Linah Rashed 900191342
Fady Salib 900191909

Simulated annealing-based placer report
Digital Design 2

In this report, we will be discussing the simulated annealing and how we used it to build a simulated annealing-based placer that minimizes the total wirelength.

First, let's start by explaining simulated annealing. Simulated annealing is an algorithm by which we put a very high temperature and then add a cooling rate, so we would avoid being stuck at a local minimum at the beginning of calculating the wire length. We start by an initial random placement, then the choosing of a high temperature and a cooling rate, and then we start swapping 2 random cells. We have to calculate the change in wirelength between cells that resulted after the swap. If the total wirelength got smaller, we accept the swap, and if it got bigger, then we reject the swap. The cooling rate helps us get the best minimum total wire length after many swaps.

Simulated annealing is based on the greedy algorithm; however, it is better as the rejection probability of the swap is very low at the beginning and higher at the end. The high temperature in the simulated annealing is what controls the rejection probability. In the greedy algorithm we get trapped in a local minimum in the beginning which makes it sometimes accepts a bad swap.

Simulated annealing is the same idea of thermal annealing. The lowest energy state of a crystal lattice is when all atoms are lined up. If we have a crystal and we heated it up, atoms will keep moving around. After we cool it, we will find that the movement is now restricted.

Code Explanation:

1) First, we start by doing the parsing of the input files. We open the input file using the fstream to read the files. If the input file exists and is read, then the number of cells, connections, rows, and columns are read in order. After that, we did a for loop so we could get the number of connections (first column) and another for loop to get the components (the rest of the row) .

2) Then we have a class “Annealing_Placer” to be able to do the grid, or the board, by having the cells, number of connections, rows, and columns. The class has the function “initialize_random” to do the initial random placement of the cells. Every component is given a random ‘x’ and ‘y’ on the grid. We can only place the cell on the grid if the place is empty.

3) The next function is for calculating the total wirelength, we are calculating it by finding the maximum and minimum x and y. We also find the netlists which the cells are in and then compute the total wire length of it

4) We have another function for computing the new wirelength which has a for loop which loops over the elements of the vector. This vector contains all the netlists that contain the random cell in them. We also check that the cell is not an empty one.

5) We continue by a function for swapping the cells randomly. We first choose 2 random cells, and the 2 cells swap their locations on the grid. Their locations are temporary as we can still accept or reject the swap.

6) If the swap is rejected, we did a function to reverse the swap and each cell will go back to its first location.

7) In, the stimulated annealing function, we first initialize the random placement of cells and the temperature as given 500 x initial cost and the final temperature as $5x(10)^{-6}$ x initial cost/number of nets, etc.

```
void simulated_anealing(){
    initialize_random();
    initial_temperature = 500 * TWL;
    final_temperature = 5 * pow(10,-6) * TWL / conections;
    temperature = initial_temperature;
    moves = 10 * cells;
```

We then do the algorithm itself, and if the temperature is greater than the final temperature, we keep swapping the cells and if the new total wirelength is less than the

original total wire length then we take it as our TWL. Else, if the swap is rejected then reject with a probability of $1-e^{(\text{TWL-new TWL})/\text{temperature}}$. We also use the cool function with a

```
int new_TWL;
while (temperature > final_temperature) {
    for (int i=0; i<moves; i++) {
        swap_random_cells();
        new_TWL = total_wirelegth();
        if (new_TWL < TWL) {
            TWL = new_TWL;
        } else {

            double probability = 1 - exp( (TWL - new_TWL) /
                temperature );
            double result = (double)rand() / RAND_MAX;
            if (result > probability) {
                TWL = new_TWL;
            } else {
                reverse_swap();
            }
        }
    }
    cool();
}
```

rate of 0.95xtemperature.

8) Finally, we have the display function which displays the results as wanted. We also display it as binary numbers.

Assumptions:

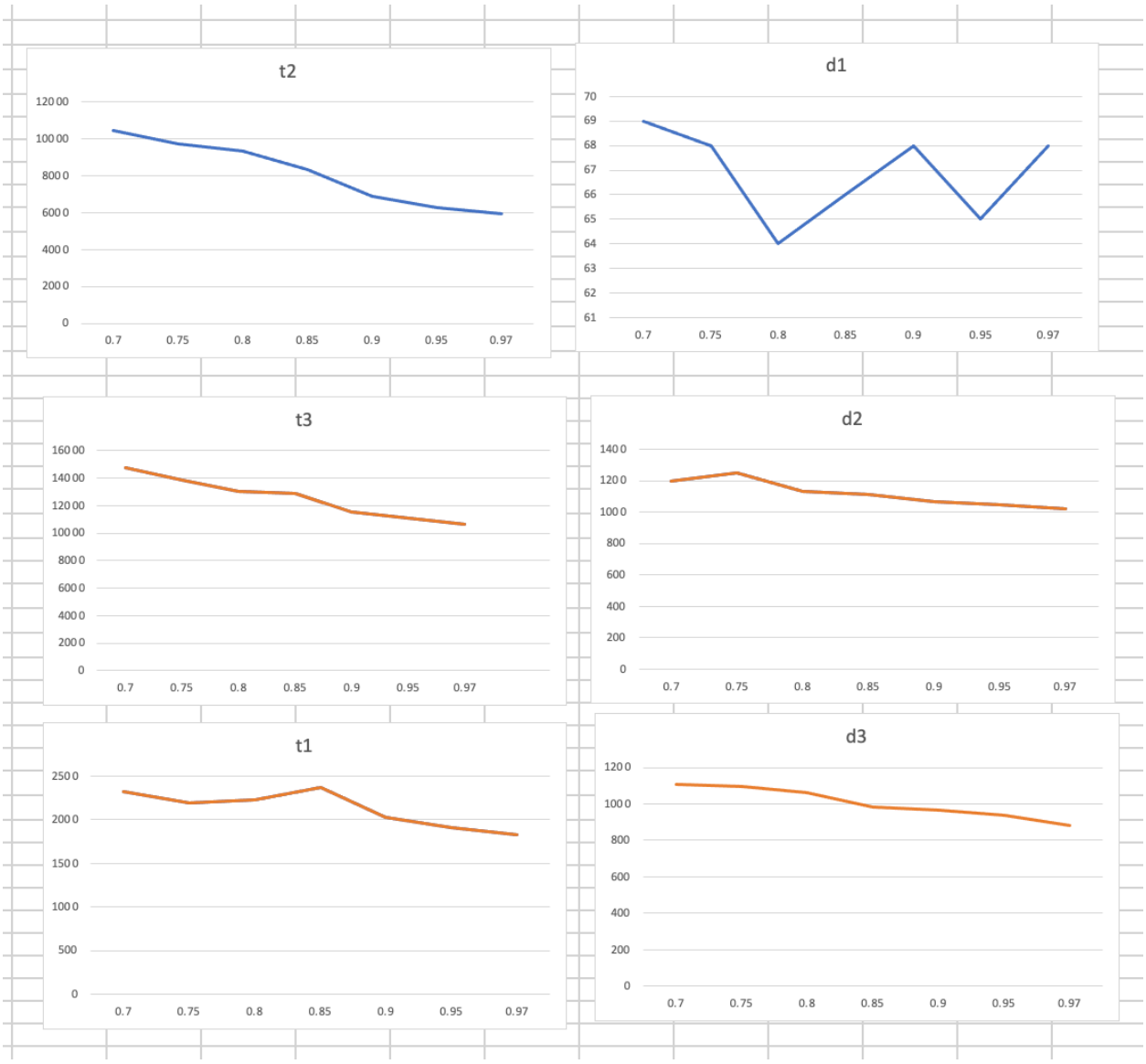
```
public:
    int cells, conections, rows, columns;
    int net_list[1290][210];
    int cell_location[1300][2];
    vector< int > cell_netlist[1300];
    int chip[70][70];
    int TWL;
    int net_wl[1290];
    int new_net_wl[1290];//
    int rand_chip_1[2], rand_chip_2[2];
    int rand_cell_1, rand_cell_2;
    int moves;
    double initial_temperature, final_temperature, temperature;
```

1- The net list is a 2D array of size of the biggest netlist we have

2- Cell location is also a 2D array with a maximum total number of cells
3- Our chip has a size of 70x70
4- The vector has in which netlist is every cell

Results:
Every time we run the code; the results are different as the simulated annealing algorithm does not find an optimal solution but a solution that is around the optimal one.

The graphs below all show the result of the algorithm. By increasing the cooling rate from 0.7to 0.97, the total wire length decreases.



These graphs show the effect of the temperature on the total wirelength. We can conclude from this that the TWL decreases as temperature decreases because it goes out of local minimum.