Artificial Intelligence – HW1 Report

Submitters:

Yuval Nahon – 206866832

Sahar Cohen – 206824088

Part A:

1) The results are as follows:

	number of permutations(no gas consideration)	number of permutations(no gas consideration)
1	1	1
2	2	10
3	6	150
4	24	3000
5	120	75000
6	720	2250000
7	5040	78750000
8	40320	3150000000
9	362880	1.4175E+11
10	3628800	7.0875E+12

2) The maximum branching factor is acquired when the delivery boy is at the starting node, and all the other nodes are reachable (orders + gas stations). There are **k** orders and **l** gas stations, each is connected to the starting node with an edge, so we get: **k+l.**

The minimum branching factor is acquired when there's not enough fuel to reach any node at all, orders or gas stations, so we get a branching factor of **0**.

3) We can get a cycle in our state space: assume we have two gas stations **A** and **B** where the only state reachable from A is B, and vice versa. So, we'll travel from A to B and back to A indefinitely, with **d, T** and **F** staying the same as we re-fuel each time, and don't deliver any delivery.
(This is an extreme case. We will get a cycle for every two of gas stations $f1, f2\ such\ that\ dist(f1, f2) \leq d_{refuel}$ in the same way).

4) The state space has infinite states since the amount of fuel **d** is a defining part of each and can be any non-zero real number. Not all states are reachable. Here are two examples:

- Some state is located too far from any other state such that it is impossible to reach it with our fuel capacity (this is case-specific).

- We can't ever reach some state (v, π, T, F) (where $\pi = 3.1415\ldots$) because $\pi \notin Q$ and the travel distance of the delivery boy is finite (this is applicable no matter the map).

5) Yes. For instance, $(v_0, d_0, [k], \phi) \rightarrow (v, 0, [k-1], \{k\})$. If we take this path: we can't travel to any other state because we're out of fuel.

6)

$$Succ\big((v_1, d_1, T_1, F_1)\big) =$$

$$\{\ (v_2, D_2, T_2, F_2)\ |$$

$$Dist(v_1, v_2) \leq d_1 \wedge$$

$$v_2 \in T1 \wedge$$

$$d_2 = d_1 - Dist(v_1, v_2) \wedge$$

$$T_2 = T_1 \backslash \{v_2\} \wedge$$

$$F_2 = F_1 \cup \{V_2\}\ \}$$

$$\cup$$

$$\{\ (v_2, D_2, T_2, F_2)\ |$$

$$Dist(v_1, v_2) \leq d_1 \wedge$$

$$v_2 \in GasStations \wedge$$

$$d_2 = d_{refuel} \wedge$$

$$T_2 = T_1 \wedge$$

$$F_2 = F_1\ \}$$

7) In the best-case scenario: the delivery boy doesn't visit a single gas station (has enough fuel for his entire trip), but still he must make all the deliveries (that is: visit k nodes). So, the lower bound for the minimal depth is **k**.

8) *load_map_from_csv: 2.42sec*

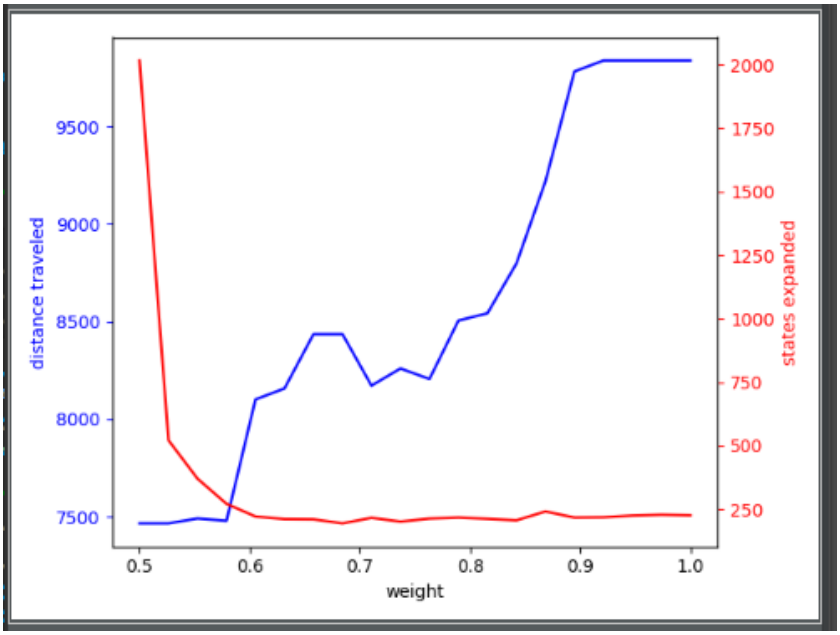
Solve the map problem.

Map(src: 54 dst: 549) UniformCost time: 0.99 #dev: 17355 total_cost: 7465.52955 |path|: 137 path: [54, 55, 56, 57, 58, 59, 60, 28893, 14580, 14590, 14591, 14592, 14593, 81892, 25814, 81, 26236, 26234, 1188, 33068, 33069, 33070, 15474, 33071, 5020, 21699, 33072, 33073, 33074, 16203, 9847, 9848, 9849, 9850, 9851, 335, 9852, 82906, 82907, 82908, 82909, 95454, 96539, 72369, 94627, 38553, 72367, 29007, 94632, 96540, 9269, 82890, 29049, 29026, 82682, 71897, 83380, 96541, 82904, 96542, 96543, 96544, 96545, 96546, 96547, 82911, 82928, 24841, 24842, 24843, 5215, 24844, 9274, 24845, 24846, 24847, 24848, 24849, 24850, 24851, 24852, 24853, 24854, 24855, 24856, 24857, 24858, 24859, 24860, 24861, 24862, 24863, 24864, 24865, 24866, 82208, 82209, 82210, 21518, 21431, 21432, 21433, 21434, 21435, 21436, 21437, 21438, 21439, 21440, 21441, 21442, 21443, 21444, 21445, 21446, 21447, 21448, 21449, 21450, 21451, 621, 21452, 21453, 21454, 21495, 21496, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549]

10) *Map(src: 54 dst: 549) A* (h=0, w=0.500) time: 1.24 #dev: 17355 total_cost: 7465.52955 |path|: 137 path: [54, 55, 56, 57, 58, 59, 60, 28893, 14580, 14590, 14591, 14592, 14593, 81892, 25814, 81, 26236, 26234, 1188, 33068, 33069, 33070, 15474, 33071, 5020, 21699, 33072, 33073, 33074, 16203, 9847, 9848, 9849, 9850, 9851, 335, 9852, 82906, 82907, 82908, 82909, 95454, 96539, 72369, 94627, 38553, 72367, 29007, 94632, 96540, 9269, 82890, 29049, 29026, 82682, 71897, 83380, 96541, 82904, 96542, 96543, 96544, 96545, 96546, 96547, 82911, 82928, 24841, 24842, 24843, 5215, 24844, 9274, 24845, 24846, 24847, 24848, 24849, 24850, 24851, 24852, 24853, 24854, 24855, 24856, 24857, 24858, 24859, 24860, 24861, 24862, 24863, 24864, 24865, 24866, 82208, 82209, 82210, 21518, 21431, 21432, 21433, 21434, 21435, 21436, 21437, 21438, 21439, 21440, 21441, 21442, 21443, 21444, 21445, 21446, 21447, 21448, 21449, 21450, 21451, 621, 21452, 21453, 21454, 21495, 21496, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549]*

11) *Map(src: 54 dst: 549) A* (h=AirDist, w=0.500) time: 16.18 #dev: 2016 total_cost: 7465.52955 |path|: 137 path: [54, 55, 56, 57, 58, 59, 60, 28893, 14580, 14590, 14591, 14592, 14593, 81892, 25814, 81, 26236, 26234, 1188, 33068, 33069, 33070, 15474, 33071, 5020, 21699, 33072, 33073, 33074, 16203, 9847, 9848, 9849, 9850, 9851, 335, 9852, 82906, 82907, 82908, 82909, 95454, 96539, 72369, 94627, 38553, 72367, 29007, 94632, 96540, 9269, 82890, 29049, 29026, 82682, 71897, 83380, 96541, 82904, 96542, 96543, 96544, 96545, 96546, 96547, 82911, 82928, 24841, 24842, 24843, 5215, 24844, 9274, 24845, 24846, 24847, 24848, 24849, 24850, 24851, 24852, 24853, 24854, 24855, 24856, 24857, 24858, 24859, 24860, 24861, 24862, 24863, 24864, 24865, 24866, 82208, 82209, 82210, 21518, 21431, 21432, 21433, 21434, 21435, 21436, 21437, 21438, 21439, 21440, 21441, 21442, 21443, 21444, 21445, 21446, 21447, 21448, 21449, 21450, 21451, 621, 21452, 21453, 21454, 21495, 21496, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549]*

12) **Weighted A-Star's solution & time quality measured by weight:**



The graph demonstrates an expected behavior:

We measure a solution's quality by the travel distance. A-star is optimal (under an admissible heuristic), so we get the global minimum for the solution's distance for weight = 0.5. However, we can also see that A-Star is relatively slow (compared to the greedy Best-First, which finds **some** solution relatively fast). As we increase the weight towards weight = 1.0: we see a decrease in the solution's quality, but the amount of expanded states gets much lower as well, so we get the solution much faster as expected.

14) The MaxAirDistance heuristic is admissible because it's optimistic: Given a state in the graph, the lowest possible distance needed to travel is at least the distance from this node to the furthest one to complete the task (which is the air distance between these two nodes), and we'll have to visit all of the delivery nodes (that includes the furthest node).

16) **The output is:**

load_map_from_csv: 2.49sec

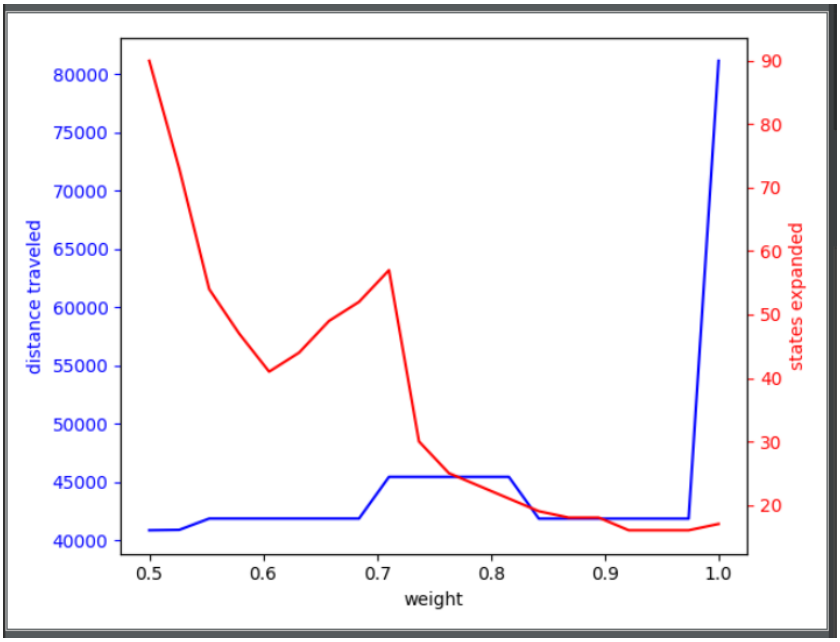
RelaxedDeliveries(big_delivery) A (h=MaxAirDist, w=0.500) time: 13.65 #dev: 6836 total_cost: 40844.21165 |path|: 11 path: [33919, 18409, 77726, 26690, 31221, 63050, 84034, 60664, 70557, 94941, 31008] gas-stations: [31221, 70557]*

17) **The output is:**

load_map_from_csv: 2.94sec

RelaxedDeliveries(big_delivery) A (h=MSTAirDist, w=0.500) time: 2.32 #dev: 90 total_cost: 40844.21165 |path|: 11 path: [33919, 18409, 77726, 26690, 31221, 63050, 84034, 60664, 70557, 94941, 31008] gas-stations: [31221, 70557]*

18) **Weighted A-Star's solution quality & time measured by weight under MSTAirDistHeuristic heuristic function:**



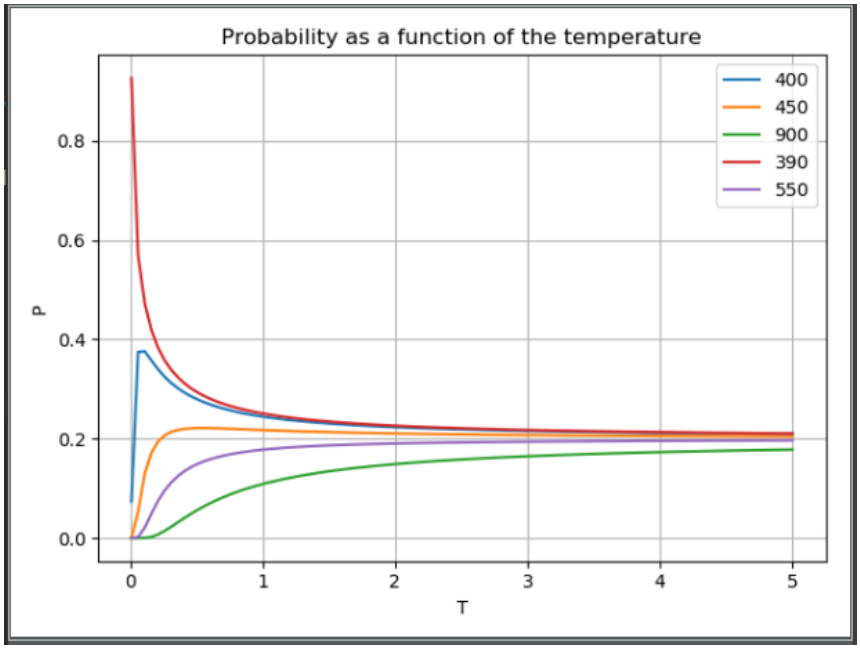
We got the expected behavior demonstrated in section #12 regarding the distance and expanded states curves. We can also see that the given heuristic function for this problem performs quite well: for weight values approaching 1.0, the solution's quality stays relatively good (compared to the optimal A-star for weight=0.5). This compromise can be very beneficial, as we expand much less states. However, we can also see at weight = 1.0 that we pay heavily for neglecting the costs altogether with this heuristic function: the solution for this case (Best-First) is two times bigger than A-star's solution.

19) $Let\ \alpha, \beta \in N^+$ be two distinct scales. We'll show that:
$\forall X_i \in X^t: \Pr_{\alpha}(X_i) = \Pr_{\beta}(X_i)$:

Let $X_i \in X^t$. We have:

$$\Pr_{\alpha}(X_i) = \frac{\left(\frac{X_i}{\alpha}\right)^{-\frac{1}{T}}}{\sum\left(\frac{X_h}{\alpha}\right)^{-\frac{1}{T}}} = \frac{\left(\frac{1}{\alpha}\right)^{-\frac{1}{T}} X_i^{-\frac{1}{T}}}{\left(\frac{1}{\alpha}\right)^{-\frac{1}{T}} \sum X_h^{-\frac{1}{T}}} = \frac{(X_i)^{-\frac{1}{T}}}{\sum X_h^{-\frac{1}{T}}} = \frac{\left(\frac{1}{\beta}\right)^{-\frac{1}{T}} X_i^{-\frac{1}{T}}}{\left(\frac{1}{\beta}\right)^{-\frac{1}{T}} \sum X_h^{-\frac{1}{T}}} = \frac{\left(\frac{X_i}{\beta}\right)^{-\frac{1}{T}}}{\sum\left(\frac{X_h}{\beta}\right)^{-\frac{1}{T}}} = \Pr_{\beta}(X_i)$$

20-22) **Distribution of vector X measured by temperature:**



As expected, this graph shows us that for very high temperatures ($T \to \infty$): the probability to choose each X_i is identical (and equal to $\frac{1}{|X|}$ which is 0.2 in our case).

This happens because for high temperatures, each member $X_i^{-\frac{1}{T}}$ in our probability formula is approaching the value of $const^0 = 1$.

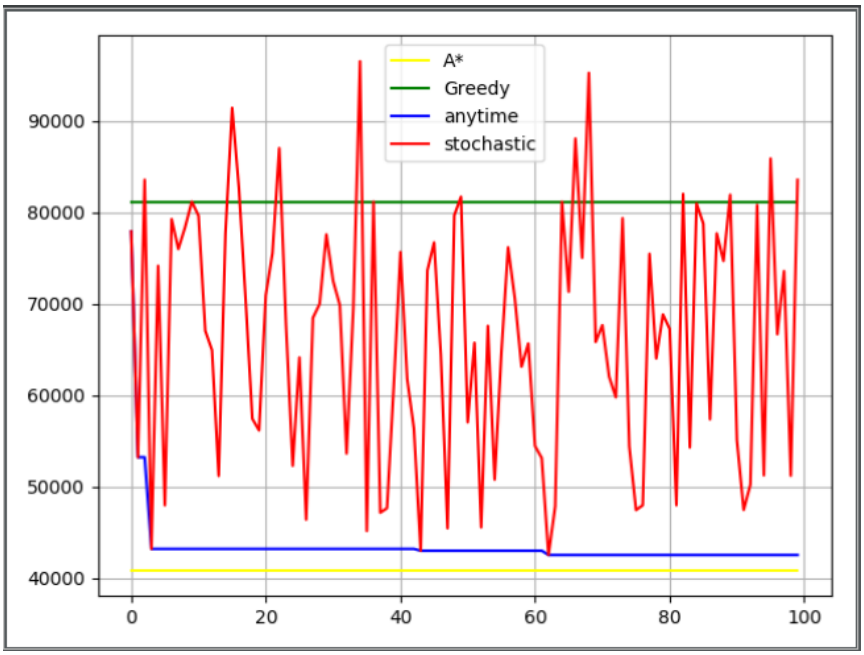
Likewise, for very low temperatures ($T \to 0^+$), we'll calculate the limit:

$$\lim_{T\to 0^+} P_m(X_m) = \lim_{T\to 0^+} \frac{\frac{1}{X_m^{\frac{1}{T}}}}{\frac{1}{X_m^{\frac{1}{T}}} + \sum_{i\neq m} \frac{1}{X_i^{\frac{1}{T}}}} = \lim_{T\to 0^+} \frac{1}{1 + \sum_{i\neq m} \frac{1}{\left(\frac{X_m}{X_i}\right)^{\frac{1}{T}}}}$$

$$= \lim_{T\to 0^+} 1 + \sum_{i\neq m} \left(\frac{X_i}{X_m}\right)^{\frac{1}{T}} = \lim_{T\to 0^+} 1 + 0 = 1$$

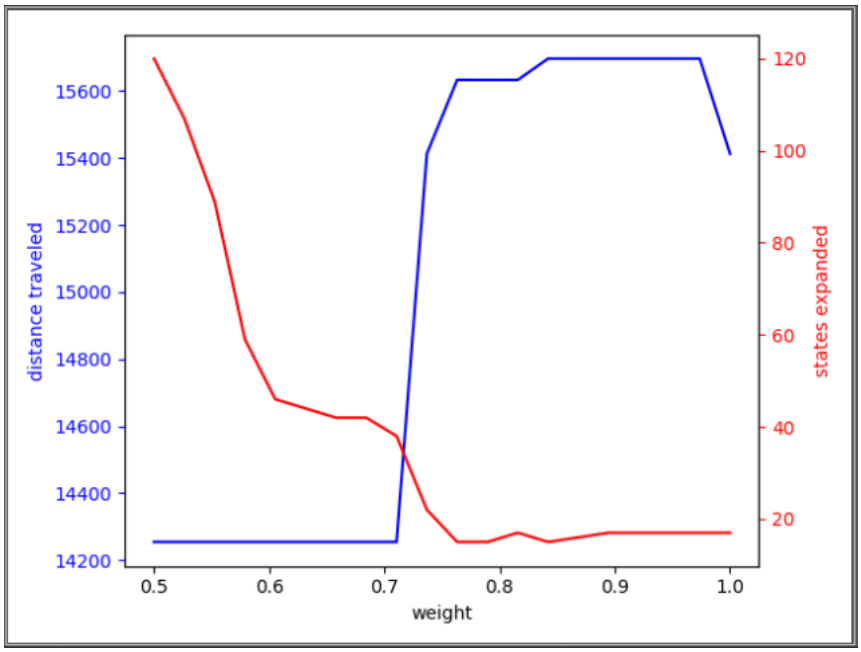
We showed that $P_m(X_m) = P(X_m)$ approaches 1. Since all the disjoint probabilities total to exactly 1: it follows that $\forall X_i \in X \backslash \{X_m\}$: $P(X_i)$ approaches 0.

24) **Anytime Greedy Stochastic algorithm compared to A-star and Greedy Best:**



The graph matches our expectations: we can see that A-star and greedy-best are both constant (as they do not make stochastic choices: their solution is the same in each iteration). We can also see that A-star is optimal whereas greedy-best is not. Anytime Greedy Stochastic finds a relatively good solution after a few iterations, and approaches A-star as we run it over more and more iterations, as expected.

26) **Weighted A-star algorithm's solution quality & time consumption for solving the strict delivery problem under MSTAirDist heuristic:**



This matches our expectations, as previously discussed in sections 12 and 18.

27) We'll define the heuristic function to be the result we get for solving the simplified **Relaxed Delivery** problem, where the input for the relaxed delivery problem is the **current state**, the **drop points** that we haven't visited yet, all the **map's gas stations**, the **gas tank capacity** and the **current gas amount**.
If we don't find a solution: we assume it's infinity (worst heuristic value).

This heuristic is applicable because:

$$\forall v1, v2\ s.t.\ isGoal(v2) = True\colon AirDistance(v1, v2) \leq minDist(v1, v2) = h^*(v1)$$

28) The output is:

StrictDeliveries(small_delivery) A* (h=RelaxedProb, w=0.500) time: 12.21 #dev: 80 total_cost: 14254.81633 |path|: 8 path: [43516, 67260, 17719, 43454, 43217, 32863, 7873, 42607] gas-stations: [17719, 32863] StrictDeliveries(small_delivery) A* (h=MSTAirDist, w=0.500) time: 0.39 #dev: 120 total_cost: 14254.81633 |path|: 8 path: [43516, 67260, 17719, 43454, 43217, 32863, 7873, 42607] gas-stations: [17719, 32863] the first weight which made the MST based heuristics better is: 0.5789473684210527

We'll observe that A-star with weight=0.5 expands more nodes under MSTAirDist heuristic than the RelaxedDevliery heuristic: the execution under RelaxedDelivery heuristic reduced the number of expanded states, **but** the heuristic function is extremely time consuming since we run A-star for each node just to calculate it's heuristic value.

The first weight we discovered that reduced the number of expanded nodes below the number of expanded nodes of the RelaxedProb heuristic is approximately 0.57. The RelaxedProb heuristic decreased our number of expanded nodes by 33.3% (for weight = 0.5), but it was over 30 times slower (so we didn't increase our run time at all).

Part B:

Let h be a partial heuristic function over S. We define:

$Applicable_h: S \to \{True, False\}$ $\boldsymbol{s.t}$ $Applicable_h(s) = True$ iff $h\ is\ defined\ on\ s$

We also assume:

$$\forall s \in S: \exists \delta > 0\ s.t: w(s) > \delta$$

$$\forall s \in S\ s.t\ Applicable_h(s) = True: h(s) \geq 0 \wedge h\ is\ applicable$$

Lastly, we define:

$$h_0(h,s) = \begin{cases} h(s), & Applicable_h(s) = True \\ 0, & Otherwise \end{cases}$$

a) **Claim: $\boldsymbol{h\ is\ applicable \Rightarrow h_0\ is\ applicable}$**

Proof:

Let $s \in S$. We'll break the problem into disjoint cases:
if $Applicable(s) = True$:
By the definition of h_0 we get: $h_0(h,s) = h(s)$.
We know that h is applicable, so: $h(s) \leq h^*(s)$. Thus, $h_0(h,s) \leq h^*(s)$

$$\Rightarrow h_0\ is\ applicable$$

if $Applicable(s) = False$:
By the definition of h_0 we get: $h_0(h,s) = 0 \leq h^*(s)$

$$\Rightarrow h_0 is\ applicable$$

■

b) We assume that the state space S is a tree.

Problem with h_0: h_0 is not informed: for every node that's not covered by h it'll return 0 with no regards to the state. We'll propose the following heuristic instead:

$$\forall s \in S\ s.t\ Applicable(s) = True:\ k(h,s) \stackrel{\text{def}}{=} h(s),$$

$$\forall s \in S\ \ s.t\ \ Applicate(s) = False:$$

$$if\ isGoal(s) = True: k(h,s) \stackrel{\text{def}}{=} 0$$

$$otherwise: k(h,s) \stackrel{\text{def}}{=} \max_{t \in Succ(s)} h_0(t)$$

We'll notice that k is defined for all nodes in S: every node s satisfies either $Applicable_h(s) = True$ or $Applicable_h(s) = False$. Moreover, $h_0(s)$ is defined for all $s \in S$ so k is well defined with domain S. Additionally, we'll notice that k is applicable since:

$$\forall s \in S\ s.t\ Applicable_h(s) = True: k(h,s) = h(s) \leq \boldsymbol{h}^*(\boldsymbol{s})\ , otherwise$$

$$if\ s \in Goal: k(h,s) = 0 = \boldsymbol{h}^*(\mathbf{0})\ , otherwise$$

$$k(h,s) = \max_{t \in Succ(s)} (h_0(t) + w(\{s.t\})) \leq \max_{t \in Succ(s)} (h^*(t) + w(\{s,t\})) \leq \boldsymbol{h}^*(\boldsymbol{s})$$

Since $\forall s \in S: |Succ(s)| \leq b$,by definition of the branching factor b we get:

$$RunTime_{k(h,s)} = O\big(\text{Applicable}_\text{h}(\text{s}) + \max\{O\big(h(s)\big), O\big(Goals(s)\big), d\big(O(h_0)\big)\}\big) =$$

$$O(1 + \max\{1, 1, d * 1\}) = O(d)$$

Alternatively, if we want O(d) runtime, we can define k as follows instead:

(we had some uncertainties with the demands of this exercise)

$$\forall s \in S\ s.t\ Applicable(s) = True:\ k(h,s) \stackrel{\text{def}}{=} h(s),$$

$$\forall s \in S\ \ s.t\ \ Applicable(s) = False:\ k(h,s) \stackrel{\text{def}}{=} \max\{h(v) - \sum_{e \in P} w(e), \qquad 0\}$$

$$where\ v\ is\ the\ closest\ father\ node\ of\ s\ such\ that$$
$$Applicable(v) = true, and\ P\ is\ the\ path\ from\ s\ to\ v$$

We'll notice that k is defined for all nodes in S: for every node s we have either $Applicable(s) = True$ or $Applicable(s) = False$. Ff $Applicable(s) = False$ and there is no such father node $v\ s.t. Applicable(v) = True$: k returns 0. k is applicable for the same reasons as before and runs in O(d) where d is the depth of the tree.

c) If we don't assume that the state space is a tree, the same heuristic function k (the first one, not the alternative option) can still be applied as we didn't assume anything about the topology of the graph.

d) **Claim: The algorithm exists.**

Proof: We'll present one such algorithm:

Starting from the root node, we'll traverse the state space in a **breadth-first** fashion (like BFS), but only expand nodes when the path from the root node to them is not greater than h(root). That is, when we pop a node from the queue, we'll push only the nodes that satisfy this rule into the queue. We can ignore the rest of the nodes because h(root) is assumed to be the perfect value, which is the optimal solution. So, if we discard paths that overwhelm this value, we're assured to still end up with **a** solution, and that solution will be the **optimal** solution. The algorithm runs for d iterations and expands b nodes at most in each iteration, so in total we have: $O(b^d)$.

We'll analyze A-star's time complexity under the h_0 heuristic: we'll notice that $h_0(s) = 0$ for every node except the root node, so:

$$\forall s \in S \backslash \{root\}: f(s) = h(s) + g(s) = 0 + g(s) = g(s)$$

What this tells us is that A-star is in fact behaving like Uniform-Cost, which runs at time complexity of $\mathrm{O}(\mathrm{b}^{\mathrm{d}} \log(b^d))$.

■