April 18, 2017

Individual Project

We learn OOP and C++ to meet practical project requirements. In our Department of Computer Science and Technology, many research projects are implemented in C++. Therefore, as the final individual project, we will practice implementing/solving scientific research papers or related research problems. We hope this type of training will prepare our students for future SRT (Student Research Team) projects, master/Ph.D. program, and industrial projects.

From the provided individual project list, please choose only **one project**, and to implement it **all by yourself**. It is **not allowed** to choose multiple projects. For each individual project, credits are given based on the relatively difficulty of the problem. Harder problems with higher difficulty values correspond to higher credits. Up to 5% bonus credits will be assigned to well-finished projects with interesting extensions and/or improvements in solution quality.

Requirement for submission:

(1) Please submit a zip/rar ball including 3 directories: **src**, **testcase**, and **doc**.

(2) All the source codes including makefile need to be put in directory **src**.

(3) All the testcases (测试用例) need to be put in directory **testcase**.

(4) Detailed description of the software design (pdf file) needs to be put in directory **doc**.

April 18, 2017

Choose one from the following projects for implementation:

(I) Paper Implementation: VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair (Difficulty: 1.0)

Requirements and TIPS:

(1) Partial source codes are provided to make the implementation easier. Please read the source codes carefully and find out how to use the code. Modify the provided source codes and add new codes as needed to implement the paper. You may also write your own codes without using the provided source codes if you want to.

(2) Search the Simulated Annealing (SA) method by Google/Baidu and implement the method to optimize the rectangle packing results as described in the paper.

(3) Rewrite the longest path algorithm for better efficiency: use Boost Library for computing the longest path to replace the provided algorithm in the source code.

(4) The optimization goal is to minimize the weighted sum of the area (A) of the bounding box of placed rectangles and the total wirelength (L), i.e., $\alpha \times A + (1-\alpha) \times L$ $(0 < \alpha < 1)$.

(5) Randomly generate 5 different testcases with up to 500 rectangles with different width and height values, and assign the rectangles to 250 pairs. For a pair of rectangles r_i and r_j whose centers are placed at (x_i,y_i) and (x_j,y_j), respectively, the wirelength $l_{i,j}$ (called Manhattan distance) of the rectangle pair is computed (r_i,r_j) as

$$|x_i - x_j| + |y_i - y_j| \qquad (1)$$

(6) Report the statistics of the experimental results (实验结果), e.g., total area, total wirelength, etc. Figures and tables on the experimental results are welcome.

(7) It is suggested that a validity checking function be implemented to verify the experimental results are indeed correct. For example, there should not be overlaps between the placed rectangles.

(II) Project: On Constructing Minimum Spanning Trees (Difficulty: 1.05)

Requirements and TIPS:

(1) Implement the MST algorithm based on Voronoi diagram for computing Euclidean minimum spanning trees in 2D plane.

(2) Please refer to the following link for MST construction algorithms. Either Prim's or Kruskal's algorithm is ok. Please refer to Page 39 for the idea of using Voronoi for MST computation.

http://www.cs.princeton.edu/courses/archive/spr07/cos226/lectures/mst.pdf

(3) Use the CGAL Library (http://www.cgal.org/) for constructing the Voronoi diagram and Delaunay triangulation.

(4) Randomly generate 5 different testcases with more than 5000 points without duplicates (输入点不重合) to test the implemented method.

(5) It is suggested that a validity checking function (验证正确性) be implemented to verify the experimental results are correct. For example, you may directly apply Prim's or Kruskal's MST algorithm on the testcase to verify that the MST trees are correct.

(6) Report the statistics of the experimental results (实验结果), e.g., total runtime, total number of points, total length of the MST edges, etc. Figures and tables on the experimental results are welcome.

(7) [This is not mandatory to finish, but it is a challenging topic] Again, can you compute the top K (1 <= K <= 20) minimum spanning trees?

(III) Project: Extension of LiquidFun (in Code Reading assignment) (Difficulty: 1.1)

Requirements and TIPS:

(1) Design a **new** game different from the given examples along with the LiquidFun library.

(2) Implement the GUI window for displaying the game.

(3) The fluid simulation library needs to be used for computing the fluid movement status.

(IV) Project: SAT-Based Two-Terminal Path Finding (Difficulty: 1.1)

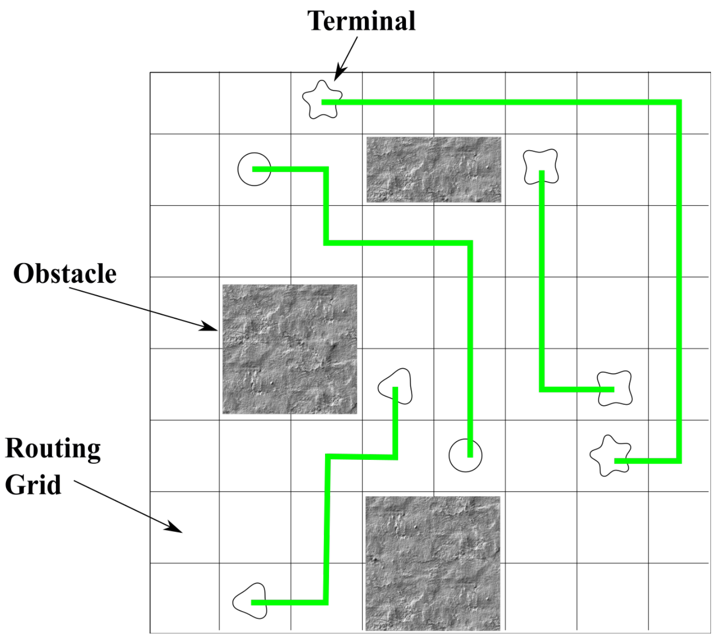

Figure 1. Example of path finding for two-terminal nets.

Requirements and TIPS:

(1) Problem description (see Figure 1):

Given: N×N routing grids, and M pairs of terminals.

Find: routing paths connecting each pair of terminals.

Constraints: different paths cannot cross each other, and no path can cross obstacles.

Objective: (a) maximize the number of connected pairs (连接最多的端点对), and (b) when all pairs of terminals can be connected, minimize the total length of all the paths (若所有端点对都成功连接，则最小化总线长，即所用 routing grid 的数目).

(2) Formulate the above path finding problem into the satisfiability (SAT) problems (可满足性问题), and then solve the problems using a SAT solver.

(3) Z3 needs to be used to as the SAT solver to solve the formulated SAT problems. Here are some useful links (if you need more references, please try to find them by yourself):

(a) https://github.com/Z3Prover/z3

(b) http://rise4fun.com/z3/tutorial/

(c)

https://github.com/Z3Prover/z3/blob/master/examples/c/test_capi.c

(d)

https://github.com/Z3Prover/z3/blob/master/examples/c%2B%2B/example.cpp

(4) Key challenges of the above path finding problem:

(a) How to formulate the path finding problem into a SAT problem: You may read the attached paper (SATExample.pdf) to find out one way of formulation. Please kindly note that you may also find other ways of SAT formulations for solving the problem.

(b) How to program your formulation into Z3 and obtain the results (i.e., paths): You are suggested to read the tutorials and sample codes carefully and find other useful materials by yourself through the documents on github.

(c) How to minimize the total length of the paths: if there is not a way to directly formulate minimization problem using Z3. You may think of giving the upper bound on the number of the used routing grids, and iteratively increase the upper bound to find a feasible solution.

(5) Randomly generate 10 different testcases with different sizes of routing grids (e.g., 8×8, 9×9, 10×10 …), different numbers of terminal pairs (e.g., 2 pairs, 3 pairs, 4 pairs …), and different number of obstacles (e.g., 2, 3, 4 …). The coordinates of the terminals and obstacles need to be randomly selected without duplicates (输入点不重合).

(6) Report the statistics of the experimental results (实验结果), e.g., total runtime, number of terminal pairs successfully connected, total path length, etc. Figures and tables on the experimental results are welcome.

(7) It is suggested that a validity checking function (验证正确性) be implemented to verify the experimental results are correct. For example, you may check the connectivity of the terminal pairs, whether different paths cross each other, whether there are loops in computed paths, etc.

(V) Project: Multiple-Terminal Path Finding (Difficulty: 1.2)

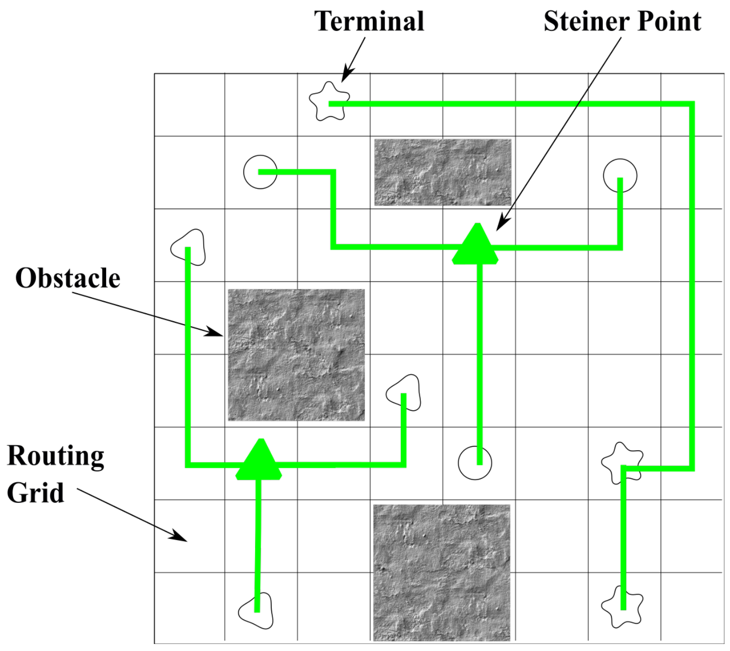

Figure 2. Example of path finding for multiple-terminal nets.

Requirements and TIPS:

(1) Problem description (see Figure 2):

Given: N×N routing grids, and M sets of terminals (each set has at least 3 terminals).

Find: routing paths connecting each set of terminals (Steiner trees are allowed).

Constraints: different paths cannot cross each other, and no path can cross obstacles.

Objective: (a) maximize the number of connected sets of terminals (连接最多的端点集合), and (b) when all sets of terminals can be connected, minimize the total length of all the paths (若所有端点对都成功连接，则最小化总线长，即所用 routing grid 的数目).

(2) Formulate the above multiple-terminal path finding problem into any formulation that you could think of. You may consider SAT formulation as discussed in Project No. IV, by the integer programming problem (ILP, 整数规划问题) and Column Generation Algorithm (please see the attached paper), by network-flow formulation, etc. **Please be warned that this is a challenging problem without known good solutions**.

(3) A problem formulation can be found in the attached paper (SimilarFormulation-FYI.pdf), where **global routing (总体布线)** is solved. In the paper, the ILP formulation is used, where the candidate path solutions for each set of terminals are generated by the column generation algorithm. However, the global routing problem is different from our multiple-terminal path finding problem in the following aspects: (a) in global routing, paths of different sets are allowed to cross each other (不同端点集合的连线可以交叉); (b) in global routing, several paths can cross one grid as long as the capacity is not exceeded (网格允许穿越多

条连线，且每个网格有穿越连线数的容量限制). Therefore, the formulation of global routing has to be modified such that it could be used to solve our multiple-terminal path finding problem.

(4) Instead of using the CPLEX solver, Gurobi (http://www.gurobi.com/) is suggested. Please find the sample codes in *Gurobi* directory for how to call Gurobi functions and compile the codes. Please try to use the academia version at http://www.gurobi.com/academia/academia-center. The license file needs to be downloaded and set up in .bashrc (Linux). Please find the documentation of Gurobi at http://www.gurobi.com/documentation/.

(5) Key challenges of the problem:

(a) How to formulate the problem into an ILP problem and how to find the candidate solutions by column generation: Please search materials by yourself for introduction to the ILP. Please try to understand how the column generation algorithm works from the attached pdf file (columnGeneration.pdf).

(b) How to program your formulation and obtain the solutions from Gurobi: Please find the sample codes in directory Gurobi and learn the programming interfaces, which were used for solving another problem using Gurobi.

(c) How to generate the initial Steiner path solutions and how to generate new candidate solutions for each set of terminals: Please download FLUTE source codes (http://home.eng.iastate.edu/~cnchu/flute.html) and learn how to call FLUTE for generating Steiner trees for randomly given points; please think of ways for generating new solution candidates by ripping-up and reroute some path segments using maze or A* routing algorithm (拆除部分连线，然后采用作业中的迷宫算法或 A*算法求解新的路径).

(6) Randomly generate 10 different testcases with different sizes of routing grids (e.g., 8×8, 9×9, 10×10 …), different numbers of sets of terminals, (e.g., 2 sets, 3 sets, 4 sets …), and different number of obstacles (e.g., 2, 3, 4 …). The coordinates of the terminals and obstacles need to be randomly selected without duplicates (输入点不重合).

(7) Report the statistics of the experimental results (实验结果), e.g., total runtime, number of sets of terminals successfully connected, total path length, etc. Figures and tables on the experimental results are welcome.

(8) It is suggested that a validity checking function (验证正确性) be implemented to verify the experimental results are correct. For example, you may check the connectivity between the sets of terminals, whether different paths cross each other, whether there are loops in the computed paths, etc.

(VI) Project: Escape Routing with Specific Obstacle Avoidance Constraints (Difficulty: 1.2)

Requirements and TIPS:

(1) Please read the attached paper, especially Figure 5 in the paper, to understand the escape routing problem in paper-based digital microfluidic biochips. In Figure 5, each CNT electrode (e.g., e_1) needs to be routed to a peripheral control pin, avoiding the neighborhood of its conflict electrodes (e.g., $C(e_1)$). **Each electrode takes 5×5 routing grids, and the spacing between adjacent electrodes is 3 routing grids.** The objective is to successfully route all the electrodes (from any routing grid on the electrode) to peripheral control pins, avoiding the specific obstacles (for

each electrode e_i, the channels adjacent to e_i's conflict electrodes $C(e_i)$ are all blocked), with total wirelength of the control lines minimized.

(2) Formulate the above problem using any type of problem formulation (e.g., by ILP with Gurobi solver, by column generation algorithm, by network flow formulation, by dynamic programming, by some good heuristic algorithm, etc.) as you like, and solve the problem efficiently (fast) and effectively (correctly with minimized wirelength).

(3) Randomly generate 20 different testcases with different sizes of electrode arrays (from 16×16 to 30×30), and different number of conflict electrodes (each electrode has 2 to 5 randomly selected electrodes).

(4) Report the statistics of the experimental results (实验结果), e.g., total runtime, size of the electrode array, total wirelength, etc. Figures and tables on the experimental results are welcome.

(5) It is suggested that a validity checking function (验证正确性) be implemented to verify the experimental results are correct. For example, you may check connectivity between the electrodes to the control pins, whether different control lines cross each other, whether the control line of an electrode passes its conflict electrodes' neighborhood, etc.

NOTE: For some projects, it is not required to plot the results. I.e., you may just output the coordinates of the computed results. However, plotting the results could be of great help in checking the correctness. For those who want to visually check the results, here are some suggestions:

(1) Write out PostScript files from the program and then open it with PS viewers. Here is a link for the tutorial on writing PS files: http://www.tailrecursive.org/postscript/drawing.html

(2) Write out BMP files from the program. Search the BMP file format using Google/Baidu to see how to write a BMP file. For example:

(a) http://stackoverflow.com/questions/2654480/writing-bmp-image-in-pure-c-c-without-other-libraries

(b) http://blog.art21.org/2011/09/13/how-to-create-a-bitmap-image-file-by-hand-without-stencils/#.U0z8uF6aIaA

(3) Learn and try to integrate existing graph drawing tools (e.g., GraphViz: http://www.graphviz.org/Gallery.php) for plotting the results. Or write your own viewer using any GUI programming language (e.g., Qt, Java, OpenGL, etc.) to open text input file and plot it.