Lab #2: Simulated Annealing

Student ID ____________________ Section # ________

Marks ____________________

Introduction

Objective: In this Lab you will become very familiar with implementing and experimenting with the Simulated Annealing algorithm. From the course lectures and previous experiences you should be fairly/somewhat familiar with optimization. In this lab you will become familiar with some techniques that you can use to solve combinatorial optimization problems. The simulated annealing process is not difficult to understand and should be somewhat familiar. Many physical process involve an annealing phase to improve (optimize) or alter material properties.

In this lab, a problem suitable for solution by a simulated annealing algorithm is investigated.

It is the same problem as in lab 1, so you likely already know and love it. The problem is denoted the Component Allocation Problem. Recall, that this is an allocation problem where the object is to allocate equal numbers (or as close as equal as possible) of components into two racks of equipment minimizing the degree of interconnect between the two racks. Review the course notes for a detailed example albeit very small example. In this lab make sure you use a considerably larger problems as you would like to monitor run time vs. problem size. Be able to solve problems of at least 100 components. This problem is similar/identical to the problem for Lab 1 and 3. This allows some degree of reuse as well as allowing for docking or comparison.

Docking is both a verification and validation procedure. So if you were getting reasonable results for the Naïve “greed” algorithm of lab 1 and results here are better, then you are likely going in the right direction. In addition, this problem is of historical significance as it was one of the first problems addressed by a non-deterministic optimization algorithm at IBM in the early 80s. Adjacency matrices are in D2L UMLearn that must be used as a benchmark or test cases. This also facilitates comparison with other peoples’ solutions. Complete problem specification can be found in the notes as well as in the preparation materials for Lab 1.

Problem Ref: Kirkpatrick, Scott, D. Gelatt Jr., and Mario P. Vecchi. "Optimization by simulated annealing." science 220, no. 4598 (1983): 671-680. Cited: 42000 times (That is a lot) In the next lab you will solve the same problem using a Genetic Algorithm.

Note: I have been using adjacency somewhat informally. This is the definition I am using here: Given a weighted graph G, the adjacency matrix is the matrix A = (aij), where aij = w(vi , vj). That is, the weight or edge cost of the edge from node I to j.

Preparation: Overview the section in the notes covering the simulated annealing algorithm section. You should have an adjacency matrix generator/reader from lab 1 and other utilities from lab 1 such as monitoring the progression of the solution and cpu time. Use that to estimate the run time versus problem size. Keep the matrix sparse as that is pretty reasonable assumption for many interconnection networks. Also use an adjacency matrix that has all the nodes in a loop or linear configuration as well as those on UMLearn. These make nice bench-mark and comparison instances.

The input is a list of components denoted Ci with connectivity represented by an adjacency matrix. The adjacency matrix represents the degree of connectivity or cost associated with connecting component i to j. The links are typically bi-directional. The objective function is to minimize the overall cost of the interconnect between two groups of components while trying to maintain an equal number of components in each set. The additional cost that may be associated with an unequal number of components, is x times the difference in the cardinality of the respective groups, if your algorithm considers roughly equal set partitions. That is, if x=5, if one group has 11 and the other 9, the difference is 2 and the increase in cost is 10. As mentioned adjacency matrices are provided in D2L for testing your algorithm and seeing how well it does.

You can also set up your problem solution such that there is always an equal number of components in each bin. No penalty term required in that case.

Questions:

1) Provide pseudo-code for your algorithm? Did it run as expected as compared to the results from lab 1. What was the stopping criteria you used?

2) What was your minimum score or solution for the test cases? What was the percent improvement from an original or initial "solution".

3) What was your cooling schedule? Why did you select that schedule?

4) How long did the algorithm take to run? If you doubled the size of your problem did the running time scale linearly, quadratically or by some other means?

5) Vary the component connectivity. How did this affect the running time? That is, consider varying the sparseness of the graph. Perhaps start at an average of one edge per node, up to an average of 25 edges per node.

6) How might your basic algorithm be improved? Some ideas may be to spend more time at temperatures that are more interesting. Of course you may have to define what that means.

This is sort of Bonus: Utility for "Visualization"

Write a utility that presents solution evolution over time(iterations) and displays the result visually. Perhaps cost vs. temperature would also be a reasonable graph.

Report

Report Submission Instructions The lab report is due at the start of the next lab, and must be submitted electronically through umLearn.

• Each student is required to write and submit a separate report.

• Include a discussion of the problem and your algorithm. In the body of the lab report include answers for each of the questions posed in the lab.

• Organize the lab write-up with corresponding sections, title, and your answers to posed questions.

• Document all programs with comments in the program code.

• Lab write-up (pdf), submitted to umLearn.

• As Appendices: Code you have written, and documentation. Zip up everything if you like.