2nd Lt David Crow

ENG/20M

CSCE 686 Advanced Algorithms, Homework 4b

­

**Problem 1**

Incorporate the ordering heuristic and the pivoting heuristic into the AFIT Graph Program (AGP) maximal independent set (MIS) function by explicitly following the top-down problem domain/algorithm domain (PD/AD) search element design approach. Show this PD/AD design process by integrating these creative heuristics top-down into the pseudocode. Extend the use of the standard search elements into the given AGP code. Additionally, extend the step notation from the pseudocode into the given code.

1. Problem Domain Requirements Specification form: (Christofides, p31)
   1. domains, D
      1. input Di - Graph G(X,Γ), X:vertices. Γ: vertex link set (adjacency)
      2. output Do - Maximal (Maximum) Independent sets
   2. I(x); input conditions on input domain satisﬁed; x in X, link in Γ
   3. O(x,z); output conditions on output/input domain satisﬁed; i.e., a feasible/optimal solution with respect to the input domain
      1. S intersection Γ(S) = φ; independent set S in X (PD eqn 3.1 Christofides)
      2. H intersection Γ(H) not = φ for all H set in S; maximal independent set z, (PD eqn 3.2 Christofides)
      3. max |S| of maximal independent sets is maximum independent set(s)   
          (PD eqn 3.3 Christofides)
2. Problem Domain/Algorithm Domain Integration Specification *”Integrate MIS problem domain with gs-dfs/bt algorithm domain”*
   1. **Basic search constructs** for gs-dfs/bt (a tree search by construction!)
      1. *next-state-generator* (Di) − > x in X; I(x)
      2. *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!)
      3. *feasibility* (x, Dp) − > boolean (if true union (x, S)), S intersection Γ(S) = φ; independent set S in X
      4. *solution* O(x,z) “maximal “; (Dp) − > boolean; z = Dp, i.e., can no longer augment S with an x in X;
      5. *objective (*Dp*) ->* Do *“ordered set/*[*well founded set*](https://en.wikipedia.org/wiki/Well-founded_relation) *of MI sets is regt’d”*
   2. **Delay Termination** from gs-dfs/bt
      1. *Find all* maximal independent solutions within tbd designed *loop*
      2. *Generate* via gs-dfs/bt all MIS solutions without duplication!
   3. imports: ADT( set, set-of-sets):Di Dp Do; Boolean; integer
   4. *Comment:*
      1. A) need a speciﬁc function/algorithm (unknown) that maps input domain to output domain
      2. B) can explicitly deﬁne axioms, A; i.e., deﬁne input/output general requirements logically for testing algorithm (including exceptions)
      3. C) consider “better” ordering in the set of candidates based upon the # of vertex connections, ….
3. Algorithm Domain Design Speciﬁcation Refinement
   1. *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?*
   2. General algorithm vs algorithm with heuristics
      1. General
         1. “Naïve” implementation
            1. Runs BK algorithm as described
         2. Pivoting heuristic
            1. Runs (1) but with pivoting heuristic
            2. Ensures we trim our search tree
            3. Selects random node for addition to current clique

As opposed to always selecting from left to right

* + - 1. Pivoting and ordering heuristics
         1. Runs (2) but with ordering heuristic
         2. Further trims the search tree for faster search
         3. Ordering heuristic

Sorts nodes in increasing order of neighbors

We should always try nodes that aren’t highly connected before trying other nodes

* 1. *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”)
  2. 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.
  3. ***Next-State Generator and Selection:*** (CREATIVE!)
     1. 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 = Q+k - Γ - xik, *(AD eqn 3.6 from PD eqn 3.1*- Christofides)
  4. ***Feasibility*** (CREATIVE!)
     1. 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  ( ) 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!*
  5. ***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)*
     1. [*One can add new gs-dfs/bt heuristics:* data structures, search constructs and algorithmic operational process refinements (improved program design in a program name: MIS gs-dfs/bt program?)]
        1. Pivoting heuristic
           1. Select random node to add next
           2. Reduces likelihood of unnecessarily repeating work
        2. Ordering heuristic
           1. Sort nodes in increasing order of neighbors
           2. Ensures we consider least-connected nodes first
           3. It’s likely that the independent sets don’t include highly-connected nodes
     2. *(Improved search = fewer search nodes/branches?)* [*NFL Thm*](https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf) *impact!?*
  6. Continuing program development by instantiating more gs-dfs/bt search elements for backtracking loop:
     1. *initialize* sets Sk = Q-k = , Q+k = X, k = 0.
     2. *Loop*
        1. *next-state-generator* (Di) − > xik in Q+k ; I(x)
        2. *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
        3. *feasibility* Q-k = Q-k – Γ(xik); , (xik, Dp) − > boolean (if true union (xik, Sk)), Sk intersection
           1. Γ(Sk) = φ; independent set Sk in Dp with Qk construction, only feasible sets are generated!
        4. *solution* O(xik,z); (Dp) − > boolean; z = Dp, i.e., can no longer augment Sk with an xik in X; Q+k = Q-k = 
        5. *ﬁnd all* maximal independent solutions within *loop* by *backtracking*
           1. \*Could be lexigraphical (Christofides); input/output degrees sorted, …
  7. imports: integer/real/character, BOOLEAN, ADT (Set, Set-of-Sets), ... (list of other design speciﬁcations, ADTs-algebraic specs
     1. data dictionary (dfs local decision creativity!)

1. Algorithm Domain Design Continuing Refinement
   1. 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]*
   2. *Creative* logic data structures Q+k and Q-k regarding backtracking condition
      1. *Creative early backtracking* If x in Q-k so that Γ  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!*)
   3. gs-dfs search constructs and algorithmic operational process *(continue refinement)*
   4. *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!)
   5. *initialize* sets Sk = Q-k = , Q+k = X, k = 0.
   6. *Loop*
      1. *next-state-generator* (Di) − > xik in Q+k ; I(x)
      2. *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, ;Γ(xik) = vertices adjacent to xik
      3. *feasibility* Q-k = Q-k – Γ (xik, 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
      4. *solution* O(xik,z); (Dp) − > boolean; z = Dp, i.e., can no longer augment Sk  with an xik in X; Q+k = Q-k = 
      5. *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.
   7. *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.
   8. ***Comments:***
      1. Could put search construct flow in a table form for ease of understanding.
      2. 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.
      3. Also, the maximum independent set(s) of vertices may be required which would need a max set operation.
2. Functional Algorithm Speciﬁcation for MIS gs-dfs/bt: *“Top-Down Design ﬂow into the Bron and Kerbosch Algorithm* *(Christoﬁdes) MIS search graph algorithm, gs-dfs/bt (page 35) [1,2]”* 
   1. “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), …**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 = Q+k - Γ - xik, where Γ = vertices adjacent to xik*Test*  
**Step 3** *Feasibility:* Q-k = Q-k – Γ , If xik in Q-k so that Γ  Q+k = , go to Step 5, else go to   
 step 4  
**Step 4** *Solution:* If (Q+k = Q-k = ) then PRINT MIS Sk, goto Step 5, If Q+k =  and   
 Q-k not =  goto Step 5, else goto 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 (dfs loop). else goto step 3.  
\*\*\*\*\*

1. Mapping to chosen computer language
   1. Cheol Kang’s 2014 BK-clique implementation in Python
      1. Computes maximal cliques
      2. By inverting the input graph beforehand, we can identify MISs
      3. Recursive implementation (as opposed to iterative)
   2. Formatting, output, and graph inversion added
   3. Files
      1. LICENSE
         1. Cheol Kang’s software license
      2. README.md
         1. A short description of the software package
      3. data.py
         1. Nine different search graphs for testing
         2. To test various graphs, follow instructions at the top of the file
      4. medium-nonplanar.txt
         1. Contains effective representation of search tree
            1. Not in standard search tree representation
            2. Easy to follow from one node to the next
            3. Backtracking clearly evident
            4. Identifies maximal independent sets
         2. Lists number of recursive calls for each version of BK
         3. Outputs execution time for each version of BK
         4. Outputs all maximal independent sets found
      5. reporter.py
         1. Class used to store and output all maximal independent sets
      6. mis.py
         1. Driver for the program
         2. Process
            1. Reads in input graph
            2. Inverts the graph
            3. Runs all three functions on graph and outputs results
         3. Computes execution time
      7. bron\_kerbosch1.py
         1. Naïve implementation of BK algorithm
      8. bron\_kerbosch2.py
         1. Naïve implementation of BK algorithm with pivoting heuristic included
      9. bron\_kerbosch3.py
         1. Naïve implementation of BK algorithm with pivoting and ordering heuristics included
   4. All code can be found in submission file

**Problem 2**

Test and compare MIS results with/without heuristics over some relatively small, medium, and large problem domain graphs. Consider planar, non-planar, perfect, circle, bipartite, permutation, and chordal graphs. Additionally, present the search tree for one medium graph.

To test the Python implementation (with and without heuristics), I created nine different graphs. Three are small, three are medium, and three are large. One of each size is planar, one of each size is nonplanar, and one of each size is bipartite. The graphs are summarized below:

|  |  |  |  |
| --- | --- | --- | --- |
| Graph | Graph Size | Vertices | Edges |
| Small planar | Small | 7 | 11 |
| Small nonplanar | Small | 4 | 6 |
| Small bipartite | Small | 8 | 11 |
| Medium planar | Medium | 10 | 12 |
| Medium nonplanar | Medium | 10 | 15 |
| Medium bipartite | Medium | 16 | 20 |
| Large planar | Large | 21 | 38 |
| Large nonplanar | Large | 26 | 60 |
| Large bipartite | Large | 24 | 28 |

After generating these graphs, I ran each of Kang’s functions on each graph. In other words, I tested each graph with the basic BK algorithm, with the BK algorithm with pivoting, and with the BK algorithm with pivoting and ordering. The execution times and clique results are summarized in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Graph | Clique Number | Cliques Found | Execution Time (s)  No Heuristics | Execution Time (s)  Pivoting | Execution Time (s)  Pivoting, Ordering |
| Small planar | 2 | 10 | 0.000117 | 4.196167e-05 | 6.103516e-05 |
| Small nonplanar | 0 | 0 | 2.312660e-05 | 1.311302e-05 | 0.000149 |
| Small bipartite | 4 | 4 | 0.000194 | 6.198883e-05 | 9.989738e-05 |
| Medium planar | 5 | 14 | 0.000291 | 0.000109 | 0.000150 |
| Medium nonplanar | 4 | 20 | 0.000222 | 0.000103 | 0.000131 |
| Medium bipartite | 9 | 52 | 0.004549 | 0.000427 | 0.000596 |
| Large planar | 9 | 193 | 0.018578 | 0.003814 | 0.002960 |
| Large nonplanar | 6 | 1533 | 0.023064 | 0.011300 | 0.011050 |
| Large bipartite | 12 | 397 | 0.214183 | 0.014868 | 0.017275 |

Additionally, the number of recursive calls for each function is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| Graph | Recursive Calls  No Heuristics | Recursive Calls  Pivoting | Recursive Calls  Pivoting, Ordering |
| Small planar | 18 | 15 | 18 |
| Small nonplanar | 5 | 5 | 5 |
| Small bipartite | 41 | 18 | 26 |
| Medium planar | 108 | 44 | 55 |
| Medium nonplanar | 71 | 46 | 49 |
| Medium bipartite | 2061 | 275 | 215 |
| Large planar | 8998 | 1445 | 925 |
| Large nonplanar | 11545 | 4293 | 4475 |
| Large bipartite | 89082 | 5104 | 4388 |

We can see that, for the most part, BK with pivoting runs faster than the basic BK algorithm. Additionally, we see that BK with both pivoting and ordering usually – but not always – runs faster than either of the other two (simpler) functions. However, note that the execution times aren’t necessarily the most important performance measure; typically, the recursive depth is more useful. In that case, we can see that, for large graphs, more heuristics typically implies a shallower recursive depth. For small and medium graphs, however, it’s often the case that employing both pivoting and ordering gives a deeper recursive depth than does using just pivoting.

In short, then, one should always employ the pivoting heuristic when evaluating maximal cliques. For small and medium graphs, this is sufficient. For large graphs, one should also employ the ordering heuristic; although this often increases the execution time, it decreases the recursive depth and thus better prunes the search tree.

For the reader’s perusal, I’ve included in my submission a file called “medium-nonplanar.txt.” In this file, we can see the search tree for one test graph. Specifically, we can see the independent sets as they grow, we can see when the algorithm reaches a maximal independent set, and we can see when the algorithm backtracks to find a new independent set. Although this file is not organized in a standard search tree format, the nodes, vertices, and traversal of the graph are clear.

**Problem 3**

Concisely discuss the above top-down design, implementation, and testing efforts.

In this assignment, we refined a provided PD/AD design process to improve the Bron-Kerbosch algorithm for finding maximal independent sets. Specifically, we added two heuristics – ordering and pivoting – to the design process and to the algorithm, and we then implemented the algorithm as a Python program.

Because most of the PD/AD process was completed for us, I don’t feel that the process was incredibly tedious for this assignment. However, I still think that, at least when the problem is relatively simple, the PD/AD process is perhaps not the most efficient use of our time when solving complex problems. In homework 3, I said that the process was straightforward because we had a goal (i.e. the Bron-Kerbosch algorithm) to work toward. I think the same applies here. We have the Kang implementation, and we have a nearly-complete PD/AD design process, so finishing the process and testing the code is not difficult. If we have a complex problem, and if we don’t already have a known algorithm to design towards, I don’t think the PD/AD process is super useful.

Although I find the PD/AD process bothersome, I certainly enjoy programming, so I found the latter half of the assignment (that is, the implementation and testing) more enjoyable.

Implementing the BK algorithm and the added heuristics was simple. Pseudocode exists on Wikipedia and in the class handouts, so translating said pseudocode to working Python would not take much effort. Kang’s program, of course, already implements the BK code in Python, so I only needed to update and refine this codebase to compute MIS and not clique. Specifically, I utilize a function called complement() to find the inverse of a graph. In this report, a graph’s *inverse* simply notes which vertices each vertex *is not connected to*. Because MIS and clique are complementary, we can solve the clique problem by deleting the complement() call (and thus reverting back to the original Kang implementation).

Testing Kang’s implementation was not difficult. In his implementation, graph generation is easy, and I was thus able to generate multiple graphs for testing. Additionally, the program is able to evaluate a given graph for the basic BK algorithm, for the BK algorithm with pivoting, and for the BK algorithm with pivoting and ordering, so I was able to gather results for various combinations of the required heuristics with little extra effort.