**7 GBL rev ax 4/20**

**CSCE686 Advanced Algorithm Design '19**

**Lecture 3 “Designing Algorithms for NP-C Problems”**

**3.0 Global Search (Deterministic vs. Stochastic)**

Global search by definition searches the entire search space implicitly or explicitly. Such searches iteratively or recursively traverse from one state space candidate to another until a feasible or optimal solution is found. Global search methods for a variety of problem reduction techniques that include *binary search, branch and bound approaches, dynamic programming, game-tree search, depth-first search with backtracking, and divide and conquer*. Many of these are discussed from the algorithm domain/problem domain perspective and related to classical NP-complete problems and beyond.

**3.1. Global Search - depth first search with backtracking**

**(gs-dfs/bt)**

- Global search gs-dfs/bt uses a depth first search approach with back-tracking to implicitly or explicitly search the entire search space in order to obtain an optimal solution (unique?). The standard search functions or constructs from the greedy approach are used again along with the additional delayed termination constraint. Also, consider efficient and effective solution representations.

***basic search functions*** - set of candidates, next state generator, feasibility, selection, solution, objective (fitness)

***delay termination***- the entire state space is searched explicitly or implicitly to find optimal solution(s) in solution space.

***representation encoding*** - (*Talbi – p35*) {[NPC Problems](https://en.wikipedia.org/wiki/List_of_NP-complete_problems) !}  
(linear, nonlinear, discrete, permutation, random, messy, diploid, quantum,…)   
 - binary: knapsack problem, SAT problem, MIS, Clique,…

* real values: continuous function optimization, …
* discrete values: assignment problem, GCP, …
* permutations: TSP, scheduling, sequencing, 8-Queens, …

***representation characteristics of solutions***  - (*Talbi – p35*)   
 - completeness (represent all solutions and partial solutions)  
 - connexity (a search path in search space must exist to sol)  
 - efficiency (search functions economical in time and space)

***May need a mapping between solution representation and problem objective function*** *(Talbi – s1.4.2):* TSP: sequence to additive value.

**3.1.0 gs-dfs/bt Template**

The following design template reflects the algorithm domain specifications from high-level design to function specification for the global search gs-dfs/bt algorithm (requirements specification is extended from the greedy (DFS) search):

***Algorithm domain requirements specification form:*** *name*: **Global-Search Depth First Search with Backtracking,gs-dfs/bt**(D*i,*D*o*)

*-* domains: D*i*  is set-of-candidates, D*o* are the sets of solutions,

*-* logic:

I(x); x in D*i*, x is a possible candidate from input set

O(x,z); x in D*i*, z in D*o*; condition on z being an optimal solution - depth first search process operators with backtracking operator  
[set of candidates, next state generator, feasibility, selection, solution, objective (fitness)]

***Algorithm domain design specification form:***

*name*: **Global-Search, gs-dfs/bt**(D*i,*D*o*)

*-* domains:D*i*  is set-of-candidates, D*o* are the sets of solutions,

D*p* is the partial solution set, boolean

*-* imports: ADT set, real/integer/character, ADT graph?, …

“other ADTs Imported depend upon application"

*-* logic: I(x); x in D*i*

O(x,z); x in D*i*, z in D*o*; condition on z being a optimal solution

I'(x,x'); x in D*i*, x' in D*p*; condition on x' being a partial solution

(usually generated by feasibility checks and set union)

- search operations:

- next-state-generator (x,D*i*)

- selection (x,D*i*) x usually from an ordered set

based explicitly/implicitly on the objective function

- feasibility (x, D*p*) feasibility check,if true union (x, D*p*)

- solution (D*p*): Is Dp solution?; if true z = D*p*;

**delayed termination – backtrack - find all optimal/optimum   
 solutions!!**

- objective (z) *“ordered set/well founded set"*

*“integration from problem domain: instantiate state, Di,*

*Do, and Dp, and set of candidates from problem domain*

*meanings/structure”* ***(hardest part!!! – creative!!!)***

*-* axioms: (How to define? Combination of operators in FOL!)

.

***Algorithm domain function specification form: (iterative)***

*function****:*****global-search-dfs-iterative** (D*i*,D*o*)

*-* Initial condition: clear(set D*p*)

*-* body

generate set of candidates x in D*i*

loop nsg: next state generator

determine value of local objective function for each x

end-loop-nsg

clear(D*p*)

while (D*i*) not exhausted gs-dfs-loop:

*“select x that max/min objective function and delete(x, Di)"*

selection(x,D*i*); x is the next candidate from in Di

*“****attempt to use PD* next state generator *to reduce set-of-***

***candidates-heuristic!”***

delete(x, D*i*); D*i* becomesD*i*/x

if feasibility (x, D*p*) then union(x, D*p*); *“add x to partial solution* D*p”*

*“attempt to generate early* ***backtrack*** *condition from PD-****heuristic****”*

if solution(D*p*) then save z = D*p*, z in D*o*, backtrack

deleting old x from D*p* and consider new x from D*i* (new gs branch)

end gs-dfs while loop

if D*o* not empty set return all z's (solutions); “also could output each solution as generated"

***Algorithm domain function specification form: (recursive)***

function: **global-search-dfs-recursive** (D*i*) sets in D*o*

*-* Initial condition: clear(D*p*)

*-* body

clear(D*p*)

find-all (D*p*)

**function** find-all (D*p*) sets in D*o*

- next state generator(x,D*i*, D*p*) xnext state in D*i*

- if feasibility(x, D*p*) then union(x, D*p*)

- if condition of D*P*, then backtrack early

- if ←solution (D*p*) then “save" z = D*p* & backtrack

- if state space exhausted then exit

else find-all(D*p*)

return all z's with each z in D*o*;

“also could output each solution as generated

“see MIS optimization using gs-dfs/bt design discussion”

**Comments:**

*-* evolving better data structures for efficient control structure

execution is paramount in binding operations and data at the

implementation level (low-level) as design is detailed.

*-* gs-dfs/bt is an extension of “Greedy” DFS (hill-climbing)

*-*  can use **“heuristics”** to limit search graph nodes (partial

states) explicitly considered. In some cases they are in

reality improved feasibility algorithms/functions. Of course, the

better an algorithm is informed about the explicit problem

domain, usually the better the search graph can be reduced.

Note that Heuristics can drive the algorithmic complexity up. Why?

*-* backtracking requires bookkeeping for retaining state

information at each search tree level (impact of large n?)

-gs-dfs/bt is applicable for solving many NPC problems:

minimal/minimum independent sets, maximal clique,

set/vertex covering, traveling salesperson problem, …

*-* why iterative over recursive control structure?

*-* does the gs-dfs/bt incorporate ALL aspects of such algorithms

*-* having considered dfs and dfs/bt it is now on to breath-first search

(ga-bfs) and best-first search; As we continue to address complex

problems, we also discuss local search in more detail (Tabu

search, simulated annealing, functional derivative-conjugate

gradient, ..) as well as stochastic search (genetic algorithms,

artificial ant systems, …) in the solution space – population based.

- a **Graph ADT** (for PD and AD design) can be found at <http://www.cs.auckland.ac.nz/~ute/220ft/graphalg/graphalg.html>

or in a paper by Guetaru and Haurat (CSCE686 ADT directory)

- observe that associated source code is available at <http://www.sourcecodeonline.com/list?q=graph_adt>

**3.1.1 Application of gs-dfs/bt to Maximum Independent Set (MIS) Problem (Christofides, Chapter 3;** *Talbi, p124***,** [**Pattillo & Butenko**](http://ise.tamu.edu/people/faculty/butenko/papers/EORMS.pdf)**)**

**MIS/Clique Problem Domain (NPC): *“optimal vs. optimum results”***

* + ***English-Symbolic-math/logic Problem Domain -- see Christofides (ch3) “others?”***
    - ***a linear model (objective + constraints)***
  + ***PD solution space complexity is 3n/3 (Moon & Moser – see handout)***
  + ***Proving MIS is in NPC class (see references)***

**MIS/Clique Algorithm Domain:**

* + ***gs-dfs/bt algorithm for the MIS – Christofides (Ch3) -- AD complexity vs. PD complexity?***
  + ***DIMACS MIS/Clique algorithmic competitions and results (1990s) – example MIS/Clique problems***
  + ***Implementation of Christofides gs-dfs/bt MIS algorithm in AFIT’s “graphprogram” (execute MIS)***

***MIS/Clique Maximal/Maximum Algorithms:*** [***Bron-Kerbosch***](https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm)***, output-sensitive,*** [***Chiba & Nishizeki,***](http://en.wikipedia.org/wiki/Clique_problem#CITEREFChibaNishizeki1985) ***Jian, Robson (fastest?); “AD complexities?” – Objective is to reduce exponent of 3n/3 (# of solutions.). Another Clique mathematical formulation -*** [***Pattillo & Butenko***](http://ise.tamu.edu/people/faculty/butenko/papers/EORMS.pdf) ***paper.***

***Approximation MIS/Clique Algorithms: Feige, Boippana & Halldorsson, parallel/distributed algorithms, …   
“parallel AD complexities?” – polynomial? (Serial Order of/ # of procs + parallel communication overhead!)***

***Related MIS/Clique Problems:*** *all maximal independent sets, independent dominating sets, maximum matching problem, maximal independent edge set, all maximal cliques, minimal vertex cover****,*** [***k-clique***](https://www.bing.com/search?q=k-clique+problem&qs=AS&pq=k-clique&sk=AS1&sc=3-8&cvid=7867BB5BFA37472AA18B6D22508A6CB3&FORM=QBRE&sp=2https://www.bing.com/search?q=k-clique+problem&qs=AS&pq=k-clique&sk=AS1&sc=3-8&cvid=7867BB5BFA37472AA18B6D22508A6CB3&FORM=QBRE&sp=2)***, …***

***MIS Applications:*** *computer vision, pattern recognition, information theory, map labeling, molecular biology, scheduling, …* ***(Military Apps?)***

***Clique Applications:*** *triangle-free graphs, bipartite graphs, interval graphs, lattice gas modeling, chordal graphs, planer graphs, complete graphs, DNA modeling, quantum physics, social networks, circuit complexity, diffusion games, risk transport, …* ***(Military Apps?)***

**3.1.1.1 Graphs for testing MIS/Clique algorithms**

**Consider finding maximum (maximal) cliques or MISs for**

**Cayley Graphs (**[**Hamming graphs**](http://mathworld.wolfram.com/HammingGraph.html)**, ...)**

**Simple graphs ( [Keller graphs](http://mathworld.wolfram.com/KellerGraph.html),** [**Lollipop graphs**](http://mathworld.wolfram.com/topics/LollipopGraphs.html) **,** [**Listing**](http://graphclasses.org/smallgraphs.html)**, ...)**

**Moon-Moser Graphs ([Turan graphs](http://en.wikipedia.org/wiki/Tur%C3%A1n_graph), …)**

**Non-Planer Graphs (**[**Listing**](http://mathworld.wolfram.com/topics/NonplanarGraphs.html)**, …)**

**Perfect Graphs (**[**Listing**](https://en.wikipedia.org/wiki/Perfect_graph)**,…) “MIS/Clique have polynomial solution”**

**...**

**Some examples area found in the DIMACS clique instances**

**(**[**http://cs.hbg.psu.edu/txn131/clique.html**](http://cs.hbg.psu.edu/txn131/clique.html) **,** [**BHOSLIB**](http://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark)**, ...)**

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

***“If your background in graph theory needs to be strengthen:***

[**http://www.utm.edu/departments/math/graph/**](http://www.utm.edu/departments/math/graph/) **or** [**http://www.youtube.com/watch?v=HmQR8Xy9DeM**](http://www.youtube.com/watch?v=HmQR8Xy9DeM) **(youtube) or**

[**http://en.wikipedia.org/wiki/Graph\_theory**](http://en.wikipedia.org/wiki/Graph_theory) **“**

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

**3.1.1.2 Some Other MIS/Clique References:   
(See CSCE686 MIS/Clique papers directory)**

**Contemporary Maximum Clique Bibliography -**

<http://iridia.ulb.ac.be/~fmascia/maximum_clique/bibliography>

**For more MIS/Clique information** see Wikipedia or <http://en.wikipedia.org/wiki/Maximal_independent_set> and

<http://en.wikipedia.org/wiki/Independent_set_(graph_theory)> and

<https://wikivisually.com/wiki/Clique_problem>

as well as Mathworld (Wolfram): <http://mathworld.wolfram.com/MaximalIndependentVertexSet.html>

-Volker Stix (2004), “Finding All Maximal Cliques in Dynamic Graphs,” Computational Optimization and Applications Vol. 7, Issue 2. Kluwer Academic Publishers: Norwell, USA.

-Lijun Chang, Wei Li, Wenjie Zhang (2017), “Computing A Near-Maximum Independent Set in Linear Time by Reducing-Peeling,” *Proceedings of the ACM SIGMOD International Conference on Management of Data* (*SIGMOD’17*)

-Liu, Lu,Yang,Xiao, Wie (2015), “Towards Maximum Independent Sets on Massive Graphs,” *Proc.VLDB Endowment,* Vol. 8, No. 13

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

***“We are covering MIS, Clique, and Coloring Problems, all NP-Complete problems in terms of PD mathematical form and integration into a gs-dfs/bt algorithm design with heuristics. Note that one could also integrate these problems into a BFS as well as a population-based metaheuristic algorithmic design”***

**Interesting Reads: Christofides***,* Chap 3 (SCP) ; **Talbi**,Sections 1.7 & 1.8; **Michalewicz & Fogel**, Chap 4, and **William & Shmoys**, Chap 1

***“Next is the set-covering problem (SCP) in Christofides, Chapter 3, that uses a gs-dfs/bt algorithm design with suggested heuristics.”***

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

**Quiz – Closed Book! (eight minutes)**

1. **What are the basic elements of a search algorithm?**
2. **Define the MIS PD math equations (English and Math).**
3. **What is the Order of the MIS/Clique solution space? Why?**

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

1. **(extra) Why is the additive combination of the MST and SPP, both P-Time, a HPC problem? Discuss in English. Does a FPTAS approximation algorithm exist for this problem?**

**A New View of Search! Approximation!**

***IMPORTANT INSERTED TOPIC:***

**Generic Approximation Algorithm Schemes Introduction**

(Talbi section 1.3.2, p21; W&S, pp5-6, [Complexity Zoo:P](https://complexityzoo.uwaterloo.ca/Complexity_Zoo:P#ptas))

*Let* X *be a minimization (respectively, maximization) problem.*

• *An approximation scheme for problem* X *is a family of* (1 + ε)*-approximation algorithms* Aε *(respectively,* (1−ε)*-approximation algorithms* Aε*) for problem* X *over all* 0 < ε < 1*.* { (1- ε) for maximization problems.}

• *A polynomial time approximation scheme* ***(***[***PTAS***](https://en.wikipedia.org/wiki/Polynomial-time_approximation_scheme)***)*** *for problem* X *is an approximation scheme if for any fixed* ε *> 0, the scheme’s time complexity is polynomial in the input size* n*.* Example: n1/ ε

• *A fully polynomial time approximation scheme* ***(FPTAS)*** *for problem* X *is an approximation scheme whose time complexity is polynomial in the* *input size n and also polynomial in* 1/ε*.* Example: n2 / ε2 .

Approximation algorithms are algorithms that run in polynomial time and give a guarantee on the quality of the solution returned***?***

***“Approximation Algorithm(input) ≤ factor \* Optimal(input)”***

Notes: For a PTAS it would be acceptable to have a time complexity for example proportional to n2/ε; although this time complexity is exponential in 1/ε, it is polynomial in the size of the input n exactly as required in the definition of a PTAS. An FPTAS cannot have a time complexity that grows exponentially in 1/ε, but a time complexity proportional to 6n/ε3 would be OK. With respect to worst case approximation, an FPTAS is the strongest possible approximation result that we can derive for an NP-hard problem. *FPTAS examples are found in weak NPC problems: 0-1 knapsack, bin-packing, subset sum problems. Strong NP-Hard problems such as TSP do NOT have a FPTAS.*

Note: Observe that the *complexity class approximable* ***(APX)*** by definition consists of all minimization problems that have a polynomial time approximation algorithm with some finite worst case ratio, and of all maximization problems that have a polynomial time approximation algorithm with some positive worst case ratio.

Observe that PSPACE-Complete problems do NOT have FPTAS algorithms.

*Exercise:* Draw a figure (Venn diagram) that illustrates the relationships between the classes NP, APX, P, the class of problems that are pseudo-polynomially solvable, and the classes of problems that have a PTAS and FPTAS [note that FPTAS ⊊ PTAS ⊊ [APX](http://en.wikipedia.org/wiki/APX)] *(can also see Wikipedia)*

Comment:

To indicate that approximation algorithms generate an answer that is “close” to the optimum value, generation of a lower bound using the approximation algorithm is required (maybe difficult). Approximation algorithms are algorithms that run in polynomial time and give a guarantee on the quality of the solution returned***?***

***“Approximation Algorithm(input) ≤ factor \* Optimal(input)”***

Lower bounds are employed to find such factors. Note that many NP-Hard problems do not have matching upper and lower bounds

with the same approximation ratio.

**Other Polynomial Approximation Algorithm Definitions:**

1. **Efficient polynomial-time approximation scheme** or **EPTAS**; the running time is required to be *O*(*nc*) for a constant *c* independent of ε.
2. **Quasi-polynomial-time approximation scheme** or **QPTAS.** A QPTAS has time complexity for each fixed .



1. **Polynomial-time randomized approximation scheme** or **PRAS**. A PRAS is an algorithm which takes an instance of an optimization or counting problem and a parameter ε > 0 and, in polynomial time, produces a solution that has a *high probability* of being within a factor ε of optimal. Conventionally, "high probability" means probability greater than ¾.
2. **Efficient polynomial-time randomized approximation scheme** or **EPRAS; …**
3. **Fully polynomial-time randomized approximation scheme** or **FPRAS; …**

**Comment:**

Note that for some combinatorial optimization problems, it is possible to prove that even the design of a ratio approximate algorithm with a small ratio is impossible, unless P = NP. Results of this kind are called “*inapproximability results.”* Examples are TSP, error correcting codes, …. (Trevisan, 2004)

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

***What would be a “good” PTAS or FPTAS approximation scheme for the*** [***MIS/Clique problem***](https://wikivisually.com/wiki/Clique_problem) ***or*** [***SCP***](https://en.wikipedia.org/wiki/Set_cover_problem)***?***

**MIS/Clique (PTAS):**

**Which approximate algorithmic approaches would be “good” for the MIS or Clique problems?**

**Best lower bound PTAS algorithm for MIS/Clique provides a maximum clique within an approximation ratio/factor of**

**O(n(log log n)2/log3n). Consider n = 216 which yields 28 .**

**No polynominal time algorithm approximates the maximum clique within a factor better than O(n1-ɛ), unless NP = ZPP!**

**For an extended reference list “see** [**Clique Problem**](https://en.wikipedia.org/wiki/Clique_problem)**,” Wikipedia and (**[**Liu et al, 2015**](http://delivery.acm.org/10.1145/2840000/2831366/p2122-liu.pdf?ip=129.92.250.40&id=2831366&acc=ACTIVE%20SERVICE&key=92909D146075CF59%2E296EECC507F02D9A%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1555959658_5bc1655f74e96ea1a726d3f6f19d6278)**)!.**

**SCP (PTAS):**

**Using a Greedy (DFS) approximation algorithm (pick the set that covers the most elements not already covered), cannot get better than (1-ɛ)Ln(n) for any 1 > ɛ > 0; (Feige, 1996). O[log(n)]**

**Bin-Packing Problem (PTAS):**

**Using a first fit Greedy algorithm (O(nlogn)), first fit descending heuristic FFD sorts objects into decreasing order by size, results in a factor of 11/9 optimal value + 1 .   
[Talbi ex. 1.16, p22; Iyer,** [***Bin Packing with FFD Heuristic***](https://www.nitt.edu/home/academics/departments/cse/faculty/kvi/Bin%20Packing%20FFD%20heuristics.pdf)***,* 2008]**

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

**“We discuss explicit approximation search algorithms for some of the classical NPC problems (MIS, CLIQUE, SCP, Coloring Problem, …) and return to the above definitions”**

**3.1.2** **Application of gs-dfs/bt to the set-covering problem (SCP)** (Christofides, Chapter 3; Hoos & Stutzle, Section 10.3, W&S, P6-22, TR p456)

**Set Covering Problem** (**SCP** is **NPC**)**: *“optimal, optimum sol.”***

* + ***Symbolic Problem Domain -- Christofides (ch3), H&S (Ch 10)***
  + ***PD complexity is 2n where n is # of sets***
  + ***An integer “programming” model (canonical form):*** 
    - ***Maximize: cTx, subject to: Ax ≤ b “linear model!”***
    - ***a 0-1 model (int): set xi is in or not in optimum set of sets***
  + ***Proving SCP is in NPC class? “reduction to ?”***
  + ***Benchmark problems (DIMACS, Beasley, Balas & Carrera,…)***
    - ***Various sizes and densities of SCP Boolean matrix***
  + ***gs-dfs/bt algorithm for the SCP – Christofides (Ch3)***
    - ***AD time and space complexity?*** *O(p(n)2n)?*
  + ***Implementation of Christofides gs-dfs/bt SCP algorithm in AFIT’s SCP Solver software on L:Drive under CSCE686***

***“Discuss gs-dfs/bt algorithmic design for SCP” HW!***

***SCP Algorithms : Christofides, Balas and Carrera, Beasley, Caprara;***

***Note:*** *# of sets and # of elements to cover is defined by a Boolean matrix*

***Approximate SCP Approaches:*** *DFS (harmonic ratio approx., local heuristic), maximize k-elements covered problem., maximum coverage problem - choose at most k sets to cover as many elements as possible, stochastic local search (SLS) algorithms (evolutionary, simulated annealing, …), iterated local search (ILS agorithms, Tabu search, SCP mapping to minimum flow problem, SCP PD relaxation ( Lagrangian,… ); “Algorithm complexities?” – polynomial?*

***Parallel SCP Approaches:*** *parallel graph search (parallel branch search), parallel SLS, Beasley, …*

***Related SCP Problems:*** [*Quadratic SCP, Set Partition Problem(SPP), Hitting Set*](http://en.wikipedia.org/wiki/Hitting_set) *is an equivalent reformulation (dual) of Set Cover.;* [*Vertex Cover*](http://en.wikipedia.org/wiki/Vertex_cover_problem) *is a special case of Hitting Set; Max Coverage Problem;* [*Edge Cover*](http://en.wikipedia.org/wiki/Edge_cover_problem) *is a special case of Set Cover.; Max* [*Set Packing*](http://en.wikipedia.org/wiki/Set_packing) *is the dual problem of Set Cover; Red-Blue SCP; Matching Problem; Geometric Set Cover; Nearest Neighbor Search; Combinatorial Auctions Winner Determination Problem (CAWDP), GENERALIZED/Capacitated SCP (every element covered multiple times, element Min-Sum…)…*

***SCP Applications:*** *scheduling of resources (deployments, crews,…), finding computer viruses, manufacturing material selection, packing and partitioning problems, network location problems, data aggregation* *(e.g., bandwidth, energy)* *in wireless sensor networks, database query sets, games (Sudoku, Kanoodle, Pentominoes, …), terrorists differentiation (terrorist characteristics vs. sensor/information cost) …*

***Some SCP References:***

*For more SCP information see Wikipedia:* [*http://en.wikipedia.org/wiki/Set\_cover\_problem*](http://en.wikipedia.org/wiki/Set_cover_problem)

[*http://mathworld.wolfram.com/SetCoveringDeployment.html*](http://mathworld.wolfram.com/SetCoveringDeployment.html) *“apps”*

*or:* [*http://www.cs.sunysb.edu/~algorith/files/set-cover.shtml*](http://www.cs.sunysb.edu/~algorith/files/set-cover.shtml)

*Also:*

*Vijay****, “Approximation Algorithms,”*** *Chapters 2 & 12;*

*Caprara, Fischette and Toth “Algorithms for the Set Covering Problem,”* ***Annals of OR****, 47(5), pp353-371, 2000;*

*Kabb,* [*Minimum Exact Cover*](http://www.csc.kth.se/~viggo/wwwcompendium/node146.html)

*Skiena,* ***“The Algorithm Design Manual,”*** *Springer, 1997 – “discussion of   
 various* [*NPC Problem Domains*](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.471.4772&rep=rep1&type=pdf)*”- SCP, page 398 – software disk and site.*

*Williamson and Shmoys,* ***Approximation Algorithms****, Chaps 1 & 7:  
 - Linear Model (Linear Programming Models – LP)*

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

***Comments on SCP: “What algorithmic heuristics?”***

To reduce the size of the problem domain, consider Christofides’ reduction techniques - heuristics (Christofides Ch. 3);

*“to seek and discover”*

To *reduce the number of partial states considered*, restructure (sort) the SCP Boolean matrix by considering the incremental cost of each item per set, keep a list (queue) of partial covers and their cost per Christofides’ “E” list SCP algorithm, etc.

SCP *approximation* algorithms (deterministic search) usually evolved DFS to find a solution; can result in worst case [a + b O(lgn)] times optimal value.

How can one *evolve better* ***“heuristic”*** data structures (linked list, prior queues, …) for efficient SCP control structure execution? This is paramount in binding operations and data at the implementation level (low-level) as algorithmic design is detailed. “Incomplete knowledge”

How can one *develop better*  ***“heuristic”*** control structures (based on a priori data structures) to limit search graph nodes (partial states) explicitly considered; i.e., reduce the cardinally of the set of candidates. In some cases these heuristics are in reality improved feasibility algorithms/functions. Again, heuristics can drive the algorithmic complexity up. Why? (polynomial function multiplying PD complexity) *“decision guiders” “much contemporary SCP research”*

DFS withbacktracking requires bookkeeping for retaining state information at each search tree level (What is the impact of large n on complexity?). Note that an SCP algorithm may or may not generate unique partial solutions only once? Of course, this is generally desired in any NPC problem solution!

gs-dfs/bt is applicable for solving many SCP type problems and, in general, all NPC problems? How does the gs-dfs-bt incorporate ALL aspects of such algorithms applied to the NPC problems? “Separate heuristics of course!”

Note that the associated vertex cover problem although NPC has a best approximation algorithm ratio/factor of 2,

***Generic Heuristic tradeoffs:*** (Wikipedia+)

* + *Optimally;* heuristic guarantees best solution found   
     (admissible vs. non-admissible heuristic-nonoptimal)
  + *Completeness;* heuristic helps find all solutions   
     (maximal, minimal)
  + *Accuracy;* confidence interval solutions - uncertainty
  + *Execution time;* heuristic complexity & convergence   
     rate
  + *Impact of No-Free Lunch Theorem!   
    “no best algorithm for solving general NPC problems!”*
  + *Develop better Heuristics for given problem!*

Reference: Romanycia et. al., [*What is a heuristic?*](http://www.sfu.ca/~jeffpell/papers/RomanyciaPelletierHeuristics85.pdf) Computer Intell., 1985

**Possible Other Readings: Christofides***,* Ch. 4. (GCP); **TR** p485; **M&F,** p61+, p89+,Ch. 8 (TSP)

**3.1.3** Application of **gs-dfs/bt** to other NPC problems

1. **General Graph Coloring Problem (GCP)**

A.1a **Definition (English)**: if a map can be colored with q colors, then its

*chromatic* number is q assuming adjacent regions cannot be colored

the same (colored regions are thus independent - independent set!)

- size of solution space: power set! 2n where n is the number of regions

- model: map each region of map to node, then adjacent nodes can-

not have the same color by definition of problem constraints. (an

independent set of nodes/vertices).

A.1b **Definition (formal)**:

- objective function: min where is 1 if node *ni* is color *j* with cost *p*j; *chromatic number* is smallest number of colors needed.

- constraints: 1)  1 , *i* = 1, …*n* assuring that a specific node

is assigned only one color.

2) , for all *k* = 1*, … m* (m edges) and *j* = 1*, … q* where [*bij* ] is the incidence matrix such that *bij* = 1 if node *ni* is connected to edge *ek* and *bij* = 0 otherwise. This feasibility condition assures that only one of the two end nodes of *ej* can be colored with color *j* (mq constraints!). Observe that Problem Domain model can be considered as an integer linear model (linear objective, linear constraints, 0-1 model).

A.2 **gs-bfs/bt solution**: (see Christofides – Ch. 4). “Heuristics from PD?”

A.3 **Other solutions**: gs-dfs/bt, best first search, linear programming-simplex, dynamic programming (bfs type), combination of independent set problem and set covering problem (Christofides-Ch. 4), …; GCP algorithm animations – see web. “Heuristics?”

A.4 **Approximate solutions:** gs-dfs/btincremental techniques;

Example - try to color next selected node with specific color (selection) and when no further nodes can be colored (feasibility constraint), then select new color and start again until all nodes colored (solution); ILS; SLS; semi dfs approaches; **constructor-analyzer-prioritizer** of set-of-candidates (see Lim et al) ; -- time and bookkeeping complexity for each?!. “Heuristics”

A.5 **Proof** of GCP being NPC, see references; for algorithm correctness, see references; 4 color problem for planer graphs proof (see Saaty and Kainen)

A.6 **Benchmarks:** DIMACS, see web.

A.7 **Related SCP Problems:** Vertex Coloring, k-colorable graphs, list coloring, road coloring, Edge Coloring, bandwidth multi-color, Rank Coloring, …

A.8 **Applications:** task scheduling, time tables, loading containers, resource allocation, map coloring, ALU register allocation – compiler, ...

A.9 **Comments:** The graph coloring problem has been variations; In general deterministic graph search, the size of the solution space is usually smaller than the size of the search space? Why?

A.10 Other **references to consider**:

Skuena, “The Algorithm Design Manual”, Springer, GCP – p329

H & S, “SLS,” Morgan Kaufmann, GCP – p468

<http://en.wikipedia.org/wiki/Graph_coloring>

<http://mathworld.wolfram.com/Four-ColorTheorem.html>

# Saaty and Kainen, “The Four-Color Problem,” Dover, 1986

Jensen and Toft, “Graph Coloring Problems,” Wiley, 1994

Lim, Zhu, and Rodriques, “Heuristic Methods for Graph Coloring   
 Problems,” ACM Symposium on Applied Computing, 2005

A.11 Can a “[perfect graph](https://en.wikipedia.org/wiki/Perfect_graph)” can be colored with two colors? The associated graph color problem can be solved in polynomial time.

1. **General Traveling Salesman Problem (TSP):**

B.1a **Defintion (English)**: Given a complete undirected graph G that has a nonnegative integer cost (weight) associated with each edge xij , and find a hamiltonian cycle (a tour that passes through all the vertices i and j only once) of G with minimum cost defined by a minimum “distance” between cities.

- size of solution space: (n-1)! ~ (n-1)n-1 (use of [Sterling’s   
 formula](https://en.wikipedia.org/wiki/Stirling%27s_approximation)), where n is the number of cities;   
- TSP is hard as any problem in NPC !

- types of TSP graphs: symmetric, asymmetric, Euclidean distance, [Manhattan distance](http://en.wikipedia.org/wiki/Manhattan_distance), Chebyshev distance, … “reduced search space size! Still NPC!”

B,1b **Definition(formal)** Find xij in Z (integer domain; {0,1}) , i,j in {0,1, …n-1}, i j, **minimize**  , where dij is the “distance” between cities,

**subject to** **“feasibility constraints!”**

xij  0 for all i,j and

xij  1 for all i,j “xij  is in {0,1}; i.e. is either a 0 or 1; an edge xij is either in a path or not”

 = 1 for all i and

= 1 for all i “ Exactly one city is visited just before and   
 exactly one city after”

ui - uj + nxij  n - 1 for all i in {0; 1; : : : ; n }, j in {1,2,…n-1}, ui in Z with i in {0, 1, n-1}

since xij  = 1 implies that uj  ui +1; ui = j means the I’th   
 is the j’th city visited.

“ Elegant representation for constraining a graph tour to only one cycle, Why?””

*Observe, that this formulation of the TSP is in the* ***integer linear problem domain*** *(see Ahuja)****.*** *For another TSP formalization see*[**http://www.me.utexas.edu/~jensen/ORMM/models/unit/combinatorics/tsp.html**](http://www.me.utexas.edu/~jensen/ORMM/models/unit/combinatorics/tsp.html)*“ Why are PD formulations useful in designing algorithms?”*

B.2 **gs-dfs/bt solution:** (seeFroushani and Yusuff) – “**Heuristics?**” (set of candidates: list of cities not visited yet; possibly sorted based upon number

of links and cost?); **constructor-analyzer-prioritizer** of set-of-candidates; **MST?**; backtracking branch and bound; Christofides heuristic length - (<http://www.ieor.berkeley.edu/~kaminsky/ieor251/notes/2-16-05.pdf> )

B.3 **Other solutions:** bfs (A\*, dynamic programming, branch and bound – DFS/BT and BFS)

B.4[**Approximation**](https://cstheory.stackexchange.com/questions/9241/approximation-algorithms-for-metric-tsp) **solutions:** Lin and Karnighan k-opt (ILS), SLS, MST, …

“% away from optimal?” TSP is a strong NP-C (Hard) problem!  
But, for a Euclidean TSP, there is a FPTAS (Arora, 1996)! A 3/2 approximation factor is possible.

B.5 **Related NPC problems:** complete weighted graph, Hamiltonian path problem, matching,

B.4 **Proof** of TSP in NPC (see Garey and Johnson);

Also for generic TSP and other NPC problem analysis techniques see <http://www.nvc.vt.edu/gregwk/cs5104/assets/course_notes/course_notes_2006-04-20.pdf>

B.5 **Benchmarks:** [TSPLIB](http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html) (see Applegate for example),DIMACS, [TSP Test Sets](http://www.math.uwaterloo.ca/tsp/data/index.html), web.

B.6 **Applications:** planning, logistics, manufacturing processes, DNA sequencing, resource allocations, printed circuit layout, traffic modeling,

B.7 **Comments:**

1) TSP is probably the most studied NPC problem because of its nature as one of the “hardest” NPC problems, O(nn).

2) Many, many algorithms are proposed in the literature for various TSP variations.

3) Most approximation TSP algorithms use DFS techniques.

4) The TSP and many of the studied NPC problems can be represented formally as an integer linear PD model as well as an optimal flow problem (remember CSCE586 discussions?).

5) Observe again that a particular class of Linear “Programming” (LP) problems of the form: min {**c**T**x** : **Ax** = **b**, **x** 0, **x** in Zn}

is called an Integer Linear “Programming” (ILP) problem model. When some of the variables are allowed to be real values, the problem is called a Mixed Integer Linear “Programming” (MILP) problem.

B.8 Other **references**: (Find additional TSP web sites and references?!)

<http://valis.cs.uiuc.edu/~sariel/research/CG/applets/tsp/TspAlg.html>

<http://en.wikipedia.org/wiki/Travelling_salesman_problem>

<http://mathworld.wolfram.com/TravelingSalesmanProblem.html>

<http://www.tsp.gatech.edu/>

<http://www.me.utexas.edu/~jensen/ORMM/models/unit/combinatorics/tsp.html>

1. Ahuja, R.K., Mangnanti, T.L., Orlin, J.B., “Network Flows,” Prentice-Hall, New Jersey. 1993
2. Applegate et al , “Certification of an optimal TSP tour through 85,900 cities,” 2006 <http://www2.isye.gatech.edu/~wcook/papers/proof.pdf>
3. Applegate, Bixby, Chvatal & Cook, “The Traveling Salesman Problem, A Computational Study.” Princeton University Press, 2006
4. Froushani and Yusuff, “Development of an Innovative Algorithm for the Traveling Salesman Problem (TSP),” Eur. J of Sci Research, v29, 2006
5. Garey and Johnson, “[Computers and Intractability: A Guide to the Theory of NP-Completeness](http://en.wikipedia.org/wiki/Computers_and_Intractability:_A_Guide_to_the_Theory_of_NP-Completeness),” W.H. Freeman, 1979

1. **Maximum Flow and Minimum Cut Problems**

*(Review discussion of the MFP from CSCE586 - KT)*

The maximum flow problem (MFP) for example is a fundamental problem in discrete optimization used for various academic studies. Many references exist exploring this model and solution. Also, rapidly solving an online sequence of MFPs arises in real-world situations.

C.1a **Description (English):** The flow on arc (i, j) must be no less than lij and can be no greater than uij. To set up the problem in the framework of an optimization problem, a unit flow cost cij, incurred by each unit of flow moving through arc (i, j), must be defined.

Flows must satisfy another condition, known as Kirchhoff’s Law (conservation of flow constraint): for every node in the network, the sum of all incoming flow plus the flow produced by the node must be equal to the sum of all incoming flow plus the flow consumed at the node.

The objective of the MCNF problem is to determine the flow on each arc of the network, such that all of the flow *(commodity)* produced in the network is moved from source nodes to the sink nodes in the most cost effective way.

**C.1b Formal model**: The Minimum Cost Network Flow Problem (MCNF)

can be formulated as the following linear program model:

**minimize:**

, where cij is the cost of transporting a unit of flow on arc (i, j),

and xij denotes the flow on arc (i, j).

**Subject to the constraints (feasibility relations):**

**,** where bj denotes the quantity of traffic   
 produced or consumed at node j; “conservation of flow”

lij xij uij , where lij and uij denote, respectively, the lower and upper bounds on flow on arc (i, j).

*with the Parameter values:*

• bj > 0: node j is a source node,

• bj < 0: node j is a sink node,

• bj = 0: node j is a trans-shipment node.

• uij = 1: uncapacitated network flow problem.

• note that lij ,the lower bound, can be set to zero.

**C.2 Related Problem Types :** MFP variations (Wikipedia), generic network flow, min edge flow, max commodity flow, …

**C.3 Algorithmic Solutions**: gs\_dfs/bt, gs\_bfs, Ford-Fulkerson, capacity scaling, … approximation algorithms; **“Integrate PD with selected standard search algorithm constructs using heuristics - insight from problem domain!”**

**C.4 Applications:** Data mining, Open-pit mining, Project selection, Airline scheduling, Bipartite graph matching, Baseball tournament elimination, Image segmentation, Network connectivity. Network reliability, Distributed computing layout,, Egalitarian stable matching, Security of statistical data, Network intrusion detection, Multi-camera scene reconstruction, network centric problems, …

**C.5 Comments:**

1) Again, observe the direct similarities between this flow model, the LIP model, and the other NPC problem formulations.

2) Thus, again, one can use algorithms for one PD to solve problems from other PDs.

**C.6 References:** *“Places to start”*

[**http://en.wikipedia.org/wiki/Maximum\_flow\_problem**](http://en.wikipedia.org/wiki/Maximum_flow_problem)

[**http://en.wikipedia.org/wiki/Max-flow\_min-cut\_theorem**](http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem)

1. Kleinberg and Tardos, “Algorithm Design,” Chap.7, Network Flow, AD
2. Mark P. Kleeman, Benjamin A. Seibert, Gary B. Lamont, Kenneth M. Hopkinson and Scott R. Graham, “Solving Multicommodity Capacitated Network Design Problems using Multiobjective Evolutionary Algorithms,” Development of various MCNDP models, IEEE paper, 2009

**3.2 Returning to General NPC Problem Classifications**

***(Optimize some objective function subject to constraints!)***

**3.2.1 General NPC Problem References**

[**http://en.wikipedia.org/wiki/List\_of\_NP-complete\_problems**](http://en.wikipedia.org/wiki/List_of_NP-complete_problems)

[**http://www.csc.liv.ac.uk/~ped/teachadmin/COMP202/annotated\_np.html**](http://www.csc.liv.ac.uk/~ped/teachadmin/COMP202/annotated_np.html)

**Examples :**

1. Constraint satisfaction problems (Circuit-SAT, 3-SAT, …)
2. Numerical NPC problems (0-1 Knapsack Problem, …)
3. More NPC Graph Problems (Steiner Graph, …)
4. Assignment Problems (Quadratic AP, Task Scheduling, …)
5. Set and String Problems (Covers, Set Packing,…)
6. String Problems ( Cryptography, Pattern Recognition, …)
7. Flow Problems (min edge flow, max commodity flow, …)
8. Max 3-SAT problem; best approx algorithm has 7/8 ratio

*“One could expand CSCE586 and CSCE686 discussions and the development of NPC problem descriptions by presenting each in a formal syntactical PD format – use references of course.*

*Then, given a real-world complex problem, define formally the application problem domain, attempt to map it to a generic NPC problem, and solve it for the moment using a deterministic dfs or bfs algorithm with generic* ***heuristics*** *– later stochastic search approaches”*

***(Don’t reinvent the “wheel” – use existing algorithm and code)***

**3.2.2 Some real-world NPC problems in applications:** (from KT, MF, Talbi, …)

Aerospace engineering: optimal mesh partitioning for finite elements.

Biology: protein folding.

Chemical engineering: heat exchanger network synthesis.

Civil engineering: equilibrium of urban traffic flow.

Computer Science: data mining, scheduling

Economics: computation of arbitrage in financial markets with friction.

Electrical engineering: VLSI layout.

Environmental engineering: optimal placement of contaminant sensors.

Financial engineering: find minimum risk portfolio of given return.

Game theory: find Nash equilibrium that maximizes social welfare.

Genomics: phylogeny reconstruction.

Mechanical engineering: structure of turbulence in sheared flows.

Medicine: reconstructing 3-D shape from biplane angiocardiogram.

Network engineering: connectivity. reliability, Distributed layout

Operations research: optimal resource allocation, scheduling

Physics: partition function of 3-D Ising model in statistical mechanics.

Politics: Shapley-Shubik voting power.

Pop culture: Minesweeper consistency.

Security: Security of information, Network intrusion detection

Statistics: optimal experimental design.

**3.3 On line vs. Off line algorithms:** ([Wikipedia](http://en.wikipedia.org/wiki/Online_algorithm), …)

Defn: An online algorithm is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start. In contrast, an offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand. (For example, selection sort requires that the entire list be given before it can sort it, while insertion sort doesn't. Others? What about search algorithms?)

**Dynamic NPC problem Characteristics:**

− *Actuality*: Dynamic variables (graph, table, …) and dimensions change with the time.

− *Continuity*: The problem changes partially and quantitatively with time.

− *Robustness*: Unexpected situations (vertex/edge.set is deleted or inserted) should meet   
 with a quick response.

− *Effective:* Dynamic NPC problems can require getting an optimal solution in reasonable time.

**3.3.1 Some Possible Online NPC or worst problems:**

***Canadian Traveller Problem (CTP):*** A problem exemplifying the concepts of online algorithms is the [Canadian Traveller Problem](http://en.wikipedia.org/wiki/Canadian_Traveller_Problem). The goal of this problem is to minimize the cost of reaching a target in a weighted graph where some of the edges are unreliable and may have been removed from the graph. However, that an edge has been removed (*failed*) is only revealed to *the traveller* when she/he reaches one of the edge's endpoints. The worst case for this problem is simply that all of the unreliable edges fail and the problem reduces to the usual [Shortest Path Problem](http://en.wikipedia.org/wiki/Shortest_path_problem). An alternative analysis of the problem can be made with the help of competitive analysis. For this method of analysis, the offline algorithm knows in advance which edges will fail and the goal is to minimize the ratio between the online and offline algorithms' performance. This problem is [PSPACE-complete](http://en.wikipedia.org/wiki/PSPACE-complete).

***Dynamic Traveling Salesman Problem (DTSP).*** The DTSP is a generalization of the classic Traveling Salesman Problem (TSP) in which customer requests arrive in an on-going fashion. Consequently, the salesman has to replan his route as soon as a new service request arrives. The objective is to minimize the expected total waiting time, i.e. the sum of the expected waiting times of all the customer demands. The DTSP falls under the broad category of real-time fleet management. A large part of the current literature is characterized by algorithms reacting to new requests only once they have occurred, while neglecting available stochastic information. Overviews of these problems can be found in, and the recent paper by Ghiani et al which builds on the work of Powell, Jaillet and Odoni, Psaraftis, and Gendreau and Potvin. Another line of research, which is particularly relevant, examines dispatching and routing policies whose performance can be determined analytically if specific assumptions are satisfied. A related area of research, often referred to as Stochastic Vehicle Routing, examines problems in which demand becomes known at the beginning of each day. In this context the problem is to determine, on the basis of a probabilistic characterization of random data, a solution of least expected cost in which the order of customers is fixed regardless of the demand realization for a particular day (an a priori solution). multivehicle stochastic vehicle routing problem. (many references)

Applet at <http://www.tjhsst.edu/~rlatimer/techlab07/Students/RWard/ProjectV1-6/Project/tsp2.html>   
Ghaini, *New Polices for dynamic TSP* at <http://poincare.unile.it/chefi/J11.pdf>

***Dynamic Routing Problem (DRP):***

Within the wide scope of logistics management, transportation plays a central role and is a crucial activity in both production and service industry. Among others, it allows for the timely distribution of goods and services between suppliers, production units, warehouses, retailers, and final customers. More specifically, Vehicle Routing Problems (VRPs) deal with the design of a set of minimal-cost routes that serve the demand for goods or services of a set of geographically spread customers, satisfying a group of operational constraints.

*Libraries (software):*

1. [**https://sites.google.com/site/vrphlibrary/**](https://sites.google.com/site/vrphlibrary/)
2. [**http://victorpillac.com/vroom/**](http://victorpillac.com/vroom/) **“general”**
3. [**http://hdl.handle.net/1721.1/59665**](http://hdl.handle.net/1721.1/59665) **“dynamic robot routing”**
4. [**http://www.roadnet.com**](http://www.roadnet.com)

***Dynamic Clique (MIS) Problems?:***

***Comment:*** Many of the approaches to these dynamic online NP-Hard problems that search the solution space. Examples include simulated annealing, Tabu search, bio-inspired population approaches [DNA-Genetic algorithms, …; Ant systems, Bees, Brain- neural nets, artificial immune systems, ..]

**“Maybe an associated CSCE686 project for solving dynamic NPC problems?**

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

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

**YouTube Discussions of NPC Problems**

**Bin Packing Problem:** [**https://www.youtube.com/watch?v=kiMFyTWqLhc**](https://www.youtube.com/watch?v=kiMFyTWqLhc) **(2D, 3D?)**

**Coloring Problem:** [**https://www.bing.com/videos/search?q=utube+coloring+problem&view=detail&mid=398A7C289F2F146EEF86398A7C289F2F146EEF86&FORM=VIRE**](https://www.bing.com/videos/search?q=utube+coloring+problem&view=detail&mid=398A7C289F2F146EEF86398A7C289F2F146EEF86&FORM=VIRE)

**MIS/Clique Problem:** [**https://www.youtube.com/watch?v=\_ZKPP5PrHOk**](https://www.youtube.com/watch?v=_ZKPP5PrHOk)

**SCP:** [**https://www.bing.com/videos/search?q=utube+set+covering+problem&view=detail&mid=7B21022A5CE9C1C0914C7B21022A5CE9C1C0914C&FORM=VIRE**](https://www.bing.com/videos/search?q=utube+set+covering+problem&view=detail&mid=7B21022A5CE9C1C0914C7B21022A5CE9C1C0914C&FORM=VIRE)

**TSP:** [**https://www.bing.com/videos/search?q=utube+TSP+problem&view=detail&mid=6E7615DBC3AF468ECF1C6E7615DBC3AF468ECF1C&FORM=VIRE**](https://www.bing.com/videos/search?q=utube+TSP+problem&view=detail&mid=6E7615DBC3AF468ECF1C6E7615DBC3AF468ECF1C&FORM=VIRE)

**Vertex Covering Problem:** [**https://www.youtube.com/watch?v=kiMFyTWqLhc**](https://www.youtube.com/watch?v=kiMFyTWqLhc)

**Max Flow Problem:** [**https://www.bing.com/videos/search?q=utube+max+flow+problem&view=detail&mid=8E7888D38A9528C1BA7E8E7888D38A9528C1BA7E&FORM=VIRE**](https://www.bing.com/videos/search?q=utube+max+flow+problem&view=detail&mid=8E7888D38A9528C1BA7E8E7888D38A9528C1BA7E&FORM=VIRE)

**“Other YouTube sites prove a particular problem is NP - Complete”**

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

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

***Remember - “Seek and Ye shall find!”***

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

**QUIZ – Closed Book! (Six Minutes)**

1. **Within an approximated MIS problem algorithm, what is the selection operation? Is this a FPTAS solution? Why?**
2. **What is the problem domain complexity for the following problems:**
   * 1. SAT
     2. MIS/Clique
     3. SCP
     4. VCP
     5. TSP
     6. VRP
     7. GCP
     8. Hitting Set
     9. Hamiltonian path
     10. Knapsack Problem
     11. Rubik’s cube solution
     12. [Canadian Traveller Problem](https://en.wikipedia.org/wiki/Canadian_traveller_problem) (PSpace-Hard)