Nicholas Denobrega
Samuel Buckler
CS375 Final Project Report

# **Optimal Development Problem**

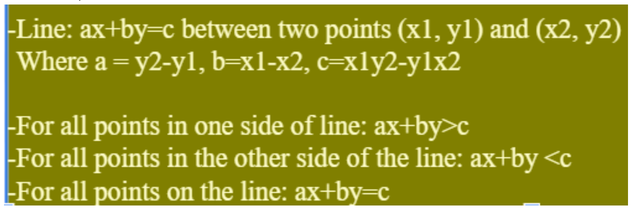
A builder is planning a residential development in a designated area and aims to include a set of points of interest (POIs)—such as parks, schools, and community centers—within the development. The builder has to work within his budget, which will cover the cost of buying/developing the land (this is related to a fixed square foot cost), as well as implementing a road network between the points of interest. In a large residential area, there are many different points that the builder may want to consider putting in his development, however it would be time-consuming for the builder to go through and run all the calculations himself. Our program aims to automate this process. Given a set of possible points for the builder to include in his development and a budget, the builder should be able to choose which points he wants, and the program decides if buying/developing the land and implementing roads would be feasible. In order to determine how much land the builder needs to buy, we can construct a convex hull that encapsulates all of the selected points of interest. We can calculate the area of the convex hull, and based on the fixed square-foot cost of buying the land, we can compute how much it would cost the builder to buy. The next step is to connect all of the points of interest by a roadway network, which will be implemented using a modified Floyd's all-paths shortest paths graph algorithm. There is also a fixed linear foot cost for building roads, which will contribute to the total cost that the builder will have to pay for his development. After the cost of buying the land is computed, and the cost of building the roads is computed, we can determine whether or not the builder can afford to build his development around all of the points of interest selected.

When breaking down the objective of the problem, we can see three main parts: constraints of the builder, land development, and roadway construction. Firstly, the builder has a budget constraint, a land cost constraint, and a road cost constraint. The budget constraint is given by the builder, the land cost constraint and road cost constraint is given based on the area the builder chooses to develop in. The combined cost of buying the land and building the roads must not exceed his given budget. In terms of land development, the builder must purchase land that encompass as many POI's as possible, and the space between them, to allow for future development of roads and other structures. This can be done by constructing a convex hull based on the set of chosen POI. The cost of buying and developing the land, excluding the roadway system, is a function of the area encapsulated by the convex hull. I.E, the cost of buying the land = Area of Convex Hull x Landcost. Finally, when considering the roadway construction, we must ensure that the roadways connect all the POI's while minimizing total distance, which in turn minimizes the total cost of roadways. This can be done using a shortest-path network, where the POI are the vertices, the roadways are the edges connecting the vertices, and the distance between the two POI's is the edge weight. The cost of the roadways is proportional to the total length of the connections x cost of building the roads.

Before our algorithm runs, the builder is given a fixed land cost, a fixed road cost, and a fixed set of POI. These parameters are based on the area of development that the builder, as different areas have different building costs and POI. In terms of our algorithm, we assume they are given before we start. The builder is then prompted to select a subset of these given points and give his desired budget. This allows for builder input, as the builder may have certain POI he wishes to include or exclude from his development, and a certain budget he wishes to use.

The points that the builder selects are sorted by x-coordinate so that the Divide and Conquer Convex Hull algorithm can break the coordinates down into left-hulls and right-hulls, before merging them all together. We loop through each subset of points, by iteratively considering 1 more point than previously, where you initially consider only the first point, then the first two points, etc. so that the subsets of points grows in size by 1 point for each iteration of the loop. This ensures that we consider every possible combination of points in the chosen set. For each subset of points, we will construct a convex hull around those points.

The convex hull construction works through Divide and Conquer by splitting the subset of points into two halves (left hull and right hull) recursively until the base case is reached: when there are 5 or less points in the left and right hull, the goal is to compute the left and right hull boundaries. When considering 5 points at a time, it is possible to compute the boundary line with the following brute force method: For every possible pair of points $A$ and $B$, create line segment $AB$ (using formula from the slides):

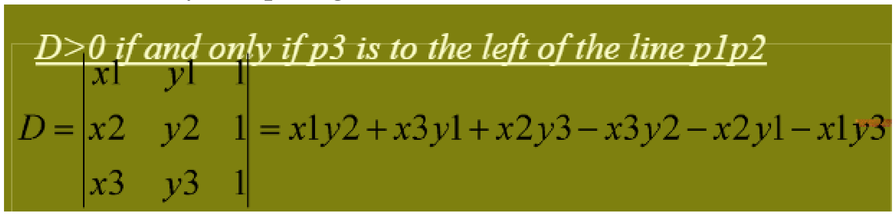


-Line: ax+by=c between two points (x1, y1) and (x2, y2)
Where a = y2-y1, b=x1-x2, c=x1y2-y1x2

-For all points in one side of line: ax+by>c
-For all points in the other side of the line: ax+by <c
-For all points on the line: ax+by=c

The line segment $AB$ represents a boundary of the hull if and only if all of the remaining points lie on either one side of the line or the other, but not both. If all the remaining points are on the same side of the line, we will add this line segment $AB$ to the boundary of the hull. We must add these points in counterclockwise order for the merging process to execute correctly. Between two points $P$ and $Q$, we can determine which point is counterclockwise to which by computing a midpoint $M$ that is the average of all the points in the hull. We then compute vectors $PM$ and $QM$, take the cross product $PM \times QM$, and the result will indicate which point is counterclockwise to which. A positive cross-product would indicate that $Q$ is counterclockwise to $P$, while a negative cross-product would indicate that $P$ is counterclockwise to $Q$. This represents the order in which they should be added to our convex hull boundary vector.

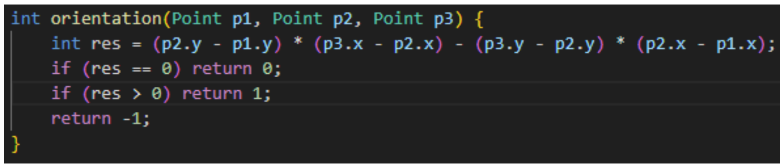
Merging two left and right convex hulls, which we can call $L$ and $R$, by taking the left half of convex hull $L$ and attaching it to the right half of convex hull $R$. Carrying out this merge involves computation of upper and lower tangent lines. The upper tangent line is the line segment between two points $A$ and $B$ where $A$ is in $L$ and $B$ is in $R$ such that $AB$ does not cross through either $L$ or $R$. Tangent line computation initializes with line $AB$ where $A$ is the rightmost point in $L$ and $B$ is the leftmost point in $R$. If the line segment $AB$ crosses through either hull, the algorithm iterates through $L$ in a counterclockwise motion and iterates through $R$ in a clockwise motion until $AB$ does not cross through either hull. Determining whether or not $AB$ crosses through either hull is done by computing the determinant with formula:



*D>0 if and only if p3 is to the left of the line p1p2*

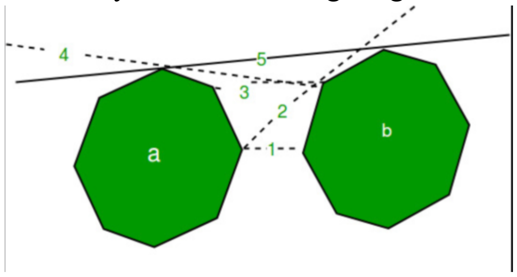
$$D = \begin{vmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{vmatrix} = x1y2 + x3y1 + x2y3 - x3y2 - x2y1 - x1y3$$

Notice our code uses this formula here:

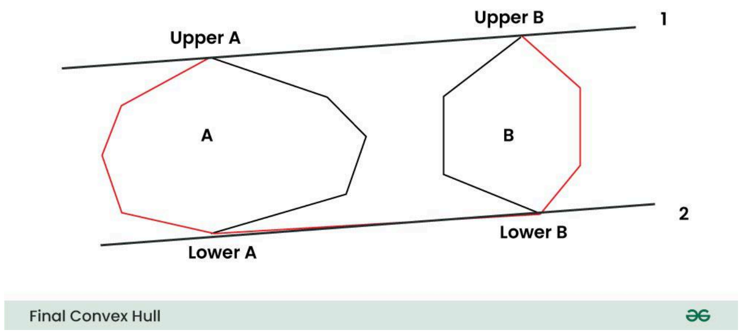


```
int orientation(Point p1, Point p2, Point p3) {
    int res = (p2.y - p1.y) * (p3.x - p2.x) - (p3.y - p2.y) * (p2.x - p1.x);
    if (res == 0) return 0;
    if (res > 0) return 1;
    return -1;
}
```

The algorithm then computes the lower tangent in the same fashion as the upper tangent, where the iteration through each hull is in the reverse direction if the line segment crosses through either hull. This can be shown visually in the following diagram:



Now that the algorithm has computed upper and lower tangent lines, the upper and lower points in *L* and *R* represent the bounds between the left half of the hull and the right half of the hull. So the points between the upper tangent point and lower tangent point in *L* in a counterclockwise motion represent the left half of the merged hull, whereas the points between the upper tangent point and lower tangent point of *R* in a clockwise motion represent the right half of the merged hull. So to finally merge these two hulls, we iterate through points of the left hull starting at upper tangent, and ending at lower tangent, adding the point to our new hull for each iteration. Then we iterate through points of right hull starting at lower tangent, and ending at lower tangent, adding the point to our new hull for each iteration. Considering the following example:



we would iteratively add points from *Upper A* to *Lower A* in counter-clockwise order, then add points from *Lower B* to *Upper B* in clockwise order. The points from *Upper A* to *Lower A* in counter-clockwise order are disregarded, and the points from *Lower B* to *Upper B* in clockwise order are disregarded.

Following the construction of our current convex hull, we now calculate the area of the convex hull, which will be used when determining the cost of buying the land. To calculate the area, we use the Shoelace Formula, or formally known as the Gauss's Area Formula. This formula is a mathematical method of calculating the area of a simple polygon, where all the vertices are known. Since a convex hull is a simple polygon (no self-intersections), and our convex hull is represented in a counterclockwise-ordered list, we are able to apply this formula. For *n* POI with x and y coordinates $(x, y)$, our list is in the form:

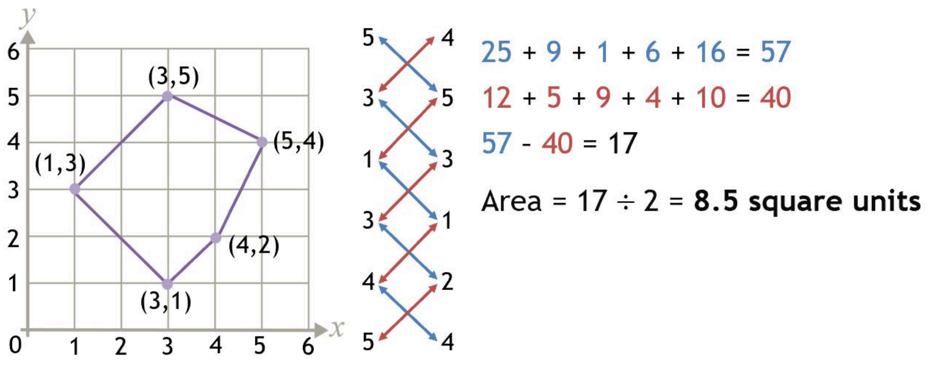
$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n), (x_{n+1}, y_{n+1})$$

$$\text{where } (x_{n+1}, y_{n+1}) = (x_1, y_1),$$

The Shoelace Formula is denoted as:

$$\text{Area} = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - y_i x_{i+1})\right|$$

Where xi and yi are the x and y coordinates of the current point considered, and xi+1 and yi+1 are the x and y coordinates of the next point in the list. We can visually see how the Shoelace Formula works in the following figure:



$25 + 9 + 1 + 6 + 16 = 57$

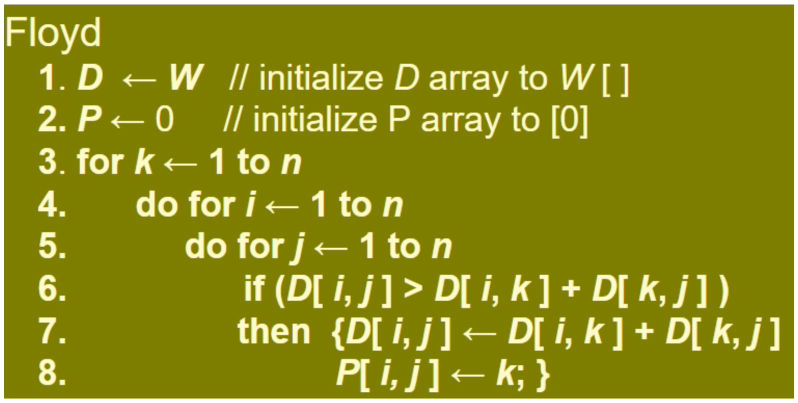
$12 + 5 + 9 + 4 + 10 = 40$

$57 - 40 = 17$

Area = 17 ÷ 2 = **8.5 square units**

Once we have the calculated area of the convex hull, it is multiplied by the fixed land cost to determine the cost of buying the land. Before proceeding to calculate the cost of constructing the roadways, we compare this cost with the budget. If the cost is greater than the budget, we know this convex hull is not the optimal hull, and move on to the next subset of points.

If the cost of buying the land for the current convex hull does not exceed the budget of the builder, we now move on to calculating the cost of the roadways. First, we take the current subset that we are considering. This is the same subset that we constructed the current convex hull with. Taking this subset, an n x n adjacency matrix is built where n is the number of points in the subset, and all values are set to INF. We then iterate through each set of points, and calculate the Euclidean distance between the two points, and update the value in the matrix to this distance value. At this point, we have a graph representation of the current subset, where every point is connected to every other point, with weights = Euclidean distance. Then using Floyd's Algorithm on the graph, we calculate the shortest distance between every set of points, using the formula from the slides:



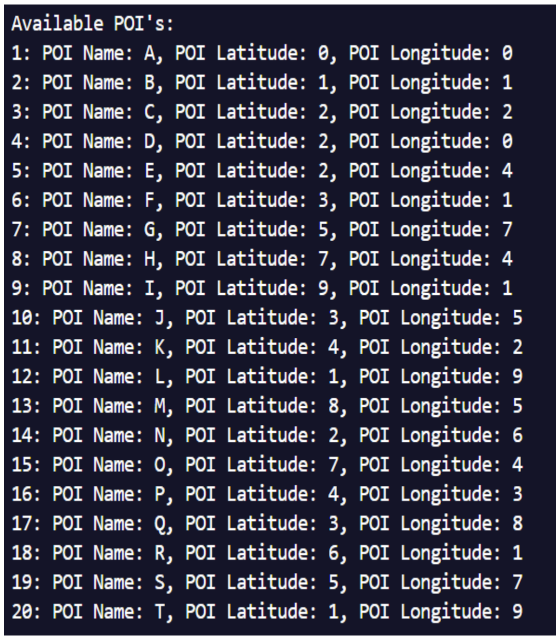
```
Floyd
  1. D ← W   // initialize D array to W[ ]
  2. P ← 0    // initialize P array to [0]
  3. for k ← 1 to n
  4.      do for i ← 1 to n
  5.          do for j ← 1 to n
  6.               if (D[ i, j ] > D[ i, k ] + D[ k, j ] )
  7.               then {D[ i, j ] ← D[ i, k ] + D[ k, j ]
  8.                       P[ i, j ] ← k; }
```

In our case, the n x n adjacency matrix we constructed is the D table, and we omit the P table because we only need the distance cost between every pair, not the specific path of points this shortest path passes through. Floyd's Algorithm considers every pair of points i and j. For each i and j, we check if the path from i to j through node k is shorter then the currently known path of i and j. If so, we update the current distance between i and j to the distance from i to k + distance from k to j. After Floyd's Algorithm is finished, we do a final loop through the adjacency matrix and sum up all distance values. This yields the total distance/roads needed to connect all the vertices/POI. This total distance value is multiplied by the fixed cost of building the roads, which gives us the final cost value of building the roadways. Finally, we add the cost of buying the land to the cost of building the roads. If the total cost is <= budget, we update the best cost to the current cost, the best hull to current hull, and move on to the next subset until we have iterated through every subset .

The overall time complexity of the program is O(2^n), as it considered every possible subset of points from the set of points chosen by the builder. For each subset of points, we compute a Convex Hull, calculates area of said Convex Hull, and runs Floyd's algorithm. Convex Hull D&C has a time complexity of O(nlogn), however being that we are calculating every possible convex hull, this program only supports the ability to select approximately ~10-12 points at a time, unfortunately making the Convex Hull Algorithm closer to O(n^3) in this application. 2^10 = 1024, while 2^20 = 1048576. So as the number of points that the builder selects increases, the program takes exponentially longer. This means that the Optimal Development problem is in NP-Complete since it cannot be computed in polynomial time.

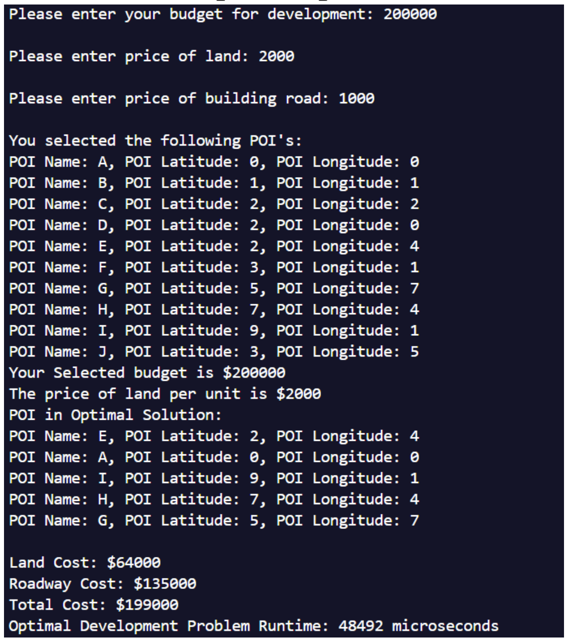
The fully implemented program successfully solves the optimization problem, based off of input from the builder (user) via the terminal, computing the area of the convex hull along with the most efficient roadway network. Our program seamlessly combines the Divide and Conquer Convex Hull algorithm, with Gauss's Area formula to calculate the area, along with Floyd's Algorithm to compute the most efficient roadway network. Each of these elements apply sophisticated mathematical concepts taught in CS375, such as how to determine orientation of a line, or how to determine the hull's boundary in the brute force base cases.

**Example Set of POI:**



```
Available POI's:
1: POI Name: A, POI Latitude: 0, POI Longitude: 0
2: POI Name: B, POI Latitude: 1, POI Longitude: 1
3: POI Name: C, POI Latitude: 2, POI Longitude: 2
4: POI Name: D, POI Latitude: 2, POI Longitude: 0
5: POI Name: E, POI Latitude: 2, POI Longitude: 4
6: POI Name: F, POI Latitude: 3, POI Longitude: 1
7: POI Name: G, POI Latitude: 5, POI Longitude: 7
8: POI Name: H, POI Latitude: 7, POI Longitude: 4
9: POI Name: I, POI Latitude: 9, POI Longitude: 1
10: POI Name: J, POI Latitude: 3, POI Longitude: 5
11: POI Name: K, POI Latitude: 4, POI Longitude: 2
12: POI Name: L, POI Latitude: 1, POI Longitude: 9
13: POI Name: M, POI Latitude: 8, POI Longitude: 5
14: POI Name: N, POI Latitude: 2, POI Longitude: 6
15: POI Name: O, POI Latitude: 7, POI Longitude: 4
16: POI Name: P, POI Latitude: 4, POI Longitude: 3
17: POI Name: Q, POI Latitude: 3, POI Longitude: 8
18: POI Name: R, POI Latitude: 6, POI Longitude: 1
19: POI Name: S, POI Latitude: 5, POI Longitude: 7
20: POI Name: T, POI Latitude: 1, POI Longitude: 9
```

**Example Output:**



```
Please enter your budget for development: 200000

Please enter price of land: 2000

Please enter price of building road: 1000

You selected the following POI's:
POI Name: A, POI Latitude: 0, POI Longitude: 0
POI Name: B, POI Latitude: 1, POI Longitude: 1
POI Name: C, POI Latitude: 2, POI Longitude: 2
POI Name: D, POI Latitude: 2, POI Longitude: 0
POI Name: E, POI Latitude: 2, POI Longitude: 4
POI Name: F, POI Latitude: 3, POI Longitude: 1
POI Name: G, POI Latitude: 5, POI Longitude: 7
POI Name: H, POI Latitude: 7, POI Longitude: 4
POI Name: I, POI Latitude: 9, POI Longitude: 1
POI Name: J, POI Latitude: 3, POI Longitude: 5
Your Selected budget is $200000
The price of land per unit is $2000
POI in Optimal Solution:
POI Name: E, POI Latitude: 2, POI Longitude: 4
POI Name: A, POI Latitude: 0, POI Longitude: 0
POI Name: I, POI Latitude: 9, POI Longitude: 1
POI Name: H, POI Latitude: 7, POI Longitude: 4
POI Name: G, POI Latitude: 5, POI Longitude: 7

Land Cost: $64000
Roadway Cost: $135000
Total Cost: $199000
Optimal Development Problem Runtime: 48492 microseconds
```

The image on the left shows the menu that prompts the builder to choose points of interest from a list. This menu also allows the builder to enter his budget, the costs of developing the land, and the cost of developing the roads. On the right are the results of running the program, which outputs the budget, price of land, and price of building road that the builder (user) enters. It shows selected points of interest, then it shows the most optimal solution, along with the land cost, roadway cost, and total cost. Note that for this entry, **99.5%** of the total budget was used. This entry ran in 0.048492 seconds, for 10 selected points with 5 of them in the optimal solution.