CS430 Final Project Report

Dynamic Programming

Matthew Knippen

Illinois Institute of Technology

Due: November 26, 2012

Algorithm Design:

To solve this problem, my algorithm is broken into two parts. Before that is done, there is a bit of setup. First, all of the targets are sorted by increasing x axis. Given that we should be able to choose how our input file is designed, we should be able to make this assumption already. We then separate all of the sensors to upper and lower sensors, meaning that a sensor is either above or below the highway. Since we know that no sensor can be on the highway, every sensor will fall into one of these two categories.The first part (called the base case) took the first target (sorted by increasing x coordinate) and looked at every possible combination of a single upper and single lower sensor, and stored the resulting weight providing that one of the two sensors covered the initial target. We also introduce an extra sensor on both the top and bottom, with infinite radius and zero weight. This is used to test a single sensor with either no top or no bottom.

For subsequent calls, we use the the following algorithm:

$$Ti(U,D) = \min_{U_1,D_1}(Ti\text{-}1(U_1,D_1) + [U_1 \neq U]cost(U) + [D_1 \neq D]cost(D))$$

where:

U_1 represents any sensor that is dominated by U at Ti
D_1 represents any sensor that is dominated by D at Ti
$[X_1 \neq X] == 0$ if $X_1 = X$, and 1 if $X_1 \neq X$
cost(X) is the cost of the sensor X

This algorithm makes use of the theory of domination, which says that one sensor has control or domination of another at a vertical line. The vertical line is formed by the x coordinate of Ti. We can say that U dominates U_1 at Ti if one of the following is true:

X = Ti(x)

- U_1 does not intersect X
- The lower intersection endpoint of U_1 and X is higher than the lower intersection endpoint of U and X.
- The lower intersection endpoints are equal, but the center of U_1 is to the right of the center of U.

Similarly, We can say that D dominates D_1 at Ti if one of the following is true:

- D_1 does not intersect X
- The higher intersection endpoint of D_1 and X is lower than the higher intersection endpoint of D and X.
- The lower intersection endpoints are equal, but the center of D_1 is to the right of the center of D.

By using domination, you can effectively limit the amount of times that the recursive algorithm is called, making the running time polynomial. (See below for additional information on running time)

Pseudocode:

```
Array M[t][u][d] //3D array initialized to all empty.
Initialize all inf/-inf sensors in M to inf

//base case
for each upper sensor u
	for each lower sensor d
		store cost(u)+cost(d) along with sensors u,d into M
	end
end

//determine final answer
minCost = inf
finalAnswer = NULL
for each upper sensor u
	for each lower sensor d
		OPT(tn, u, d)
		if (cost < minCost)
			minCost = cost
			finalAnswer = solution returned from OPT
		end
	end
end

//recursive method being called above
OPT(ti,u,d) {
	check if solution already exists in memory. If so, return that.

	minCost = inf
	answer = NULL
	for each upper sensor u’
		if (u dominates u’ at t)
			for each upper sensor d’
				if (d dominates d’ at t && both u’ & d’ != inf)
					if (at least one sensor covers t)
						OPT(ti-1,u’,d’)
						if cost != inf
							if (U != U’)
								cost += cost(U)
							end
							if (D != D’)
								cost += cost(D)
							end

							if (cost < minCost)
```

```
                                                  minCost = cost
                                                  answer = solution from OPT
                                             end
                                        end
                                   end
                              end
                         end
                    end
          add U and D to answer if they are not already present
          store the answer and minCost in M
          return minCost and answer
}
```

Proof of Correctness:

By filling in every option for the base case, we know that we have covered every possible option for the first target, which means for the base case, we know that the ideal solution is present. (If all solutions are present, than the optimal one must be there) For every iterative step, we search for the optimal solution by building on every possible solution that came before it, and store the most optimal solution. We only have to check the disks that are dominated due to the transitive property. If s1 is dominated by s2, and s2 is dominated by s3, than s1 dominates s3. Using this principle, if we prove that one sensor (si) dominates another (si+1), we have proved that it dominates every sensor that si dominates.

Running Time:

Running time for the base case is approximately $(s/2 +1)(s/2+1)-1$, or $O(s^2)$. This is due to multiplying all of the upper sensors (with the infinity sensor) with all of the lower sensors (with the infinity sensor) minus the option where the two infinity sensors are together. For each iterative step, we must run the same $O(s^2)$ calculations, plus compare each one of those against all of the the previous calculations, which is also $O(s^2)$. Thus we have $(s^2)^2 = O(s^4)$ running time for each remaining target. Thus, the total running time is $O(t*s^4)$, or simply $O(s^4)$.

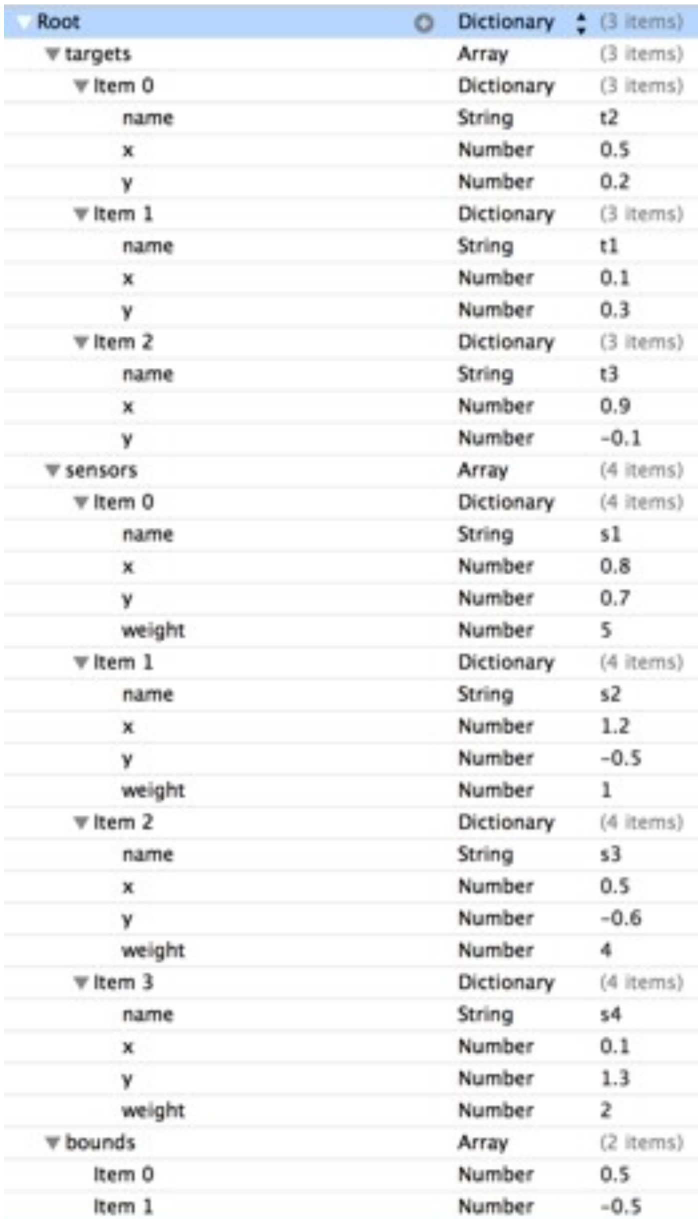
Source Code:

Included in ZIP file. Also accessible from GitHub: https://github.com/mknippen/IITCS430Project

Test Source Data and Output:

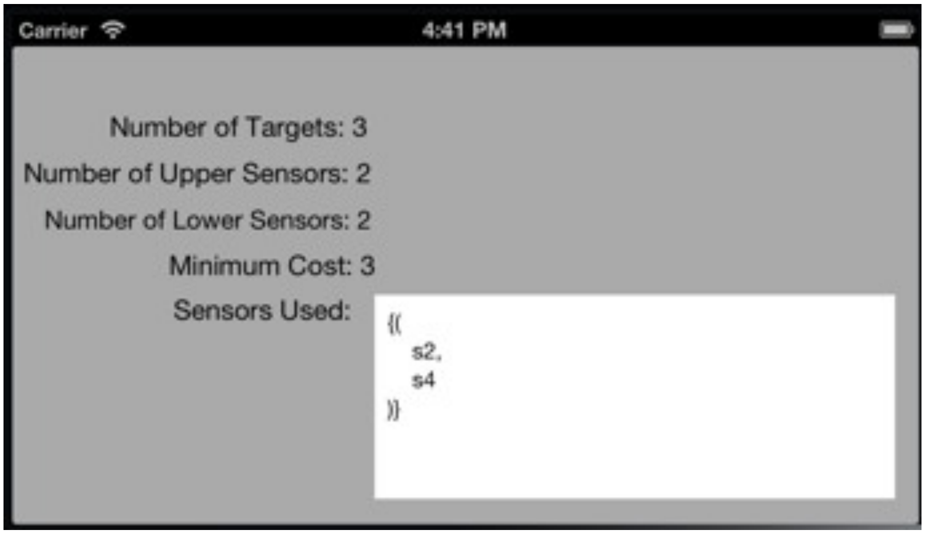
Source data included a Property List (plist) of information. Plists are extremely simple to create, and extremely simple to build into an application. Plists can be created/edited in any text editor, but using Xcode makes it much simpler. Screenshots of source data and output are provided below.

Input:



Root	Dictionary	(3 items)
targets	Array	(3 items)
Item 0	Dictionary	(3 items)
name	String	t2
x	Number	0.5
y	Number	0.2
Item 1	Dictionary	(3 items)
name	String	t1
x	Number	0.1
y	Number	0.3
Item 2	Dictionary	(3 items)
name	String	t3
x	Number	0.9
y	Number	-0.1
sensors	Array	(4 items)
Item 0	Dictionary	(4 items)
name	String	s1
x	Number	0.8
y	Number	0.7
weight	Number	5
Item 1	Dictionary	(4 items)
name	String	s2
x	Number	1.2
y	Number	-0.5
weight	Number	1
Item 2	Dictionary	(4 items)
name	String	s3
x	Number	0.5
y	Number	-0.6
weight	Number	4
Item 3	Dictionary	(4 items)
name	String	s4
x	Number	0.1
y	Number	1.3
weight	Number	2
bounds	Array	(2 items)
Item 0	Number	0.5
Item 1	Number	-0.5

Output:



README File is Included