4>:
[1 1 2 1] A <g(A) = 7, h(A) = 6>:
[1 1 2 1 1] B <g(B) = 10, h(B) = 5>:
[1 1 2 1 2] C <g(C) = 9, h(C) = 4>:
[1 1 2 1 2 1] A <g(A) = 11, h(A) = 6>:
[1 1 2 1 2 2] B <g(B) = 11, h(B) = 5>:
[1 1 2 1 2 3] E <g(E) = 18, h(E) = 2>:
```

```
[1 1 2 1 3] D <g(D) = 15, h(D) = 3>:
[1 1 2 2] B <g(B) = 7, h(B) = 5>:
[1 1 2 2 1] A <g(A) = 10, h(A) = 6>:
[1 1 2 2 2] C <g(C) = 9, h(C) = 4>:
[1 1 2 2 2 1] A <g(A) = 11, h(A) = 6>:
[1 1 2 2 2 2] B <g(B) = 11, h(B) = 5>:
[1 1 2 2 2 3] E <g(E) = 18, h(E) = 2>:
[1 1 2 2 3] F <g(F) = 9, h(F) = 1>:
[1 1 2 2 3 1] B <g(B) = 11, h(B) = 5>:
[1 1 2 2 3 2] D <g(D) = 11, h(D) = 3>:
[1 1 2 2 3 2 1] A <g(A) = 19, h(A) = 6>:
[1 1 2 2 3 2 2] E <g(E) = 14, h(E) = 2>:
[1 1 2 2 3 2 3] F <g(F) = 13, h(F) = 1>:
[1 1 2 2 3 3] E <g(E) = 15, h(E) = 2>:
[1 1 2 2 3 4] G <g(G) = 19, h(G) = 0>:
[1 1 2 3] E <g(E) = 14, h(E) = 2>:
[1 1 3] F <g(F) = 5, h(F) = 1>:
[1 1 3 1] B <g(B) = 7, h(B) = 5>:
[1 1 3 1 1] A <g(A) = 10, h(A) = 6>:
[1 1 3 1 2] C <g(C) = 9, h(C) = 4>:
[1 1 3 1 2 1] A <g(A) = 11, h(A) = 6>:
[1 1 3 1 2 2] B <g(B) = 11, h(B) = 5>:
[1 1 3 1 2 3] E <g(E) = 18, h(E) = 2>:
[1 1 3 1 3] F <g(F) = 9, h(F) = 1>:
[1 1 3 1 3 1] B <g(B) = 11, h(B) = 5>:
[1 1 3 1 3 2] D <g(D) = 11, h(D) = 3>:
[1 1 3 1 3 2 1] A <g(A) = 19, h(A) = 6>:
[1 1 3 1 3 2 2] E <g(E) = 14, h(E) = 2>:
[1 1 3 1 3 2 3] F <g(F) = 13, h(F) = 1>:
[1 1 3 1 3 3] E <g(E) = 15, h(E) = 2>:
[1 1 3 1 3 4] G <g(G) = 19, h(G) = 0>:
[1 1 3 2] D <g(D) = 7, h(D) = 3>:
[1 1 3 2 1] A <g(A) = 15, h(A) = 6>:
[1 1 3 2 2] E <g(E) = 10, h(E) = 2>:
[1 1 3 2 2 1] C <g(C) = 19, h(C) = 4>:
[1 1 3 2 2 2] D <g(D) = 13, h(D) = 3>:
[1 1 3 2 2 3] F <g(F) = 16, h(F) = 1>:
[1 1 3 2 2 4] G <g(G) = 14, h(G) = 0>:
[1 1 3 2 3] F <g(F) = 9, h(F) = 1>:
[1 1 3 2 3 1] B <g(B) = 11, h(B) = 5>:
[1 1 3 2 3 2] D <g(D) = 11, h(D) = 3>:
[1 1 3 2 3 2 1] A <g(A) = 19, h(A) = 6>:
[1 1 3 2 3 2 2] E <g(E) = 14, h(E) = 2>:
[1 1 3 2 3 2 3] F <g(F) = 13, h(F) = 1>:
[1 1 3 2 3 3] E <g(E) = 15, h(E) = 2>:
[1 1 3 2 3 4] G <g(G) = 19, h(G) = 0>:
[1 1 3 3] E <g(E) = 11, h(E) = 2>:
[1 1 3 3 1] C <g(C) = 20, h(C) = 4>:
[1 1 3 3 2] D <g(D) = 14, h(D) = 3>:
[1 1 3 3 3] F <g(F) = 17, h(F) = 1>:
[1 1 3 3 4] G <g(G) = 15, h(G) = 0>:
```

```
[1 1 3 4] G <g(G) = 15, h(G) = 0>:
[1 2] C <g(C) = 2, h(C) = 4>:
[1 2 1] A <g(A) = 4, h(A) = 6>:
[1 2 1 1] B <g(B) = 7, h(B) = 5>:
[1 2 1 1 1] A <g(A) = 10, h(A) = 6>:
[1 2 1 1 2] C <g(C) = 9, h(C) = 4>:
[1 2 1 1 2 1] A <g(A) = 11, h(A) = 6>:
[1 2 1 1 2 2] B <g(B) = 11, h(B) = 5>:
[1 2 1 1 2 3] E <g(E) = 18, h(E) = 2>:
[1 2 1 1 3] F <g(F) = 9, h(F) = 1>:
[1 2 1 1 3 1] B <g(B) = 11, h(B) = 5>:
[1 2 1 1 3 2] D <g(D) = 11, h(D) = 3>:
[1 2 1 1 3 2 1] A <g(A) = 19, h(A) = 6>:
[1 2 1 1 3 2 2] E <g(E) = 14, h(E) = 2>:
[1 2 1 1 3 2 3] F <g(F) = 13, h(F) = 1>:
[1 2 1 1 3 3] E <g(E) = 15, h(E) = 2>:
[1 2 1 1 3 4] G <g(G) = 19, h(G) = 0>:
[1 2 1 2] C <g(C) = 6, h(C) = 4>:
[1 2 1 2 1] A <g(A) = 8, h(A) = 6>:
[1 2 1 2 1 1] B <g(B) = 11, h(B) = 5>:
[1 2 1 2 1 2] C <g(C) = 10, h(C) = 4>:
[1 2 1 2 1 3] D <g(D) = 16, h(D) = 3>:
[1 2 1 2 2] B <g(B) = 8, h(B) = 5>:
[1 2 1 2 2 1] A <g(A) = 11, h(A) = 6>:
[1 2 1 2 2 2] C <g(C) = 10, h(C) = 4>:
[1 2 1 2 2 3] F <g(F) = 10, h(F) = 1>:
[1 2 1 2 2 3 1] B <g(B) = 12, h(B) = 5>:
[1 2 1 2 2 3 2] D <g(D) = 12, h(D) = 3>:
[1 2 1 2 2 3 3] E <g(E) = 16, h(E) = 2>:
[1 2 1 2 2 3 4] G <g(G) = 20, h(G) = 0>:
[1 2 1 2 3] E <g(E) = 15, h(E) = 2>:
[1 2 1 3] D <g(D) = 12, h(D) = 3>:
[1 2 2] B <g(B) = 4, h(B) = 5>:
[1 2 2 1] A <g(A) = 7, h(A) = 6>:
[1 2 2 1 1] B <g(B) = 10, h(B) = 5>:
[1 2 2 1 2] C <g(C) = 9, h(C) = 4>:
[1 2 2 1 2 1] A <g(A) = 11, h(A) = 6>:
[1 2 2 1 2 2] B <g(B) = 11, h(B) = 5>:
[1 2 2 1 2 3] E <g(E) = 18, h(E) = 2>:
[1 2 2 1 3] D <g(D) = 15, h(D) = 3>:
[1 2 2 2] C <g(C) = 6, h(C) = 4>:
[1 2 2 2 1] A <g(A) = 8, h(A) = 6>:
[1 2 2 2 1 1] B <g(B) = 11, h(B) = 5>:
[1 2 2 2 1 2] C <g(C) = 10, h(C) = 4>:
[1 2 2 2 1 3] D <g(D) = 16, h(D) = 3>:
[1 2 2 2 2] B <g(B) = 8, h(B) = 5>:
[1 2 2 2 2 1] A <g(A) = 11, h(A) = 6>:
[1 2 2 2 2 2] C <g(C) = 10, h(C) = 4>:
[1 2 2 2 2 3] F <g(F) = 10, h(F) = 1>:
[1 2 2 2 2 3 1] B <g(B) = 12, h(B) = 5>:
[1 2 2 2 2 3 2] D <g(D) = 12, h(D) = 3>:
```

```
[1 2 2 2 2 3 3] E <g(E) = 16, h(E) = 2>:
[1 2 2 2 2 3 4] G <g(G) = 20, h(G) = 0>:
[1 2 2 2 3] E <g(E) = 15, h(E) = 2>:
[1 2 2 3] F <g(F) = 6, h(F) = 1>:
[1 2 2 3 1] B <g(B) = 8, h(B) = 5>:
[1 2 2 3 1 1] A <g(A) = 11, h(A) = 6>:
[1 2 2 3 1 2] C <g(C) = 10, h(C) = 4>:
[1 2 2 3 1 3] F <g(F) = 10, h(F) = 1>:
[1 2 2 3 1 3 1] B <g(B) = 12, h(B) = 5>:
[1 2 2 3 1 3 2] D <g(D) = 12, h(D) = 3>:
[1 2 2 3 1 3 3] E <g(E) = 16, h(E) = 2>:
[1 2 2 3 1 3 4] G <g(G) = 20, h(G) = 0>:
[1 2 2 3 2] D <g(D) = 8, h(D) = 3>:
[1 2 2 3 2 1] A <g(A) = 16, h(A) = 6>:
[1 2 2 3 2 2] E <g(E) = 11, h(E) = 2>:
[1 2 2 3 2 2 1] C <g(C) = 20, h(C) = 4>:
[1 2 2 3 2 2 2] D <g(D) = 14, h(D) = 3>:
[1 2 2 3 2 2 3] F <g(F) = 17, h(F) = 1>:
[1 2 2 3 2 2 4] G <g(G) = 15, h(G) = 0>:
[1 2 2 3 2 3] F <g(F) = 10, h(F) = 1>:
[1 2 2 3 2 3 1] B <g(B) = 12, h(B) = 5>:
[1 2 2 3 2 3 2] D <g(D) = 12, h(D) = 3>:
[1 2 2 3 2 3 3] E <g(E) = 16, h(E) = 2>:
[1 2 2 3 2 3 4] G <g(G) = 20, h(G) = 0>:
[1 2 2 3 3] E <g(E) = 12, h(E) = 2>:
[1 2 2 3 3 1] C <g(C) = 21, h(C) = 4>:
[1 2 2 3 3 2] D <g(D) = 15, h(D) = 3>:
[1 2 2 3 3 3] F <g(F) = 18, h(F) = 1>:
[1 2 2 3 3 4] G <g(G) = 16, h(G) = 0>:
[1 2 2 3 4] G <g(G) = 16, h(G) = 0>:
[1 2 3] E <g(E) = 11, h(E) = 2>:
[1 2 3 1] C <g(C) = 20, h(C) = 4>:
[1 2 3 2] D <g(D) = 14, h(D) = 3>:
[1 2 3 3] F <g(F) = 17, h(F) = 1>:
[1 2 3 4] G <g(G) = 15, h(G) = 0>:
[1 3] D <g(D) = 8, h(D) = 3>:
[1 3 1] A <g(A) = 16, h(A) = 6>:
[1 3 2] E <g(E) = 11, h(E) = 2>:
[1 3 2 1] C <g(C) = 20, h(C) = 4>:
[1 3 2 2] D <g(D) = 14, h(D) = 3>:
[1 3 2 3] F <g(F) = 17, h(F) = 1>:
[1 3 2 4] G <g(G) = 15, h(G) = 0>:
[1 3 3] F <g(F) = 10, h(F) = 1>:
[1 3 3 1] B <g(B) = 12, h(B) = 5>:
[1 3 3 2] D <g(D) = 12, h(D) = 3>:
[1 3 3 3] E <g(E) = 16, h(E) = 2>:
[1 3 3 4] G <g(G) = 20, h(G) = 0>:
```

- *The explored search tree with the original function h is shown in Fig. 1.*

The solutions have been generated using the following lisp-code:

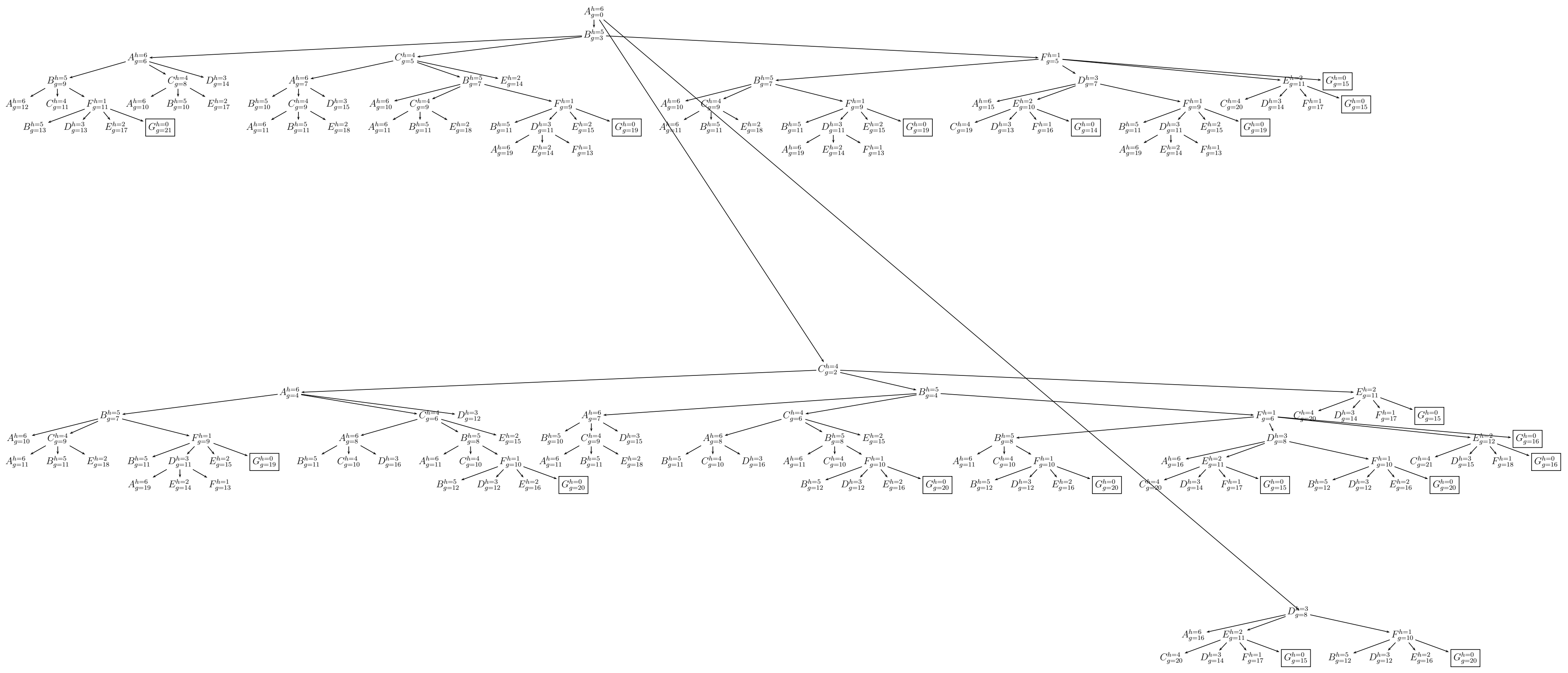


Figure 1: Search tree for A^* with function h

```
;; Serge Autexier
;; 2005

;; (a*-search 'a 'cost-estimate-1 'get-successors 'goal? t)
;; (a*-search 'a 'cost-estimate-2 'get-successors 'goal? t)

(defun a*-search (root-node estimate-costs get-successors goal? latex?)

  (search-tree-reset)
  (a*=search (list (cons root-node 0)) estimate-costs get-successors goal? 0 0)
  (search-tree-print root-node estimate-costs goal? t latex?)
  )


(defun a*=search (node-queue estimate-costs get-successors goal? &optional (expanded-nodes 0) (generated-nodes 0))

  (let* ((minimal-hnode (apply #'min (mapcar #'(lambda (node.cost) (+ (funcall estimate-costs (car node.cost)) (cdr node.cos
 (minimal-node.cost  (find-if #'(lambda (node.cost) (= (+ (funcall estimate-costs (car node.cost)) (cdr node.cost)) minimal-
      node-queue))
 (minimal-node (car minimal-node.cost))
 (minimal-cost (cdr minimal-node.cost)))
    (warn "Selected node ~A with f(~A) = g(~A) + h(~A) = ~D + ~D = ~D"
  minimal-node minimal-node minimal-node minimal-node
  minimal-cost (funcall estimate-costs minimal-node)
  (+ (funcall estimate-costs minimal-node) minimal-cost))
    (if (funcall goal? minimal-node)
(progn
  (warn "Goal ~A reached with costs ~D" minimal-node minimal-cost)
  (warn "Expanded ~D nodes~%Generated ~D nodes" expanded-nodes generated-nodes))
      (let* ((subnodes (funcall get-successors minimal-node) ;; resturns a list of dotted pairs (successor . cost)
       )
     (subnodes.real-costs (mapcar #'(lambda (node.costs) (cons (car node.costs) (+ (cdr node.costs)
   minimal-cost)))
  subnodes)))
(setq expanded-nodes (1+ expanded-nodes)
      generated-nodes (+ generated-nodes (length subnodes.real-costs)))
(search-tree-insert-expansion minimal-node.cost estimate-costs subnodes.real-costs)
(warn "Expanding node (~A, g(~A)=~D, h(~A)=~D) into ~A" minimal-node minimal-node minimal-cost minimal-node
      (funcall estimate-costs minimal-node) subnodes.real-costs)
(a*=search (append (remove minimal-node.cost node-queue) subnodes.real-costs) estimate-costs get-successors goal?
   expanded-nodes generated-nodes)))))

(let ((search-tree nil))
  (defun node-cost-id (node.cost cost-estimate)
    (list (car node.cost) (cdr node.cost) (funcall cost-estimate (car node.cost))))


  (defun find-node-id-successors (node-id)
    (cdr (assoc node-id search-tree :test #'equal)))

  (defun search-tree-reset ()
    (setq search-tree nil))

  (defun search-tree-insert-expansion (node.cost cost-estimate subnodes.cost)
    (Setq search-tree
  (acons (node-cost-id node.cost cost-estimate)
 (mapcar #'(lambda (subnode.cost)
     (node-cost-id subnode.cost cost-estimate))
 subnodes.cost)
 search-tree)))

  (defun search-tree-print (root cost-estimate goal? &optional (stream t) (latex? nil))
    (format stream "~%")
    (search-tree-print-internal (node-cost-id (cons root 0) cost-estimate) goal? stream (list 1) latex?))

  (defun search-tree-print-internal (node-id goal? stream &optional depthlist latex?)
    (if latex?
(if (find-node-id-successors node-id)
    (format stream "
pstree
Tr~A_g=~D^h=~D~%"
    (first node-id) (second node-id) (third node-id))
  (if (funcall goal? (car node-id))
      (format stream "
Tr
psframebox[linestyle=solid,linecolor=black]~A_g=~D^h=~D~%"
      (first node-id) (second node-id) (third node-id))
    (format stream "
Tr~A_g=~D^h=~D~%"
    (first node-id) (second node-id) (third node-id))))
      (format stream "~%[~~D~^ ~] ~A <g(~A) = ~D, h(~A) = ~D>:" depthlist
      (first node-id) (first node-id) (second node-id) (first node-id) (third node-id)))
    (let ((counter 0))
      (dolist (subnode-id (find-node-id-successors node-id))
(search-tree-print-internal subnode-id goal? stream (append depthlist (list (incf counter))) latex?))
      (when (and latex? (find-node-id-successors node-id))
(format stream "~%"))
      ))

  )



(defun cost-estimate-1 (node)

  (case node
    (A 6)
    (B 5)
    (C 4)
    (D 3)
    (E 2)
    (F 1)
    (G 0))
  )

(defun cost-estimate-2 (node)

  (case node
    (A 30)
    (B 25)
    (C 20)
    (D 15)
    (E 10)
    (F 5)
    (G 0))
  )

(defun goal? (node)

  (eq node 'g)
  )


(defun get-successors (node)

  (case node
    (A '((B . 3) (C . 2) (D . 8)))
    (B '((A . 3) (C . 2) (F . 2)))
    (C '((A . 2) (B . 2) (E . 9)))
    (D '((A . 8) (E . 3) (F . 2)))
    (E '((C . 9) (D . 3) (F . 6) (G . 4)))
    (F '((B . 2) (D . 2) (E . 6) (G . 10)))
    (G '((E . 4) (F . 10))))
  )
```