Optimization

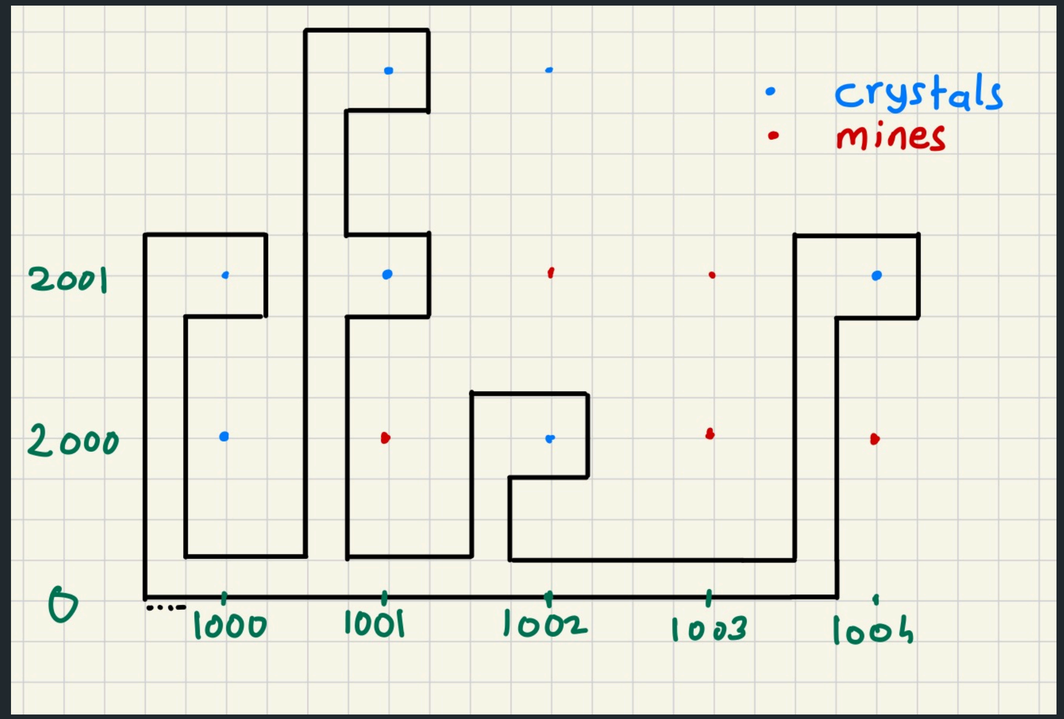
Ideation: Brute Force

- We took highest 166 crystals and tried to connect them with a ‘pipeline’ on the x axis
- Pipeline: A long rectangle of very small width containing no points inside.

Using our pipeline, we require 6 vertices for each crystal

Given max 1000 vertices,

1000/6 crystals = 166 crystals

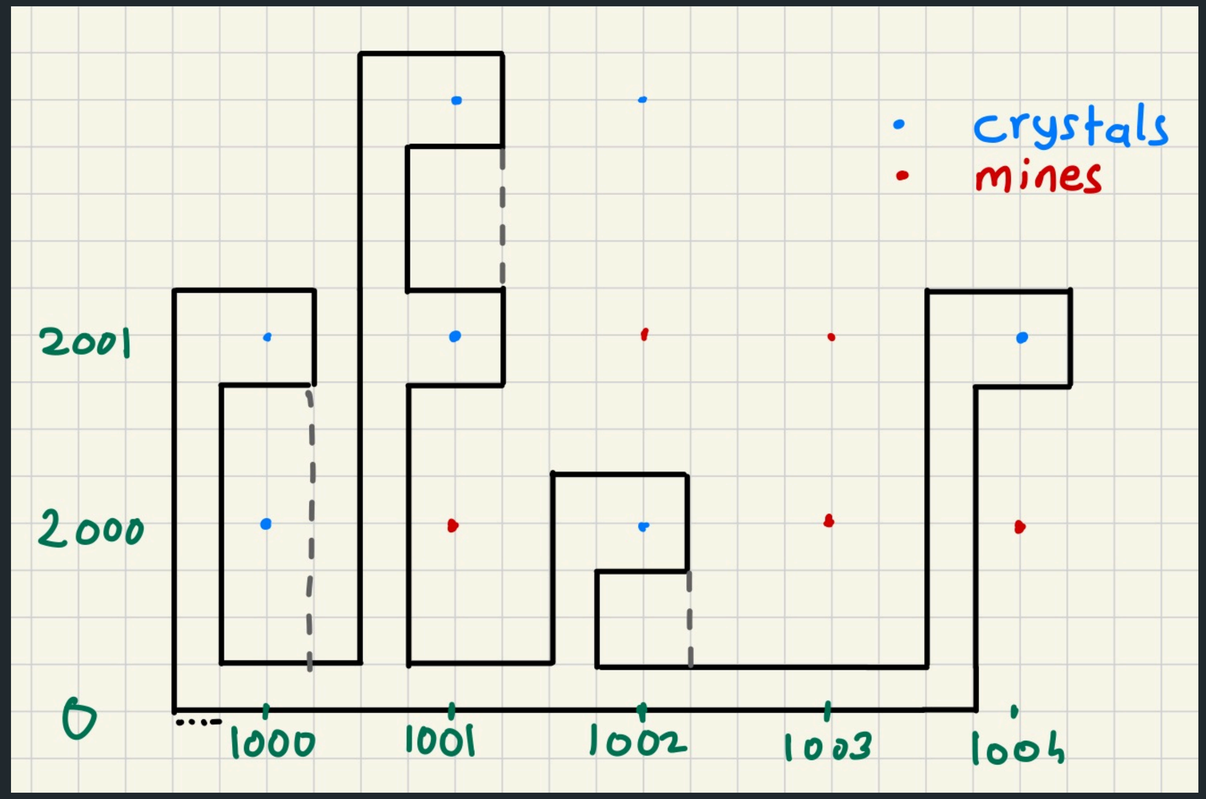


Pipeline Optimization

This Brute Force is not ideal most of the times and leads to a lot of wastage of vertices when we try to be careful.

Hence, we check if it is the last point on the given x, if yes, then we can connect it with only 4 vertices , saving 2 vertices in the process.

Then, we can use the best 166 + (vertices saved)/6 crystals and get an optimal solution.

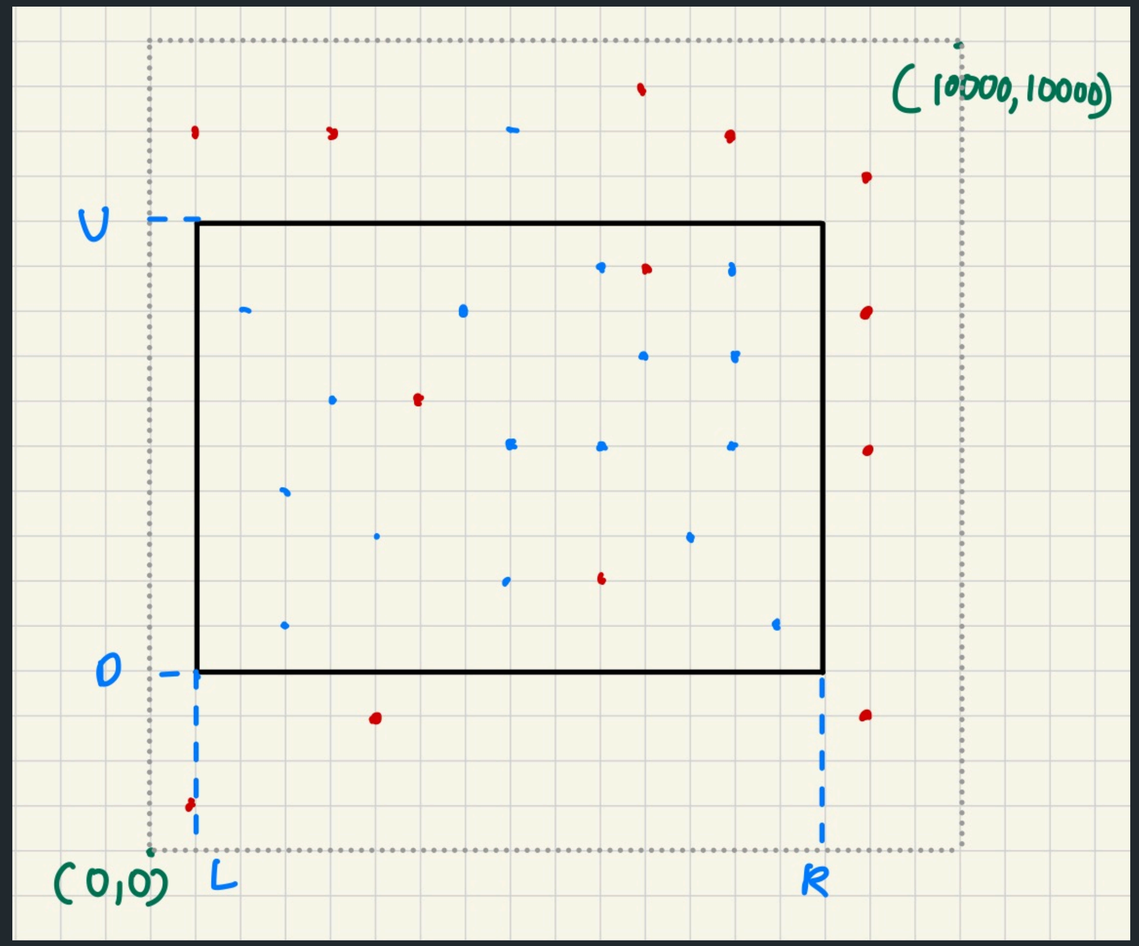


Ideation: Rectangle

This approach consists of finding the best rectangle containing the highest value of points in a given grid.

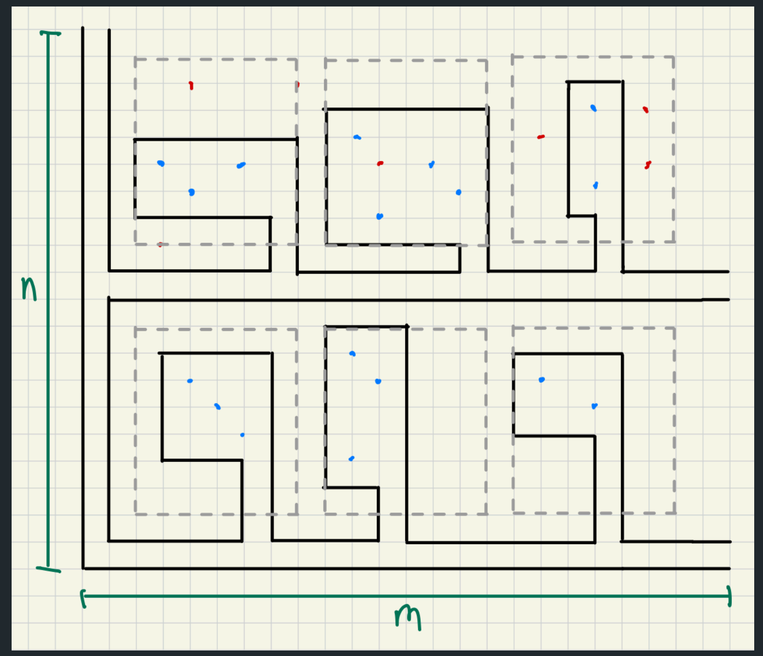
We use only 4 vertices in this approach.

Since we can’t iterate through every single x and y, twice, in our grid we limit the number of x and y by choosing them in an interval so that our total number of x and y are less than a specific value - MAX_CAND.



Ideation: Divide and Conquer

- Since we are allowed to use 1000 vertices we can divide our grid into smaller grids and find the best rectangle in each one of them.
- This will take 4n + 6mn vertices using our pipeline approach. (We use n = m = 10)
- Time complexity = $\sum a_i * c_i^4$ where a_i = number of points in the sub grid and c_i = MAX_CAND of that grid.
- We notice that increasing MAX_CAND increases our total score but takes more time up to a certain limit.



2D Prefix Sum Optimization

We were iterating through every single point inside the sub grid to calculate our score for the given possible rectangle. However, using 2D prefix sum we can preprocess it.

```
for(auto &p:allPoints){
    if(p.second.x>R)break;
    if(p.second.x >= L && p.second.y >= B && p.second.y <= T)currans += p.first;
}
```

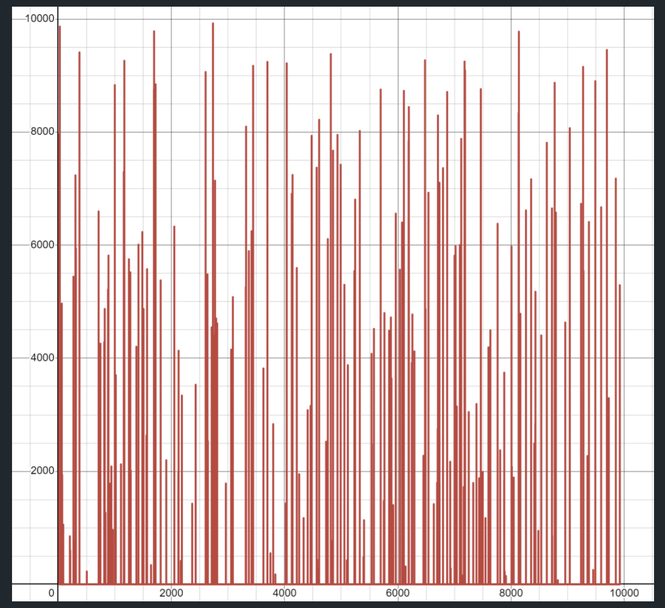
becomes

```
ll currans = pre[(R%4000)+1][(T%4000)+1]-pre[(L%4000)][(T%4000)+1]-pre[(R%4000)+1][(B%4000)]+pre[L%4000][B%4000];
```

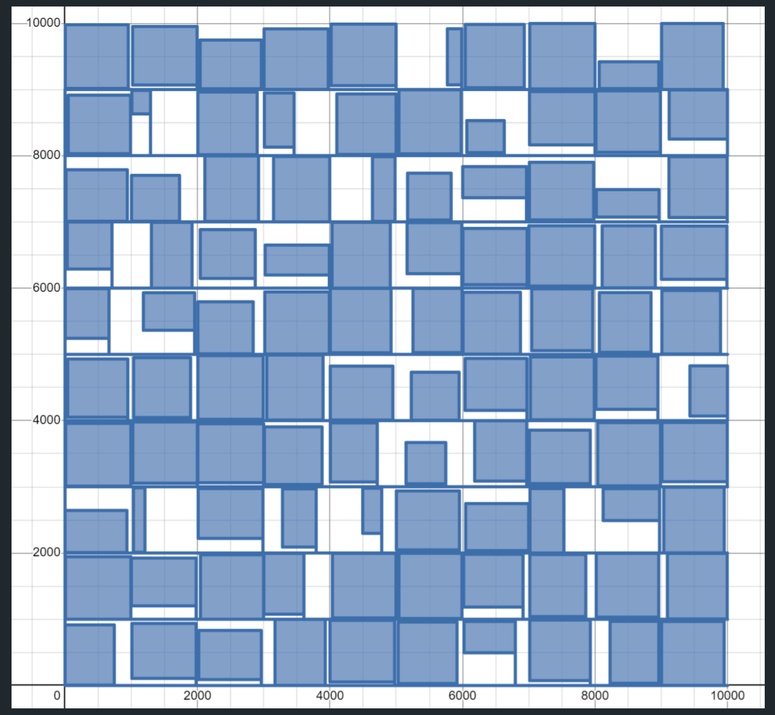
Testing

- To test our outputs we created a testcase generator and an output tester.
- Our tester checks if the score is correct and if the polygon is being formed correctly with all of the edges connecting on endpoints
- Our tester code also generates an output which you can use to plot in Desmos to help us visualize our output.

Optimized Pipeline :



Divide and Conquer:



Scalability

- Our Brute Force approach scales only based of maximum number of vertices and does good when there are less crystals.
- Our divide and conquer approach can be scaled based of vertices and time both. It does good when all the crystals are clustered together with less mines near them.
- We have divided our grid into 100 1000x1000 sub grids which only consumes 640 vertices out of total allowed 1000 vertices.

Thank You