Initial analysis

After reading the case, I found the case somewhat similar to the Twitter lab that was conducted earlier in the semester. An initial brainstorming session resulted in several of the theories below:

Methodlogy	Psueudo-random distribution	Search and margin in clusters	Segmenting the map
Description	1. Pseudo-randomly distribute stops according to the number required using some sort of distribution 2. Score these separate distributions and find which particular one works out best 3. Find some sort of method to find points which do well?	1. Find optimal location to place the first stop 2. Subsequently find the optimal location for the next best stop taking into account the first stop 3. Continue until number of pokestops is found	1. Segment the map into smaller areas depending on number of pokestops 2. Rank the different areas depending on the summation of population densities 3. Assign pokestops, 1 to each area, and determine the best location to place them based on score calculation of the score within the area
Pros	1. If ran a good number of times, is completely comprehensive with the types of formations given 2. Accuracy increases with a larger stop to (array size) ratio? 3. Is better at even maps	1. Identifies the best location for the initial stop 2. Seems like a good method to accurately place stops if there are several dense clusters	1. Ensures even distribution of pokestops on the map 2. Complexity would not be high as it would not involve such strenuous iterations
Cons	1. The lower the number of stops the higher the required complexity needed to find the best alternative 2. Is in essence random…. Variability could pose an issue 3. Don't know when is the best number to stop at! 4. Timing will always be an issue	1. Placement of stops is dependent on the placement of the initial stop. Which is likely near the middle 2. The code becomes more intensive quickly as the number of pokestops to assign increases	1. If the map is uneven, pokestops would be very misplaced 2. The number of segments to split into could be trick to decipher

After some deliberation I decided to proceed with the "Search and margin in clusters".

Midpoint evaluation

After successfully coding the algorithm I found myself struck with several issues. My cod e was running decently on the server, with a score of 4100000~ but at a timing of about 40 seconds. This is visualised on my results with map2.csv. The main factors contribution to the sup-par results are outlined below:

```
Summary:
        stops   score   time taken
        1       167476  1.180326
        2       129258  1.580116
        3       120999  1.237374
        4       92361   1.058263
        5       95885   0.801076
        6       64635   0.784531
        7       63526   0.746528
        8       59332   0.633462
        9       58903   0.728017
        10      59416   0.548388
        11      55751   0.620438
        12      49199   0.69999
        13      44854   0.820066
        14      42051   0.896649
        15      43932   0.62197
        16      42651   0.689002
        17      41571   0.781541
        18      40222   0.881176
        19      39113   0.622441
        20      37907   0.704998
        21      37494   0.782091
        22      34875   0.825608
        23      34649   0.90215
        24      33855   0.960216
        25      33257   1.031905
        26      32999   1.123794
        27      32697   1.194878
        28      32233   1.277423
        29      27703   1.303382
        30      26954   1.39799
=> true
```

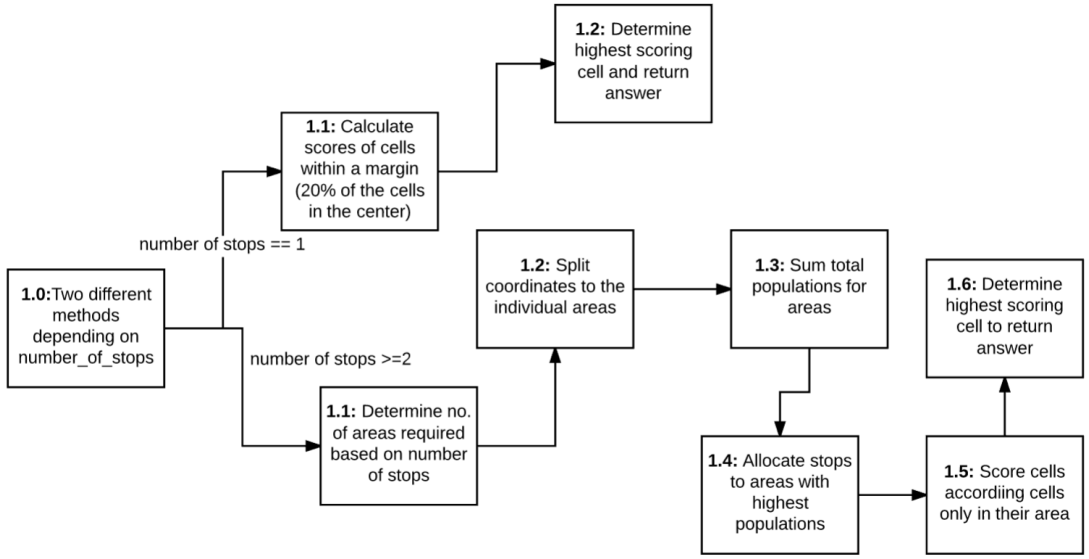
- **High complexity with size of the map –** my initial code had to go through, in a brute force manner, every cell on the map. It calculated score for every cell, making its complexity easily O(n^2) for the scoring system alone. I made do with several constraints (creating a no-placing new stops margin around stops placed formerly) to get the code running, but could not fulfil step 2, to its optimal capacity.
- **Dependent on initial stops –** as expected, how the code ran made it highly dependent placings. But this was not an optimal formation. A simple example of when placing two pokestops: the first stop is usually placed at the centre of the map, the second stop would have to accommodate to that being placed on its perimeter. The optimal formation of both stops would obviously involve the placement of first stop not at the center.

Ultimately what contributed to my change of direction was when I observed a visual representation of the centre of map2.csv according to my scoring metric (higher the better):

1334.012	1359.679	1362.703	1389.231	1435.39	1482.878	1508.509	1489.561	1484.44	1518.724	1582.342	1626.95	1636.71	1649.172
1224.041	1250.157	1272.552	1306.602	1347.399	1405.08	1423.289	1431.568	1439.062	1507.456	1583.223	1623.223	1644.569	1650.733
1133.596	1161.205	1189.088	1223.265	1266.436	1305.394	1340.616	1357.053	1410.314	1476.876	1543.209	1595.067	1606.755	1624.866
1079.127	1106.107	1134.426	1165.558	1199.402	1233.768	1267.18	1302.517	1356.924	1425.402	1477.906	1496.315	1530.194	1520.792
1041.858	1068.272	1096.511	1127.122	1160.153	1194.99	1230.995	1268.666	1310.356	1354.061	1386.11	1408.767	1416.769	1423.089
1015.624	1041.84	1070.361	1101.975	1137.209	1176.106	1217.759	1259.535	1296.898	1324.329	1338.38	1342.512	1340.633	1335.271
998.5677	1024.259	1052.977	1086.469	1126.131	1173.6	1229.15	1288.24	1334.448	1356.996	1343.973	1325.068	1307.689	1291.556
988.7246	1014.249	1042.14	1077.681	1123.3	1186.519	1274.792	1344.876	1401.781	1421.577	1405.716	1344.981	1304.607	1274.231
982.7692	1009.211	1036.063	1072.881	1123.006	1203.936	1288.686	1379.666	1442.739	1476.78	1449.837	1399.815	1330.066	1279.189
982.2284	1008.633	1034.558	1070.543	1120.816	1192.359	1287.336	1359.087	1425.19	1469.624	1481.832	1446.082	1381.859	1307.2
984.1031	1010.196	1038.206	1070.739	1120.285	1191.308	1278.323	1350.191	1397.809	1440.17	1476.707	1476.171	1443.679	1362.08
983.5754	1014.468	1039.081	1071.803	1122.221	1208.672	1302.237	1375.264	1411.058	1440.133	1475.615	1498.37	1470.172	1396.855
982.8787	1009.916	1035.606	1067.929	1121.408	1216.022	1330.561	1406.801	1451.067	1470.055	1490.215	1506.13	1460.327	1384.789
979.1824	1001.278	1025.458	1057.942	1115.5	1213.091	1332.111	1436.166	1473.626	1512.193	1506.285	1497.733	1443.038	1353.015

From this I noticed that the map was more evenly spread then I had first thought out. The “red” on the left still had substantial scores, not so different from the “green”. Also there is a gradient of slow transition from left to right. Clusters were not dense enough. “Search and Margin in clusters” was not suited for this map. Therefore, my project swapped gears to “Segmenting the map”.

The final algorithm – Segmenting the map



Above is a brief description of the code, which ultimately did much better than the previous. I believe the complexity of the code is (if map size was y and x; and number of stops was n):

$$Big\ O\ Complexity = O(\frac{Y * X}{N})$$

Largest steps being in the search that is required in Step 1.5. As the number of areas increase as the number of pokestops increases, and that in turn reduces the total amount of cells needed to be searched on the map when considering the scoring of a cell. Each cell is always only searched once.

All in all, this was a great experience. I feel like if I were to do a project similar to this again, I would first analyse the data I was dealt with more scrutiny. Ultimately if complexity was not a concern, I feel that “Search and margin clustering” is better at denser clusters, while the “Segmenting the map” is better at more even maps.