copyright GBL rev s 5/20

**MIS Global Depth-First-Search Back-Tracking Design  
*[Use our Disciplined PD/AD Algorithm Top-Down Design Technique!]***

The required gs-dfs/bt (global search depth-ﬁrst-search with backtracking) by definition searches the entire MIS search space implicitly or explicitly. The dfs process searches iteratively or recursively by traversing from one partial solution state space candidate to another until maximal solutions are found. It executes backtracking to find **all** maximal/maximum (or minimal/minimum) solutions.

1. **General discussion of graph MIS problem domain:**

*“The maximal (maximum) independent set problem –* [*MIS*](https://en.wikipedia.org/wiki/Maximal_independent_set)*”*

*The MIS problem is the following: given a graph G = (V;E) and an independent set in G*

*of maximal cardinality. In the weighted case, each node v in V has an associated non-negative weight w(v) and the goal is to find maximal weight independent sets S. {v is in S and the neighborhood of v intersecting with S is not empty. This problem is NP-complete and thus it is natural to look for MIS/Clique approximation algorithms.*

- MIS informal description: (Christoﬁdes, p30-31 [1]) “internally stable set,”   
 (cliques of complementary graph) – a NP-Complete problem (NP-Hard)

- MIS formal model with n is the number of vertices: (Christoﬁdes, p31) ;   
- solution space: O(3n/3) (Christoﬁdes, p53; Reingold, Nievergelt and Deo, p353)   
- graph search space: O(2n), power set of possible vertex partial combinations

- applications: NPC, constrainted resource task scheduling, communica­tions, clique, vertex cover, ... *(note that one should be able to map in order of (p(n)) from any of these problems to and from the MIS formulation - NPC)*

**Proof that MIS is NP-Complete:**

First, we need to show that the MIS problem is in **NP**. We can show that a set |S| = k is may be an independent set or not in polynomial time.Second,the data structure of the 3-SAT problem (NP-Complete) can be reduced in polynomial time to the MIS problem, Thus, the 3-SAT can be solved using the MIS problem and therefore MIS is **NP-Complete** (GJ)**.**

**MIS Algorithm Domain (gs-dfs/bt) Development**

- selected algorithm: gs-bfs/bt search (Christoﬁdes, p33-35); set operations needed

The standard problem domain requirements analysis and generation of the high level design can be approached in the general pattern as indicated in the previous lecture notes. Mapping the global gs-dfs/bt algorithm to the design of a MIS search algorithm results in a tree search control structure (Christoﬁdes, pp33-35). The algorithmic design philosophy of Talbi [4] and Michaelewicz and Fogel [5] are also followed throughout this development. What are “good” heuristics?

* *Comment:* we design and analyze the MIS problem and study in detail the graph search (tree search) algorithm (Christoﬁdes) using the gs-dfs/bt search algorithm template. Individually you can study other algorithm implementation and applications for solving the MIS problem based upon other selected graph data structures (see references). *Observe that ADT(D,F,A) needed for sets in this algorithmic design*. Also consider [Clique problem](https://en.wikipedia.org/wiki/Clique_problem) description! Or a [Clique graph theory](https://en.wikipedia.org/wiki/Clique_(graph_theory)) description.

**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, ….

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)
* ***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)*

[*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?)]

*(Improved search = fewer search nodes/branches?)* [*NFL Thm*](https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf) *impact!?*

* 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  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), …**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).  
\*\*\*\*\*  
6. Mapping to chosen computer language

Carry standard search elements into computer language as comments. One should be able top to follow the design from problem domain and algorithm domain integration explicitly to implementation.

1. Test and Evaluation Report of Software Execution

Report on [software testing](https://en.wikipedia.org/wiki/Software_testing) evaluation per Barr’s approach and suggestions of others. Show MIS problem domain graphs and algorithm domain search graphs.

***“ Analyze design flow mapping into AFIT “graphprogram” coded in C++? Search Control Structures indicated with comments using standard search elements? Or map to a Java object-oriented implementation? Or other programming language, or parallelize algorithm?” Are there any special MIS PD cases? (planar graphs, Interval graphs, …)*** *\*\*\*\*\*\*\*\*\*\*\*\*\** ***- Comments:***

1. We have attempted for complete design documentation to explicitly reflect each symbolic algorithmic step leading to the functional low-level de­sign; this structure is more in the form of the functional decomposition design paradigm, could also define an object-oriented structure. Algorithm Complexity is of order (n22n) per code documentation? Check! Or is it of order (n23n/3)? NO!

2) Redesign gs-dfs/bt algorithm for the MIS/Clique using the suggested a priori sorting routine. Also consider appropriate data structures and processing for a large number of nodes?

3)Also, could output the search process per each search tree node for pedagogical insight. In addition, what are the restrictions to be placed on graph size? Define test objectives of MIS gs-bfs/bt program (Use Axioms?).

4) Mapping to the low-level algorithm design. Note that the integration of the Qks could be inside or outside the objects depending upon the designers con­straints/creativity. Also, the graph input process could be embedded in the graph-ADT object or designed as a separate module as the AFIT case. What are some other MIS object-oriented design possibilities? Discuss Approximation algorithms for MIS/Clique problem solutions – DFS.

5) Select another gs-dfs/bt algorithmic design for the maximal/maximum MIS or Clique problem and indicate the creative design data structures and control structures used. [ For example, [kiyama et al. (1977](http://en.wikipedia.org/wiki/Clique_problem#CITEREFTsukiyamaIdeAriyoshiShirakawa1977)) [Chiba & Nishizeki (1985](http://en.wikipedia.org/wiki/Clique_problem#CITEREFChibaNishizeki1985)) [Johnson & Yannakakis (1988](http://en.wikipedia.org/wiki/Clique_problem#CITEREFJohnsonYannakakis1988)) ]. Also, consider specialized MIS/Clique problems as listed previously using gs-dfs/bt. [wikipedia]

6) Test MIS/Clique algorithms and compare over various MIS/Clique graphs ([DIMACS](http://dimacs.rutgers.edu/), [Dimacs(clique)](http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark), [BHOSLIB](http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark), Conte’s BK paper in 2013, …)

7) Add an algorithm to implement Christofides Equation 3.9, page 34. This is defined as “pivoting” in some descriptions of the BK algorithm (see Wikipedia and other references). “Heuristic impact?” [complexity?]

8) Also, add the “simple” implementation of “vertex ordering” based upon vertex degree. That is, select those nodes with “smaller degrees” first (*degeneracy ordering*) in the DFS/BT search process. “Heuristic impact?” [complexity?]

***\*\*\*\*\*\*\*\*\*\*\*\*\*\****

**A view of a gs-bfs/bt refinement (pseudo code) using a stack** *(from literature – How was it designed? – difficult to understand design!)*

Input G(V,E) *[a connected graph]*

*v* *[start vertex]*

Algorithm Depth-First Search/Backtracking

visit *v*

*V′* ← {*v*} *[V′ is the set of vertices already visited]*

Put *v* on *S* *[S is a stack – set of candidates?]*

*u ← v*

repeat while *S* not empty;

if *A(u)* − *V′* is not empty ;

then Choose *w* ε *A(u) − V′*

visit *w*

*V′* = *V′* union {*w*}

Put *w* on stack *S [what if we want to consider each S element directly?]*

*u ← w*

else *u ← top(S)* *[Pop the stack – careful with ADT!]*

endif

endrepeat

DFS uses backtracking

• Go as far as you can until you get stuck(?)

• Then go back to the first point you had an untried choice *[quite general!]*

*“Now embed our standard search element functions into this gs-dfs/bt algorithm – yet another example of binding too early to a specific data structure – justify stack”*

***Some References:*** *(also consider papers in MIS\_Clique paper directory)*

[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

***Some MIS/Clique Software:*** *(see CSCE686 MIS\_Clique Software directory)*

* *C++:* [*https://www.sanfoundry.com/cpp-program-find-maximum-size-clique-graph/*](https://www.sanfoundry.com/cpp-program-find-maximum-size-clique-graph/)
* *JAVA:* Prosser, Patrick, *Exact Algorithms for Maximum Clique, a Computational Study*, TR-2012-33, Glasgow University, Scotland, 2012
* *Python:* [*https://networkx.github.io/documentation/networkx-1.9.1/reference/algorithms.clique.html*](https://networkx.github.io/documentation/networkx-1.9.1/reference/algorithms.clique.html)
* *JAVA:* [*https://sites.google.com/site/indy256/algo/bron\_kerbosh*](https://sites.google.com/site/indy256/algo/bron_kerbosh)