Principles of Artificial Intelligence
Fall 2014
Heuristic Problem Solvers

Vasant Honavar
Artificial Intelligence Research Laboratory
College of Information Sciences and Technology
Pennsylvania State University
State College, PA 16802

Last revised: September 7, 2014

1 Heuristic Search

Most interesting AI problems are susceptible to combinatorial explosion. If we use blind search, we run out of space, time or both very quickly. An alternative is to try to estimate the cost of the solution by using heuristics, i.e. methods that improve the performance of search algorithms in practice.

Judea Pearl, who has studied heuristic search in a lot of detail, had this to say about the role of heuristics in search in general and AI in particular:

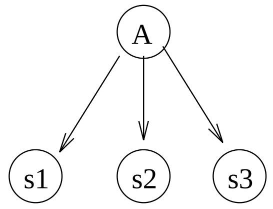
> Heuristics, patient rules of thumb! So often scorned, sloppy! dumb! Yet slowly, common sense become. - Ode to AI Judea Pearl

When given several choices that can be made at any given time to affect a future outcome, the only known strategy that is absolutely guaranteed to work (identify the best choice) under *all* circumstances is one that examines the entire search space. Such a strategy, although completely rational, is impractical in all but the most trivial problems. If we were to apply this principle in real life, we would spend all our time thinking what to do next and be utterly incapable of *doing* anything.

Informally, heuristics are criteria or principles for deciding which among a set of alternatives "promises" to be the "best" with the use of *modest* computational resources. This approach to decision making is closely related in philosophy to *the principle of limited rationality* proposed by Herbert Simon, one of the founders of AI and a Nobel laureate in economics.

We could design a heuristic which looks at all choices and chooses the best but it wouldn't be a good one because computing such a heuristic would be as expensive as doing an exhaustive search! Typically, we would be interested in heuristic functions that are easy to compute and are informative. We will formalize these notions as well as study some ways to discover such heuristics later.

Definition: A heuristic function $h : \Psi \longrightarrow \mathbf{R}^+$, where Ψ is set of all states and $\mathbf{R}^+$ is the set of positive real numbers, maps each state s in the state space Ψ into a measurement $h(s)$ which is an *estimate* of the cost of a path obtained by extending the partial path from the start node S to s, the node under consideration.



In the graph above, node A has 3 children. Suppose the heuristic values associated with these nodes are as follows:

$$\begin{aligned} h(s1) &= 0.8 \\ h(s2) &= 2.0 \\ h(s3) &= 1.6 \end{aligned}$$

Each of these values is an *estimate* of the cost of a path to goal obtained by extending the current path through the corresponding nodes. Without loss of generality, we will restrict ourselves to heuristic functions that are cost functions i.e., $\forall$ states n in the state space Ψ, $h(n) \geq 0$. We don't allow negative costs. The lower the value of $h(n)$ is, the smaller is the estimated cost of reaching a goal node via a path through n.

S0 - Start

2	8	3
1	6	4
7		5

g - Goal

1	2	3
8		4
7	6	5

In the above 8-puzzle example, suppose $h(s) =$ "the number of out-of-place tiles (not counting the blank tile) in state s (relative to the goal)."

$$\begin{aligned} h(S0) &= 4 \\ h(g) &= 0 \end{aligned}$$

$h(s)$ satisfies our definition and criteria for a heuristic; It doesn't take much effort to compute $h(n)$ and $\forall n\ h(n) \geq 0$. Since it takes at least one move to fix a misplaced tile, this function also underestimates the amount of work needed. We will later show that such optimism is a desirable property of heuristic functions.

Heuristics can guide BF or DF search by using h values to order the list of partial paths and select then the best path. If we do this in DF order, we get Hill Climbing search; if we do this in BF order, we get Best First search. To further study *Heuristic Search*, we look at these 2 particular examples of search that use the method of heuristic estimates:

1. **Hill Climbing:** (refer to fig. 1)

```
S
SB SC SA
SBF SBH SC SA
SC SA
SCG SCI SA
```

2. **Best-First Search:**

```
S
SB SC SA
SC SA SBF SBH
SCG SCI SA SBF SBH
```

Hill Climbing is essentially a Depth-First Search. However, the choices are ordered according to some heuristic measure of the remaining distance to the goal. In this case, we order the children according to the "h" (heuristic function) values.

Since Hill Climbing is just a Depth-First Search (with quality measurements), it is associated with the same problems as Depth First Search.

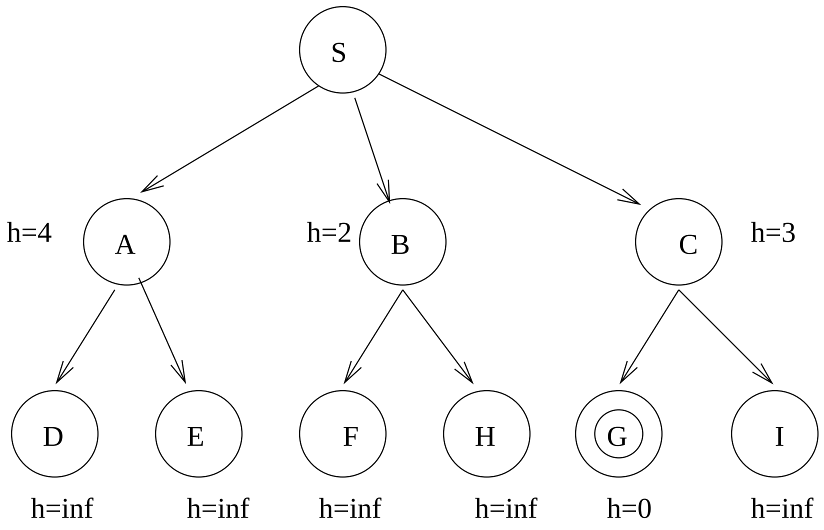


Figure 1: Heuristic search

1. the search may be following a potentially infinite path;
2. maximum memory requirement $\sim \infty$ (bd);
3. worst case time complex analysis is $O(b^d)$;
4. run into the problem of getting stuck in a local minima and need to backtrack.

Observation: Hill Climbing search is not complete nor admissible.

2 A* Search

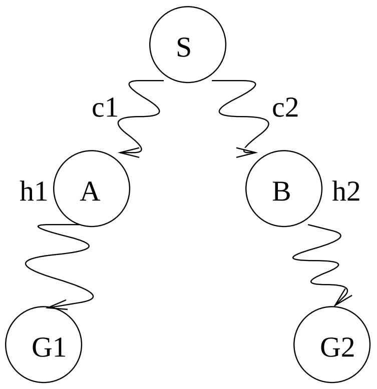
This brings us to our goal of arriving at an optimal heuristic search algorithm; one that finds an optimal (cheapest or of highest quality) solution with minimal search effort given the information provided. To enable us to do so, we first have to quantify things like the information available, the quality of a solution, and the search effort.

2.1 A* Algorithm

By combining the *Best-First Heuristic Search*, using the estimated cost of completing a partial solution as measured by h values, and the *Branch-and-Bound Search*, where we make use of the cost information, we get the A* Algorithm. This can be seen in the evaluation function of the A* Algorithm $f(n) = g(n) + h(n)$ where $g(n)$ represents the known least-cost path from s to n, $h(n)$ is the estimated least-cost path from n to a goal node, and $f(n)$ represents the estimated cost of the path from the start node s to a goal node g that is constrained to pass through n.

Essentially, A* is an algorithm for finding an optimal solution with minimal effort. It is basically a refinement of B&B (Branch-and-Bound) search with dynamic programming by adding guesses (or heuristic estimates) of the cost of reaching a goal node from the current state by extending the current partial path. The estimated total path length for such a search would be the distance already traveled plus the estimate of remaining distance (measured by "h").

The basic idea of A* is to always extend the partial path with the cheapest estimated path length:

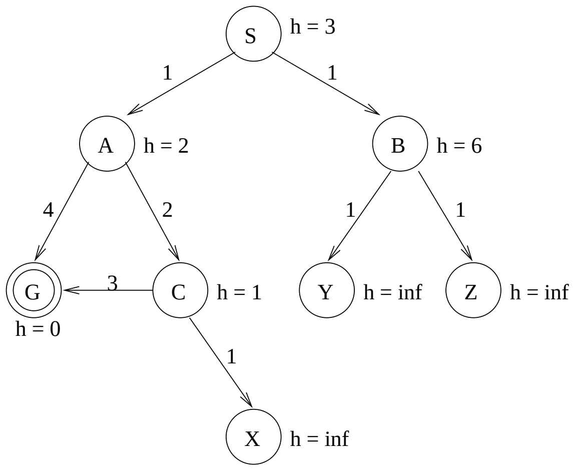


[(S)]
[(SA),(SB)]
If (c1+h1)<(c2+h2) expand SA next
else expand SB next

2.2 Pseudo Code for A*

```
1. Initialize $L$ to a path of length 0 containing $s$.
2. If the list $L$ is empty return FAILURE.
3. Pick the node $x \epsilon L$ that has the least $f$ value among all nodes in $L$.
4. If $x \epsilon G$ (where $G$ is the goal set) then return the path $s \ldots x$ and stop else expand $x$ and place its
   children on $L$.
5. Go to step 2.
```

2.3 Example of A*

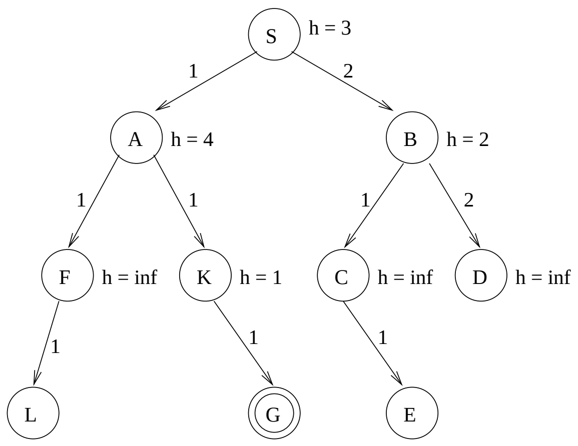


(path, f(n))
[(S, 3)]
[(SA, 3), (SB, 7)]
[(SAC, 4), (SAG, 5), (SB, 7)]
[(SAG, 5), (SACG,6), (SB, 7), (SACX, inf)]

Graph of Example using A*.

2.4 Examples of A* Search

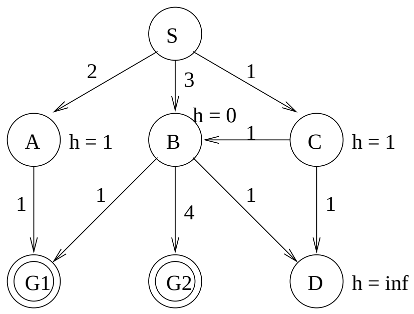
Exercise 1: Write down the list of partial path maintained by A*:



[(S, 3)]
[(SB, 4), (SA, 5)]
[(SA, 5), (SBC, inf), (SBD, inf)]
[(SAK, 3), (SAF, inf), (SBC, inf), (SBD, inf)]
[(SAKG, 3), (SAF, inf), (SBC, inf), (SBD, inf)]

Graph for Exercise 1.

Exercise 2: Calculate the cost of finding the goal node. $f(n) = g(n) + h(n)$, where $g(n)$ = cost of the cheapest *known* path to n, and $h(n)$ = *estimated* cheapest cost of going from n to a goal.



[(S,?)]
[(SC,2), (SA,3), (SB,3)]
[(SCB,2), (SA,3), (SB,3), (SCD,inf)]
[(SCBG1,3), (SA,3), (SB,3), (SCBG2,6), (SCBD,inf),(SCD,inf)]

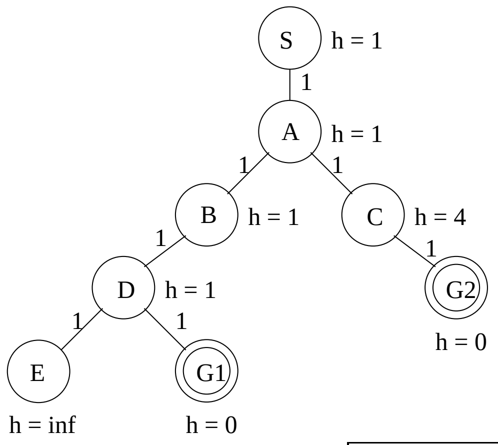
Graph for Exercise 2.

2.5 Properties of A*

We consider the following questions:

1. What can we say about A* if h is a perfect heuristic? It turns out that when provided with the perfect heuristic, A* performs essentially no search. The algorithm focuses on the best path.

2. What if h overestimates the actual cost (of reaching a goal)? We will see that in this case it is possible to miss an optimal path (in favor of a worse path).



[(S, 1)]
[(SA, 2)]
[(SAB, 3), (SAC, 6)]
[(SABD, 4), (SAC, 6)]
[(SABDG1, 4), (SAC, 6), (SABDE, inf)]

As a result of over-estimation of h,
G1 will be found instead of G2.

Graph with overestimate.

3. What if h is guaranteed to be an underestimate (of the actual costs of reaching a goal)? In this case, we will prove that in this case, A* is guaranteed to find an optimal path (if a path exists).

Examples of Heuristic Functions

1. In trying to find the shortest path between two cities by road, a heuristic function which is an underestimate is the straight line distance between the two cities.

2. In the puzzle problem with 9 positions and 8 numbered tiles, the number of tiles in the incorrect position is a heuristic function which is an underestimate.

The evaluation function of A* Algorithm is $f(n) = g(n) + h(n)$ where $g(n)$ represents the cost of the cheapest known path from s to n and $h(n)$ is the *estimated* least-cost path from n to a goal node.

Definition. A search algorithm is said to be *complete* if it is guaranteed to terminate with a solution whenever a solution exists.

Definition. A search algorithm is said to be *admissible* if it is guaranteed to terminate with an optimal (i.e., cheapest) solution whenever a solution exists.

Search	Complete	Admissible
Depth First	No	No
Breadth First	Yes	No (in general) Yes (for uniform cost arcs)
IDS	Yes	No (in general) Yes (for uniform cost arcs)
BBS	Yes	Yes
Best First	No	No
A*	Yes (for locally finite graphs)	Yes (provided $h(n)$ is an underestimate)

We will now prove that A* is admissible.

2.6 Admissibility of A*

Theorem. A* is admissible (i.e. it is guaranteed to terminate with an optimal solution if a solution exists) if:

1. the branching factor is finite.

2. for all n_i, n_j in Ψ, $k(n_i, n_j) \geq \delta > 0$ (i.e. all costs are positive) where $k(n_i, n_j)$ is the actual cost of the minimum cost path from n_i to n_j (infinite or undefined if no path).

3. for all n in Ψ, $0 \leq h(n) \leq h^*(n)$ (i.e. all heuristics are underestimates).

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

- $g(n)$ = cost of cheapest known path from s to n;
- $h(n)$ = estimate of cheapest cost from n to a goal;
- $f(n)$ = estimated cost of cheapest path from s to g through n.

Let $c(n_i, n_j)$ = cost of an arc connecting node n_i to n_j $\boxed{c(n_i, n_j) \geq \delta > 0}$.

The actual cost of the minimum cost path from n to a goal $g_i = k(n, g_i)$ and $h^*(n) = \underset{g_i \in G}{min}\{k(n, g_i)\}$

Clearly, any path from n to a goal in G (the set of all goal nodes) with cost $= h^*(n)$ is an optimal path from n to that goal.

Definition: Let $g^*(n) = k(s, n) \forall n$ accessible from s. Define $f^*(n) = g^*(n) + h^*(n)$. (The cost of the optimal path from s to a goal that is constrained to pass through node n).

$f(n)$, $g(n)$ and $h(n)$ are estimates of $f^*(n)$, $g^*(n)$ and $h^*(n)$ respectively, where $f^*(n)$, $g^*(n)$ and $h^*(n)$ are optimal.

Note that $g(n)$ = "sum of arc costs along the shortest known path from s to n" and can be modified as we discover cheaper/better paths to n. Hence, $g^*(n) \leq g(n)$.

In order to prove that A^* is *admissible*, we've to first prove that it is *complete*. We do so using proof by contradiction. That is we assume that A^* is *not* complete, i.e. it can fail to terminate with a solution even if one exists. This implies that new paths are forever being added to the list of partial paths, thus making the f values of such a path $\to \infty$. Let us formalize this argument:

Lemma: A* is complete.

Proof: (by contradiction) Suppose A* is not complete. This implies that A* does not terminate with a solution even if one exists. This means that new paths are forever being added to the list of partial paths.

> *Claim:* The f-values for such paths must approach infinity.
> *Proof:* Let $d^*(n)$ be the number of arcs along an optimal path from s to n. Then $g^*(n) \geq \delta d^*(n)$, because all arc costs are greater than δ. $g(n) \geq g^*(n)$ by definition, so $g(n) \geq \delta d^*(n)$. Since $f(n) = g(n) + h(n)$, $f(n) \geq \delta d^*(n) + h(n)$. So if an infinite path is being pursued,
> $\boxed{d^*(n) \to \infty \text{ and so } f^*(n) \to \infty \text{ along such a path.}}$
>
> *Claim:* If there is a finite path from s to a goal, then, at any time before the termination of A*, there exists some partial path from s to n on the list of partial paths such that $\underline{f(n) \leq f^*(s)}$, which is the cost of an optimal solution.
>
> *Proof:* Let $(s, n_1, n_2, \ldots, g)$ be an optimal path from s to a goal. Let n' be the first node in the sequence such that $(s, n_1, n_2, \ldots, n')$ appears on the list of partial paths at some time during the execution of A*. Note that $f(n') = g(n') + h(n')$. Now, since $g(n') = g^*(n')$ (because it is on the optimal path), $f(n') = g^*(n') + h(n') \leq g^*(n') + h^*(n') = f^*(n')$, so $f(n') \leq f^*(n')$. Because n' is on the optimal path, $f^*(n') = f^*(s)$ and $\boxed{f(n') \leq f^*(s)}$. So at any time before the termination of A*, there is a partial path on the list whose cost is bounded.

Since there is a partial path whose cost is bounded, and the costs of all infinite paths grow without bound, A* will eventually select the partial path whose cost is bounded over a potentially infinite path whose cost is growing without bound. This contradicts the supposition that A* is not complete. Thus A* must be complete. $\square$

Now that we have shown that A* is complete, it is easy to show that it is admissible.

Proof: (by contradiction)
Assume A* has terminated with a non-optimal solution $f(g) > f^*(s)$. But since there must be a partial path $(s \cdots n')$ on the list at any time before the termination of A* with $f(n') \leq f^*(s)$, A* would have picked that path over the non-optimal solution and hence could not have terminated with a non-optimal path $(s \cdots g)$, contrary to our assumption. Therefore, A* must be admissible. $\square$

2.7 Comparing Heuristic Functions

How can we choose between two heuristic functions when we know nothing else about the search space?

Definition: A heuristic function is said to be *admissible* if, for all $n \in \Psi$, $0 \leq h(n) \leq h^*(n)$.

Consider two admissible heuristics $h_1(n)$ and $h_2(n)$, where $\forall n \in \Psi$, $0 \leq h_1(n) \leq h^*(n)$ and $0 \leq h_2(n) \leq h^*(n)$.

Intuitively, if for all n, $h(n) = 0$ then the function provides zero heuristic information, and A* is simply branch and bound search with dynamic programming. Alternatively, if for all n, $h(n) = h^*(n)$ then the function provides perfect heuristic information, and search is virtually avoided.

Definition: Given two admissible heuristic functions h_1 and h_2, we say that $h_2(n)$ is *more informed* than $h_1(n)$ if, for all non-goal nodes in the search space Ψ, $h_2(n) > h_1(n)$. If A_1 is an A* algorithm that uses $f_1(n) = g_1(n) + h_1(n)$ and A_2 is an A* algorithm that uses $f_2(n) = g_2(n) + h_2(n)$, we say that A_2 is *more informed* than A_1.

Intuitively, we would expect more information to translate to less search effort.

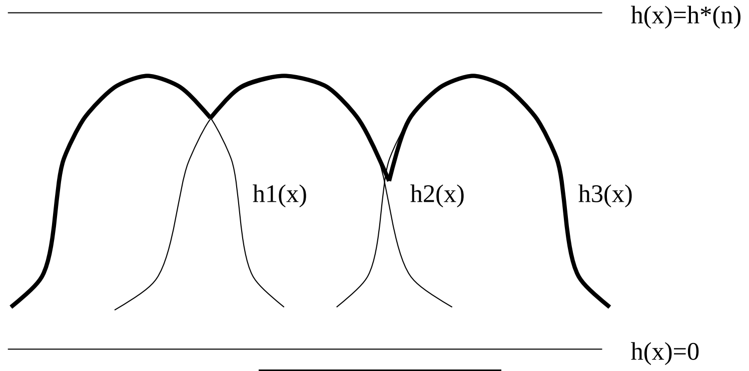
Theorem: If A_1 and A_2 are versions of A* using admissible heuristic functions h_1 and h_2 respectively, and A_2 is more informed than A_1, then upon termination with a solution, every path extended by A_2 is also necessarily extended by A_1.

(Remark: this does not necessarily imply that A_2 takes less time than A_1 to find an optimal solution, unless the work involved in computing $h_2(n)$ is no more that that involved in computing $h_1(n)$).

Proof: (by induction)
Consider the search trees generated by A$_1$ and A$_2$ up to some depth k. Clearly, both extend the partial path (s) unless s is a goal (base case). Suppose that A$_1$ extends every partial path extended by A$_2$ up to some depth k (induction hypothesis). Then we prove that A$_1$ extends every partial path extended by A$_2$ up to depth $k+1$ (induction step).

The induction hypothesis implies that A$_1$ necessarily finds a path from s to some node n at depth k that is no worse than that found by A$_2$, or that $g_1(n) \leq g_2(n)$ (because A_1 has extended every path extended by A_2 up to depth k). Assume the opposite, that A$_1$ did not extend s to some node n (at depth $k+1$) while A$_2$ did. This implies that, upon termination, the path $(s, n_1 \dots n)$ is on the list of partial paths for A$_1$, or that $f_1(n) \geq f^*(s)$. Then $f^*(s) \leq g_1(n) + h_1(n)$. If $(s, n_1 \dots n)$ is extended by A$_2$ then $f^*(s) \geq f_2(n) = g_2(n) + h_2(n)$. This implies that $g_1(n) + h_1(n) \geq g_2(n) + h_2(n)$, or $g_1(n) - g_2(n) \geq h_2(n) - h_1(n) \geq 0$, which contradicts $g_1(n) \leq g_2(n)$ (established by the induction hypothesis). So A$_1$ extends every partial path extended by A$_2$.

Corollary: Suppose we have admissible heuristic functions $h_1(n), \dots h_m(n)$, then a combined heuristic function $h(n) = max[h_1(n), \dots h_m(n)]$ would result in no more search effort (loosely speaking) than any of the individual heuristic functions.



Multiple heuristics.

3 Problem Solving Through Problem Reduction

Consider an agent that is given a goal to be achieved. Suppose the agent knows how to successively transform the goal into simpler subproblems. The goal is achieved by successively decomposing or transforming it into subproblems until we end up with *primitive* problems (i.e. those problems whose solutions are known or can be looked up in a table or can be obtained using a known algorithm). The example that follows illustrates this process.
Example:

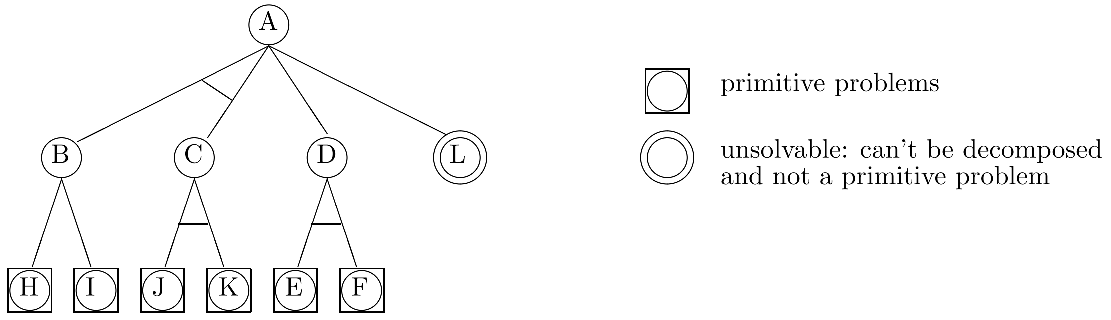
$$\begin{tikzcd}
 & \int (2t + sec^2 t\, sin^2 t + 7)\,dt \arrow[dl] \arrow[d] \arrow[dr] & \text{AND} \\
\int 2t\,dt + & \int sec^2 t\, sin^2 t\,dt + \arrow[dl] \arrow[dr] & \int 7\,dt \\
\int sec^2 t\,(1 - cos^2 t)\,dt & \text{OR} & \int (1 + tan^2 t) sin^2 t\,dt
\end{tikzcd}$$

Such a decomposition is called problem reduction representation (PRR). A PRR is specified by the 3-tuple (G, O, P), where

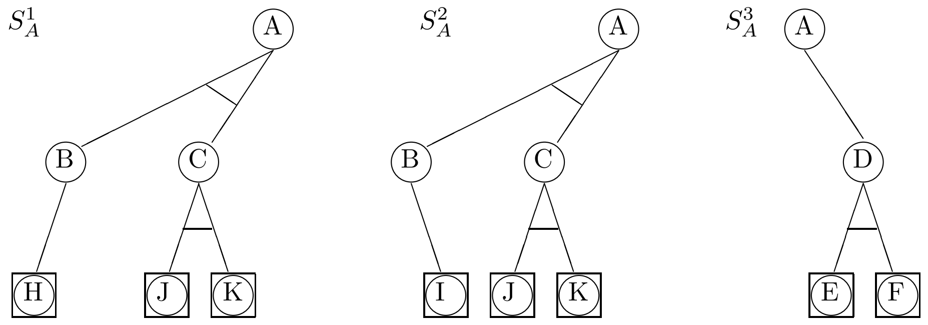
G = Problem to be solved
O = Set of operators for transforming a problem into subproblems using AND
    or OR decompositions
P = Set of primitive problems (those we know how to solve)

Such problem decompositions can be represented using **AND-OR graphs** in which the nodes represent problems and subproblems to be solved and the arcs have been generalized to AND and OR **connectors**. This was shown with the integral calculus example above.

Example of AND-OR graph:



There are 3 solutions to the above example:



Definition: A *subgraph* S_x of an AND-OR graph (G, O, P) is said to be a solution of problem x iff:

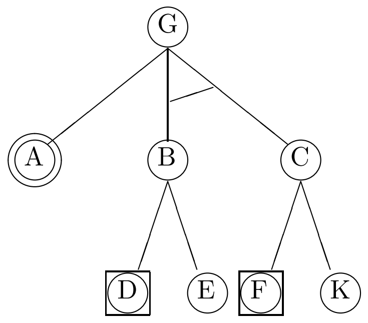
1. x is the root of S_x,
2. for every non-leaf node $y \in S_x$, exactly one connector k going out of y along with all the nodes at the tip of the connector is in S_x,
3. all leaf nodes in S_x are elements of P (the set of primitive problems).

Definition: We say that a node x is *solvable* iff

1. $x \in P$ (i.e., x is a primitive problem) **or**
2. x is a non-leaf node such that all its children at the tip of at least one of its connectors are solvable.

Essentially, solving a problem G using AND-OR graphs entails finding a solution to S_G. Now how can we generalize our search algorithms to include AND-OR graphs? Instead of placing nodes on a list maintained by the search algorithm, we place connectors or partial solution graphs on the list.

Example:



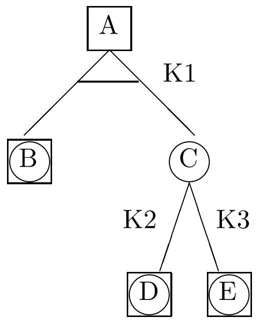
BFS

(G)
(A) (B & C)
if you find out A can't be solved then you get:
(B & C)
(D & F) (E & F) (D & K) (E & K)
Note that when the search terminates in this example, nodes E and K have not yet been checked; they may be primitive, but we do not yet know.

Exercise: Solve same example using DFS.

3.1 Heuristic Search of And-Or graphs

Example:

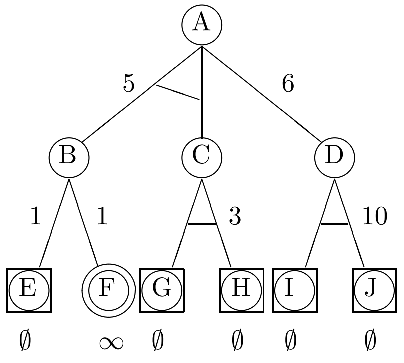


$Cost(S_A) = Cost(k_1) + Cost(S_B) + Cost(S_C)$
$Cost(S_C) = Min\{(Cost(k_2) + Cost(S_D), (Cost(k_3) + Cost(S_E)\}$

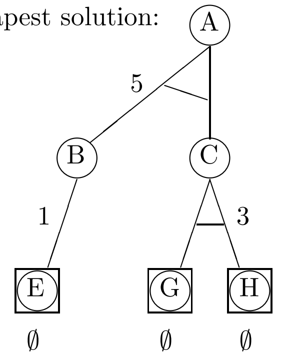
Cost of connectors and primitive problems are assumed to be positive and bounded.

$Cost(n) = \infty$ if n is unsolvable.

Example:



Cheapest solution:

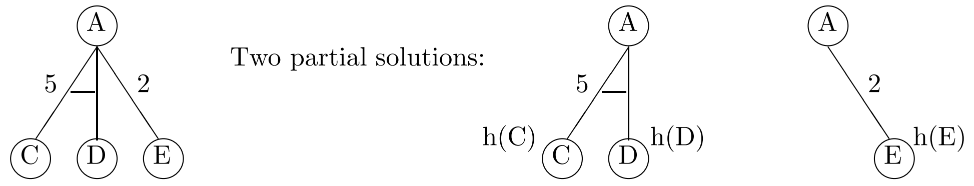


$S_{Amin} = 5 + 1 + 3 = 9$

3.2 Searching for an optimal solution

We consider defining a heuristic function $h(x)$ for a solution graph rooted at x. If $h(n) \leq h^*(x)$ (where $h^*(x)$ is the cost of the cheapest (optimal) solution rooted at n), we say that $h(n)$ is **admissible** and we can easily adapt A* to an analogous algorithm AO* to work on AND-OR graphs.

Example: Calculation of $f(x)$

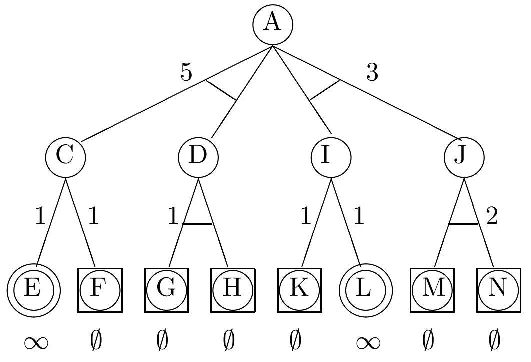


$f(A) = min[f_1(A), f_2(A)]$ is the estimated cost of the cheapest solution rooted at A.
The above graph has two partial solutions (as shown) with the following costs:
$f_1(A) = 5 + h(C) + h(D)$

$f_2(A) = 2 + h(E)$

Example:

$h(C) = h(D) = h(I) = h(J) = 1$



Search List:

(A)
(I & J) (C & D)
   5   7
(K & M & N) (C & D)
   6
(M & N) (C & D) because K is terminal
(N) (C & D) because M is terminal
() (C & D) because N is terminal
Algorithm terminates when the problem list under consideration becomes empty
(i.e., every subproblem that needed to be solved is solved).

We state the following results about AO*:

- AO* is admissible given an admissible heuristic function.
- AO* is an optimal search algorithm among all admissible search algorithms that use an additive evaluation function.

- Therefore, AO* is an optimal admissible search algorithm for AND-OR graphs.

The proof of these results is left as an exercise for the reader.