John Beighle
CSCE 686
HW4

a). Task: Develop the PD/AD design using the ordering and pivoting heuristics for an MIS/Clique algorithm implemented in a different programming language. Of course, include standard search elements throughout the new heuristic design development as appropriate (including implementation).

PD/AD Approach and Implementation
Note: Task objective is Maximum Clique specific algorithm and focus versus Maximum Independent Set with the understanding the two are interchangeable with graph compliments. The source and executable (Linux only) for this solution are available here:
https://github.com/beighle/AFIT686/blob/master/bronkerbosch.zip

Augmented design elements utilize mapping equivalence symbol notation: P=Q+, R=Q^-, X=Q^+, Gamma=N

AD Design Specification Requirement (3) Additions to Next-State Generator and Selection
Subtask a1: Discuss "vertex ordering"

Selection step is changed to implement a degeneracy ordering. This is an ordering on the vertices that results from repeatedly removing a vertex of minimum degree in the graph. The degeneracy is the smallest number d such that every vertex has at most d neighbors that are later than it in the ordering. Among all possible orderings, the degeneracy orderings minimize this number.

New Step 2: Compute Degeneracy

Subtask a2: Discuss "pivoting"

The role of pivoting is two-fold. Primarily, it promotes dividing the search space into branches that are likely to conflict with one another, meaning that they yield different solutions. As a result, pivot selection can allow earlier pruning of non-productive branches from the search space. Secondly, the choice of pivot may also reduce the branching factor such that the search tree is narrow and deep, rather than wide and shallow. Keeping the branching factor small favors the possibility of pruning larger sections of the search space. This is the intention of the pivot selection strategy widely attributed to Tomita et al. [6].

New Step 2: chooses $u \in P \cup X$ maximizing $|P \cap N(u)|$

Any maximal clique must include either u or one of its non-neighbors, for otherwise the clique could be augmented by adding u to it. Therefore, only u and its non-neighbors need to be tested as the choices for the vertex v that is added to R in each recursive call to the algorithm.

"This pivot rule has been introduced by Bron and Kerbosch (1973) and ensures that when the algorithm searches for maximal cliques that sets containing v are listed only once and not multiple times. Tomita et al. (2006) proved that this strategy reduces the running time to O(3n/3), where n is the number of vertices in the input graph. This is optimal as a function of n because there are graphs with 3n/3 cliques in a graph (Moon and Moser, 1965). Cazals and Karande (2008) provide a coherent overview of the work of Bron and Kerbosch, Koch and Tomita et al..” [5]

AD Design Specification Requirement (3) Additions to Solution
Subtask a3: Discuss “pivoting” with math/symbols and algorithmic process

New Step 2: chooses $u \in P \cup X$ maximizing $|P \cap N(u)|$. As explained above below is the pseudocode.

```
ChoosePivot(S, P, X) is
  if (P != 0)
    let $q \in P \cup X$ maximizing $|P \cap N(q)|$
    return $P - N(q)$ # or simply $P \cap K(q)$
  else
    return 0
```

AD Design Continuing Refinement (4) Additions to Next-State Generator
Subtask a4: Discuss “vertex ordering” with COMPLETE math/symbols and process

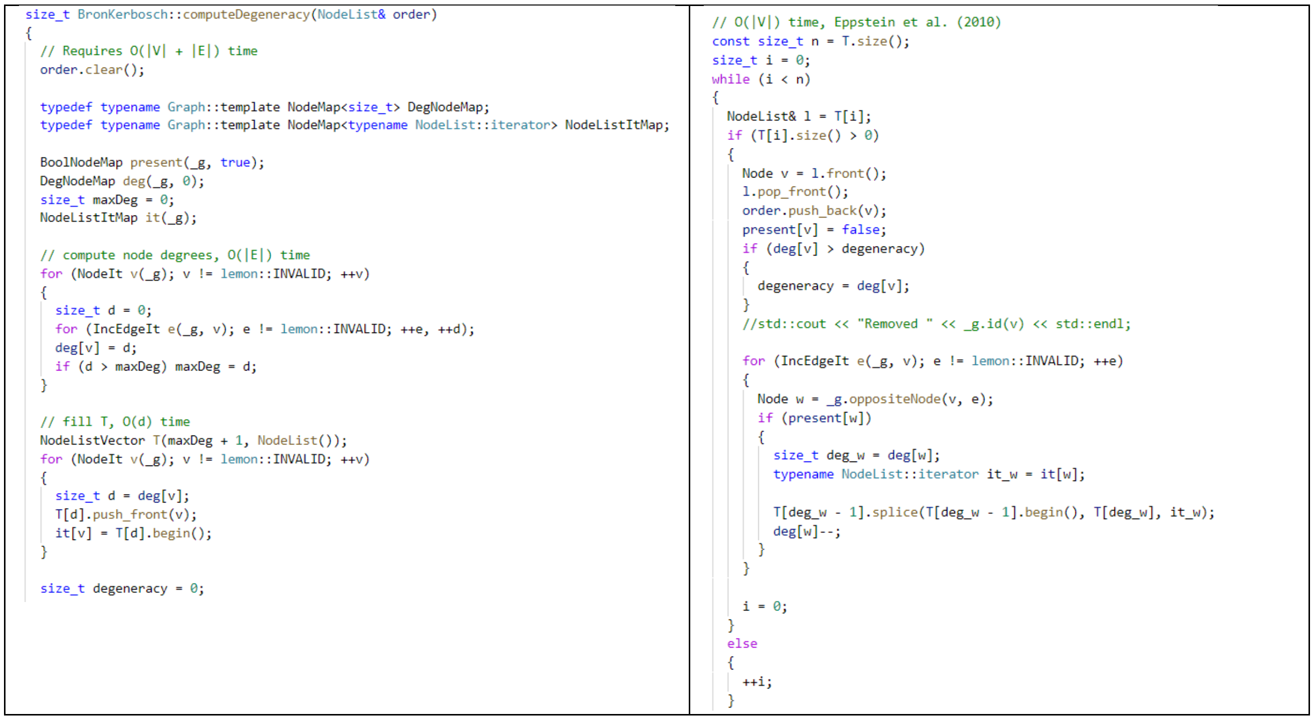
New Step 2: Compute Degeneracy Ordering. As explained above below is the pseudocode.

```
$G \leftarrow$ ComputeDegeneracy(1, n[, m])
clear order
while $\Delta(G) > 0$ do
        $u \leftarrow$ smallest vertex with $d(u) < k$
        $v \leftarrow$ largest vertex with $d(v) > k$
        $x \leftarrow$ smallest vertex in $N(v) \setminus N(u)$
        $G \leftarrow (G - \{v, x\}) + \{u, x\}$
return G
```

Functional Algorithm Specification for Clique (5)
Subtask a5: Add “vertex ordering” with math/symbols and complete algorithmic process in proper position.

As seen from the implementation below the direct mapping from pseudocode.

John Beighle
CSCE 686
HW4

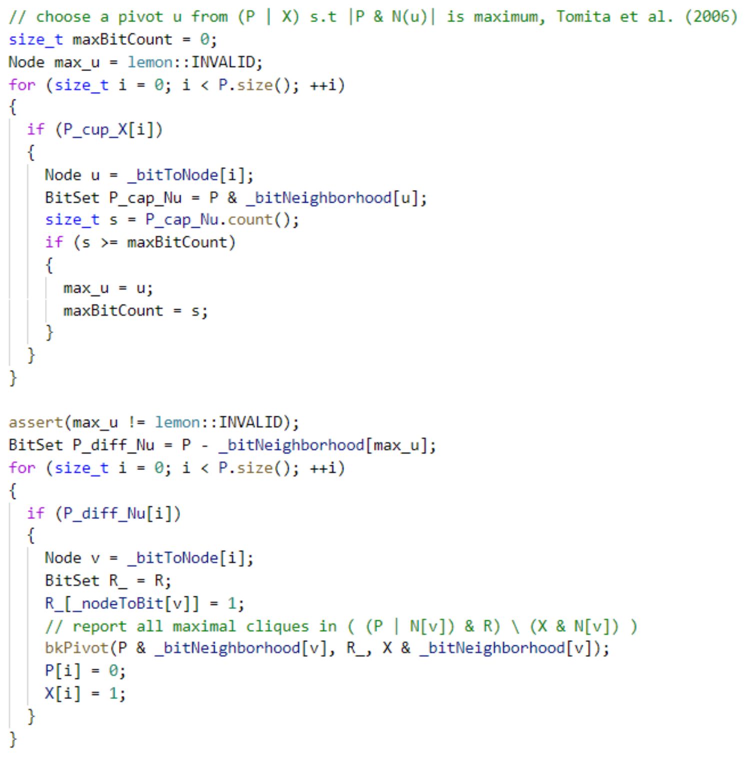


```
size_t BronKerbosch::computeDegeneracy(NodeList& order)
{
  // Requires O(|V| + |E|) time
  order.clear();

  typedef typename Graph::template NodeMap<size_t> DegNodeMap;
  typedef typename Graph::template NodeMap<typename NodeList::iterator> NodeListItMap;

  BoolNodeMap present(_g, true);
  DegNodeMap deg(_g, 0);
  size_t maxDeg = 0;
  NodeListItMap it(_g);

  // compute node degrees, O(|E|) time
  for (NodeIt v(_g); v != lemon::INVALID; ++v)
  {
    size_t d = 0;
    for (IncEdgeIt e(_g, v); e != lemon::INVALID; ++e, ++d);
    deg[v] = d;
    if (d > maxDeg) maxDeg = d;
  }

  // fill T, O(d) time
  NodeListVector T(maxDeg + 1, NodeList());
  for (NodeIt v(_g); v != lemon::INVALID; ++v)
  {
    size_t d = deg[v];
    T[d].push_front(v);
    it[v] = T[d].begin();
  }

  size_t degeneracy = 0;
```

```
  // O(|V|) time, Eppstein et al. (2010)
  const size_t n = T.size();
  size_t i = 0;
  while (i < n)
  {
    NodeList& l = T[i];
    if (T[i].size() > 0)
    {
      Node v = l.front();
      l.pop_front();
      order.push_back(v);
      present[v] = false;
      if (deg[v] > degeneracy)
      {
        degeneracy = deg[v];
      }
      //std::cout << "Removed " << _g.id(v) << std::endl;

      for (IncEdgeIt e(_g, v); e != lemon::INVALID; ++e)
      {
        Node w = _g.oppositeNode(v, e);
        if (present[w])
        {
          size_t deg_w = deg[w];
          typename NodeList::iterator it_w = it[w];

          T[deg_w - 1].splice(T[deg_w - 1].begin(), T[deg_w], it_w);
          deg[w]--;
        }
      }

      i = 0;
    }
    else
    {
      ++i;
    }
```

Subtask a6: Add “pivoting” with math/symbols and complete algorithmic process in proper position

As seen from the below code segment the mapping from pseudocode.

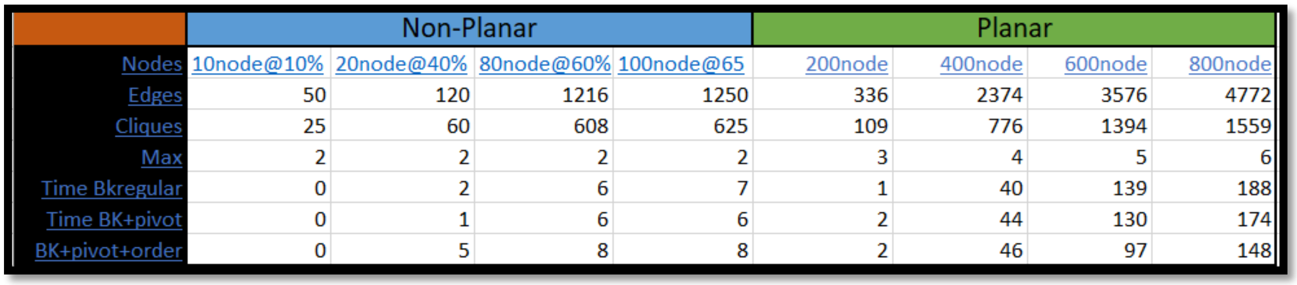


```
// choose a pivot u from (P | X) s.t |P & N(u)| is maximum, Tomita et al. (2006)
size_t maxBitCount = 0;
Node max_u = lemon::INVALID;
for (size_t i = 0; i < P.size(); ++i)
{
  if (P_cup_X[i])
  {
    Node u = _bitToNode[i];
    BitSet P_cap_Nu = P & _bitNeighborhood[u];
    size_t s = P_cap_Nu.count();
    if (s >= maxBitCount)
    {
      max_u = u;
      maxBitCount = s;
    }
  }
}

assert(max_u != lemon::INVALID);
BitSet P_diff_Nu = P - _bitNeighborhood[max_u];
for (size_t i = 0; i < P.size(); ++i)
{
  if (P_diff_Nu[i])
  {
    Node v = _bitToNode[i];
    BitSet R_ = R;
    R_[_nodeToBit[v]] = 1;
    // report all maximal cliques in ( (P | N[v]) & R) \ (X & N[v]) )
    bkPivot(P & _bitNeighborhood[v], R_, X & _bitNeighborhood[v]);
    P[i] = 0;
    X[i] = 1;
  }
}
```

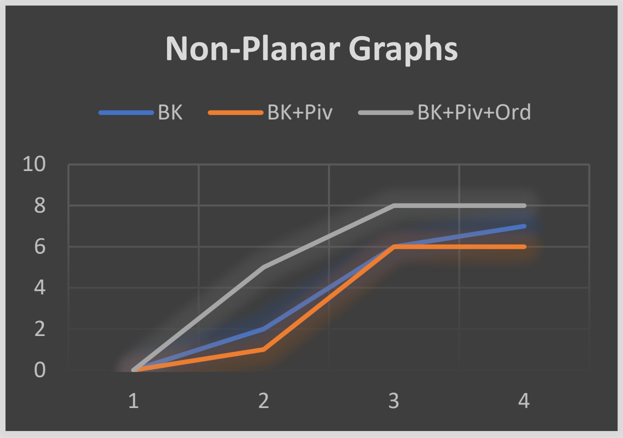
John Beighle
CSCE 686
HW4

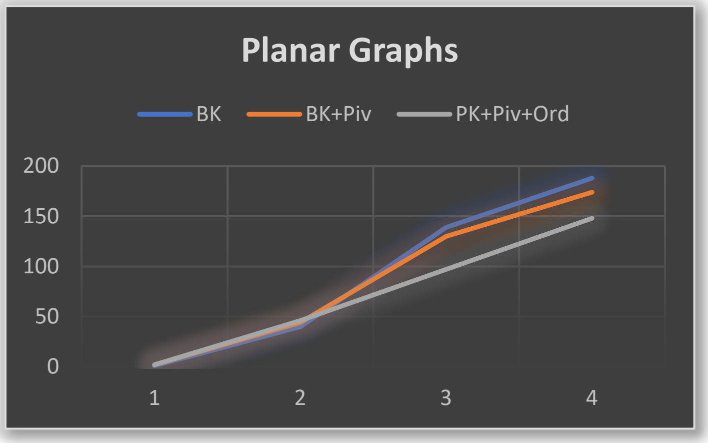
b). Task: Test and compare results with and without added Heuristics.

In terms of optimality Tomita et al. (2006) thoroughly covers the solution improvements that can be gained from these to BK optimizations. Theoretical best cases of O(3n/3). What we see below with limited data analysis is some overhead to the specific solution space dimension in preprocessing with ordering spcecifically. These overhead computational penalties are included in the sourcecode lists above. E.g. O(|V| + |E|) clear, ComputeNodeDegree O(|E|) , filling nodelists + O(d) and O(|V|). The charts below portray these facts by BKregular performing better on some graphs due to this additional processing. You will see the algorithm go to additional backtracking trouble with the Planar graphs as the processing time in general is much higher. However, the optimizations are starting to become clear in this space in the last two trials. It is clear that here the optimizations are beginning to pay dividends. The search space dimension is the number of solutions using the algorithm for n nodes best case O(n) and worst O(2^n).

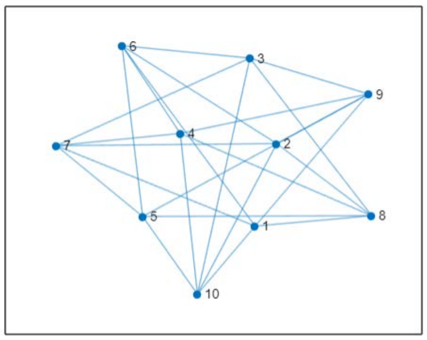
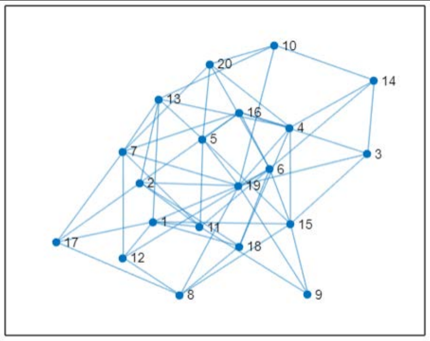
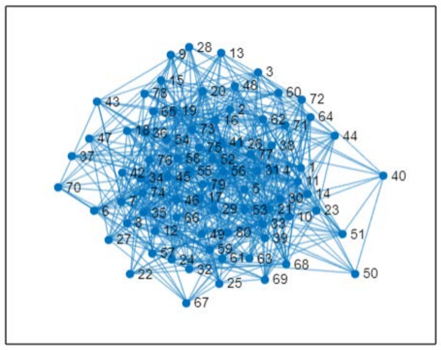
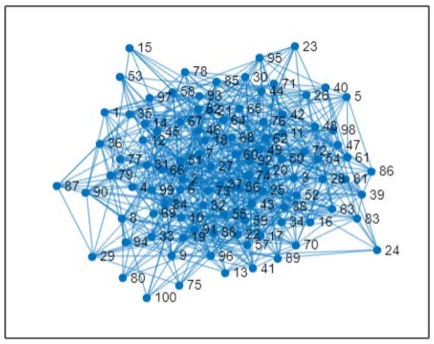
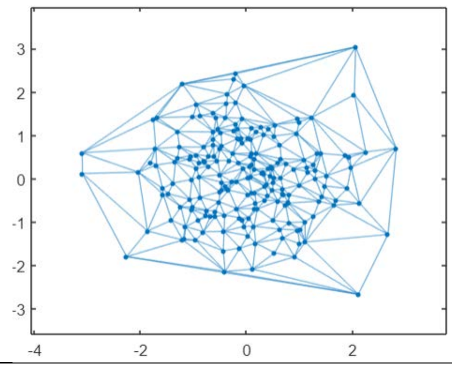
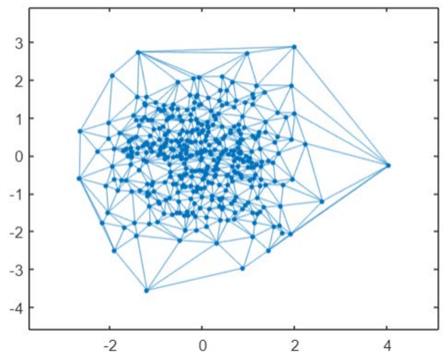
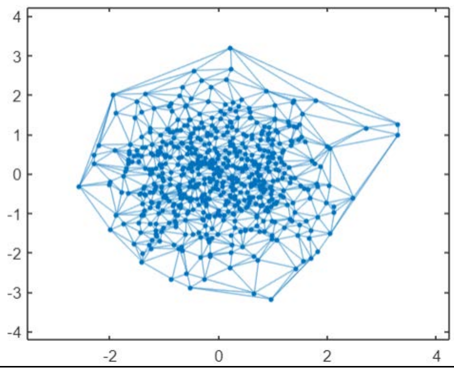
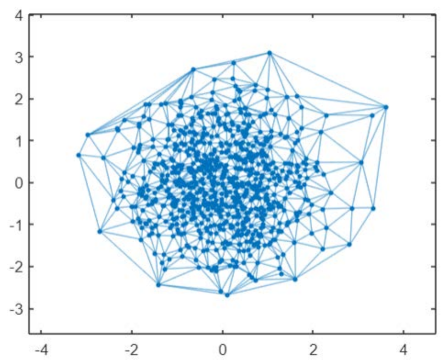


	Non-Planar				Planar			
Nodes	10node@10%	20node@40%	80node@60%	100node@65	200node	400node	600node	800node
Edges	50	120	1216	1250	336	2374	3576	4772
Cliques	25	60	608	625	109	776	1394	1559
Max	2	2	2	2	3	4	5	6
Time Bkregular	0	2	6	7	1	40	139	188
Time BK+pivot	0	1	6	6	2	44	130	174
BK+pivot+order	0	5	8	8	2	46	97	148





John Beighle
CSCE 686
HW4
Appendix A: Planar and Non-Planar Graphs used in this experiment

Nonplanar 10 Node @ 10% Saturation	Nonplanar 20 node @ 40%
	
Nonplanar 80 node @ 60%	Nonplanar 100 node @ 75%
	
Planar 200 node	Planar 400 node
	
Planar 600 node	Planar 800 node
	

John Beighle
CSCE 686
HW4

References:

[1] https://en.wikipedia.org/wiki/Tur%C3%A1n_graph

[2] *Bron-Kerbosch Algorithm* http://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm

[3] *DIMACS,* MIS/Clique Benchmarks http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark

[4] *Talbi:* Section1.7.2.1/.2/.3

[5] Enumerating common molecular substructures Martin S. Engler, Mohammed El-Kebir, Jelmer Mulder, Alan E. Mark, Daan P. Geerke, and Gunnar W. Klau