10/10/2024

Lab 04 Report

Ryan Beck
Jared Bronson

Objectives

Write a simulated annealing program in C or C++ that solves the FPGA placement problem. The program will ingest a text file with a grid size, number of vertices, and edges between two nodes. Using the values given in the text file, the program will place the edges on a grid. The nodes will then go through an annealing process to find a solution to the placement problem. When the program finds an acceptable solution, it will print to an output file.

Procedures

Create a C++ file with global variables for the initial temperature, cooling rate, and stop threshold. In the main function, take the given input file and read the parameters. In the input file, the type of input will be denoted by the first character on each line. The characters used will be g for grid size, v for vertices, and e for edges. Use the parameters given by the input file to make a properly sized grid, and place the given nodes at random coordinates on the grid. Do not allow a node to be placed in the same spot as another in the beginning.

After the nodes are placed, pass them into the anneal function. First, the anneal function will evaluate the initial score of the placement. Then, while the temperature is greater than the stop threshold, the next nodes are set to equal the current nodes. After changing the next node, it goes to be altered. In the alter function a random number from 0-2 for both the x and y positions. These random numbers will be used to change the x and y positions. If the number is two, the position will be moved one in the negative direction. The new positions are put into evaluation to check the new score. The old score and new score are then compared to determine if the new score will be accepted. If the old score is better, there is still a chance the new score will be accepted because it may lead to a better solution. Then temperature is reduced through the cooling function. This will repeat until the temperature is less than the stop threshold.

After the annealing process, the results need to be printed to an output file. The output file should show the new placement of each node, as well as the length of the edges between two nodes.

Statistical Analysis

The team ran a statistical analysis to assess the relationship between the cooling rate and both the execution time and solution quality. This analysis is not complete, as results were only collected from the first input.txt file due to time constraints. In addition, more executions at each cooling rate would also provide more accurate data.

The program was run with 5 different cooling rates, and the corresponding solutions tried and final score are recorded. The number of solutions tried is proportional to the execution time, as more loops through the anneal function are required. As a note, the INITIAL_TEMPERATURE is 100000, and STOP_THRESHOLD is 0.001.

Cooling Rate: 0.9

Run Number	Solutions Tried	Final Score
1	182	16
2	182	22
3	182	20
4	182	19
5	182	17

Cooling Rate: 0.99

Run Number	Solutions Tried	Final Score
1	1840	13
2	1840	11
3	1840	14
4	1840	12
5	1840	12

Cooling Rate: 0.999

Run Number	Solutions Tried	Final Score
1	18419	9
2	18419	12
3	18419	10
4	18419	9
5	18419	8

Cooling Rate: 0.9999

Run Number	Solutions Tried	Final Score
1	184205	8
2	184205	9
3	184205	7
4	184205	8
5	184205	9

Cooling Rate: 0.99999

Run Number	Solutions Tried	Final Score
1	1842066	7
2	1842066	7
3	1842066	8
4	1842066	8
5	1842066	7

Results

The place.cpp file and header file, which can be found in figures, successfully takes the input of an input file. It takes the input and creates a grid with nodes placed randomly throughout. These nodes are sent through an annealing process to find the best possible solution allowed given the time restraints. When done cooling off, the function prints the node placement and edge length to an output file. Using an input file with six edges and running the program, the best score achieved is seven.

The statistical analysis shows a positive correlation between cooling rate, solutions tried, and final score. As the cooling rate is increased (approaches 1), more solutions are checked and the score is improved. It can also be noted that as another 9 is added to the cooling rate, the solutions tried, and therefore the execution time, increases by a factor of ~10.

The execution time for cooling rates below 0.99999 were visually indistinguishable. However, with cooling rate of 0.99999 and 1842066 solutions tried, the team observed nearly 2 seconds of delay in program execution.

Conclusion

In conclusion, we were able to implement a program to take an input file with parameters for a grid, vertices, and edges of nodes for an FPGA. The program places the nodes at random locations on the grid, and sends them through an annealing process. The nodes are moved randomly, one space away from their current location. Each new node location is then given a score, and assessed if it is better than the previous placement. The program will continue to do this until the cooling process is complete, and the results are printed to an output file. For a 6x6 graph, the best score the program was able to achieve was seven.

Figures

```cpp
/*
 * ECE 5730
 * Lab4 - Simulated Annealing
 * 10/10/24
 * Ryan Beck, Jared Bronson
 *
 */
#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <math.h>

#include "place.h"

#define INITIAL_TEMPERATURE 100000
#define COOLING_RATE 0.99999
#define STOP_THRESHOLD 0.001

int nodes, num_rows, num_cols;

std::vector<std::vector<bool>> edges; // 2D vector for edge graph

int main(int argc, char *argv[]) {
    srand(time(NULL));
    /*************************************
     * READ INPUT CONTENTS
     *************************************/
    // Open file
    std::ifstream file(argv[1]);
    if (!file.is_open()) {
        std::cerr << "Failed to open the file.\n";
        return 1;
    }

    // Variables for input parsing
    std::vector<std::string> lines;
    std::string line;

    // Enter each line as string into lines vector
    while (std::getline(file,line)) {
        lines.push_back(line);
    }

    // Close file
    file.close();

    char ch; // For switching on first char of line
    int row;      // tracking row for edge graph
    int col;      // tracking col for edge graph

```

Figure 1: Main function Pt. 1 in Place.cpp

```cpp

    // Iterate through lines vector
    for (const auto& l : lines) {
        // Grab first char of line
        ch = l[0];

        switch(ch) {
            // If initializing grid
            case 'g': {
                num_rows = l[2]-48;
                num_cols = l[4]-48;
                break;
            }
            // Determine node count
            case 'v': {
                nodes = l[2]-48;
                std::cout << "nodes: " << nodes << std::endl << std::endl;
                for(int i=0; i<nodes; i++) {
                    edges.push_back(std::vector<bool>(nodes,false));
                }
                break;
            }
            // Create edges
            case 'e': {
                // Select edge in graph
                row = l[2]-48;
                col = l[4]-48;
                // Set edge
                edges[row][col] = true;
                break;
            }
            default: {
                std::cout << "Invalid line! ch: " << ch << std::endl;
                std::cout << "--> " << l << std::endl;
                break;
            }
        }
    }
    // END READ INPUT CONTENTS
    printEdges(&edges);

    std::vector<int> current_x_pos; // current_x_pos[n] = x coord for node n
    std::vector<int> current_y_pos; // current_y_pos[n] = y coord for node n

    // Populate x and y pos vectors with initial positions
    std::cout << "Placing Nodes" << std::endl;
    placeNodes(current_x_pos,current_y_pos);

    // Print Node locations
    std::cout << "Initial Node Locations" << std::endl;
    printNodes(current_x_pos, current_y_pos);

    // By this point, we have two int vectors with node coords and an edge map
    // We can now begin the annealing process
    anneal(current_x_pos, current_y_pos);

    // Print results to console and table
    FILE* outputFile = fopen("output.txt", "w");

    for(int i=0; i<nodes; i++) {
        printf("Node %d placed at (%d, %d)\n",i, current_x_pos[i], current_y_pos[i]);
        fprintf(outputFile, "Node %d placed at (%d, %d)\n",i, current_x_pos[i], current_y_pos[i]);
    }

    int distance, n1, n2;

    for (n1 = 0; n1 < nodes; n1++){
        for(n2 = 0; n2 < nodes; n2++){
            if (edges[n1][n2]){
                distance = abs(current_x_pos[n1] - current_x_pos[n2])
                + abs(current_y_pos[n1] - current_y_pos[n2]);
                printf("Edge from %d to %d has length %d\n", n1, n2, distance);
                fprintf(outputFile, "Edge from %d to %d has length %d\n", n1, n2, distance);
            }
        }
    }

    return 0;
}

```

Figure 2: Main function Pt. 2 in Place.cpp

```cpp
/*****************************
* Simulated Annealing Functions
*****************************/

void anneal(std::vector<int> &current_x_pos, std::vector<int> &current_y_pos)
{
    double temperature = INITIAL_TEMPERATURE;
    int current_val, next_val, i;
    std::vector<int> next_x_pos;
    std::vector<int> next_y_pos;

    current_val = evaluate(current_x_pos, current_y_pos);
    printf("\nInitial score: %d\n", current_val);

    while (temperature > STOP_THRESHOLD)
    {
        copy(current_x_pos, current_y_pos, next_x_pos, next_y_pos);
        //std::cout << "Finished copy" << std::endl;

        alter(next_x_pos, next_y_pos);
        //std::cout << "Finished alter" << std::endl;

        next_val = evaluate(next_x_pos, next_y_pos);
        //std::cout << "Finished evaluate" << std::endl;

        accept(current_val, next_val, current_x_pos, current_y_pos,
               next_x_pos, next_y_pos, temperature);
        //std::cout << "Finished accept" << std::endl;

        temperature = cooling();
        i++;
    }
    printf("\nExplored %d solutions\n", i);
    printf("Final score: %d\n\n", current_val);
}

void copy(std::vector<int> &current_x_pos, std::vector<int> &current_y_pos,
          std::vector<int> &next_x_pos,    std::vector<int> &next_y_pos)
{
    int i;
    for (i = 0; i < nodes; i++){
        next_x_pos.push_back(current_x_pos[i]);
        next_y_pos.push_back(current_y_pos[i]);
    }
}

void alter(std::vector<int> &next_x_pos, std::vector<int> &next_y_pos)
{
    int n, new_x_pos, new_y_pos;
    // Repeat until move is on the graph
    do
    {
        // Pick random directions to move node
        do
        {
            n = rand() % nodes;
            new_x_pos = rand() % 3;
            new_y_pos = rand() % 3;


            if(new_x_pos == 2) new_x_pos = -1;
            if(new_y_pos == 2) new_y_pos = -1;

        }
        while(!((new_x_pos == 0 && new_y_pos != 0) || (new_x_pos != 0 && new_y_pos == 0)));
    }
    while(!((next_x_pos[n] + new_x_pos >= 0) && (next_x_pos[n] + new_x_pos < num_cols) &&
    (next_y_pos[n] + new_y_pos >= 0) && (next_y_pos[n] + new_y_pos < num_cols)));

    //printf("Old coords: (%d,%d)\tNew coords: (%d,%d)\n", next_x_pos[n], next_y_pos[n],
    //new_x_pos, new_y_pos);
    next_x_pos[n] = next_x_pos[n] + new_x_pos;
    next_y_pos[n] = next_y_pos[n] + new_y_pos;
}

```

Figure 3: Annealing Functions Pt. 1 in Place.cpp

```cpp
int evaluate(std::vector<int> &next_x_pos, std::vector<int> &next_y_pos)
{
    int distance, n1, n2;
    bool penalty = false;

    distance = 0;
    for (n1 = 0; n1 < nodes; n1++){
        for(n2 = 0; n2 < nodes; n2++){
            if(n1 != n2) {
                if (edges[n1][n2]){
                    distance += abs(next_x_pos[n1] - next_x_pos[n2]) +
                                abs(next_y_pos[n1] - next_y_pos[n2]);
                    //printf("Node1: %d\t Node2: %d\t N1(%d,%d)\t N2(%d,%d)\nDistance:
                    //%d\n", i,j,next_x_pos[i],next_y_pos[i],next_x_pos[j],next_y_pos[j],
                    //abs(next_x_pos[i] - next_x_pos[j]) +abs(next_y_pos[i] - next_y_pos[j]));
                }
                // If invalid nodes exist (nodes in same place) add penalty (3 times the max distance
                //of the grid)
                if((next_x_pos[n1] == next_x_pos[n2]) && (next_y_pos[n1] == next_y_pos[n2])) {
                    distance += 1*(num_rows + num_cols);
                    penalty = true;
                    //printf("Penalty Applied\t");
                }
                //printf("N1(%d,%d)\tN2(%d,%d)\n", next_x_pos[n1],next_y_pos[n1],next_x_pos[n2],
                //next_y_pos[n2]);
            }
        }
    }
    /*
    if(penalty)
        printf("Penalty applied to this solution\n");
    else
        printf("Penalty not applied\n");
    */
    return distance;
}

void accept(int &current_val, int next_val, std::vector<int> &current_x_pos,
std::vector<int> &current_y_pos, std::vector<int> &next_x_pos, std::vector<int> &next_y_pos,
int temperature)
{
    int delta_e, i;
    double p, r;

    delta_e = next_val - current_val;
    // If new result is better
    if (delta_e <= 0){
        for (i = 0; i < nodes; i++){
            current_x_pos[i] = next_x_pos[i];
            current_y_pos[i] = next_y_pos[i];
        }
        current_val = next_val;
    }
    // Have a chance to take worse result
    else{
        //printf("Current val: %d\t Next val: %d\n", current_val, next_val);
        p = exp(-((double)delta_e)/temperature);
        r = (double)rand() / RAND_MAX;
        if (r < p){
            for (i = 0; i < nodes; i++){
                current_x_pos[i] = next_x_pos[i];
                current_y_pos[i] = next_y_pos[i];
            }
            current_val = next_val;
        }
    }
}

double cooling()
{
    static double temperature = INITIAL_TEMPERATURE;
    temperature *= COOLING_RATE;
    return temperature;
}
```

Figure 4: Annealing Functions Pt. 2 in Place.cpp

```cpp
/****************************
* Graph Management Functions
****************************/

void printEdges(std::vector<std::vector<bool>> * graph) {
    printf("Edges of provided graph\n   ");
    for(int i=0; i<nodes; i++)
        printf("%d ", i);
    printf("\n");

    for(int i=0; i<nodes; i++) {
        printf("%d: ",i);
        for(int j=0; j<nodes; j++) {
            std::cout << edges[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

void placeNodes(std::vector<int> &x_pos, std::vector<int> &y_pos) {
    int node_x, node_y;

    // Loop for all nodes
    for(int n=0; n<nodes;) {

        bool duplicate = false;

        // Select random coords
        node_x = rand() % num_rows;
        node_y = rand() % num_cols;
        // Place first node
        if(n == 0) {
            x_pos.push_back(node_x);
            y_pos.push_back(node_y);
            n++;
        }
        else {
            // Loop through existing nodes
            for(int existingNode = 0; existingNode<n; existingNode++) {
                // If existing node coords do match new coords
                if(x_pos[existingNode] == node_x && y_pos[existingNode] == node_y) {
                    // Break from loop without incrementing n count
                    duplicate = true;
                    break;
                }
                //printf("New node %d (%d,%d)\t Existing at %d (%d,%d)\n",n,node_x,node_y,existingNode,
                //x_pos[existingNode], y_pos[existingNode]);
            }
            if(!duplicate) {
                // Place node
                x_pos.push_back(node_x);
                y_pos.push_back(node_y);
                // Increment n count
                n++;
            }
        }
    }
    std::cout << std::endl;;
}

void printNodes(std::vector<int> x_pos, std::vector<int> y_pos) {
    for(int i=0; i<nodes; i++) {
        printf("Node %d placed at (%d, %d)\n",i, x_pos[i], y_pos[i]);
    }
}
```

Figure 5: Grid Management functions in Place.cpp

```
#ifndef PLACE_H
#define PLACE_H

/******************************************
* Graph management functions
******************************************/
void printEdges(std::vector<std::vector<bool>> * graph); // Prints edges of provided graph(1 for edge, 0 no edge)
void printNodes(std::vector<int> x_pos, std::vector<int> y_pos); // Prints (x,y) coords of each node
void placeNodes(std::vector<int> &x_pos, std::vector<int> &y_pos); // Place nodes in initial locations


/******************************************
* Annealing functions
******************************************/
void anneal(std::vector<int> &current_x_pos, std::vector<int> &current_y_pos)

void copy(std::vector<int> &current_x_pos, std::vector<int> &current_y_pos,
          std::vector<int> &next_x_pos,    std::vector<int> &next_y_pos);

void alter(std::vector<int> &next_x_pos, std::vector<int> &next_y_pos);

int evaluate(std::vector<int> &next_x_pos, std::vector<int> &next_y_pos);

void accept(int &current_val, int next_val, std::vector<int> &current_x_pos,
            std::vector<int> &current_y_pos, std::vector<int> &next_x_pos,
            std::vector<int> &next_y_pos, int temperature);

double cooling();

#endif

```

Figure 6: Place.h header file