Optimisation of Architectural Layout of an Office/Workplace

Group Members:
Luv Singh 20AR10019
Little John Saroniya 20AR10018
Nidam Kumar Jha 20AR10023

Problem Statement

Problem Description-

Optimisation of Office/Workplace layout in reference to area, feasibility, climate and topology i.e., creating a layout which when provided with the data of the given location, provides set of solutions as of how the layout of a workplace at that location should be designed.

Mathematical Interpretation-

A given location comes with sets of data of climate and micro-climate of that place, further the government provided and topological setbacks. Taking in all the data provides us with set of constraints that one has to keep in mind while designing the layout of a workplace in order to minimise problems and maximise comfort. The placement of shades, ventilation, windows, corridors, cabins, cubicles, washrooms, etc.., must be done accordingly.

Input: Latitude and Longitude, Annual Rainfall Data, Annual Cloud Coverage Data, Annual wind-rose data, altitude, Annual Temperature Data, nearest water body distance direction information, Available Area, Boundary setback according to the govern. Bye laws, employee employer information, information about the neighbouring infrastructure.

Output: Set of solutions regarding direction of window, ventilation, shades, amount of manual ventilation required for sustainable energy usage, no. of cubicles and cabins on a floor for optimised space usage and their respective placement.

The process would be based on prioritisation, therefore basically involving set of preference constraints while deciding the location of windows, ventilation and wall thickness. Also, no. of cubicles and cabins that could be fit on a floor would have to fit according to the employee-employee supervisor ratio on each floor.

AI Mapping & Approach

Constraint Satisfaction Problem.

1. Window Placement:

Priority 1(P1): Sun path- determined by latitude information. Depending being on the northern or the southern hemisphere, the yearly sun path would be mostly around most of the southern and northern regions of the sky respectively.
The window should majorly be placed on the opposite side to the sun path.

Constraint 1.1: The P1 region is blocked by a building or an infrastructure and hence could not be used for placing the windows.

Priority 2(P2): The sunrise side i.e., the east side would have to become the majorly glass façade wall. This is because the heat turns sharp later in the day and hence won't west won't really be the best option.

Constraint 1.2: Presence of a water body/coastal area, providing scenery and breeze.

Just as constrain 1.2 comes in, P2 changes. Hence, we'll need to assign cost values to constraint 1.1 and 1.2 in order to decide the best direction to place the windows.

Constraint 1.3: Prevailing wind direction. Although this factor won't be affecting across major part of the year, it still carries some weight, as windows provide for natural ventilation.

Domain: north, east, south, west
Variable: WP1, WP2 (for top 2 preferrable directions)
Constraint: ((WP1, WP2), C1, C2, C3, WP1 not equal to WP2)

2. Inlets for Natural Ventilation:

P1: Prevailing wind direction across the year.
P2: Presence of water body, as in due to breeze.

Constraint 2.1: An inlet for natural ventilation to occur could not be provided as it would let the noise enter as well if there is a presence of major roads/factory/etc right outside the wall.

If sufficient natural ventilation could somehow be possible by making inlets on less noisy sides of walls, energy could be saved on providing artificial ventilation. And if not, then the requirement would become heavy.

Also, a binary constraint relating points 1. and 2. to see if we could accommodate the window and ventilation both in one

3. Area Optimisation:

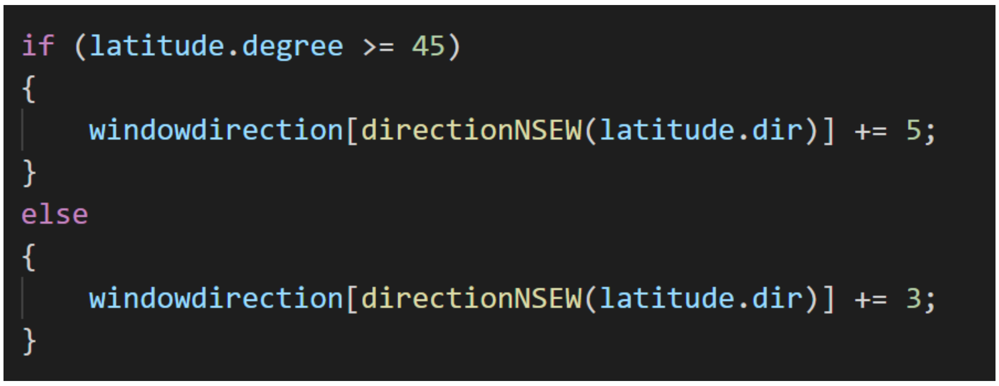
Given the Supervisor : Employee ratio, will give us the cabin : cubicle ratio for the floor. Putting in the value of the optimal area for a single cabin and cubicle, we can find how much we can accommodate on a single floor.

Constraint 3.1: The time period of an average employee working plus the stress the occupation carries, when set in obvious priorities of the combinations formed will help us monitor the % free area we'd require to provide on each floor as for the spacious environment helps perform better in more stress, plus more no. of entertainment areas would be required on floor to function as stress reliever.

Programming Language- C++

Solution, Logic and Code

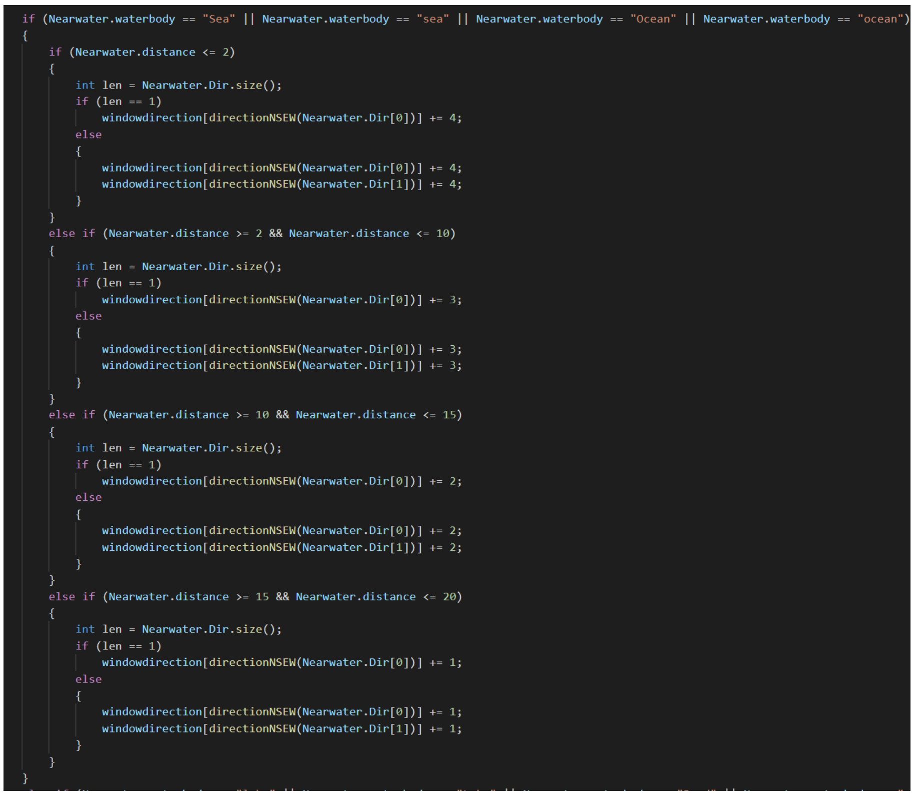
For window placement, 3 factors were taken to consideration, the sun-path (from the latitude information), sea breeze, and prevailing wind direction.



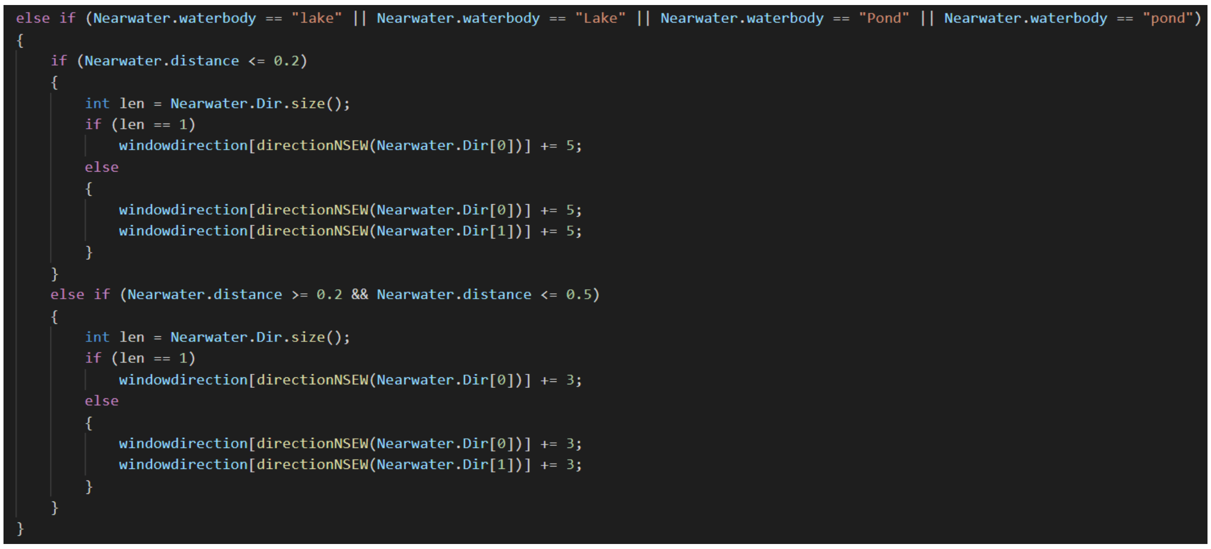
```cpp
if (latitude.degree >= 45)
{
    windowdirection[directionNSEW(latitude.dir)] += 5;
}
else
{
    windowdirection[directionNSEW(latitude.dir)] += 3;
}
```

Different weights have been given to different factors as per what would affect the placement more. In the end, the values for each direction are added up, and we get the preference order for the best direction to place windows (glass façades).

Below is the preference order in preference to breeze that a building would get depending on the type of water body and its distance from the building.

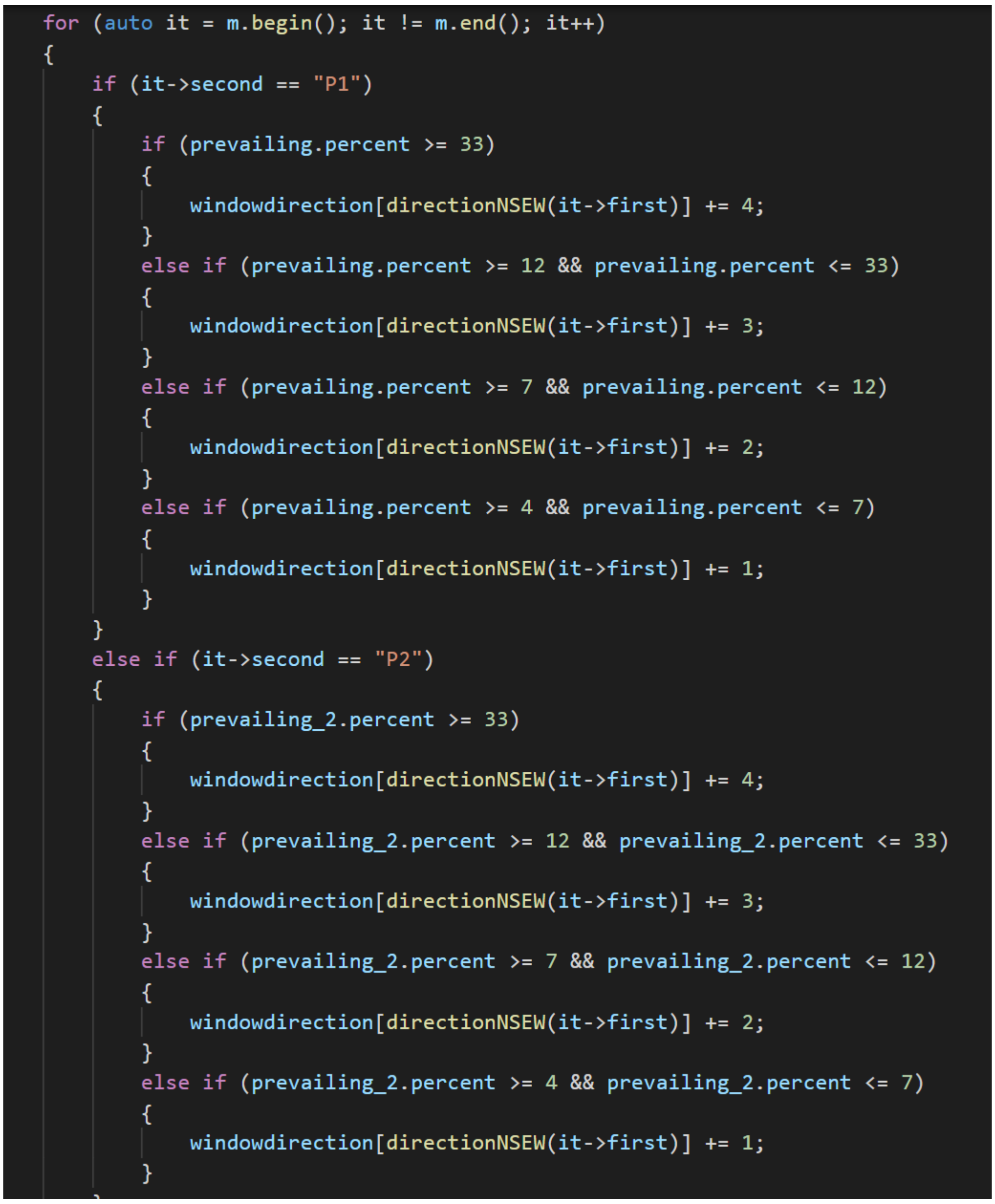


```
if (Nearwater.waterbody == "Sea" || Nearwater.waterbody == "sea" || Nearwater.waterbody == "Ocean" || Nearwater.waterbody == "ocean")
{
    if (Nearwater.distance <= 2)
    {
        int len = Nearwater.Dir.size();
        if (len == 1)
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 4;
        else
        {
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 4;
            windowdirection[directionNSEW(Nearwater.Dir[1])] += 4;
        }
    }
    else if (Nearwater.distance >= 2 && Nearwater.distance <= 10)
    {
        int len = Nearwater.Dir.size();
        if (len == 1)
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 3;
        else
        {
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 3;
            windowdirection[directionNSEW(Nearwater.Dir[1])] += 3;
        }
    }
    else if (Nearwater.distance >= 10 && Nearwater.distance <= 15)
    {
        int len = Nearwater.Dir.size();
        if (len == 1)
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 2;
        else
        {
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 2;
            windowdirection[directionNSEW(Nearwater.Dir[1])] += 2;
        }
    }
    else if (Nearwater.distance >= 15 && Nearwater.distance <= 20)
    {
        int len = Nearwater.Dir.size();
        if (len == 1)
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 1;
        else
        {
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 1;
            windowdirection[directionNSEW(Nearwater.Dir[1])] += 1;
        }
    }
}
```



```
else if (Nearwater.waterbody == "lake" || Nearwater.waterbody == "Lake" || Nearwater.waterbody == "Pond" || Nearwater.waterbody == "pond")
{
    if (Nearwater.distance <= 0.2)
    {
        int len = Nearwater.Dir.size();
        if (len == 1)
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 5;
        else
        {
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 5;
            windowdirection[directionNSEW(Nearwater.Dir[1])] += 5;
        }
    }
    else if (Nearwater.distance >= 0.2 && Nearwater.distance <= 0.5)
    {
        int len = Nearwater.Dir.size();
        if (len == 1)
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 3;
        else
        {
            windowdirection[directionNSEW(Nearwater.Dir[0])] += 3;
            windowdirection[directionNSEW(Nearwater.Dir[1])] += 3;
        }
    }
}
```

The weightage value is written in each conditional statement. And now below is the weightage as per the prevailing wind direction.



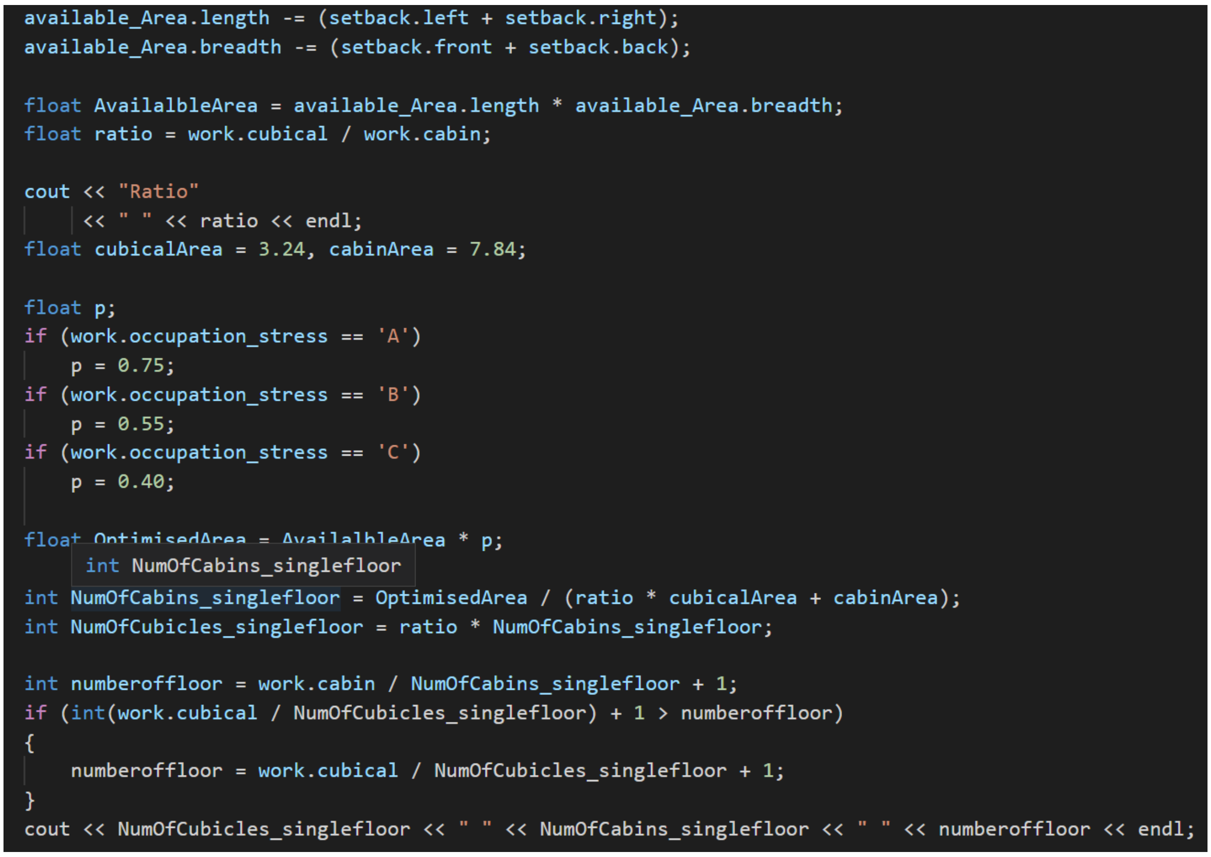
```
for (auto it = m.begin(); it != m.end(); it++)
{
    if (it->second == "P1")
    {
        if (prevailing.percent >= 33)
        {
            windowdirection[directionNSEW(it->first)] += 4;
        }
        else if (prevailing.percent >= 12 && prevailing.percent <= 33)
        {
            windowdirection[directionNSEW(it->first)] += 3;
        }
        else if (prevailing.percent >= 7 && prevailing.percent <= 12)
        {
            windowdirection[directionNSEW(it->first)] += 2;
        }
        else if (prevailing.percent >= 4 && prevailing.percent <= 7)
        {
            windowdirection[directionNSEW(it->first)] += 1;
        }
    }
    else if (it->second == "P2")
    {
        if (prevailing_2.percent >= 33)
        {
            windowdirection[directionNSEW(it->first)] += 4;
        }
        else if (prevailing_2.percent >= 12 && prevailing_2.percent <= 33)
        {
            windowdirection[directionNSEW(it->first)] += 3;
        }
        else if (prevailing_2.percent >= 7 && prevailing_2.percent <= 12)
        {
            windowdirection[directionNSEW(it->first)] += 2;
        }
        else if (prevailing_2.percent >= 4 && prevailing_2.percent <= 7)
        {
            windowdirection[directionNSEW(it->first)] += 1;
        }
```

For ventilation, only the sun-path factor is removed and the rest stays the same. Hence again preferential CSP.

Further, if the order of window and ventilation comes out to be the same, some intervals in glass facades only can be used as in for natural ventilation.

Road, and infrastructure information is also taken in as input, this information could majorly influence the preferential direction and would directly nullify one whole direction, if blocked by a major infrastructure on one side.

Under Area optimisation, the constraint is the job type, that'd affect the spacing inside the workplace.



```
available_Area.length -= (setback.left + setback.right);
available_Area.breadth -= (setback.front + setback.back);

float AvailalbleArea = available_Area.length * available_Area.breadth;
float ratio = work.cubical / work.cabin;

cout << "Ratio"
        << " " << ratio << endl;
float cubicalArea = 3.24, cabinArea = 7.84;

float p;
if (work.occupation_stress == 'A')
    p = 0.75;
if (work.occupation_stress == 'B')
    p = 0.55;
if (work.occupation_stress == 'C')
    p = 0.40;

float OptimisedArea = AvailalbleArea * p;
     int NumOfCabins_singlefloor
int NumOfCabins_singlefloor = OptimisedArea / (ratio * cubicalArea + cabinArea);
int NumOfCubicles_singlefloor = ratio * NumOfCabins_singlefloor;

int numberoffloor = work.cabin / NumOfCabins_singlefloor + 1;
if (int(work.cubical / NumOfCubicles_singlefloor) + 1 > numberoffloor)
{
    numberoffloor = work.cubical / NumOfCubicles_singlefloor + 1;
}
cout << NumOfCubicles_singlefloor << " " << NumOfCabins_singlefloor << " " << numberoffloor << endl;
```