The A* Algorithm

We walk through an example computation of the A^* algorithm for solving the single pair minpath problem on a weighted directed graph. The pair is (S,T). The *weight* of an arc (x,y) is written $w(x,y)$.

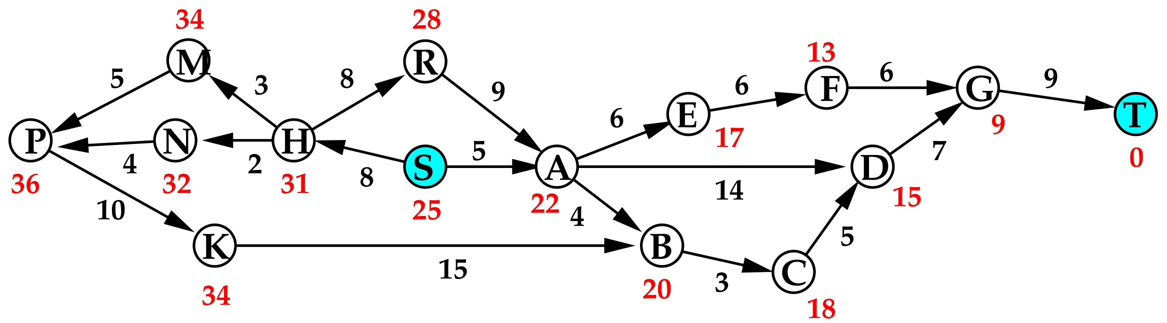

Figure 1: Example single pair minpath problem.

To work A^*, we must define a heuristic $h(x)$ for each vertex x. $h(x)$ must be less than or equal to the minimum distance from x to T, and must also be *consistent*, that is, $h(x) \le w(x,y) + h(y)$ for any arc (x,y). The closer $h(x)$ is to the true distance from x to T, the faster the A^* algorithm will converge.

In Figure 1, a consistent heuristic is given, shown by red numerals.

Steps of A^*

Just as for Dijkstra's algorithm, we maintain three sets of vertices: processed, partially processed, and unprocessed. Initially there is no processed vertex and only S is partially processed. If x is processed or partialy processed, $g(x)$ is the shortest distance discovered so far from S to x. Value of g are indicted by blue numerals in our figures. After each $g(x)$ is computed, we let $f(x) = g(x) + h(x)$. Values of f are indicated by green numerals.

At each step, the vertex V with the minimum value of f is selected, and becomes fully processed. In the figures, fully processed vertices are indicated by heavy circles. All outneighbors of V are updated, becoming partially processed. An outneighbor which was already partially processed could possibly acquire a new, smaller, value of g, hence a new value of f, and a new backpointer.

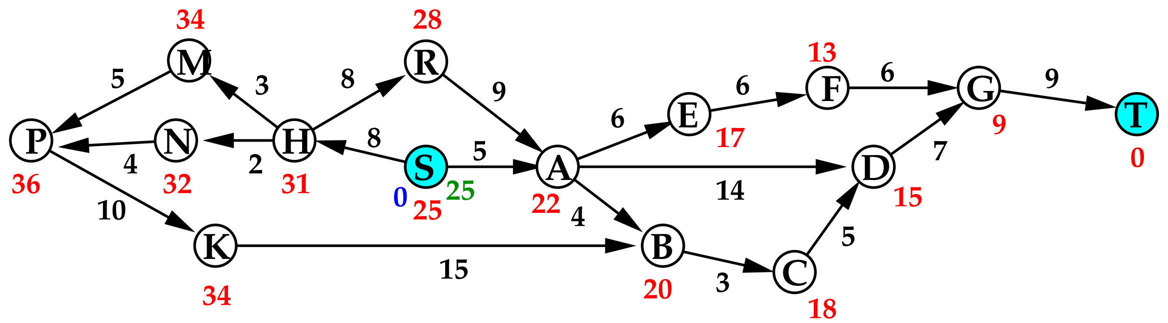

Figure 2

In Figure 2, S is the only partially processed vertex. $h(S)$ is given to be 25. $g(S) = 0$, hence $f(x) = 25$.

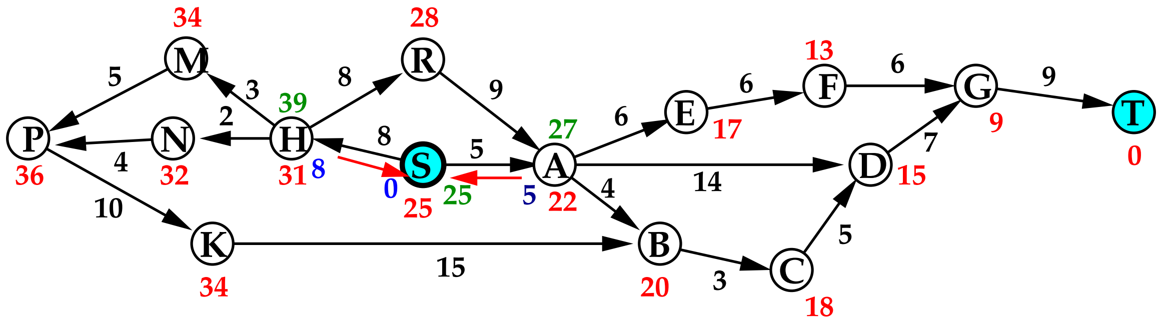

Figure 3

In Figure 3, S becomes processed, as indicated by the darker circle. Its outneighbors A and H become partially processed. Backpointers are indicated as red arrows.

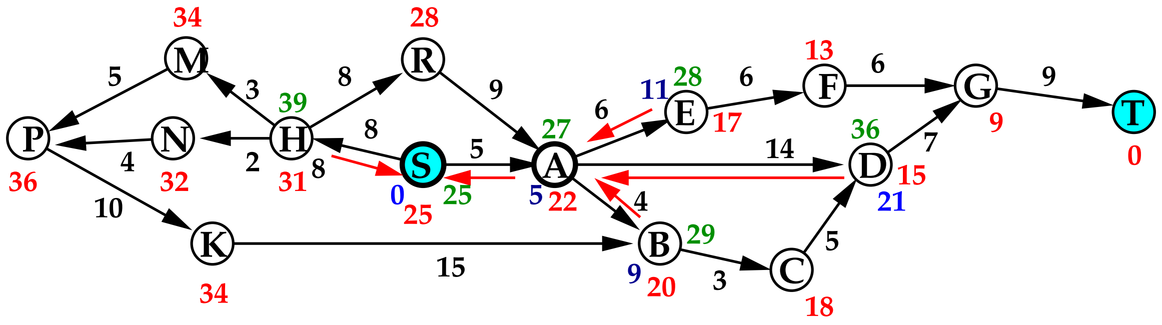

Figure 4

In Figure 4 A becomes fully processed, while B, D, and E become partially processed.

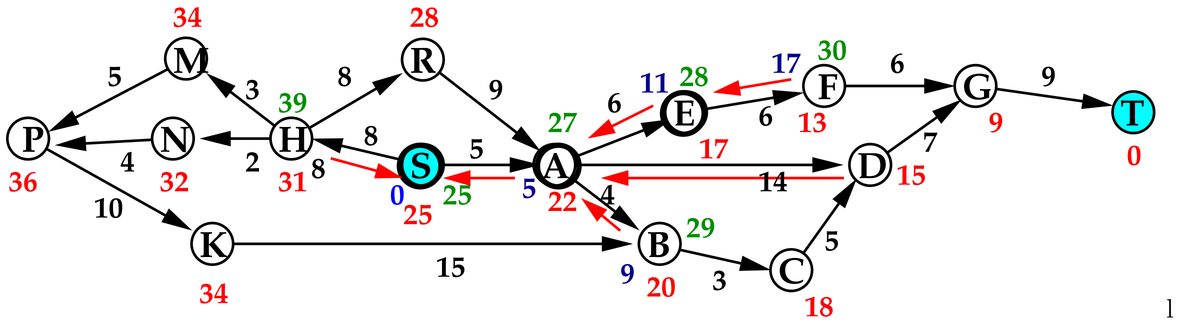

Figure 5

E becomes fully processed, while F becomes partially processed.

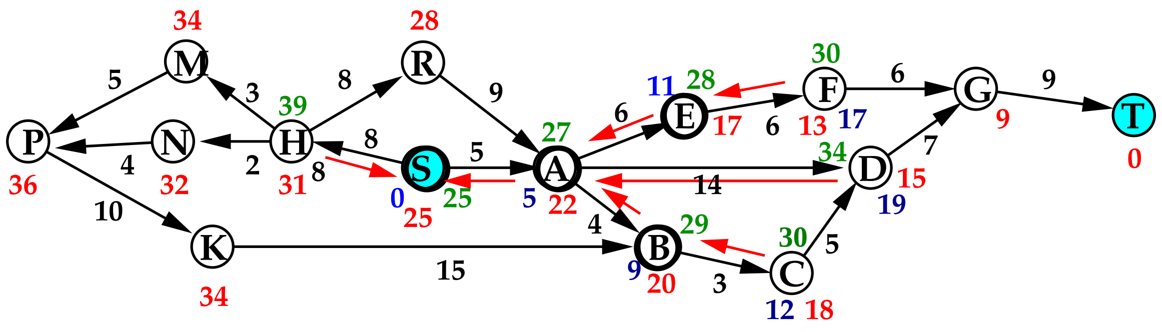
Figure 6

B becomes fully processed, while *C* becomes partially processed.

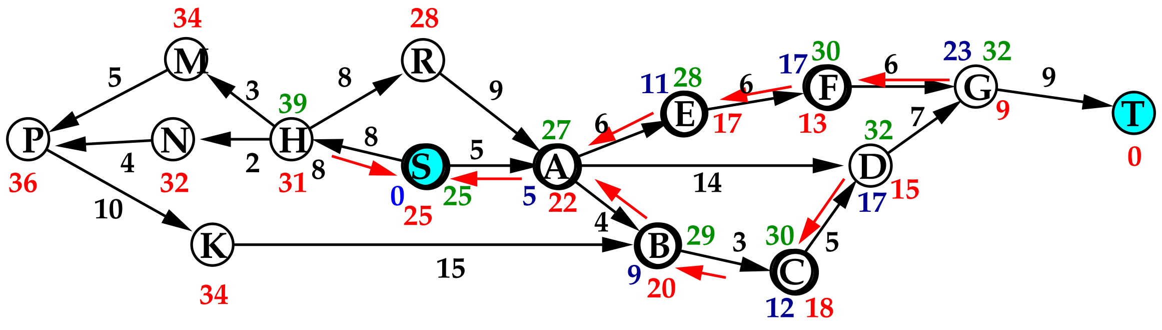
Figure 7

Now, *C* and *F* are fully processed. *D* acquires a new, smaller value of *g*, and its backpointer changes to *C*.

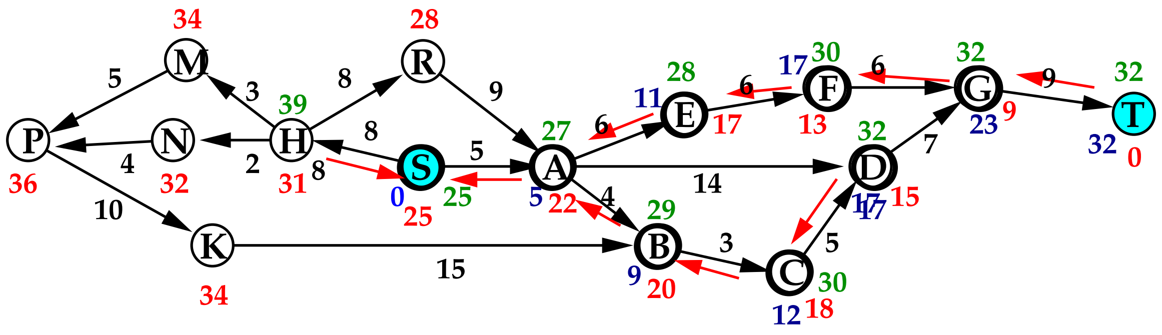
Figure 8

D and *G* become fully processed, while *T* becomes partially processed.

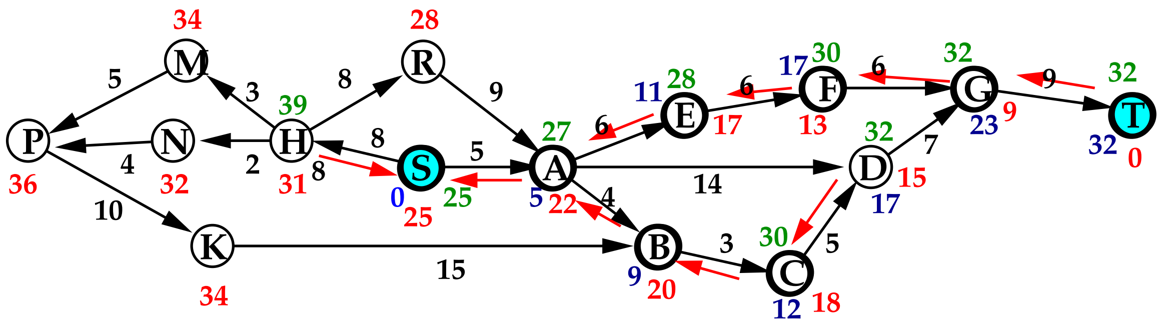

Figure 9

It seems unnecessary, but the algorithm only stops when T becomes fully processed. Although not in this example, it is possible that T would acquire a new backpointer after being partially processed for the first time.

Wikipedia Page

There is a Wikipedia article at `https://en.wikipedia.org/wiki/A*_search_algorithm` titled "A* search algorithm," which covers the algorithm I've presented to you, as well as variations. In order to bring the notation of this document into line with the notation in the Wikipedia page, I have exchanged f and g, so that $g(x)$ is the length of shortest known path from x to T, and $f(x) = g(x) + h(x)$. That document also uses "open" to describe partially processed vertices, and the backpointer of a vertex x points to the "predecessor" of x. There is other notation you should be able to figure out. The figures in this document have not been altered: a blue numeral now represents $g(x)$, and a green numeral $f(x)$.