CSE 586 Assignment 2

Caleb Alexander

Problem 1. Suppose we want to construct three new airports in a given country. The objective is to minimize the distance from each city to its nearest airport. Describe the gradient descent and explain how this technique can help us to solve this problem. Write your solution mathematically as we discussed in class. (8 points)

Answer 1. As we did in class and in the slides, we will let C_i be the set of all cities with such that the ith airport is closes to them. Each city and airport location will be represented as (x_i, y_i) and (x_c, y_c) respectively. The summed distances for this problem can be represented as:

$$\sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2.$$

Our goal is to minimize this summation. It may be useful to think about why minimizing this summation will necessarily minimize the distance between each city and its nearest airport.

The distance between an arbitrary city (at (x_c, y_c)) and arbitrary airport (at (x_i, y_i)) can be formulated as

$$\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \quad \text{This is the usual euclidean distance.}$$

However, when $x \geq 0$, $\sqrt{x}$ is minimized precisely when x is minimized. Thus, we can discard the root. Now, since we potentially have multiple cities which are closest to airport i we must begin to worry about a larger sum. So for cities c_1 and c_2 we must minimize

$$(x_i - x_{c_1})^2 + (y_i - y_{c_1})^2 + (x_i - x_{c_2})^2 + (y_i - y_{c_2})^2.$$

Clearly, across all cities we have

$$\sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2.$$

The original sum now ought to be clear. To actually perform the minimization on this sum, we use the following optimization algorithm

$$x \leftarrow x - H_f^{-1}(x) \Delta f(x).$$

It now suffices to compute the gradient and hessian matrix for f.

$$\Delta f(x) = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}]$$

$$\begin{aligned}
&\text{see that for any } x_i \quad \frac{\partial f}{\partial x_i} = \frac{\partial}{\partial x_i}\sum_{i=1}^{3}\sum_{c\in C_i}(x_i - x_c)^2 + (y_i - y_c)^2 \\
&= \frac{\partial}{\partial x_i}\sum_{c\in C_i}(x_i - x_c)^2 \quad \text{discard terms without } x_i \\
&= \frac{\partial}{\partial x_i}\sum_{c\in C_i} x_i^2 - 2x_i x_c + x_c^2 \\
&= \sum_{c\in C_i} 2x_i - 2x_c \quad \text{differentiate} \\
&= 2\left(\sum_{c\in C_i} x_i - x_c\right).
\end{aligned}$$

For the hessian matrix notice that for any off-diagonal element we have the following second derivative

$$\frac{\partial}{\partial x_j}2\left(\sum_{c\in C_i} x_i - x_c\right) \quad \text{where } x_i \neq x_j.$$

So clearly, every off-diagonal element ought to be 0. The diagonal elements give us

$$\frac{\partial}{\partial x_i}2\left(\sum_{c\in C_i} x_i - x_c\right) = 2\sum_{c\in C_i} 1 = 2|c_i|.$$

Hence,

$$H_f(x) = \begin{bmatrix} 2|C_1| & 0 & 0 & 0 & 0 & 0 \\ 0 & 2|C_1| & 0 & 0 & 0 & 0 \\ 0 & 0 & 2|C_2| & 0 & 0 & 0 \\ 0 & 0 & 0 & 2|C_2| & 0 & 0 \\ 0 & 0 & 0 & 0 & 2|C_3| & 0 \\ 0 & 0 & 0 & 0 & 0 & 2|C_3| \end{bmatrix}$$

and

$$H_f^{-1}(x) = \begin{bmatrix} \frac{1}{2|C_1|} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2|C_1|} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2|C_2|} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2|C_2|} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2|C_3|} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2|C_3|} \end{bmatrix}.$$

Notice that in this case our Hessian matrices are constant, symmetric, and diagonal. In general, none of this need be true. The reason why this Hessian matrix has only constant values is because we have no terms with a power greater than 2 in our summation. Our matrix is symmetric because each second derivative is continuous; in general $\frac{\partial f}{\partial x_i}\frac{\partial f}{\partial x_j}f(x) = \frac{\partial f}{\partial x_j}\frac{\partial f}{\partial x_i}f(x)$ iff our function has continuous second derivatives. Our matrix is diagonal because we have no terms in our summation such that variables are paired (i.e. there are no $x_i x_j$ terms in our function).

Problem 2. Implement the gradient descent for a given scenario. (Optional group problem, 1 EXTRA credit, Due: November 10, 2024, show me your code in person)

Answer 2. TBD.

Problem 3. Design an AI that plays chess using the minimax algorithm and alpha-beta pruning. (Optional group problem, 3 EXTRA credits, Due: November 10, 2024, show me your code in person).

Answer 3. TBD.

Problem 4. (**a**) Give a set of broad conditions under which A* search reduces to BFS. (2 points)

(**b**) Does A* always return an optimal solution? Explain. (2 points)

Answer 4. (**a**) Consider any scenario in which $h(n)$ is uniform for all n. Our formula for A* is

$$f(n) = g(n) + h(n).$$

Notice that if $h(n)$ is constant then we can effectively remove the term since for any n_1, n_2 and constant c,

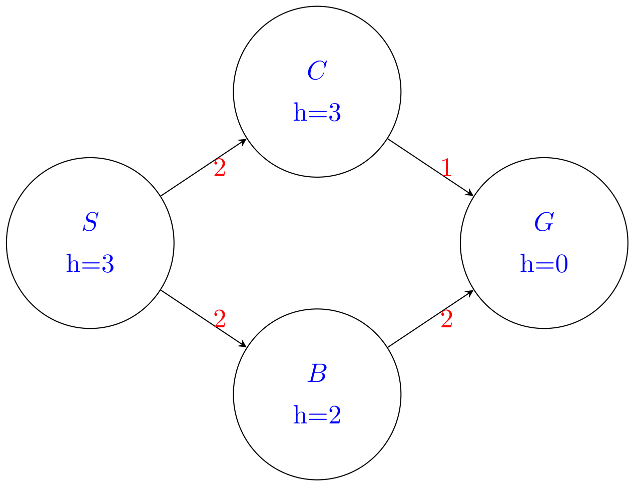
$$g(n_1) + c > g(n_2) + c \iff g(n_1) > g(n_2).$$

Hence we have that

$$f(n) = g(n),$$

which is exactly the formula for UCS. Now it suffices to find the condition in which UCS is equivalent to BFS. From previous material we know that UCS is equivalent to BFS when the $g(n)$ is uniform. Hence; if $h(n)$ and $g(n)$ are individually uniform then A* is equivalent to BFS.

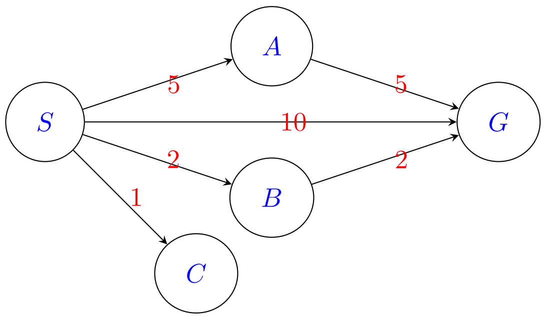
(**b**) No, A* is not always optimal. In particular, if h is not admissable then it is possible that A* will not be optimal. Consider the graph below.



A* would find the path $S - B - G$ with cost 4; however the optimal path is clearly $S - C - G$ with cost 3.

Problem 5. (**a**) When would DFS be a better choice than A* search? (3 points)

(**b**) Which path will A* return for the following search problem? (3 points)



Node	h
S	4
A	3
B	2
C	100
G	0

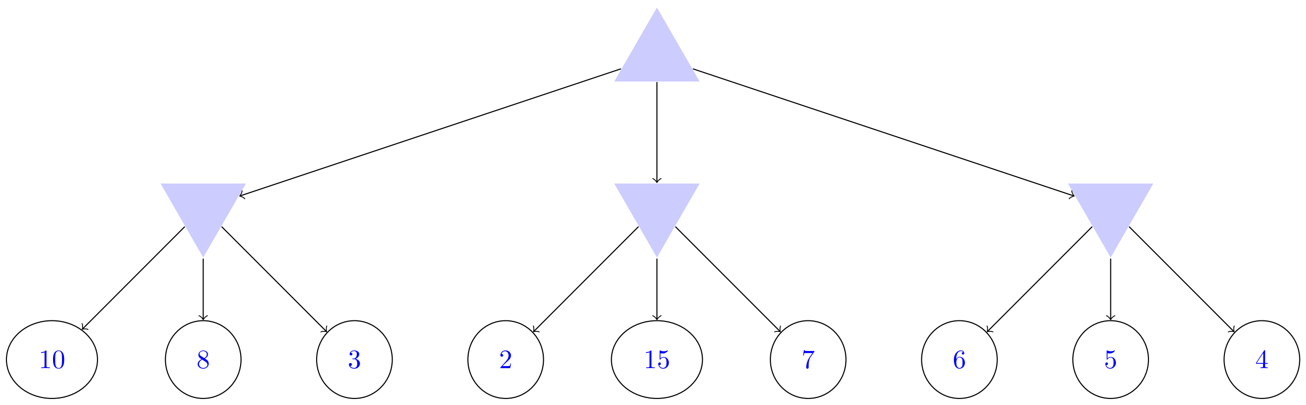
Answer 5. (**a**) DFS could be described as better than A* in a few scenarios, two of which include:

- You care about space efficiency but not necessarily time efficiency. In this case DFS may be better since you don't need to hold the whole tree.
- Your tree takes a DFS suited shape (i.e. low branching factor). In this case DFS could be described as better since; while it is not guaranteed to find an optimal solution, it is likely to find **a solution** very quickly.

(**b**) The path found is $S - B - G$ with a cost of 4.

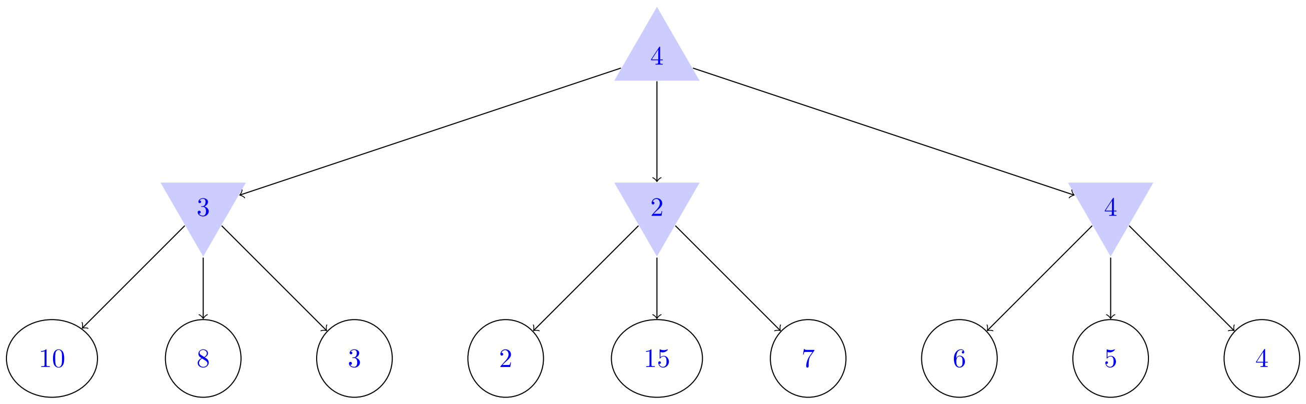
Expand	Frontier
S	$\{B : 2 + 2,\ A : 5 + 3,\ G : 10 + 0,\ C : 1 + 100\}$
B	$\{G : 2 + 2 + 0,\ A : 5 + 3,\ C : 1 + 100\}$

Problem 6. (**a**) Consider the zero-sum game tree shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. Assuming both players act optimally, fill in the minimax value of each node. (4 points)



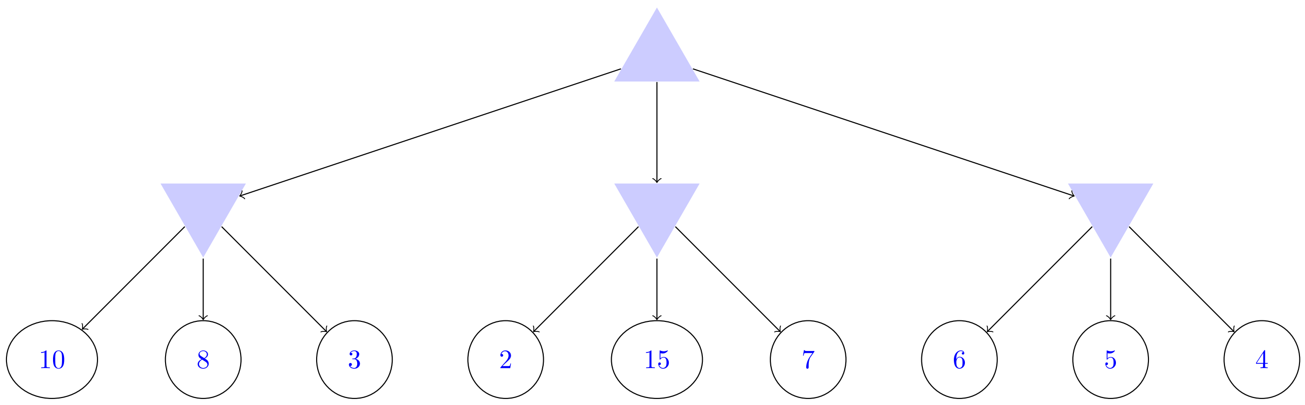
(**b**) b) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. Assume the search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child. (2 points)

Answer 6. (**a**) See the diagram below.



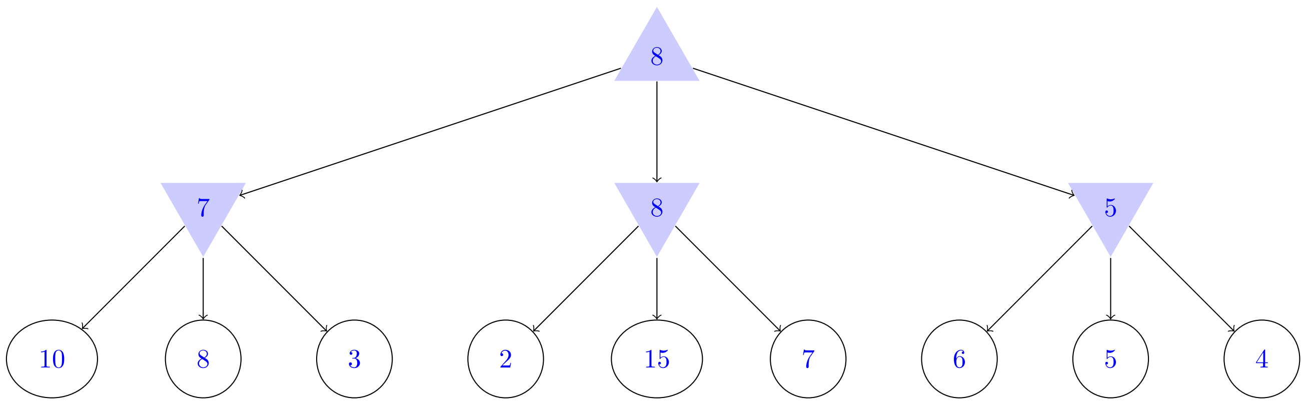
(**b**) We can prune the middle branches to 15 and 7 since we know that once we have found 2 as the first branch of that subtree; the value of said subtree will be less than or equal to 2. But, we have already found 3 as the minimum of the leftmost subtree; hence, the maximizer will always choose the left subtree over the middle subtree. We can prune no other branches.

Problem 7. (**a**) Again, consider the same zero-sum game tree, except that now, instead of a minimizing player, we have a chance node that will select one of the three values uniformly at random. Fill in the expectiminimax value of each node. The game tree is redrawn below for your convenience. (4 points)



(b) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. (2 points)

Answer 7. **(a)** Since the probability that the randomizing player will choose each node is uniform, we can write the expectiminmax as the usual arithmetic mean.



(b) No nodes can be pruned because as we traverse the tree we cannot be sure of the mean of the branches until we have examined every node. To demonstrate this; imagine that we are traversing the tree and have traversed to the node 15 in the middle subtree. We know that the mean of the previous subtree was 7 and the current mean of the current subtree is $2 + 15/2 = 8.5$. We cannot prune the next branch because we can always find a number x such that x could raise or lower the mean of the current subtree. This logic holds for every branch of the tree.

Problem 8. The evaluation function of a well-known search algorithm is

$$f(n) = (2 - w)g(n) + wh(n).$$

For what values of w is this search algorithm optimal, assuming that h is admissible? What kind of search does this perform for $w = 0$ and $w = 1$? (Only for CSE 586 students)

Answer 8. Assuming that h is admissible, we can guarantee that f is optimal for both $w = 0$ and $w = 1$. This is because if $w = 0$, then

$$f(n) = 2g(n).$$

Notice that this is UCS since $g(n) > g(n') \iff 2g(n) > 2g(n')$. We know that UCS is optimal; hence why the algorithm must be optimal in this scenario. Similarly, when $w = 1$,

$$f(n) = g(n) + h(n).$$

Clearly, this is just A*. Since we know that A* is optimal when h is admissible and we have assumed that h is admissible, we can say that f is optimal.

More generally, notice that if $w \in [0, 1]$ each $g(n)$ is scaled by an identical proportion and $h(n)$ is never overestimated (since h is admissible from the start and we scale it by a factor ≤ 1). Hence, f must be optimal when $w \in [0, 1]$. If $w > 1$, then we cannot guarantee that h is admissible anymore; and thus cannot guarantee optimality. Finally, clearly if $w < 0$ we cannot guarantee optimality since this would value nodes with higher h values over nodes with lower h values.