Programming Problem
for the
1999 Comprehensive Exam

Out: Monday 1/25/99 at 9:00 am
Due: Friday 1/29/99 at 5:00 pm

Department of Computer Science
Brown University
Providence, RI 02912

1.0 The Task

This year you are going to study the effectiveness of approximation algorithms to NP-complete problems. In particular you are going to look at the MAX CUT problem and empirically investigate several different approximation algorithms acting on several different types of graphs.

2.0 The MAX CUT Problem

MAX CUT is one of the large class of NP-complete graph problems. Here one is given an undirected graph with weighted edges. The task is to divide the nodes of the graph into two distinct sets such that the sum of the weights of the edges that go from a node in one set to one in the other set is maximized.

Formally, given an undirected graph $G = (V, E)$ and weights $w_{i,j}$ on the edges $(i, j) \in E$, the maximum cut problem (MAX CUT) is that of finding the set of vertices S that maximizes the weight of the edges in the *cut* $(S, \bar{S})$; that is the weight of the edges with one endpoint in S and the other in $\bar{S}$. For simplicity, one usually sets $w_{i,j} = 0$ for $(i, j) \notin E$ and denote the weight of a cut $(S, \bar{S})$ by

$$w(S, \bar{S}) = \sum_{i \in S,\, j \notin S} w_{i,j}$$

Thus, one wants to find S such that $w(S, \bar{S})$ is maximized.

This problem has been known to be NP-complete for a long time, even in the unweighted case. However, it is an important problem with applications in circuit layout design and statistical physics and hence often needs to be approximated.

Most of the approximation algorithms that have been proposed only guaranteed to get within 1/2 of the optimal solution. In 1994 a new algorithm was proposed based on semidefinite programming (article attached) that offered a 0.87856 guarantee. It is this algorithm that you are going to compare to three much simpler ones on a bunch of graphs you will generate yourself to see if the additional complexity is actually worth it.

3.0 Task Specifics

You are to code up four approximation algorithms to the MAX CUT problem. These are described in the next section. Then you are to generate a series of graphs using the criteria in the following section. You are to run each of your algorithms on each of the graphs (it would probably be easiest if this was all one within one program) and compare the results. At the end you should produce a summary showing how well each of the algorithms did.

4.0 Approximation Algorithm

In this section we detail the four approximation algorithms you are to use.

4.1 Random Cut

The simplest algorithm is just to randomly select the set of nodes S. A random selection of S actually is not a bad approximation: it has an average case value of 1/2 of the optimal solution. Note that the easiest way of randomly choosing S is to take each node and randomly decide if it is in S or not in S.

4.2 Greedy

Here one looks at each node in turn and then assigns it to either S or $\bar{S}$ based on whichever increases the size of the cut the most. One can either look at the nodes in a random order or in the order they are given.

4.3 Your Algorithm

The third algorithm is one of your own devising. It need not be all that complex — it can simply be a modification of one of the above — but you should try to have it get good results (i.e. it should be able to do better than either of the above two naive algorithms).

4.4 Semidefinite Programming

This algorithm gets somewhat complex if you try to understand how and why it works, but you really only need to understand how to code it to successfully

complete this problem. Moreover, we are not expecting you to write a semidefinite programming solver; that is being provided to you.

The algorithm is as follows:

1. Set up the semidefinite programming problem to obtain the matrix Y:

$$max \frac{1}{2} \sum_{i<j} w_{i,j}(1 - y_{i,j})$$

subject to: $y_{i,i} = 1$

$Y = \left[y_{i,j} \right]$ symmetric and positive semidefinite

To make this tractable, we have provided you with a subroutine for semidefinite programming. The library `libsemidef.so` contains the routine:

```
extern int solveSemidef(int n,      // dim of W, D, Y
      double * w,                   // N x N Matrix W (see below)
      double * d,                   // N Vector D
      double * y)                   // N x N Matrix Y (see below)
```

which solves the semidefinite programming problem of maximizing

$$\sum_{i<j} (W_{i,j} \times Y_{i,j})$$

where $W_{i,j}$ is a symmetric NxN matrix of constants and $Y_{i,j}$ is an NxN matrix of terms to be found subject to the constraints that

Y is symmetric positive semidefinite

$Y_{i,i} = D_i$

for the given set of constants $D_i \geq 0$

The matrices Y and W are stored in row-major, upper triangular form, i.e.

```
w[0] = W[1,2]
w[1] = W[1,3]
...
w[n-2] = W[1,n]
w[n-1] = W[2,3]
...
```

to save space. (Note that the diagonal elements are not needed: the W elements are never used while the Y diagonal is assumed equal to D.)

2. Use a Cholesky decomposition to find the lower triangular matrix B such that $Y = B^T B$:

$$B_{k,k} = \left(Y_{k,k} - \sum_{p=1}^{k-1} B_{k,p}^2 \right)^{\frac{1}{2}}$$

$$B_{i,k} = \frac{Y_{i,k} - \sum_{p=1}^{k-1} B_{i,p} B_{k,p}}{B_{k,k}} \quad , \quad i = k+1,\ldots,n$$

3. Let v_i be the ith column of B.
4. Choose a random vector r uniformly distributed on the unit sphere S_n. This can be done by selecting n values $x_1, x_2, \ldots, x_n$ randomly from a normal (not uniform) distribution and then normalizing the vector so obtained.
5. Set $S = \{i \mid v_i \cdot r \geq 0\}$ where dot indicates the dot product of the two vectors.

5.0 Testing Graphs

You should compare these algorithms on a variety of graphs. The graphs should be of moderate size, but not overly large. (The code for semidefinite programming is polynomial, but not necessarily small polynomial.) More importantly, you want to try a variety of graphs (and several instances of each variety — enough to get a good basis for comparison).

The different types of graphs to try include:

- **Random graphs**. You should generate random graphs of various sizes with different densities of edges. You can try various different ways of generating random graphs if you are so inclined.
- **Bipartite graphs**. Just for fun and to test your algorithms you should try some bipartite graphs (here the MAX CUT solution is obvious).
- **Grid graphs**. Divide the unit square into four quadrants. Now generate random points in the square. Each point represents a node of the graph. Two points have an edge between them if the distance between them is less than δ for some δ. The weight of the edge is w_i if both points are in quadrant i, and $w_i \times w_j$ if one point is in quadrant i and the other in a different quadrant j. You can choose different values for δ and the w_i.

- **Choice graphs**. Choose m. Let $t \leq m/2$. Then the nodes of the graph are the $\binom{m}{t}$ t-element subsets of (1...m). Two nodes have an edge between them (with weight 1) if the size of their intersection is exactly b for some small b. Note that you probably cannot let m get too big and that b will have to be small in order for this to be practical.

6.0 Mechanics

You can write your program in any language you want. We are providing the code for semidefinite programming as a shared library callable from C or C++ and as a Java class. If you want to use some other language you will have to interface to this code yourself. (I would not suggest that you try to write your own semidefinite programming solver — it would be hard to get done in a month, much less a week.) We are providing the libraries on the departmental suns. If you must use another machine for this project, we will provide the source for our library and the other libraries that it uses. Note here that some of these libraries require a Fortran compiler and that we will make no guarantees that things will work.

The header file `semidef.h` and the library `libsemidef.so` are both located in `/map/aux1fred/comps/semidef` in the `include` and `lib` directories respectively. The source as we have it is in the `src` directory. For Java, the directory `/map/aux1fred/comps/java` should be added to the `CLASSPATH` and the class is accessed as `brown.comps.SemiDef` (it should be a self-loading native method).

You should hand in your program, sample runs, and whatever output you have to demonstrate your conclusions about the different algorithms on the different classes of graphs. You should also include a brief write-up that describes your own algorithm. Handins should be done electronically if at all possible into the directory `/map/aux1fred/comps/handins/###` where ### is the random number you assign yourself. You should have permissions to create this directory and should protect it accordingly.

All work that you hand in is expected to be your own. You should not discuss your work or ideas with anyone else. You should not share code or ask others for advice on your code. You should not read others code (whether those here or code obtained elsewhere) that duplicates in any way what you are to write. You may look at the source of the semidefinite solver that is provided if you think that would be helpful. You may ask Dr. Reiss questions either in person or via email. Any answers or information that should be known to all will be posted in the comps news group which is your responsibility to read.

7.0 Caveats

This problem involves a bit of mathematics (not all that much) and a bit of numerical analysis. You should be careful when dealing with floating point numbers. The values you get back from the semidefinite programming solver, for example, will not be exact (we are only giving you three or four digits accuracy). It is possible then for the Cholesky decomposition to then get negative numbers where the numbers should be zero or close there to, and the system does not like taking square roots of negative numbers.