Incremental cost backoff in A*

Assume C(n), the cost from the start to node n
Assume H(n), the heuristic (distance based cost estimate) from n to the goal.

A* explanation

In A*, when selecting nodes to consider from the open set, you pick the node which minimizes C(n)+H(n). This is effectively selecting the most promising node, the node that minimizes cost from start and cost to goal.

However, the heuristic function needs to be finely tuned so that it doesn't consistently overestimate or underestimate in general. If the heuristic function underestimates cost, A* performs like Dijkstra—an equally expanding sphere nodes. (if the heuristic function is always zero, A* literally is the same as dijkstra) If the heuristic function overestimates cost, it will almost completely ignore cost and just pick the path that minimizes the heuristic distance.

Difficulties with heuristic cost functions in our application

In our application, it is very difficult to have an accurate heuristic function because the heuristic function doesn't take into account the terrain it is passing over (because that would be computationally prohibitive). Sometimes our heuristic grossly overestimates the cost to goal, such as when the terrain is extremely easy to navigate (e.g. superflat worlds), and sometimes it grossly underestimates the cost to goal (e.g. extremely rough terrain, like climbing mountains).

Our implementation

In our implementation of A*, it uses the standard equation of C(n)+H(n) to select nodes to consider. However, since we have a time-limited application, once it runs out of time we need to select the most promising node to navigate to (called "partial solution selection").Of course, if it selects a node in the goal while running, it will immediately return that node and it won't have to go partial solution selection.

Partial solution selection overview

Partial solution selection is when the graph search runs out of time without finding a full solution to the goal node. This can happen for many reasons: maybe the goal is so far away that that chunk isn't loaded, or maybe it's so far away that it couldn't get all the way there in the graph search in time. So, we need to select one of the nodes we considered as a partial solution, and we will navigate to there. In this selection, we aren't limited to the open set, we can select any node that the search has considered in its run time. Once we arrive, we will calculate another path to the goal. We call this "path stitching". Once the current path finishes and we aren't in the goal node, we recalculate.

Partial solution selection implementation

The standard way to select an A* partial solution is picking the node with the lowest overall C(n)+H(n).

When the cost heuristic underestimates, the node with the lowest C(n)+H(n) will be the starting node, or one very close to it. (see chart 1 for example of graph with positive resulting slope)

This is why we decided to use an incremental backoff of the cost heuristic.
Instead of keeping track of the best node so far by C(n)+H(n) during the search, we keep track of the best node by various metrics: H(n)+C(n)/x for various value of x.
Then, at the end, we start with the lowest value of x we kept track of and move upwards. Once we find a value of x for which the best node is more than 5 blocks from the start, we select that one.
Here's why this works.
Note that H(n) and C(n) are roughly linear functions with respect to taxicab blocks to goal and start, respectively. Therefore, H(n)+C(n)/x are all noisy linear functions.
By modifying x, we are modifying the slope of H(n)+C(n)/x with the goal that the function decreases as you get farther from the start and closer to the goal. (this way the node with the lowest value in this function will be close to the goal)
The reason why we slowly increase x in the evaluation is this allows us to pick the value of x that pays attention to C(n) the most (high values of x basically result in ignoring C(n) because C(n)/x is almost zero compared to H(n)). And the reason why we pick the first one that results in the best node being more than 5 blocks from the start is that if the best node is more than 5 blocks from the start, it's very likely that the slope of H(n)+C(n)/x is negative. (see chart 2 for example with negative resulting slope) Of course, if it was a perfect linear function, the best node would be the start node if the slope is positive, and the best node would be the goal node if the slope was negative. However, these are very noisy linear functions. It's possible that a positive slope equation could result in the best node not quite being the start node, but one that is very close to it, possibly adjacent. This is why we pick the first one that results in the best node being more than 5 blocks away; in testing we found that if the best node is more than 5 blocks away, it's probably going to go for as far as it can (probably dozens of blocks), because the slope of H(n)+C(n)/x is negative. So basically we are trying to pay attention to cost as much as possible, while still making sure that we get at least five blocks, which means that it will probably go as far as it can.

Chart 1, example of graph with positive slope

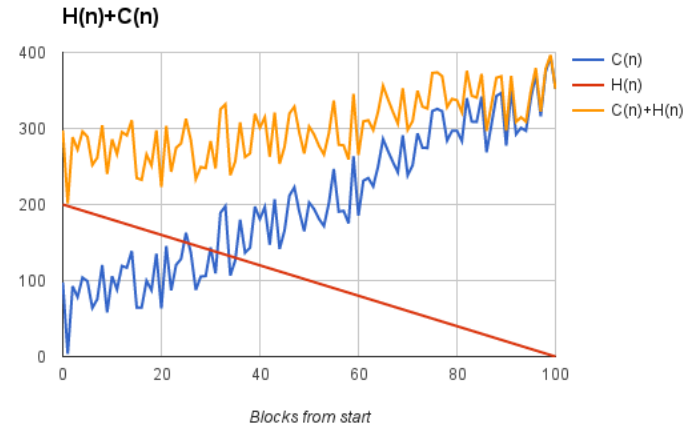

Chart 2, example of graph with negative slope