Team Project: Projects and Scientific Research Paper Implementation

One of the following projects needs to be finished in team. As different projects have different degrees of difficulty (难度系数). Just like in the diving competition (跳水比赛), higher degree of difficulty corresponds to more efforts as well as more credits (分数). Bottom line (底线) is to correctly finish one project with the lowest degree of difficulty. For each project, an extra bonus of up to 10% will be granted considering the degree of difficulty, as well as any interesting extension and/or improvement in solution quality.

Please choose only ONE from the following projects to implement in team. When implementing the research papers, you may use any open source codes you find through Google/Baidu searching engine. You may not be able to find the exact source codes of the project, but you may find source codes of algorithms used in the paper, which make your implementation easier and faster. Any such open source codes (or libraries) are allowed in implementing the research papers. However, for projects (i.e., Problem No. 2 and No. 4), it is not allowed to copy from open source codes.

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**.

(5) Each team may have 2-3 students, and each student needs to submit a separate document (the pdf file) describing his own work in the project.

Choose one from the following projects for implementation:

(I) Paper: On Constructing Minimum Spanning Trees in k-dimensional Spaces and Related Problems (degree of difficulty: 1.0)

Requirements and TIPS:

(1) Implement the paper using OOP design (e.g., you may want to use the strategy design pattern for multiple MST algorithms). Only Euclidean minimum spanning trees in 2D plane is required (Section 3 in the paper).

(2) Please refer to the following link for MST construction algorithms. Either Prim's or Kruskal's algorithm is ok. It is not required to implement the O(|E|loglog|V|) MST algorithm.

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

(3) The $O(n^2)$-time preprocessing for nearest point query (reference [15] in the paper) is not required.

(4) Randomly generate 10 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. For example, you may directly apply Prim's or Kruskal's MST algorithm on the testcase to verify that the correctness of the results.

(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. For those who have implemented the Voronoi diagram for MST computation (in the individual project), you may report the results of both methods for comparison.

(II) Project: Code Optimizer (degree of difficulty: 1.0)

Requirements and TIPS:

(1) Write a program using OOP design. For a given C++ project, your program accepts multiple source (.cpp or .cxx) and header files (.h) of the project, and outputs the optimized (优化的) source and header files.

(2) Items that can be optimized include but not limited to (包括但不限于):

(a) Automatically indent (缩进) each line of the source codes. You need to provide a way for the user to configure the shift width for indentation (允许用户设置缩进的空格数);

(b) Remove all the redundant (冗余的) variables, classes and functions that are not used in the program;

(c) This is not required but it is encouraged to have a try (不要求完成): identify code blocks that can be improved in efficiency (提高代码运行效率) and replace them with improved codes. As a simple example, we may replace "i++" with "++i" for better efficiency if it does not cause wrong outputs.

(3) You may want to learn something about the lexical analysis (词法分析) and syntactic analysis (语法分析) to finish the project well. Please design the way by yourself for loading in all the source and header files of the project. And please note that your code optimizer should not cause errors in the program.

(4) Find at least 5 open-source C++ projects from github or other sources to test your code optimizer. You may also manually modify the source codes to

verify that your code optimizer works. Please report the statistics results of your optimizer, e.g., how many variables are reduced, how many lines are indented, how much runtime is reduced, etc.

(III) Paper: 3D Placement Using the T-tree Formulation (degree of difficulty: 1.1)

Requirements and TIPS:

(1) Implement the paper using OOP design. You only need to implement the 3D placement algorithm. The defect-tolerance in digital microfluidic biochips is not required. Please refer to the other two attached papers for more detailed description of the T-Tree formulation.

(2) Search the Simulated Annealing (SA) method by Google/Baidu and implement this method to optimize the volume (体积) of the bounding cube (包围立方体), i.e., $x \times y \times h$ (completion time) as described in the paper.

(3) It is suggested that a validity checking function be implemented to verify (验证正确性) the experimental results. For example, there should not be any overlap between the placed blocks/operations.

(4) Randomly generate 10 different testcases with cubes of different sizes ($x \in [1,10]$, $y \in [1,10]$, $h \in [1,10]$) to test your 3D placement method.

(5) Report the statistics of the experimental results (实验结果), e.g., total runtime, total number of cubes, total volume of bounding cube after placement, wasted volume (= "total volume of bounding cube" – "total volume of the placed cubes"), etc. Figures and tables on the experimental results are welcome. Please experimentally find out the maximum number of cubes

that can be placed in 10 minutes with fairly good solution quality, e.g., "wasted volume"/"total volume" < 10%.

(6) It is suggested that a validity checking function (验证正确性) be implemented to verify the experimental results, e.g., no two cubes intersect with each other after placement.

(7) This is not required but you are encouraged to have a try: implement any other alternative 3D placement methods. For example, is it possible to extend the idea of sequence-pair (see our Individual Project) for 3D placement?

(IV) Project: Software Plagiarism Detection (软件剽窃检测, degree of difficulty: 1.1)

Requirements and TIPS:

(1) Implement the Software Plagiarism Detection project using OOP design. Please read the attached exercise, one of our future homeworks, to understand the background. Read the related references as needed for how to realize software plagiarism detection.

(2) Please note that you may learn from existing open-source software plagiarism detection systems (search for them through Baidu/Google by yourself). But you should not copy from their codes.

(3) Manually design at least 5 single-file or multiple-file projects (you may even use homeworks of your team members) to test your software plagiarism detector.

(V) Project: Large-Scale Multiple-Terminal Path Finding (degree of difficulty: 1.2)

Requirements and TIPS:

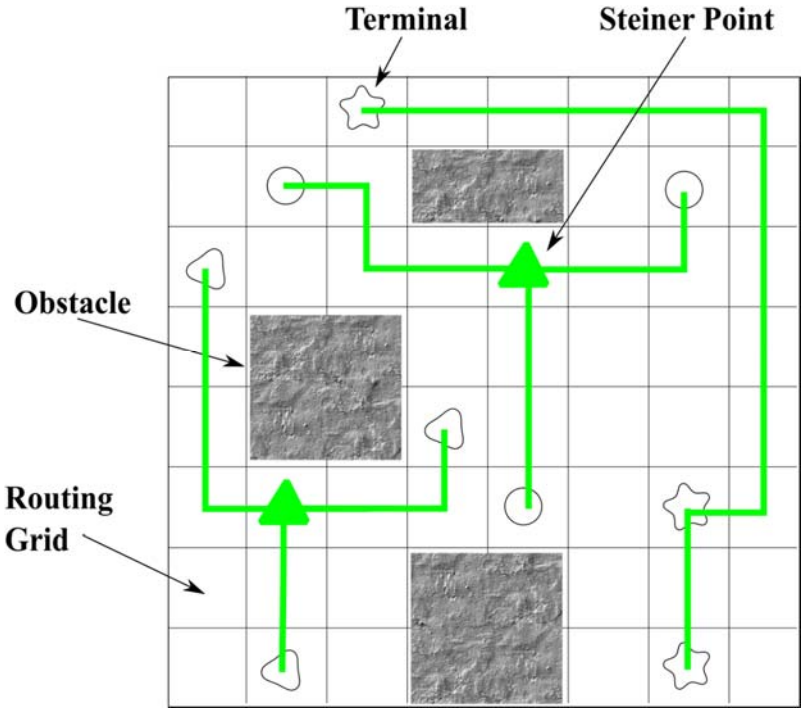

Figure 1. Example of the path-finding problem.

(1) Write the program using OOP design. In our individual projects (Problem IV and Problem V), the path finding problems are solved using different formulations, i.e., SAT and column generation. In this project, our objective is to solve the multiple-terminal path finding problem (please read our individual project No. 5 for details) on much larger number of routing grids. Here, for the N×N routing grids and M sets of terminals (each set has at least 3 terminals), we require that N >= 300, and M >= 10.

(2) The SAT and column generation methods in the individual projects can also be used in solving this project. However, due to the large problem size

(问题规模大) in this project, you probably need to think of some divide-and-conquer (分而治之) strategy. For example, is it possible to merge the adjacent 2×2 or 10×10 grids into a larger grid to reduce the problem size (将相邻的 2×2 或 10×10 个网格合并为一个大网格，从而降低问题规模), and then map back the solutions from the reduced problem to the original problem (将缩小后问题的解映射到原问题)? Besides, any other methods can be used provided that it solves the problem with good solution quality and runtime (可接受的运行时间). Due to the difficulty in solving this problem, it is not required to obtain the optimal solution.

(3) Randomly generate 10 different testcases with different sizes of routing grids (e.g., 300×300, 350×350 …), different sets of terminals, (e.g., 10 sets, 11 sets …), and different number of obstacles (e.g., 5, 6 …). The coordinates of the terminals and obstacles need to be randomly selected without duplicates (输入点不重合).

(4) 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.

(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 the connectivity between the sets of terminals, whether different paths cross each other, whether there are loops in the computed paths, etc.

(VI) Paper: Mastering the Game of Go with Deep Neural Networks and Tree Search (degree of difficulty: 1.2)

Requirements and TIPS:

(1) Write the program using OOP design. Only those who are familiar with machine learning (especially deep learning) methods are suggested to try this project.

(2) Besides Google DeepMind's homepage (https://deepmind.com/alpha-go.html) and the python source codes (https://github.com/Rochester-NRT/RocAlphaGo), you may want to search for more materials through Baidu/Google to further understand the algorithms' details.

(3) Any open-source algorithms and/or libraries can be used in your implementation. For example, you may use TensorFlow or Caffe you have learned during our code reading homework, or any other deep learning frameworks.

(4) Objective: as the computing resource (计算资源) needed for training the game of Go is huge, you are not required to train your deep learning model for playing the game of Go. Instead, you are required to try five-in-a-row (五子棋), or the rectilinear Steiner tree (直角斯坦纳树) problem with 11 input points. (五子棋和直角斯坦纳树任选其一即可)

(a) If you cannot find good computing resource by yourself, we can provide one GPU server with one account for each team (please send me an email and ask for the account). Please note that it is not guaranteed that the GPU server always works without crashes. And please make backups (备份) of your source codes and try your algorithms as soon as possible.

(b) Intuitively, five-in-a-row should be much easier than the rectilinear Steiner tree problem. So successful algorithms for the rectilinear Steiner tree problem will correspond to more bonus credits.

(c) Learn the rule of five-in-a-row by yourself and design your program in such a way that the user may play the game with the computer. GUI (Graphical User Interface, 图形用户界面) is not required. But if you want, you may develop the GUI for better user experience (不要求 GUI，但可以开发 GUI 界面以改善用户体验).

(d) For those who want to challenge the rectilinear Steiner tree problem, please search using Google/Baidu to learn some basics on rectilinear Steiner tree. To make the problem easier to solve, you only need to consider the problem with 11 terminal points (只需要考虑 11 个端点的直角斯坦纳树求解问题), i.e., for any randomly given 11 terminal points, your trained robot can easily solve the problem and output one optimal rectilinear Steiner tree. Please note that it is an open problem whether this problem is solvable by directly using the algorithms for the game of Go, and you are suggested to think about the solutions (原论文中的算法未必能直接用于求解该直角斯坦纳树问题，答案未知，需要您思考如何才能有效进行求解). The sources of GeoSteiner-3.1 (http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.diku.dk/geosteiner/) can be used to obtain the optimal solutions for given input terminals, which may be used for generating the training samples.

NOTE: For some projects, 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) Write your own viewer using any GUI programming language (e.g., Qt, Java, OpenGL, etc.) to open text input file and plot it.