RESEARCH STATEMENT (CIRCUIT COMPACTION)

Aditya Chauhan chauhanaditya3628@gmail.com

INTRODUCTION

'*Building Wiring Problem*' could have wide range of application such as minimizing the length of circuit wires in a VLSI circuit board, electrical wiring of building floor planning, Total Wiring Length Minimization of Neural Network [1] etc.

Consider performing one-dimensional minimization of wires along the horizontal direction or vertical direction. As long as constraints are not violated, the layout elements can be allowed to slide horizontally or vertically [2] . However, we are dealing with two dimensional graph that needs to be minimized in both X and Y directions, leading to a tree in a graph reaching every sub-rectangle with minimum value. Since this novel problem does not have a well defined solution, the analysis of this problem is done based on existing techniques and graph algorithms. From the analysis, we have devised a model and proved the hardness of the problem with necessary proof and have shown the possible algorithms to solve this problem. Problem transformation method is used to convert this new problem into existing problems.

==
==

METHODOLOGY

There are four different algorithm proposed from approximation to optimal with polynomial time complexity.

A. GREEDY APPROACH

```
R = {r1, r2 ,r3 ,r4 , ......,rn}        // Set of sub Rectangles
V = {ra,rb,rc, .....}                   // sub rectangles which are already visited

LL = (xi,yii)                           // current lower left point in the set of all the sub rectangles


Answer = {e1,e2,....ep}                 // selected edges for the final answer


While(V doesn't include every sub rectangle in R){
        Select LL and find the closest point;
        Add this edge into Answer;
        If(no next unvisited point is found) // cycle/loop found in the traversal
                /* cycle/loop found in the traversal*/
        {
                Remove all the visited rectangles from the set R;
                Update the value of llbreak;
                Start the second traversal with new LL value;
        }
}
Answer contains our solution for the problem
```

Algorithm

1. Rectangle Set R containing all the sub-rectangles $r_1, r_2, r_3 \ldots . r_n$ is given
2. Start from a particular fixed point every time(left bottom in this case) and select the next point which is closest to that point.
3. Mark all the sub rectangles which are connected with this edge as visited.
4. When a situation comes when we cannot select any other points and we still haven't visited all the sub rectangles ie the traversal forms a loop in the graph, remove the all the visited rectangles so far and go back to step 2.
5. Once we visit every sub rectangles in the set, stop.

Time Complexity : **O(E Log V)**

==

B. MINIMUM SPANNING TREE

```
R = {r1, r2 ,r3 ,r4 , ......,rn}        // Set of sub Rectangles
V = {ra,rb,rc, ......}                  // sub rectangles which are already visited

E= {all the edges of the sub rectangles}

Answer = {e1,e2,....ep}                 // selected edges for the final answer

Sort (set E by their length);


While(V doesn't include every sub rectangle in R){
    select smallest edge ei from E;
    If(all the connected rectangles with edge ei are visited){
        neglect the edge ei and select the next smallest edge;
    }
    Else{
        Mark all the rect. conncetd to this edge as visited and inlude in V.
        Add edge ei in the answer set

    }

}
Answer Set contains all the selected edges.

If(any edge is not touching the surface of outer rectangle){
    Get smallest possible edges from set E to extend the final Answer to the
    surface of the Outer rectangle;

}
Now the Answer set has the final solution of the problem.
```

Algorithm

1. Given a set R of sub rectangles $r_1, r_2, r_3 \ldots r_n$, sort their edges in ascending order of their length
2. Pick the smallest edge from the set and include it in the final solution.
3. Mark all the sub rectangles connected with that edge as visited and then select the next smallest edge from the set.
4. if all the rectangles connected with a selected edge are visited already, neglect that and don't include that in the final solution .
5. Once all the sub rectangles are marked as visited we will have a set of disjoint edges.
6. Connect them using the smallest edges possible.
7. At the end if the solution don't touch the surface of the outer rectangle, select edge(s) to connect the solution to the outer surface.

Time Complexity : **O(E log V)**

==

C. VARIANT OF SET COVER THEORY

```
I = {Xi,Yi} ;
/*set containing all the verticies of bigger rectangle {(X1,Y1),(X2,Y2),(X1,Y2),(X2,Y1)}*/

U = {r1,r2,.....,rn} ;        //set conatining all the samller rectangles
r1 = {Xj1,Yj1} ;              // set containing all the verticies of smaller recatngle one
r2 = {Xj2,Yj2} ;              // set containing all the verticies of smaller recatngle two
.                             .
.                             .
.                             .
.                             .
rjn= {Xjn,Yjn} ;              // set containing all the verticies of smaller recatngle n

Vr = { } ;
/* set containing the final output set  of smaller recatngle in order of preference initialised to be
empty*/

W(rj) = perimeter(rj) ;   // W represents the weight  so W(rj) denotes the weight of rect.
L = Yi-1 – Yi ;           // L denotes the vertical length of rectangle
If  L is less than 10{
    N = L ;               // N denotes no. of sets used to implement set cover algorithm
}
Else If  L is greater than 10^x {        // x is an  integer that is greater than or equla to 2
    N = ceilf(L/10^(x-1)) ;
/* ceilf is a function in c used to convert float to its upper bound integer*/
}
/* All N sets formed are of almost same vertical length and same horizontal length
 In example figue these sets are of 10 unit vertical height each and are represented by red line
partitions.
*/
Now divide the big rectangle in smaller sets according to above rules and mark
there Y coordinates value. Name these sets(Sy) :  S1, S2 ,...., SN  , all these sets are
empty sets.
/* Yz represents the value of Y coordinates for different sets*/
```

Time Complexity : **O(nN)+O(m log*n)**

```
/* =====================Start from top of rectangle======================*/
int y = 1;
While N is not 0{
    For  (int a = 2 ; a ≤ N + 1; a + + ){
        For  all the rectangles(rj) in set U{
            If  Y_z(a−1) ≤ Y_j−1 < Y_za {
                Sy = { rj};
            }
        }
    }
}
Now N sets are formed each containing rectangles that are partially or
completely part of that set according to constraints applied.
W(Sy) = ∑W(rj) ;        // weight of each set is the sum of weights of all the rect. in that set
C(Sy) = W(Sy)/(number of elements in set Sy);      // cover value for each set.
While Vr is not equal to U{
    LOOP : C(Sy) = W(Sy)/(number of elements in set Sy); // cover value for each set.
        Take Si with min(C(Si)) and union with Vr set;
        Vr = Vr ∪ Si ;
        Plot the path that covers all the rectangles via edge or just vertex such
        That all the rect. are covered with minimum length(i.e take a path that is
         Going through inside part not touching outer rectangle);
        Remove set Si and also elements of Si from remaining N-1 sets.
        If  Vr is equal to U
            break;
        Else
            goto  LOOP;
}
FINALLY :
If output path touches bigger rectangle
    END;
Else
   Connect one of the point on path with shortest possible and available line to
bigger rectangle;

/* ===============================END===============================*/
```

Algorithm

1. Input set U is set of all the smaller rectangles in the larger rectangle.
   - $U = \{r_1, r_2, r_3 \dots r_n$ I r represent rectangles and n represents number of rectangles}
2. Weight of rectangle in set U is a perimeter of that rectangle defined by set W.
   - $W = \{w_1, w_2, w_3 \dots w_n$ I wi = perimeter(ri)}
3. Initialise solution set to be F = {}
4. Now divide the bigger rectangle in multiple horizontal strips of equal dimension.
5. Each strip represent on set and elements of set are all the rectangles in that set.
6. Let $S_1, S_2, S_3 \dots S_k$ represent all the strips of sets and weight of each set is sum of weights of rectangles in that set.
7. W(Si) = w_j {j represent rectangles in S_i}
8. Find cover of all the sets(S)
   - $C(S_i) = W(S_i)$/No. of element in S_i
9. Take a set with $\min(C(S_i))$ and union that set with F.
   - F = F U S_i
10. Remove S_i and also the elements of S_i from remaining sets but the value of W(S) is still the same for all the sets.
11. Start making a connection based on the current F set such that all the rectangles are at least touched at one vertex and connection inside the big rectangle not touching any of its edges.
12. Repeat steps 8 to 11 until all the rectangles are covered.
13. If the resulting connection does not touches outer rectangle connect one of the free ends with the outer rectangle with the shortest distance in our grid.

==

D. DIVIDE AND CONQUER

```
I = {X_i,Y_i} ;
/*set containing all the verticies of bigger rectangle {(X1,Y1),(X2,Y2),(X1,Y2),(X2,Y1)}*/

U = {r_1,r_2,.....,r_n} ;          //set conatining all the samller rectangles

/* set containing all the verticies of smaller recatngle one . Second representation shows that
each reactangle comprised of 4 vertices so P is representing those verticies for each rectangle*/
r_1 = {X_j1,Y_j1} ; = {P_j11, P_j12, P_j13, P_j14};
r_2 = {X_j2,Y_j2} ; = { P_j21, P_j22, P_j23, P_j24};
.
.
.
.
r_jn= {X_jn,Y_jn} ; = {P_jn1, P_jn2, P_jn3, P_jn4};

V_r = { } ;
/* set containing the final output set  of smaller recatngle in order of preference initialised to be
empty*/

P = {P_k1, P_k2,P_k3, P_k4}  //this set represents collection of all the points in given input
/* k goes from 1 to n so all the points for all n smaller rectangles are contained in set P*/

/* Now create a set of points that have common X coordinate*/
For a from 1 to n{
        For b from 2 to n{
                If X(P_a) is equla to X(P_b) && Y(P_a) is not equal to Y(P_b){
                        S(P_a) = {P_b};
                }
        }
}

/*count number of elements of each set created in above iterartion */
Count(S(P_i)) = number of elements in set S((P_i))
P_f = { }        //P_f is a final set containing all the points used to make path of shortest length

While V_r is not equla to U{
        LOOP : Select set(S(P_i)) with max(Count(S(P_i)));
        For all elements in S(P_i){
                Select point common to max number of rectangles.
                P_f = P_f ∪ {P_i};
                V_r = V_r ∪ {r_i}; // put all the rectangles that are common to that point in V_r
                Select point closest to selected before and covers remaining rect.;
                P_f = P_f ∪ {P_i};
                Continue until all rectangles having X(P_i) as a point are covered.
        }
        Remove set S(P_i) ;
        If V_r is equal to U
                break;
        Else
                goto LOOP;
}
Connect all the points in P_f according to available lines in input.

FINALLY :
If output path touches bigger rectangle
        END;
Else
        Connect one of the point on path with shortest possible and available line
to bigger rectangle;
```

Time Complexity : **$O(n^2)+O(m \log^* n)$**

Algorithm

1. Plot all the points in rectangular grid on coordinate plane and Initialise a input set(I) containing all those points.
   - $I = \{p_1, p_2, p_3 \dots p_n\}$
2. Select a vertical line(L) with maximum number of points.
   - $L = \max(\text{count}(p_i))$
3. Find minimum number of points on that line such that all the rectangles common to that line are covered and the distance between two adjacent points in minimum.
4. Put those points in final set (F)
   - $F = F \cup \{p_i\}$
5. Select next line with second max number of points repeat steps 3 and 4.
6. Continue until all the rectangles are covered.
7. Finally join all the points in set F with vertical and horizontal lines as represented in input grid.
8. If the final result does not touch any of the edges in outer rectangle than join the free end with one of the edges with shortest distance.

==
==

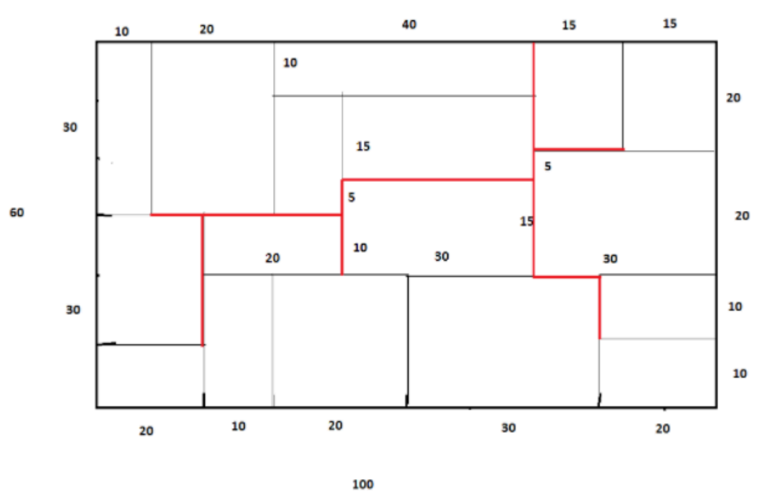
RESULT

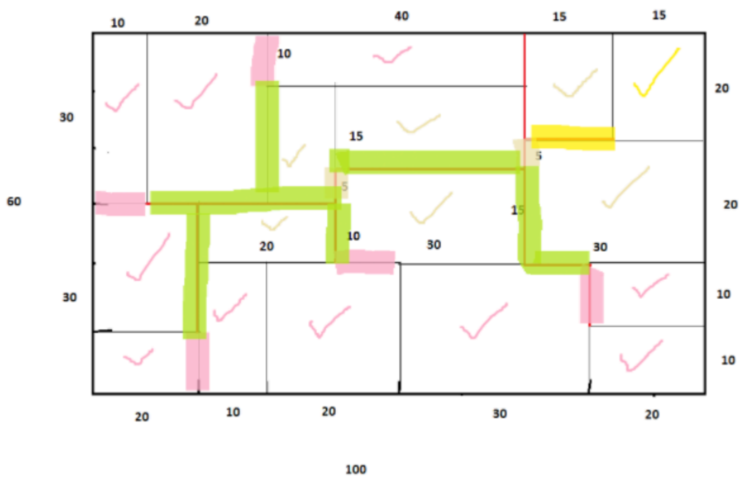
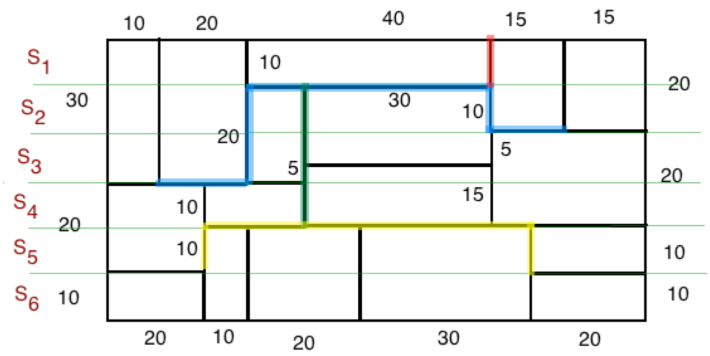
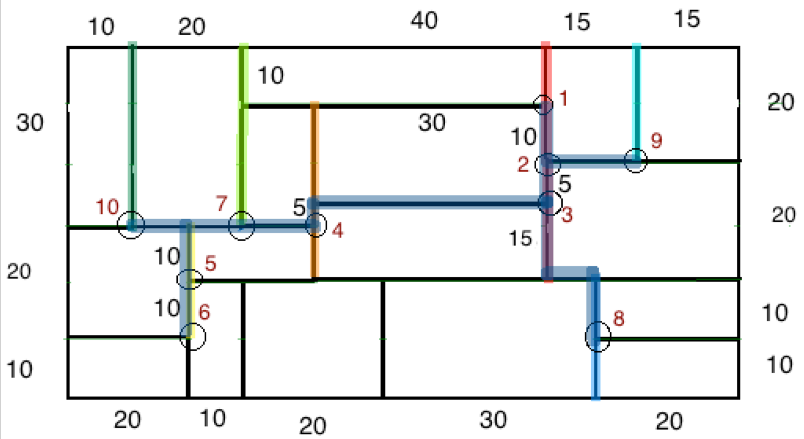
- Among all these suggested algorithms *Greedy Algorithm* works but gives the possible worst case result as instance grids get more complex.
- *MST* and *Variant of Set Cover* gives the average case results.
- *Set Cover* for some instance grids also gives optimal results.
- *Divide and Conquer* in each case gives the most optimal solution to this problem.

Below is one of the example instance grid and its resulting path and length as output for different algorithms suggested above

As suggested and expected divide and conquer approach gives the most optimal solution

Figure 1 : Test instance and its output

	**Greedy Algorithm**	**Total length = 375 units**

<table>
<tr>
<td>
</td>
<td>Minimum Spanning Tree Algorithm</td>
<td>Total length = 200 units</td>
</tr>
<tr>
<td>
</td>
<td>Variant of Set Cover Algorithm</td>
<td>Total length = 230 units</td>
</tr>
<tr>
<td>
</td>
<td>Divide and conquer Algorithm</td>
<td>Total length = 170 units</td>
</tr>
</table>

REFERANCES

[1] Gushchin A, T. A. (2015, December 14). Total Wiring Length Minimization of C. elegans Neural Network: A Constrained Optimization Approach. http://dx.doi.org/10.1371/journal.pone.0145029 , 12.

[2] Hambrusch, Susanne E. and Tu, Hung-Yi(1993). New Algorithms for Minimizing the Longest Wire Length During Circuit Compaction. Computer Science Technical Reports.
h p://docs.lib.purdue.edu/cstech/1031 Paper 1031

[3] R.A.Rutman (1964, April) . An Algorithm for Placement of Interconnected Elements based on minimum wire length. AC Spark Plug Division . General Motor Coprportion , El Sengundo, California. ACM Digital Library, 477-491.

[4] Ramachandran Varadarajan (1985, Dec). Algorithm for Circuit layout compaction of building blocks. Texas Tech University

[5] Sastry,S. , Parker,A (1982, Sep.). The complexity of two-Dimensional compaction of VLSI layouts. Proceedings of the International Conference on Circuits and Computers, 402-406.

[6] Schlag,M. , Liao,Y.Z. and Wong,C.K (1983). An algorithm for optimal two-dimensional compaction of VLSI layouts. Integration the VLSI Journal, 1 , 179-209.

[7] Yinghai Lu, Hai Zhou, Li Shang, Xuan Zeng (2010). Multicore Parallelization of Min-Cost Flow for CAD Applications. Computer-Aided Design of Integrated Circuits and Systems IEEE Transactions on, vol. 29, ISSN 0278-0070, 1546-1557.

[8] R.K Pal, S.P Pal , M.M.Das, A.Pal (1995, Jan.). Computing area and wire length efficient routes for channels. VLSI Design, Proceedings of the 8th International Conference, ISBN 1063-9667.

/ END ***/**