The Mathematical Foundation of A*

Olav Geil
Department of Mathematical Sciences
Aalborg University

This note was written for use in a mathematical course for students in computer game programming at Department of Computer Science at Aalborg University. The literature on A* can be rather confusing and it is the aim of the note to present the complete mathematical foundation for A*. Having established the foundation of A* it becomes clear which errors in the literature one should omit to implement.

1 The basic concepts

Consider a finite weighted graph $G = (V, E)$, meaning that $\#V < \infty$ and that we are given a map $w : E \rightarrow \mathbb{R}_+$. Let $v_a, v_z \in V$. We want to establish a most inexpensive path from v_a to v_z if such one exists. When the function w has some physical meaning a good candidate for an algorithm to solve this problem is the A* algorithm. As we will see this algorithm can be viewed as a generalization of Dijkstra's algorithm. The A* algorithm is an example of an informed search. More precisely we have as input to the algorithm for every $v \in V$ a value $h(v) \in \mathbb{R}_+ \cup \{0\}$ estimating in advance the weight of the most inexpensive path from v to v_z. The function $h : V \rightarrow \mathbb{R}_+ \cup \{0\}$ is called a heuristic function. We are particular interested in admissible heuristics and monotone heuristics.

Definition 1 *A heuristic function $h : V \rightarrow \mathbb{R}_+ \cup \{0\}$ is called admissible with respect to v_z if for every $v \in V$, $h(v)$ is not larger than the weight of a most inexpensive path from v to v_z.*

Observe, that in particular $h(v_z)$ needs to be equal to 0 if h is admissible with respect to v_z.

Definition 2 *A heuristic function $h : V \rightarrow \mathbb{R}_+ \cup \{0\}$ is called monotone if for every pair of neighbors v_i, v_j in G we have*

$$|| \, h(v_i) - h(v_j) \, || \leq w(v_i, v_j)$$

The following proposition shows that being monotone is basically a stronger requirement than being admissible.

Proposition 1 *A monotone heuristic function h satisfying $h(v_z) = 0$ is admissible with respect to v_z.*

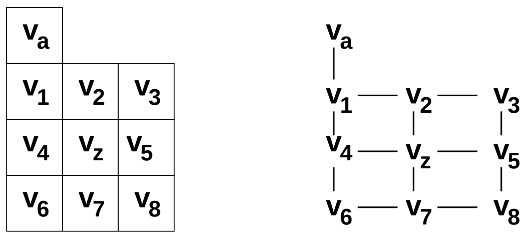

Figure 1:

Proof: Consider any vertex $v \in V$. If there is no path from v to v_z then certainly the condition in Definition 1 is satisfied. Assume a most inexpensive path is given by $P : v = v_1, v_2, \ldots, v_s = v_z$. We have

$$\begin{aligned} w(P) &= w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_{s-1}, v_s) \\ &\geq (h(v_1) - h(v_2)) + (h(v_2) - h(v_3)) + \cdots + (h(v_{s-1}) - h(v_s)) \\ &= h(v_1) - h(v_z) = h(v_1) \end{aligned}$$

and again the condition in Definition 1 is satisfied. □

Example 1 *Given n towns identify each of them with a vertex, say $V = \{v_1, \ldots, v_n\}$. Let there be an edge between v_i and v_j, $i \neq j$ if there is a road from v_i to v_j that does not pass any other town. For every edge (v_i, v_j) let $w(v_i, v_j)$ be the length of the most inexpensive route from v_i to v_j not passing any other town. An obvious heuristic is to choose $h(v)$ to be simply the bee line distance between v and v_z. In this way one obviously derive an admissible heuristic. We now show that the heuristic is actually also monotone. Let v_i, v_j be neighbors. The usual triangle equality tells us that the bee line distance from v_i to v_z is less than or equal to the sum of the bee line distances from v_j to v_z plus the be line distance from v_i to v_j. The latter one being less than or equal to $w(v_i, v_j)$. Hence, we get the equality*

$$\begin{aligned} & h(v_i) \leq h(v_j) + w(v_i, v_j) \\ \Downarrow & \\ & w(v_i, v_j) \geq h(v_i) - h(v_j). \end{aligned}$$

By symmetry this tells us

$$w(v_i, v_j) \geq || \, h(v_i) - h(v_j) \, ||$$

and we are through,

Example 2 *To use the A^* algorithm we must have a graphical model of our world. Assume the world under consideration is a bounded area in 2D. We can place a square grid on the world consisting of squares. Hence, we identify every square with a vertex and we have an edge between two vertices if the squares they present share a side.*

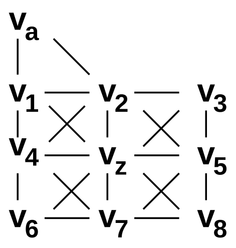

Figure 2:

If we imagine that the vertices are placed in the center of the squares we can calculate the distance between any two adjacent vertices. Identify now coordinates for vertices as follows. The vertex in the lower left corner is $(0, 0)$ *and the vertex* v *has coordinates* (v_x, v_y) *if it is in position of* v_x *squares to the right and* v_y *squares up relative to the lower left corner. Hence, in Figure 1 the coordinates of* v_z *are* $(1, 1)$*. Let* D *be the (real world) minimal price for moving between to adjacent vertices in the graph. The Manhattan heuristic is given as*

$$h(v) = D(|| \; v_x - (v_z)_x \; || + || \; v_y - (v_z)_y \; ||).$$

Clearly, the Manhattan heuristic is monotone with $h(v_z) = 0$ *and thereby also admissible. Note, that it is important to include the scaling factor* D*.*

Assume next that we make a little different model of our world by having an edge between any two vertices for which the squares touch each other. In our example we produce in this way Figure 2. If we try to use the Manhattan heuristic in this model (which is sometimes faulty done) we might very well get into serious troubles. Assume as an example that the physical distance between two adjacent vertices are given in the following natural way. Two adjacent vertices in horizontal or vertical position to each other are assumed to be in distance 1*. Two adjacent vertices in diagonal position to each other are assumed to be in distance* $\sqrt{2}$ *(coming from the Pythagorean equality). The minimal price for moving between adjacent vertices is now* $D = 1$ *but we may for instance move from* $(1, 1)$ *to* $(2, 2)$ *in only one step instead of two. The price will only be* $\sqrt{2}$ *instead of* $1 + 1 = 2$*. Analyzing the situation we see that an adapted Manhattan heuristic would be*

$$h(v) = \frac{\sqrt{2}}{2}(|| \; v_x - (v_z)_x \; || + || \; v_y - (v_z)_y \; ||)$$

or

$$h(v) = 0.7(|| \; v_x - (v_z)_x \; || + || \; v_y - (v_z)_y \; ||).$$

These adapted Manhattan heuristics are both monotone and admissible.

In situations where we are allowed to make diagonal moves we could also choose to use instead of the Manhattan heuristic the diagonal heuristic defined as follows

$$h(v) = D \max\{|| \; v_x - (v_z)_x \; ||, || \; v_y - (v_z)_y \; ||\}.$$

This heuristic is of course again monotone and admissible.

If the minimal price for a horizontal move or a vertical move is D_1 and the minimal price for making a diagonal move is $D_2 = \sqrt{2}D_1$ we could make a more refined heuristic as follows. To move from v to v_z by only horizontal or vertical moves requires at least $\| v_x - (v_z)_x \| + \| v_y - (v_z)_y \|$ moves. However, by allowing diagonal moves one, horizontal move plus one vertical move might be replaced by a single diagonal move. In this way we get the heuristic

$$\begin{aligned} h(v) &= D_2(\min\{\| v_x - (v_z)_x \|, \| v_y - (v_z)_y \|\}) \\ &\quad + D_1 (\| v_x - (v_z)_x \| + \| v_y - (v_z)_y \| - 2\min\{\| v_x - (v_z)_x \|, \| v_y - (v_z)_y \|\}) \\ &= D_1 \left(\| v_x - (v_z)_x \| + \| v_y - (v_z)_y \| - (2-\sqrt{2})\min\{\| v_x - (v_z)_x \|, \| v_y - (v_z)_y \|\}\right). \end{aligned}$$

This heuristic of course is again monotone and admissible. The heuristics in the present example of course can be generalized to work in 3D world.

2 The algorithm

In this note we will present two versions of A*. Namely, Algorithm 1 and Algorithm 2 below. Algorithm 1 is the most general one as it only requires the heuristic to be admissible to always find a shortest path. If in addition the heuristic is monotone we can skip some parts of Algorithm 1 giving us Algorithm 2. The reader should keep in mind that the algorithms are just generalizations of Dijkstra's algorithm. More precisely, using the heuristic function $h(v) = 0$ for all $v \in V$ will make Algorithm 1 as well as Algorithm 2 into nothing but Dijkstra's algorithm. Recall, that in Dijkstra's algorithm we build up a search tree routed in v_a. In every loop we update information about the cost of coming from v_a to vertices that has not yet been included in the tree. At the end of the loop we include in the tree the most inexpensive one. We keep on building up the tree until v_z is included. The shortest path from v_a to v_z found by the algorithm will then be the unique path in the tree from v_a to v_z.

A* does something very much similar. However, instead of only calculating the cost $g(v)$ for moving from v_a to v we add the heuristic $h(v)$ to get $f(v) = g(v) + h(v)$. The point added to the search tree are the one with lowest f-value. Clearly, doing this corresponds to choosing the point which looks to be on a most inexpensive path from v_a to v_z rather than just being the one closest to v_a. Except for $f = g + h$ being used rather than only g, Algorithm 2 is very close to Dijkstra's algorithm. Algorithm 1 works for relatively general heuristics and here things can be a little complicated. It might happen that a point v which is already included in the search tree need to be updated, meaning that it should be connected somehow different to the other vertices in the tree. This happens when a new and more inexpensive path is found from v_a to v. What we do then is to remove it from the search tree making the search tree possible into a forest. To handle this the A* algorithm works with a number of lists. The list Open from the beginning contains only v_a. At later times in the run it contains every vertex that has not yet been included in the tree (possible forest) but is either connected to the tree (possible forest) or is adjacent to a vertex that has at some earlier point been in the tree (forest). The set Closed are the points in the search tree (forest).

Algorithm 1:
Input: A weighted graph $G = (V, E)$ and vertices $v_a, v_z \in V$. A heuristic function $h : E \to \mathbb{R}_+ \cup \{0\}$.

Output: A path from v_a to v_z or "failure".

```
Step 1 (initialization):

$g(v_a) = 0$, $f(v_a) = g(v_a) + h(v_a)$, $Open = \{v_a\}$, $Closed = \emptyset$.

Step 2:

If $Open = \emptyset$ then

begin

    Return("failure") and Quit

end

If Open is nonempty then

begin

    Let $B$ be a vertex in $Open$ with smallest $f$-value.

    If $B = v_z$ then

    begin

       Let $P$ be the path

     $P : v_z, Parent(v_z), Parent(Parent(v_z)), \ldots, Parent(\cdots(Parent(v_z))) = v_a$

       Return($P$ reversed) and Quit

    end

end
```

Step 3:

```
Let Successor be the list with all vertices adjacent to $B$.
For all $C$ in Successor:
begin
    If $C \in Closed$ or $C \in Open$
    begin
        Let $g'(C) = g(B) + w(B,C)$
        If $g'(C) < g(C)$
        begin
            let $g(C) = g'(C)$, $f(C) = g(C) + h(C)$ and $Parent(C) = B$
            If $C \in Closed$
            begin
                $Closed = Closed \setminus \{C\}$
                $Open = Open \cup \{C\}$
            end
        end
    end
    If neither $C \in Closed$ nor $C \in Open$
    begin
        Let $g(C) = g(B) + w(B,C)$, $f(C) = g(C) + h(C)$ and $Parent(C) = B$.
        $Open = Open \cup \{C\}$
    end
end
```

Step 4:

$Open = Open \setminus \{B\}$

$Closed = Closed \cup \{B\}$

Go to step 2

Algorithm 2:
Input: A weighted graph $G = (V, E)$ and vertices $v_a, v_z \in V$. A heuristic function $h : E \to \mathbb{R}_+ \cup \{0\}$.

Output: A path from v_a to v_z or "failure".

Step 1 (initialization):

$g(v_a) = 0$, $f(v_a) = g(v_a) + h(v_a)$, $Open = \{v_a\}$, $Closed = \emptyset$.

Step 2:

```
If $Open = \emptyset$ then
begin
    Return("failure") and Quit
end

If Open is nonempty then
begin
    Let $B$ be a vertex in $Open$ with smallest $f$-value.
    If $B = v_z$ then
    begin
       Let $P$ be the path
     $P : v_z, Parent(v_z), Parent(Parent(v_z)), \ldots, Parent(\cdots(Parent(v_z))) = v_a$
       Return($P$ reversed) and Quit
    end
end
```

Step 3:

```
Let Successor be the list with all vertices adjacent to $B$.
For all $C$ in Successor:
begin
    If $C \in Open$
    begin
        Let $g'(C) = g(B) + w(B, C)$
        If $g'(C) < g(C)$
        begin
            let $g(C) = g'(C)$, $f(C) = g(C) + h(C)$ and $Parent(C) = B$
        end
    end
    If neither $C \in Closed$ nor $C \in Open$
    begin
        Let $g(C) = g(B) + w(B, C)$, $f(C) = g(C) + h(C)$ and $Parent(C) = B$.
        $Open = Open \cup \{C\}$
    end
end
```

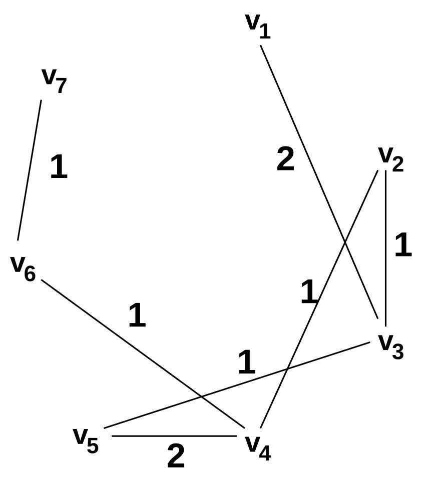

Figure 3:

Step 4:

$Open = Open \backslash \{B\}$

$Closed = Closed \cup \{B\}$

Go to step 2

In the remaining part of this note we will often refer to the vertex B as the active vertex.

3 Examples

To illustrate the algorithms we treat two small examples. To really experience the strength of A* one should investigate larger examples and heuristics with some physical meaning.

Example 3 *Consider the weighted graph in Figure 3 The heuristic* $h(v_1) = 4$,$h(v_2) = 3$,$h(v_3) = 2$,$h(v_4) = 2$,$h(v_5) = 1$,$h(v_6) = 1$,$h(v_7) = 0$ *is monotone with* $h(v_7) = 0$. *Hence, we can use Algorithm 2 to find a shortest path from* $v_a = v_3$ *to* $v_z = v_7$. *We get:*

$Open = \{v_3\}, Closed = \emptyset, B = v_3$

$Successors = \{v_1, v_2, v_5\}$

$v_1 \notin Closed, Open$

$g(v_1) = 2, h(v_1) = 4, f(v_1) = 6, Parent(v_1) = v_3$

$v_2 \notin Closed, Open$

$g(v_2) = 1, h(v_2) = 3, f(v_2) = 4, Parent(v_2) = v_3$

$v_5 \notin Closed, Open$

$g(v_5) = 1, h(v_5) = 1, f(v_5) = 2, Parent(v_5) = v_3$

$Closed = \{v_3\}, Open = \{v_1, v_2, v_5\}$

$B = v_5$

$Successors = \{v_3, v_4\}$

$v_3 \in Closed$

$v_4 \notin Closed, Open$

$g(v_4) = 3, h(v_4) = 2, f(v_4) = 5, Parent(v_4) = v_5$

$Closed = \{v_3, v_5\}, Open = \{v_1, v_2, v_4\}$

$B = v_2$

$Successors = \{v_3, v_4\}$

$v_3 \in Closed$

$v_4 \notin Closed, v_4 \in Open$

$g'(v_4) = 2 < g(v_4)$

$g(v_4) = 2, h(v_4) = 2, f(v_4) = 4, Parent(v_4) = v_2$

$Closed = \{v_2, v_3, v_5\}, Open = \{v_1, v_4\}$

$B = v_4$

$Successors = \{v_2, v_5, v_6\}$

$v_2 \in Closed$

$v_5 \in Closed$

$g(v_6) = 3, h(v_6) = 1, f(v_6) = 4, Parent(v_6) = v_4$

$Closed = \{v_2, v_3.v_4, v_5\}Open = \{v_1, v_6\}$

$B = v_6$

$Successors = \{v_4, v_7\}$

$v_4 \in Closed$

$g(v_7) = 4, h(v_7) = 0, f(v_7) = 4, Parent(v_7) = v_6$

$Closed = \{v_2, v_3, v_4, v_5, v_6\}Open = \{v_1, v_7\}$

$B = v_7$

$Return(P : v_3, v_2, v_4, v_6, V_7)$

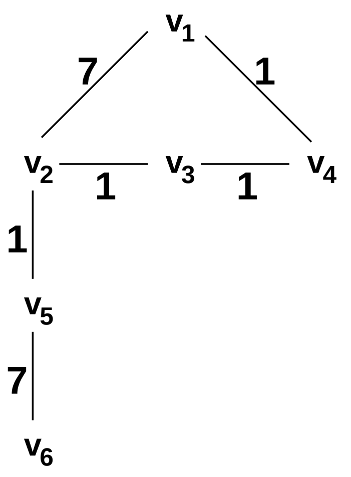

Figure 4:

Example 4 *Consider the weighted graph in Figure 4 The heuristic* $h(v_1) = 11$,$h(v_2) = 2$,$h(v_3) = 2$,$h(v_4) = 10$,$h(v_5) = 7$,$h(v_6) = 0$ *is not monotone. However, it is admissible with respect to* v_6*. Hence, we can use Algorithm 1 to find a shortest path from* $v_a = v_1$ *to* $v_z = v_6$*. We get:*

$$
\begin{aligned}
&Open = \{v_1\}, Closed = \emptyset, B = v_1\\
&Successors = \{v_2, v_4\}\\
&v_2 \notin Closed, Open\\
&g(v_2) = 7, h(v_2) = 2, f(v_2) = 9, Parent(v_2) = v_1\\
&v_4 \notin Closed, Open\\
&g(v_4) = 1, h(v_4) = 10, f(v_4) = 11, Parent(v_4) = v_1\\
&Closed = \{v_1\}, Open = \{v_2, v_4\}\\
&B = \{v_2\}\\
&Successors = \{v_1, v_3, v_5\}\\
&v_1 = Closed\\
&g'(v_1) = 14 \geq g(v_1)\\
&v_3 \notin Closed, Open\\
&g(v_3) = 8, h(v_3) = 2, f(v_3) = 10, Parent(v_3) = v_2\\
&v_5 \notin Closed, Open\\
&g(v_5) = 8, h(v_5) = 7, f(v_5) = 15, Parent(v_5) = v_2\\
&Closed = \{v_1, v_2\}, Open = \{v_3, v_4, v_5\}
\end{aligned}
$$

$B = v_3$

$Successors = \{v_2, v_4\}$

$v_2 \in Closed$

$g'(v_2) = 9 \geq g(v_2)$

$v_4 \in Open$

$g'(v_4) = 9 \geq g(v_4)$

$Closed = \{v_1, v_2, v_3\}, Open = \{v_4, v_5\}$

$B = v_4$

$Successors = \{v_1, v_3\}$

$v_1 \in Closed$

$g'(v_1) = 2 \geq g(v_1)$

$v_3 \in Closed$

$g'(v_3) = 2 < g(v_3)$

$g(v_3) = 2, h(v_3) = 2, f(v_3) = 4, Parent(v_3) = v_4$

$Closed = \{v_1, v_2\}, Open = \{v_3, v_4, v_5\}$

$Closed = \{v_1, v_2, v_4\}, Open = \{v_3, v_5\}$

$B = v_3$

$Successors = \{v_2, v_4\}$

$v_2 \in Closed$

$g'(v_2) = 3 < g(v_2)$

$g(v_2) = 3, h(v_2) = 2, f(v_2) = 5, Parent(v_2) = v_3$

$Closed = \{v_1, v_4\}, Open = \{v_2, v_3, v_5\}$

$v_4 \in Closed$

$g'(v_4) = 3 \geq g(v_4)$

$Closed = \{v_1, v_3, v_4\}, Open = \{v_2, v_5\}$

$B = v_2$

$Successors = \{v_1, v_3, v_5\}$

$v_1 \in Closed$

$g'(v_1) = 10 \geq g(v_1)$

$v_3 \in Closed$

$g'(v_3) = 4 \geq g(v_3)$

$v_5 \in Open$

$g'(v_5) = 4 < g(v_5)$

$g(v_5) = 4, h(v_5) = 7, f(v_5) = 11, Parent(v_5) = \{v_2\}$

$Closed = \{v_1, v_2, v_3, v_4\}, Open = \{v_5\}$

$B = v_5$
$Successors = \{v_2, v_6\}$
$v_2 \in Closed$
$g'(v_2) = 5 \geq g(v_2)$
$v_6 \notin Closed, Open$
$g(v_6) = 11, h(v_6) = 0, f(v_6) = 11, Parent(v_6) = v_5$
$Closed = \{v_1, v_2, v_3, v_4, v_5\}, Open = \{v_6\}$
$B = v_6$
$Return(P : v_1, v_4, v_3, v_2, v_5, v_6)$

The path found by the algorithm is of length 11 *and is surely the most inexpensive path from* v_1 *to* v_6. *It is important that when updating the g-value of a vertex in Closed we remember to move the vertex to Open. Failing to do this would produce the path* $P' : v_1, v_2, v_3, v_6$ *of length 15.*

4 The mathematical foundation

In this section we show first that Algorithm 1 is guaranteed to find a most inexpensive path (provided there is a path) whenever the heuristic is admissible. This result is then used to show that Algorithm 2 finds a most inexpensive path (provided there is a path) whenever the heuristic is monotone with $h(v_z) = 0$.

Theorem 1 *Consider a weighted graph* $G = (V, E)$ *and vertices* $v_a, v_z \in V$ *Let the heuristic* $h : V \to \mathbb{R}_+ \cup \{0\}$ *be admissible with respect to* v_z. *If a path from* v_a *to* v_z *exist then Algorithm 1 finds a most inexpensive such one. If no path exist the algorithm returns "failure".*

Proof: The algorithm builds up a tree (or forest) containing v_a. It can stop of one of two reasons. Either it stops because Open is empty. In this case we have in Closed all vertices connected to v_a by some path. The vertex v_z cannot be in Closed as this would have caused the algorithm to stop earlier. Therefore there is no path from v_a to v_z and the algorithm answers correctly by returning "failure". The other possibility for the algorithm to stop is that $B = v_z$. In this case the algorithm has established a path from v_a to v_z. We have to show that there cannot be a less expensive path from v_a to v_z then the one found by the algorithm.

Before doing this we note that the algorithm will eventually stop. This is seen as follows. let v be a vertex that is in Open at some point of the the run. The value $g(v)$ corresponds to the length of some path from v_a to v. Every time v is moved from Open to Closed and back to Open again it is because a less expensive path has been detected from v_a to v. However, there are only finitely many paths from v_a to v and therefore v can move forth and back between Closed and Open only finitely many times. In every loop some vertex moves from Open to Closed and therefore the algorithm will either stop because $B = v_z$ or it will stop because Open is empty.

It remains to be shown that it the algorithm finds a path P from v_a to v_z and if

$$Q : v_a = v_1, v_2, \ldots, v_{t-1}, v_t = v_z$$

is a most inexpensive path from v_a to v_z then the cost of Q is not less than the cost of P. We will assume that actually the cost of Q is less than the cost of P and arrive at a contradiction. Consider the step in the algorithm just before B is chosen to be v_z (all g and f values in the following will be with respect to this step of the algorithm). Let at this point $v_1, v_2, \ldots, v_s \in \mathit{Closed}$ but $v_{s+1} \notin \mathit{Closed}$. We note that according to our assumptions that the path Q was not established such an s exists. We also observe that v_{s+1} must be in Open.

By assumption the edge (v_a, v_2) is most inexpensive path from v_a to v_2. Therefore $g(v_2)$ equals the length of the shortest path from v_a to v_2 already after the very first loop of the algorithm. At some later loop v_2 must have been active (as v_2 is in Closed). At this point $g(v_3)$ must have been assigned a value which equals the length of a most inexpensive path from v_a to v_3. Continuing this way we see that $g(v_u)$ equals the length of a most inexpensive path from v_a to v_u for $u = 1, 2, \ldots, s+1$.

If now we denote by $\text{dist}_w(x, y)$ the weight of a most inexpensive path from x to y we therefore get

$$\begin{aligned} f(v_{s+1}) &= g(v_{s+1}) + h(v_{s+1}) \\ &= \text{dist}_w(v_a, v_{s+1}) + h(v_{s+1}) \\ &\leq \text{dist}_w(v_a, v_{s+1}) + \text{dist}_w(v_{s+1}, v_z) \\ &= w(Q) \qquad (1) \end{aligned}$$

Here $w(Q)$ denotes the weight of the path Q. In the third equality we used the fact that the heuristic is admissible. In the last equality we used the fact that Q is a least expensive path from v_a to v_z.

Having estimated $f(v_{s+1})$ we next consider $f(v_z)$. As the algorithm is just about to choose v_z as active vertex and thereby establish the path P we have

$$f(v_z) = g(v_z) + h(v_z) = g(v_z) = w(P). \qquad (2)$$

Comparing (1) and (2) we see that it cannot be true that v_z is chosen as active vertex before v_{s+1} is. We have arrived at a contradiction. □

Theorem 2 *Consider a weighted graph $G = (V, E)$ and vertices $v_a, v_z \in V$. Let the heuristic $h : V \to \mathbb{R}_+ \cup \{0\}$ be monotone with $h(v_z) = 0$. If a path from*

v_a to v_z exists then Algorithm 2 finds a most inexpensive such one. If no path exist the algorithm returns "failure".

Proof: By Proposition 1 the heuristic is admissible. According to Theorem 1 we therefore could apply Algorithm 1 on the problem and thereby find a most inexpensive path if such one exist. We will show that if Algorithm 1 is used, then once a vertex has been moved to Closed its g-value will never again be updated. In other words it will never be moved back to Open. But then Algorithm 1 becomes identical to Algorithm 2 and the theorem follows.

We will (faulty) assume that a vertex v_i exists that is moved from Closed to Open when Algorithm 1 is applied and arrive at a contradiction. Without loss of generality we may assume that v_i is the very first vertex to be moved from Closed to Open. We consider the algorithm just before this is done. Denote $B = v_t$. By assumption we have

$$g(v_i) > g(v_t) + w(v_t, v_i). \tag{3}$$

To establish a contradiction to (3) we will need to prove that

$$g(v_t) + h(v_t) \geq g(v_i) + h(v_i). \tag{4}$$

This is where we will need the assumption that no vertex has been moved from Closed at the present step of the algorithm. Assume the path found from v_a to v_t at the present step of the algorithm is

$$R : v_a = v_{j_0}, v_{j_1}, \ldots, v_{j_k}, v_t = v_{j_{k+1}}.$$

By monotonicity and the assumption that no vertex have been moved from Closed to Open we must have

$$\begin{aligned} g(v_a) + h(v_a) &\leq g(v_{j_1}) + h(v_{j_1}) \leq \cdots \\ &\leq g(v_{j_k}) + h(v_{j_k}) \leq g(v_t) + h(v_t). \end{aligned}$$

If v_i is in R, then (4) is established. If not let v_{j_l} be the first vertex in R added to Closed after v_i was added to Closed. But then $v_{j_{l-1}}$ was in Closed before v_i was added to Closed and therefore according to the algorithm

$$g(v_i) + h(v_i) \leq g(v_{j_l}) + h(v_{j_l})$$

must hold. But

$$g(v_{j_l}) + h(v_{j_l}) \leq g(v_t) + h(v_t)$$

and (4) follows.

Using (4) and the fact that the heuristic function is monotone we get

$$\begin{aligned} g(v_i + h(v_i) &\leq g(v_t) + h(v_t) \\ &\leq g(v_t) + h(v_t) + w(v_t, v_i) - (h(v_t) - h(v_i)) \\ &= g(v_t) + w(v_t, v_i) + h(v_i) \\ &= g'(v_i) + h(v_i). \end{aligned}$$

Therefore, $g(v_i) \leq g'(v_i)$ in contradiction with (3).

We have shown that no vertex v_i is ever moved from Closed to Open and therefore it suffices to use Algorithm 2 instead of Algorithm 1. □

5 Concluding remarks

The task of finding shortest paths by use of artificial intelligence algorithms is the object of active research. A lot of interesting papers have been written including papers on robotics, chemistry and of course game programming. There are many resources on A* in particular a lot of homepages. Be careful not to believe everything that is written. It is quite common to faulty use Algorithm 2 for non-monotonic admissible heuristics. As we have seen this may cause the algorithm to not find a shortest path. In particular it is not uncommon to use Algorithm 2 with non-monotonic Manhattan heuristics. In some descriptions of Algorithm 1 it is forgotten to move the vertex C from Closed to Open when its g value is updated (as we mentioned in Example 4 this may cause the algorithm to find a wrong path). Some authors insist that it is checked that

$$|| \, h(v_i) - h(v_j) \, || \leq \mathrm{dist}_w(v_i, v_j) \tag{5}$$

holds for all pairs of vertices $v - i, v_j \in V$ for h to be called monotone. From the proof of Proposition 1 it can be seen that whenever (5) holds for all neighbors it automatically holds for all pairs of vertices. Hence, there is not reason for checking more than the neighbors.

Among the very many interesting results that where not covered in this introductionary note we like to mention a few. It may happen when applying A* that more vertices C in Open are assigned the same f-value. When this is also the smallest f-value for vertices in Open the algorithm has more choices of B. To deal with his some versions of A* have implemented some intelligent ways of breaking ties. The D* algorithm is a generalization of A*. it works in unknown, partially unknown or even changing environments. The Adaptive A* algorithm and the LPA* algorithm are very efficient when the landscape is fixed but one needs to find shortest paths from v_a to v_z for a series of different pairs (v_a, v_z).

The A* algorithm was invented by P. Hart, N. Nilsson and B. Raphael in 1968. It is said to be discovered by the commercial computer game industry in 1999. We conclude the note by mentioning that the results described here easily generalizes to directed graphs.