Mark Demore, 2d Lt

CSCE 686 – Dr. Lamont

Homework 4

**MIS Global Depth-First-Search Back-Tracking Design with Pivoting and Vertex Ordering Heuristics**

**MIS Problem Domain & Algorithm Domain Formal Development**

**1. Problem Domain Requirements Specification form: (Christofides, p31)**

- domains, D

input Di - Graph G(X,Γ), X:vertices. Γ: vertex link set (adjacency)

output Do - Maximal (Maximum) Independent sets per Christofides

- I(x); input conditions on input domain satisﬁed; x in X, link in Γ

- O(x,z); output conditions on output/input domain satisﬁed; i.e.,

a feasible/optimal solution with respect to the input domain   
-- S intersection Γ(S) = φ; independent set S in X (PD eqn 3.1 Christofides)  
-- H intersection Γ(H) not = φ for all H set in S; maximal independent set z,   
 (PD eqn 3.2 Christofides)  
-- max |S| of maximal independent sets is maximum independent set(s)   
 (PD eqn 3.3 Christofides)

1. **Problem Domain/Algorithm Domain Integration Specification**  
     
   *”Integrate MIS problem domain with gs-dfs/bt algorithm domain”*

* **Basic search constructs** for gs-dfs/bt (a tree search by construction!)

* *next-state-generator* (Di) − > x in X; I(x)
* *selection* (Di) − > x; x in X (usually from an ordered/sorted set based   
   explicitly/implicitly MIS criteria-desire terminal nodes to be MIS

(call the set Qk = X for each level/search-stage k of search tree!)

* *feasibility* (x, Dp) − > boolean (if true union (x, S)), S intersection Γ(S) = φ; independent set S in X
* *solution* O(x,z) “maximal “; (Dp) − > boolean; z = Dp, i.e., can no longer   
   augment S with an x in X;
* *objective (*Dp*) ->* Do *“ordered set/*[*well founded set*](https://en.wikipedia.org/wiki/Well-founded_relation) *of MI sets is regt’d”*
* **Delay Termination** from gs-dfs/bt
* *Find all* maximal independent solutions within tbd designed *loop*
* *Generate* via gs-dfs/bt all MIS solutions without duplication!
* imports: ADT( set, set-of-sets):Di Dp Do; Boolean; integer

- *Comment:*   
A) need a speciﬁc function/algorithm (unknown) that maps input domain to output domain

B) can explicitly deﬁne axioms, A; i.e., deﬁne input/output general requirements logically for testing algorithm (including exceptions)

C) consider “better” ordering in the set of candidates based upon the # of vertex connections, …. (vertex ordering)

**3. Algorithm Domain Design Speciﬁcation Refinement**

* *Possibly Sort a priori nodes/vertices in* the set of candidates *Qk based upon # of connections to other nodes? How to handle (store, process) PDs with very large number of nodes? Distributed or parallel computation?*
* *Creative data structure augmentation* of the set of candidates Qk  into Q+k and Q-k in gs-dfs/bt that provides for *generating sets without duplication*; a search tree vs. search graph (Christofides, pp 33-34; “Bron-Kerbosch Algorithm”)
* Observe that k is the stage index (level in gs-dfs/bt search tree): S = Sk is defined as the independent set of PD graph vertices at stage k in the tree search; Sk is a partial MIS solution.
* ***Next-State Generator and Selection:*** (CREATIVE!)
  + Q+k : set of vertices not selected previously at state (level) k or higher in search tree to augment Sk : updated with forward search *selecting* xik from Q+k ; Q+k+1 = Q+k – Γ(xik)- {xik}, *(AD eqn 3.6 from PD eqn 3.1*- Christofides)
* ***Vertex ordering uses the degeneracy ordering of a graph to feed to the Bron Kerbosch algorithm. The degeneracy ordering orders the vertices based on the degree of the node, its number of neighbors.***
  + ***P = V(G)***
  + ***R = X = empty***
  + ***For each vertex in degeneracy ordering of G***
* ***Feasibility*** (CREATIVE!)
  + Q-k set of vertices which have been selected previously at state k – 1 or higher in search tree to augment Sk; removal of Γ(xik ) and xik added when backtracking from Q−k  (Q-k+1 = Q-k – Γ(xik) ) where Γ(xik) = vertices adjacent to xik ). *This is a very creative selection of a “reﬁned” data structure. (updated with equation 3.5 - Christofides with backward search when deselecting xik from Q+k ; addition of xik to Q-k and minus Γ(xik ).* *WHY?!* *Generates sets without duplication!*
* ***Solution:*** if Q+k = Q-k = : a set Sk is a MIS solution if it cannot be augmented further, and since sets are generated without duplication, Sk is a MIS solution if and only if Q+k = Q-k =   *“again a very creative insight from AD to PD!” – (indirectly from PD eqn 3.2; see Christofides for more discussion details)*
* ***Pivoting limits the search of the Bron Kerbosch algorithm by excluding neighbors of the pivot vertex from the search, since the neighbors of a node will not be part of an independent set with that node.***
  + ***For each vertex in P \ N(u)***
* Continuing program development by instantiating more gs-dfs/bt search elements for backtracking loop:
* *initialize* sets Sk = Q-k = , Q+k = X, k = 0.
* *loop*
* *next-state-generator* (Di) − > xik in Q+k ; I(x)
* *selection* (Di) − > xik; xik in Q+k (usually from an ordered\* set based explicitly/implicitly MIS criteria-desire terminal nodes to be MIS)

update Q+k+1 = Q+k – Γ(xik) - xik, ; Γ(xik) = vertices adjacent to xik

* *feasibility* Q-k+1 = Q-k – Γ(xik); (xik in Dp) − > boolean (if true union (xik, Sk)), Sk   
   intersection Γ(Sk) = φ; independent set Sk in Dp with Qk construction, only   
   feasible sets are generated!
* *solution* O(xik,z); (xik in Dp) − > boolean; z = Dp, i.e., can no longer   
   augment Sk with an xik in X; Q+k = Q-k = 
* *ﬁnd all* maximal independent solutions within *loop* by *backtracking*

\*Could be lexigraphical (Christofides); input/output degrees sorted, …

* imports: integer/real/character, BOOLEAN, ADT (Set, Set-of-Sets), ...

(list of other design speciﬁcations, ADTs-algebraic specs

* data dictionary (dfs local decision creativity!)

1. **Algorithm Domain Design Continuing Refinement**

* Design Speciﬁcation Name: (list of parameter speciﬁcations) domains: Di,Do“MIS gs-dfs/bt Program”  
   *[Christofides algorithm does not use a priori sorting or consider # of nodes]*
* *Creative* logic data structures Q+k and Q-k regarding backtracking condition
* ***Creative*** *early backtracking* If x in Q-k so that Γ(x)  Q+k = ; i.e., if for some x in Q-k exists for which Γ(x)  Q+k = , then regardless of which x vertex is taken from Q+k to augment Sk forward, x can never be removed from Q-k (*creative equation 3.8!*)
* gs-dfs search constructs and algorithmic operational process *(continue refinement)*
* *imports:* integer/real/character, BOOLEAN, ADT (SET, SET-OF-SETS, graph), ...

(list of other design speciﬁcations, ADTs-algebraic specs,

data dictionary (dfs local decision creativity!)

* *initialize* sets Sk = Q-k = , Q+k = X, k = 0.

*loop*

* *next-state-generator* (Di) − > xik in Q+k ; I(x)
* *selection* (Di) − > xik; xik in Q+k (usually from an ordered set based explicitly/implicitly MIS criteria-desire terminal nodes to be MIS)

update Q+k = Q+k - Γ(xik) - xik, ;Γ(xik) = vertices adjacent to xik

* *feasibility* Q-k = Q-k – Γ(xik); (xik in Dp) − > boolean (if true union (xik, Sk)), Sk   
   intersection Γ(Sk) = φ; independent set Sk in Dp with Qk construction, only   
   feasible sets are generated!   
   If xik in Q-k so that Γ(xik)  Q+k = , then backtrack
* *solution* O(xik,z); (xik in Dp) − > boolean; z = Dp, i.e., can no longer   
   augment Sk with an xik in X; Q+k = Q-k = 
* *backtrack to loop* until all possible combinations (states) are check implicitly or explicitly; backtrack to previous level k-1 search tree level   
   and loop; if all PD vertices have been used at the k = 0 level; i.e.,   
   Q+k =  for k = 0, then STOP.
* *axioms*: tbd (list of axioms relating parameters, types, imports, and operations) for all x in Di, if I(x) then there exists a function Fn(x) = z with z in Do that satisﬁes O(x,z); desired to find a specific function(x)/operational mapping.
* ***Comments:***   
  ***a***. Could put search construct flow in a table form for ease of understanding.

***b***. Observe that at this design level, the details of the functional implementa­tion are yet to be deﬁned; i.e., one must reﬁne the AD into a gs-dfs/bt low level design for mapping to a given computer language.   
  
***c.*** Also, the maximum independent set(s) of vertices may be required which would need a max set operation.

1. **Functional Algorithm Speciﬁcation for MIS gs-dfs/bt:**

*”Top-Down Design ﬂow into the Bron and Kerbosch Algorithm* *(Christoﬁdes)   
MIS dfs-bt search graph algorithm, gs-dfs/bt (page 35) [1,2]”*

“Functional MIS gs-dfs/bt Algorithm Psuedo code found in Christofides;  
 **Note:** algorithmic step-by-step math/symbolic notation! ”  
\*\*\*\*\*  
Name: **MIS gs-dfs/bt Algorithm** *(Christofides, Bron and Kerbosch)*  
*Declaration and Initialization*Step 0 *declaration: i*nteger/real/character, Boolean, ADT (set, set-of-sets), …

proc BronKerboschOrdering(V , E)

1: for each vertex vi in a degeneracy ordering v0, v1, v2, . . . of (V, E) do

2: P ← Γ(vi ) ∩ {vi+1, . . . , vn−1 }

3: X ← Γ(vi ) ∩ {v0, . . . , vi−1 }

4: BronKerboschPivot(P, {vi }, X)

5: end for

**Step 1** *Initiation*: Set Sk = Q-k = , Q+k = X, k = 0.  
*Forward Step* (dfs loop)   
**Step 2** *Selection:* Choose a vertex xik in Q+k, Sk+1 = Sk  xik, k = k + 1   
 Update Q+k+1 = Q+k - Γ(xik) - xik, where Γ(xik) = vertices adjacent to xik*Test*  
**Step 3** *Feasibility:* Q-k+1 = Q-k – Γ(xik). If xik in Q-k so that Γ(xik) Q+k = , go to Step 5,   
 else go to step 4  
**Step 4** *Solution:* If (Q+k = Q-k = ) then PRINT MIS Sk, go to Step 5, If Q+k =  and   
 Q-k not =  go to Step 5, else go to Step 2.  
*Backtrack* **Step 5** *Loop Backtrack:* Set *k = k - 1*. Sk = Sk+1 - xik, Q+k = Q+k - xik, Q-k = Q-k + xik,   
 if k = 0 and Q+k = , STOP, else go to step 3 (dfs loop).

proc BronKerboschPivot(P, R, X)

1: if P ∪ X = ∅ then

2: report R as a maximal clique

3: end if

4: choose a pivot u ∈ P ∪ X to minimize |P \ Γ(u)|

5: for each vertex v ∈ P \ Γ(u) do

6: BronKerboschPivot(P ∩ Γ(v), R ∪ {v}, X ∩ Γ(v))

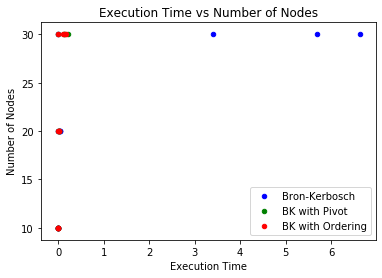
7: P ← P \ {v} 8: X ← X ∪ {v} 9: end for T  
\*\*\*\*\*  
**6. Mapping to chosen computer language**

Source code for the implementation of the Bron-Kerbosch algorithm as well as with pivoting and vertex ordering in the Python programming language is enumerated in Appendix A. It is also available at github.com/mdemore2.

1. **Test and Evaluation Report of Software Execution**

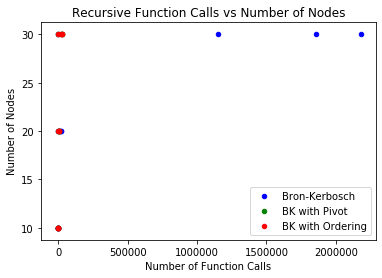
The Bron-Kerbosch algorithm was implemented in Python. In order to use the Bron-Kerbosch algorithm to calculate the MIS of a graph, the input graph is complemented before it is used in the Bron-Kerbosch algorithm. The maximum clique of the graphs complementary graph is its maximum independent set. The original Bron-Kerbosh algorithm was implemented, as well as one using the pivoting heuristic and another using the vertex ordering heuristic. The algorithms were tested using 4 graph types and 3 graph sizes. The graph types were: complete, cycle, path, and wheel. The graph sizes were: 10 nodes, 20 nodes, and 30 nodes.

The plot below shows the execution time based on the number of nodes, points are colored with the different algorithms that were used.

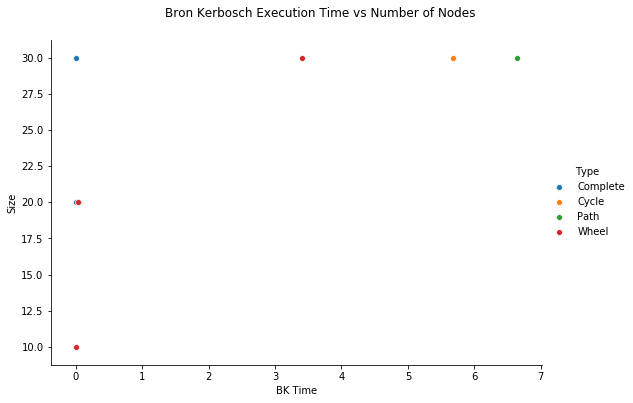
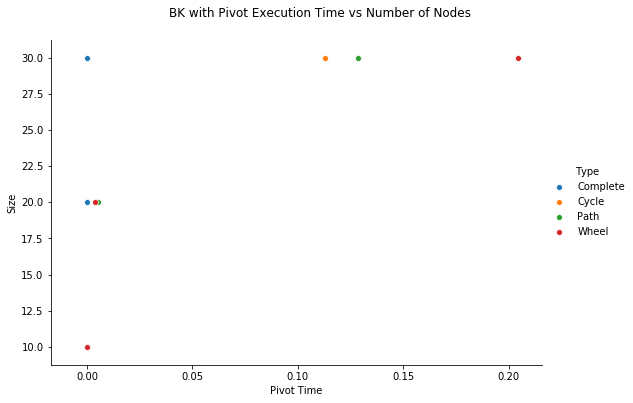
****

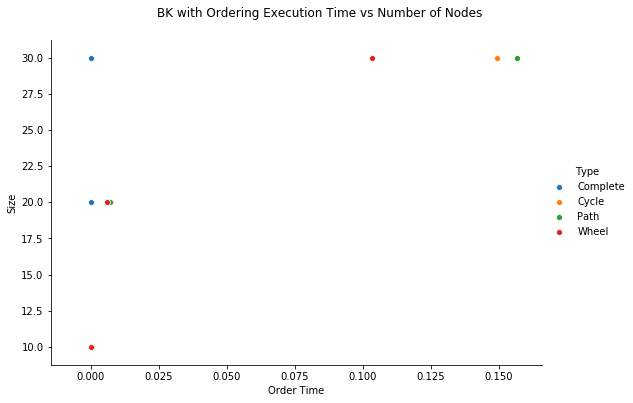
The above plot shows that time does increase slightly with the size, but more importantly, that the original algorithm takes much longer than it does with the added heuristics.

The plot below shows the number of recursive calls to the algorithm based on the number of nodes, and points are colored with the different algorithms that were used.

****

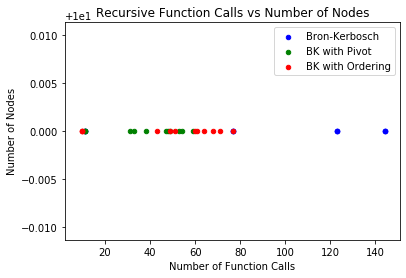
The above plot shows that the number of recursive calls to the algorithm also increases with graph size, but again more importantly that the original algorithm makes much more calls than the algorithm with the added heuristic.

The three plots below show the execution time for each of the algorithm relative to the size of the graph, and enumerates the types of the graph through the color of the points. ** **

****

The above graphs show that size increases the time for each of the algorithms, and also that certain graph types yield worse performance, namely the path, cycle, and wheel graphs which have larger independent sets and whose connectedness create more for the algorithm to search through, particularly when excluding neighbors.

When referencing the scaling of the last three graphs, it is easy to note that the algorithm with vertex ordering performs the fastest and with the least calls. While both algorithms with heuristics perform much better than the original algorithm. This is also show in the graph below, showing number of function calls for graphs of only size 10 nodes, again across the 4 different graph types.

****

This result makes sense because the complexity of the Bron-Kerbosch algorithm is O(3*n*/3) in the worst case, because a graph with n-vertices can have at most 3*n*/3 maximal cliques. The pivot can improve upon this, but worst-case complexity remains O(3*n*/3). The vertex ordering version of the algorithm can reduce the complexity to O(*dn*3*d*/3) where d is the degeneracy of the graph. Therefore, the vertex-ordering approach is particularly efficient for sparse graphs, but both heuristics remain effective in improving upon the original algorithm.

***References:***

[1] Nicos Christofides. *Graph theory: An algorithmic approach (Computer science and applied mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1975.

[2] Alessio Conte, [*Review of the Bron-Kerbosch algorithm and variations*](http://www.dcs.gla.ac.uk/~pat/jchoco/clique/enumeration/report.pdf), Univ of Glasgow,   
School of Computing Science, May, 2013

[3] Edward Reingold, Jurg Nieverelt, and Narsing Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, 1977.

[4] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. *The worst-case time complexity for generating all maximal cliques and computational experiments.* **Theor. Comput. Sci.,** 363:28–42, October 2006.

[5] El-Ghazali Talbi. *Metaheurisics From Design to Implementation*. Wiley and Sons, 2009

[6] Michaelewicz and Fogel, *How to Solve it: Modern Heuristics*, 2ed, Springer, ‘04

[7] Wikipedia, MIS and Clique

[8] Robson, [*Algorithms for Maximum Independent Sets*](https://www.cs.umd.edu/~gasarch/TOPICS/sat/robson.pdf), Journal of Algorithms 7, pp 425-440, 1986

[9] <https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm>

[10] <https://iq.opengenus.org/bron-kerbosch-algorithm/>

[11] <https://www.ics.uci.edu/~goodrich/teach/graph/notes/Strash.pdf>

[12] Manoussakis, George. “The Bron-Kerbosch Algorithm with Vertex Ordering Is Output-Sensitive.” *ArXiv:1911.01951 [Cs]*, Nov. 2019. *arXiv.org*, <http://arxiv.org/abs/1911.01951>.

**Appendix A.**

import networkx as nx

import numpy as np

import matplotlib.pyplot as plt

import random

from timeit import default\_timer as timer

import pandas as pd

from pandas.plotting import scatter\_matrix

import seaborn as sns

def bk(graph, P, R=set(), X=set()):

bk.count += 1

if not any((P, X)):

yield R

else:

for node in P.copy():

for r in bk(graph, P.intersection(graph.neighbors(node)),

R=R.union(set([node])), X=X.intersection(graph.neighbors(node))):

yield r

P.remove(node)

X.add(node)

def bk\_pivot(graph, P, R=set(), X=set()):

bk\_pivot.count += 1

if not any((P, X)):

yield R

else:

P\_copy = P.copy()

pivot = random.choice(list(P.union(X)))

for node in P\_copy.difference((graph.neighbors(pivot))):

for r in bk\_pivot(graph, P.intersection(graph.neighbors(node)),

R=R.union(set([node])), X=X.intersection(graph.neighbors(node))):

yield r

P.remove(node)

X.add(node)

def bk\_order\_pivot(graph, P, R=set(), X=set()):

bk\_order\_pivot.count += 1

if not any((P, X)):

yield R

else:

P\_copy = P.copy()

pivot = random.choice(list(P.union(X)))

for node in P\_copy.difference((graph.neighbors(pivot))):

for r in bk\_order\_pivot(graph, P.intersection(graph.neighbors(node)),

R=R.union(set([node])), X=X.intersection(graph.neighbors(node))):

yield r

P.remove(node)

X.add(node)

def bk\_order(graph, P, R=set(), X=set()):

for node in P.copy():

for r in bk\_order\_pivot(graph, P.intersection(graph.neighbors(node), set(), set()),

R=R.union(set([node])), X=X.intersection(graph.neighbors(node))):

yield r

P.remove(node)

X.add(node)

def run\_test():

results = pd.DataFrame()

size\_l = (10, 20, 30)

type\_l = ('Complete', 'Cycle', 'Path', 'Wheel')

for type in type\_l:

for size in size\_l:

if type == 'Complete':

graph = nx.complete\_graph(size)

elif type == 'Cycle':

graph = nx.cycle\_graph(size)

elif type == 'Star':

graph = nx.star\_graph(size)

elif type == 'Path':

graph = nx.path\_graph(size)

else:

graph = nx.wheel\_graph(size)

graph\_o = graph

graph = nx.complement(graph)

bk\_list = list()

bk.count = 0

start = timer()

for clique in bk(graph, set(graph.nodes), set(), set()):

bk\_list.append(clique)

end = timer()

bk\_time = end - start

pivot\_list = list()

bk\_pivot.count = 0

start = timer()

for clique in bk\_pivot(graph, set(graph.nodes), set(), set()):

pivot\_list.append(clique)

end = timer()

pivot\_time = end - start

order\_list = list()

bk\_order\_pivot.count = 0

start = timer()

for clique in bk\_order(graph, set(nx.algorithms.coloring.strategy\_largest\_first(graph, 1))):

order\_list.append(clique)

end = timer()

order\_time = end - start

to\_add = pd.DataFrame({'Size': [size],

'Type': [type],

'Graph': [graph\_o],

'BK Calls': [bk.count],

'BK Time': [bk\_time],

'BK MIS': [bk\_list],

'Pivot Calls': [bk\_pivot.count],

'Pivot Time': [pivot\_time],

'Pivot MIS': [pivot\_list],

'Order Calls': [bk\_order\_pivot.count],

'Order Time': [order\_time],

'Order MIS': [order\_list]})

results = results.append(to\_add)

return results

def analyze(results):

#print(results.head)

#scatter\_matrix(results)

#results.plot.scatter(x='BK Time',y='Size',color='Blue',label='Bron-Kerbosch')

#plt.show()

#results.plot.scatter(x='Pivot Time',y='Size',color='Green',label='BK with Pivot')

#plt.show()

#results.plot.scatter(x='Order Time',y='Size',color='Red',label='BK with Ordering')

#plt.show()

sns.pairplot(x\_vars=["BK Time"], y\_vars=["Size"], data=results, hue="Type", height=5, aspect=1.5)

plt.show()

sns.pairplot(x\_vars=["Pivot Time"], y\_vars=["Size"], data=results, hue="Type", height=5, aspect=1.5)

plt.show()

sns.pairplot(x\_vars=["Order Time"], y\_vars=["Size"], data=results, hue="Type", height=5, aspect=1.5)

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

results = run\_test()

analyze(results)