

CMSC/AMSC 660-001 Fall 2004

Final Project

Author: Aditya Kalyanpur

Show all work.

All work must be your own (i.e., no group efforts are allowed).

If you use a reference book, cite it, or you will lose credit!

In this homework, we shall investigate the problem of **finding the maximum independent set in a graph** using various search methods, with primary focus on the **Metropolis Algorithm** seen in class.

Background

Definition: An independent set (IS) of a graph is a set of vertices of the graph that have **no edges in common**. The number of vertices in the set gives the cardinality of the IS and the IS with the maximum cardinality is called the maximum (or maximal) independent set (MIS) of the graph.

Note that in general, the solution may not be unique, i.e. there could be more than one MIS in a graph (depending on its structure). In this assignment, however, we are concerned with finding any one MIS. Also note that the problem of finding the MIS of a graph is a classical graph theory problem and its complexity is known to be NP-Complete [1].

Significance: A number of real world problems (e.g. in coding theory, scheduling) can be reduced to the problem of finding independent sets in a graph, where the goal is to seek a set of mutually separated points and avoid any conflicts between elements. For example, suppose you are trying to identify locations for a new franchise service such that no two locations are close enough to compete with each other. In this case, you can construct a graph where the vertices are possible locations, and add edges between any two locations deemed close enough to interfere. The maximum independent set gives you the maximum number of franchises you can sell without jeopardizing sales. [2]

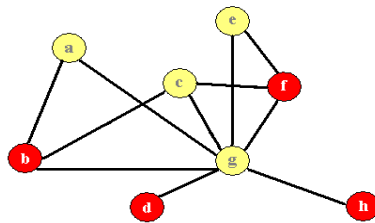


Figure 1: In the sample graph shown, the vertices $\{b, d, f, h\}$ form an independent set (IS) since there are no direct edges interconnecting them

Problem 1. (5) In Figure 1, is the IS highlighted (with cardinality 4) the maximum IS of the graph shown? List all independent sets with cardinality ≥ 4 (if any), and give any one MIS of the graph.

Data for our Problem: The graph data to be used in the problems below is given in the file `graph.mat`. Basically, it specifies a square matrix G that

represents an *undirected graph* (edges don't have directions on them) with 100 vertices. Dimensions of G are 100 X 100, and the matrix contains only 1's and 0's. The notation for the elements of G is as follows: the row (i) and column (j) of G specify vertices of the graph (i, j), such that if:

- $G(i, j) = 0$ *implies* No edge exists between vertices i, j of the graph
- $G(i, j) = 1$ *implies* An edge exists between vertices i, j of the graph

Additionally, G satisfies two properties:

- $G(i, i) = 1$, *since a vertex is implicitly connected to itself*
- $G(i, j) = G(j, i)$, *since if vertex i is (not) connected to j , obviously vertex j is (not) connected to i*

Thus, the square matrix G is symmetric and all its diagonal elements are 1.

Note on Graph Representation: Various data structures can be used to represent graphs such as the *adjacency list*, *incidence matrix* etc [3]. The choice of data structure can play a crucial role in deciding performance–efficiency of the algorithm, depending on the size of the graph, density, directed or undirected, and operations to be performed on it. The above representation corresponds to an *adjacency matrix* and is chosen to ease computation (at the expense of storage) in our problem solutions.

Techniques to find Independent Sets in a Graph

Since our graph has 100 vertices, the natural state space for the maximal IS solution is 2^n ($n = 100$) subsets of vertices, too large for an exhaustive search. Hence, we need to explore alternate search strategies to find the optimal solution.

A greedy approach: A greedy algorithm is a systematic step-wise search algorithm that always takes the best immediate, or local, solution while finding an answer. Greedy methods are usually quicker, since they converge to a local optimum without considering all possible alternatives. On the other hand, the approach may not always find the global optima. In this case, the algorithm can be rerun with different starting points in the hope of finding the global optima.

The following is a greedy algorithm to solve our problem:

1. Represent the solution by a bit vector \mathbf{s} of length n ($n = 100$) such that: for $1 \leq i \leq n$, if $\mathbf{s}(i) = 1$, vertex i is part of the IS; else $\mathbf{s}(i) = 0$ and vertex i is excluded from the IS. **Thus, a solution vector corresponds to a subgraph of G , say G_s**
2. Define the objective function for the solution vector (\mathbf{s}) as:

$$\text{Obj}(\mathbf{s}) = (\text{No. of vertices} - \text{No. of edges}) \text{ in subgraph } G_s$$
Our goal is to **maximize** this objective function
3. Set the solution vector s_k (initially $k = 0$) randomly and compute its objective value $\text{Obj}(s_k)$

4. Find a neighboring solution s_{k+1} by flipping one of the bits (chosen at random) in s_k
5. Compute the objective value of the neighbor as $Obj(s_{k+1})$:
Only if $Obj(s_{k+1}) > Obj(s_k)$, set $s_k = s_{k+1}$ (*Greedy!*)
6. Return to step 4. Continue the loop (4-6) till no improvement is obtained in s_k for a consecutive number of tries (100). Our final solution is s_k

Problem 2(a). (8) Using the greedy approach described above, write the MATLAB code (documented appropriately) to solve the Maximum IS problem for the given graph G. Run the greedy algorithm over 10 cycles starting with a different (random) solution vector s_0 each time, and for each cycle determine:

1. Solution found at the end of the cycle
2. Number of iterations taken to find it
3. Variation of objective function
4. Also, store the **best** solution i.e. one with the current highest objective value (along with 2 and 3 above)

Finally (at the end of 10 cycles), print the best solution, corresponding iteration count and plot its objective function values. Explain your results.

Hint: Note that the objective function $Obj(s)$ is evaluated in each iteration and hence you need to optimize this function code (e.g. compute difference in number of vertices/edges between solutions)

Problem 2(b). (2) Calculate the total run time (order) of the greedy algorithm (without the random restarts).

Problem 2(c). (5) What is a key limitation of the greedy method above (besides not being able to find the global optima) due to the way in which the objective function is defined? How can the objective function be modified to overcome this limitation? What are the drawbacks of doing so, if any?

Hint: Does the solution found at the end of a greedy cycle **always** represent an IS?

Simulated Annealing approach: A drawback of the greedy approach seen in Problem 2 is that it gets trapped at the local optimum and thus may fail to find the global optima for the given problem (in spite of the random restarts). In this case, we need to use other search algorithms which explore a greater fragment of the solution space without getting trapped. One such method is the **Metropolis algorithm seen in class** that is based on the notion of annealing. This method inspired by the process of cooling molten materials down to a solid state, uses a temperature variable to determine the probability of accepting 'bad' transitions (initially they are accepted with a higher probability, corresponding to global exploration) and an annealing schedule that gradually decreases the temperature, causing the system to gradually accept fewer and fewer 'bad' changes (corresponding to local optimizations/tuning). Note that in all iterations, better solutions are always accepted.

The algorithm consists of the following key components, each of which needs to be selected carefully for the given problem:

1. Representation of the solution space and a computable objective function for each intermediate solution.
2. Transition mechanism between intermediate solutions

Note: The above two components were accounted for in Problem 2.

3. *Cooling schedule* which determines the probability of accepting bad transitions and consists of:
 - (a) Initial temperature (T_0)
 - (b) Temperature decrement function (dT)
 - (c) Number of iterations between temperature change (dn)
 - (d) Acceptance criteria for bad transitions ($P(\delta Obj(s), T)$)
 - (e) Stop Criteria or final temperature (T_f)

Unfortunately, there are no *generic* optimal settings for the parameters above. Choosing an appropriate annealing strategy is **problem-dependent** and usually determined by a trial and error process. Below, we consider three sample strategies:

Strategy I	Strategy II	Strategy III
$Obj(s) = N_v - (N_e * p)$	$Obj(s) = N_v - (N_e * p * T_0/T_k)$	$Obj(s) = N_v - (N_e * p)$
$p = 4$	$p = 1$	$p = 0.75 + (T_0/T_k)/375$
$T_0 = 45$	$T_0 = 45$	$T_0 = 45$
$dT : linear$ $T_{k+1} = T_k - 1.25$	$dT : exponential$ $T_{k+1} = 0.875 * T_k$	$dT : logarithmic$ $T_k = 0.75 * T_0 / (k + \log(k)); (T_k > 1)$ $T_{k+1} = T_k - 0.1; (T_k < 1), linear$
$dn = 250$	$dn = 250$	$dn = 250/T_k$
$P(\delta Obj, T_k) = e^{-\delta Obj/T_k}$	$P(\delta Obj, T_k) = e^{-\delta Obj/T_k}$	$P(\delta Obj, T_k) = e^{-\delta Obj/T_k}$
$T_f = 0$	$T_f = 0$	$T_f = 0$
Transition: Flip 1 bit	Transition: Flip 1 bit	Transition: $n_{FB} = \min(\text{round}(T_k/5) + 1, 8)$

Some notes on the strategies listed in the table above:

1. In all cases, we choose the same representation of solution space as in *Problem 2*.
2. The variable p used in the objective functions is an error penalty for edges in the subgraph induced by the solution.
3. The term δObj appearing in the probability acceptance function $P(\delta Obj, T)$ corresponds to a change in the objective value from the current solution (s_k) to the next one (s_{k+1}). **If this change is positive, we always accept s_{k+1}** , else we select s_{k+1} with a probability returned by $P(\delta Obj, T)$.
4. The transition mechanism between intermediate solutions (referring to the last row in the table above) is the same for strategies I and II i.e. one bit flipped randomly between solutions, whereas its a variable parameter in strategy III i.e. number of bits flipped (n_{FB}) dependent on current temperature (T_k) as shown, i.e. minimum of two values, $\text{round}(T_k/5) + 1$ and 8.

5. The stopping criteria for a single annealing cycle is one of three conditions:
 - (a) Final temperature $T_f(=0)$ is reached
 - (b) No improvement in solution (i.e. system is frozen) for a consecutive set of iterations (100)
 - (c) A max limit (10^4) on the total number of iterations is reached

Our next task is to **compare the above three annealing strategies** to derive an intuition into the working of the simulated annealing process for our specific problem.

Problem 3(a). (15) Write the MATLAB code to solve the maximum IS problem for the given graph G using Simulated Annealing based on the three annealing strategies (cases) described in the table. For each case, plot the variation of the objective function ($Obj(s)$). Also, store the final solution obtained and the total number of iterations required to reach the solution. Comment on the quality of the results by determining the **best** and **worst** annealing strategy.

Problem 3(b). (5) Based on the results you get in Problem 3(a), explain the effects of (and discuss the tradeoffs in) selecting the following parameters:

1. Objective function $Obj(s)$ and edge penalty p
2. Transition mechanism between intermediate solutions
3. Cooling schedule, specifically parameters: dT and dn

Hint: For example, consider how changing the number of bits to flip between solutions (as done in Strategy III) can help improve search space coverage

Problem 4. (5) Compare the result obtained in Problem 2 with the best result of Problem 3 by answering the following questions: How does the quality of results compare? How does the amount of work compare?

Tools:

1. One of the most comprehensive books on simulated annealing is ‘*Simulated Annealing: Theory and Applications (Mathematics and Its Applications (Kluwer))*’ by P. J. M. Van Laarhoven, E. H. L. Aarts
2. The standard reference text book for algorithms (covers greedy and randomized algorithms) is ‘*Introduction to Algorithms, Second Edition*’ by T. Cormen, C. Leiserson, R. Rivest, C Stein
3. For a good introduction to simulated annealing on the web see:
 1. <http://members.aol.com/btluke/simann1.htm>
 2. http://en.wikipedia.org/wiki/Simulated_annealing
3. Numerous heuristics have been suggested to improve the regular annealing process such as *adaptive* annealing, *nested* annealing etc. In general, you need to study problem-specific characteristics in accordance with varying effects of annealing parameters (maybe using such heuristics) in order to obtain the best annealing strategy.

References:

- [1] Karp, R. M. "Reducibility Among Combinatorial Problems." In Complexity of Computer Computations, (Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972). New York: Plenum, pp. 85-103, 1972. (For a definition of NP-Complete problems see <http://en.wikipedia.org/wiki/NP-complete>)
- [2] Example of practical use of independent sets taken from: <http://www.cs.sunysb.edu/algorithm/files/independent-set.shtml>
- [3] For more on graph data structures, see <http://www.mathworld.com>