Paradigms of Computing

- Centralized computation: The RAM machine
  - One processor: The one we have seen so far
- Parallel computation
  - Several processors, shared memory
- Distributed computation
  - Several processor, exchange of messages
Networks

- Example of distributed computing
- $n > 1$ nodes are joined by $m$ links
  - $m$ is usually $> n-1$
- To perform a common task the nodes exchange messages
  - Often the messages are packetized
- Another measure of complexity: Message complexity
Applications of Graphs

- Graphs are often used to model everyday problems.
- The Internet can be modeled as a graph:
  - Routers and computers on the Internet are nodes.
  - Links between the nodes are the edges.
- Graphs are used to represent networks.
Unit Disk Graphs

UDGs are a particular kind of graphs

- A graph is a UDG if and only if its vertices can be put in a one to one correspondence with equisized circle in a plane so that two vertices are joined by an edge if and only if the corresponding circles intersect
- Tangent circles intersect
- The radius of each circle (disk) is assumed to be 1
Networks and UDGs

The kind of networks modeled by UDGs are wireless networks

- Two nodes are joined by a wireless link if and only if they are in each other transmission range
- We need generalized disk graphs (radius > 1)
- Typically corresponding to the maximum range of a specific technology
Modeling Wireless Networks

What kind of wireless networks?
- Cellular networks
- Ad hoc networks

UDGs are best suited for ad hoc networks
- Each node is a router
- There is no general rule for their distribution on the plane/space
Typical Use for Ad Hoc Networks

- Disaster recovery
- Military networks
- Sensor networks
- Monitoring/Surveillance
- Law enforcement
- Vehicular technology
Generating Random Ad Hoc Topology

- We use graphs
- We generate:
  - n random points in a given area (say, a square in the plane)
  - We determine which nodes are neighbors based on a TX range \( r > 0 \)
  - We build the adjacency list or the corresponding adjacency matrix (network topology graph)
A C++ Program for Topology Generation, 1

```cpp
int main() {
    // MAIN VARIABLES of interest (possibly from the command line)
    // File name
    char fileNameCoord[80];
    // Total number of nodes in the network
    int nn = 5;
    // The "world" is a rectangle grid of MAXROWxMAXCOL size.
    // 1 grid unit = 1m
    int mrow = 500;
    int mcol = 500;
    // Transmission power of nodes in grid units (m)
    float tx = 30.0;
    // The number of topologies generated for the current run
    unsigned int topologies = 3;
```
unsigned int nets = 0;
// Main loop
while ( nets < topologies ) {
  vector< float > X( nn, -1.0 );
  vector< float > Y( nn, -1.0 );
  initCoord( nn, mrow, mcol, X, Y );
  // This topology adjacency list
  vector< deque< int > > G( nn );
  int md = 0;
  float ad = 0.0;
  int ll = 0;
  buildRep( nn, X, Y, tx, G, md, ad, ll );
if ( isConnected( nn, G ) ) {
    cout << nets << " " << md << " " << ad << " " << ll << endl;
    // Here file name construction statements
    for( int i = 0; i < nn; i++ )
        for( unsigned int j = 0; j < G[ i ].size(); j++ )
            outFileCoord << i << " " << G[ i ][ j ] << endl;
    outFileCoord.close();
    nets++;
} // End of if isConnected
} // End of topologies
return 0;
} // End main
// Initialization of the nodes' coordinates

void initCoord( int n, int mr, int mc, vector< float >& x, vector< float >& y ) {
    // Random stream for the positions
    static Rstream pos( 477 );

    for( int i = 0; i < n ; i++ ) {
        x[ i ] = pos.uniform( 0.0, (double)( mr ) );
        y[ i ] = pos.uniform( 0.0, (double)( mc ) );
    }
}

} // End initCoord
// Distance (here Euclidean)
float distance( float x1, float y1, float x2, float y2 ) {
    return sqrt( pow( x1-x2, 2 ) + pow( y1-y2, 2 ) );
}

// Neighbor test
bool isNeighbor( float x1, float y1, float x2, float y2, float txp ) {
    return ( distance( x1, y1, x2, y2 ) <= txp );
}

// End isNeighbor
void buildRep( int n, vector< float > x, vector< float > y, float tpx,
               vector< deque< int > > &adjl, int &mDeg, float &aDeg, int &l ) {

    int nl = 0; // Counts the links
    // Build up the adj. list
    for( int a = 0; a < n; a++ )
        for( int b = a + 1; b < n; b++ )
            if ( isNeighbor( x[ a ], y[ a ], x[ b ], y[ b ], tpx ) )
            {
                (adjl[ a ]).push_front( b );
                (adjl[ b ]).push_front( a );
                nl++;
            }
// Compute the max degree and avg. degree of the network
int sumDeg = 0;
for ( int c = 0; c < n; c++ ) {
    sumDeg += adjl[ c ].size();
    if ( mDeg < (int)adjl[ c ].size() )
        mDeg = (int)adjl[ c ].size();
}
aDeg = (float)sumDeg / n;

// Number of links
l = nl;

} // End buildRep
void DFS( int v, vector< deque< int > > al, vector< unsigned int > &c, int &vis ) {
    c[ v ] = 1; // The node v is grayed
    for( unsigned int a = 0; a < al[ v ].size(); a++ )
        if ( c[ al[ v ][ a ] ] == 0 )
            DFS( al[ v ][ a ], al, c, vis );
    c[ v ] = 2; // The node v is blackened
    vis++;
} // End DFS

bool isConnected( int n, vector< deque< int > > adjL ) {
    vector< unsigned int > color( n, 0 ); // 0=white, 1=gray, 2=black
    int visited = 0;
    DFS( 0, adjL, color, visited );
    return ( visited == n );
} // End isConnected
Assignments

* Updated information on the class web page:
  www.ece.neu.edu/courses/eceg205/2004fa