

CMSC/AMSC 660-001 Fall 2004
Final Project
Author: Aditya Kalyanpur
Solution Sheet for Homework Assignment

Solution 1. Problem 1 shows a graph with an independent set (b, d, f, h) of cardinality 4. However, this is **not** the MIS of the graph. Other IS with cardinality ≥ 4 are

- (a, c, d, e)
- (a, c, d, h)
- (a, d, e, h)
- (a, d, f, h)
- (b, d, e, h)
- (a, c, d, e, h) **MIS of the graph with cardinality 5, see Figure 1**

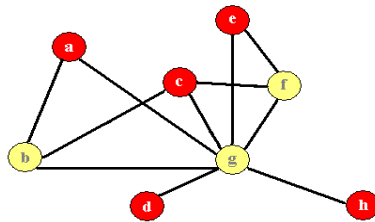


Figure 1: MIS of the graph is highlighted in red

Solution 2(a). The MATLAB code to solve the MIS problem for the given graph (G) using the greedy approach is attached (`greedy.m`). The code follows the specifications given in the problem statement and is appropriately documented. It uses a function `updateParams.m` to:

- compute the objective value of the next intermediate solution
- determine the vertices vector of the new solution subgraph
- find the number of edges in the new solution subgraph

For this, it looks at the bit that was flipped between the two intermediate solutions, and updates the number of vertices/edges in the new solution depending on:

1. whether the vertex corresponding to the flipped bit was added/removed from the old solution (obtained by checking the value of the bit in the solution vector)
2. number of edges that were introduced or eliminated by adding/removing this vertex (obtained by checking for edges between the vertex in question and other remaining vertices in the old solution; hence takes $O(n_v)$ steps, where n_v is the number of vertices in the old solution)

It then computes the new objective function value using the updated counts for vertices and edges, and also determines the new vertices vector using the old one (additional $\approx O(n_v)$ steps) . Hence, *running time* of `updateParams.m` for the greedy algorithm is $2 * O(n_v)$. Optimizing this function is critical since objective value is evaluated in each iteration of the greedy search.

Also, since the algorithm is greedy, only a test case that causes an *increase* in the objective function value is accepted as a next intermediate solution, otherwise its simply rejected. Thereafter, another bit is selected randomly to flip (using the precomputed random matrix) providing a new neighbor to test and the process goes on till there is no improvement in the solution for 100 consecutive iterations i.e. system is trapped. Finally, the entire greedy loop is run for 10 cycles, each initiated with a different (random) solution or starting point.

In order to analyze the system better, two different sample runs of the program are presented.

Sample Run 1:

```
No IS of graph found
Best Solution contains 16 vertices and 2 edge(s)
Columns 1 through 16
15 20 23 38 45 56 57 67 75 83 87 89 95 96 98 99
No. of iterations: 325
```

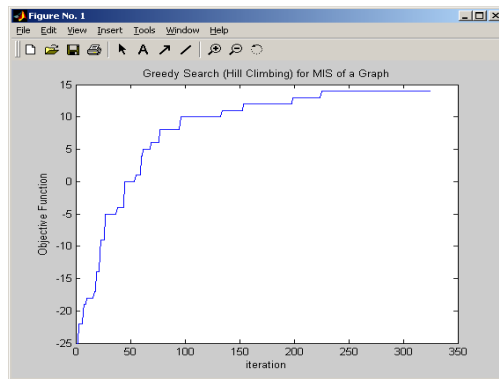


Figure 2: Objective Func. for **Sample Run 1** using the **Greedy Algorithm**

Sample Run 2:

```
IS of graph found
Best Solution contains 15 vertices and 0 edge(s)
Columns 1 through 15
2 4 29 32 35 36 37 56 64 67 73 76 89 94 96
No. of iterations: 338
```

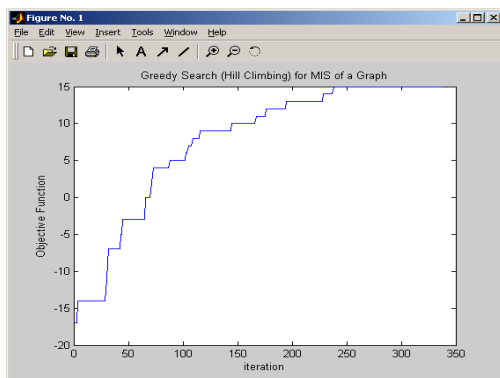


Figure 3: Objective Func. for **Sample Run 2** using the **Greedy Algorithm**

Explanation of results:

1. In a greedy algorithm, the objective function (or optimization metric in general) never makes a negative or bad transition during the search process. Moreover, it remains constant over a set of iterations for which no *immediate* better solution is found. Hence the incremental-step-like curve representing the function to be optimized is a distinct characteristic of the greedy method (also known as **Hill Climbing** due to its shape).
2. As seen in the first sample run, the final solution obtained at the end of the greedy search is **not** an independent set since the subgraph has 2 edges. At this point, the algorithm is trapped and it fails to progress towards a better solution. On the second sample run, the program succeeds in finding an IS with 15 vertices. However, due to the nature of the greedy method, we cannot guarantee that this is the MIS of the graph, only that the solution corresponds to the local maxima.
3. The greedy method is reasonably fast (especially compared to simulated annealing as observed later). The number of iterations taken to find a local maxima is roughly between 300 to 400 (average ≈ 350). This small number is due to the fact that the algorithm proceeds rapidly to the best solution it can find without considering possible alternatives. It may or may not produce accurate results depending on the choices made in the starting solution and the transitions between neighbors.
4. Finally, depending on the problem and/or system constraints, a greedy method is usually combined with random restarts to achieve better results without compromising speed altogether. In our case, the advantage gained by restarting (10 times) is evident by observing the output of a single greedy cycle alone (12 vertices with 1 edge, 13 vertices with 3 edges etc) versus the best output obtained over 10 cycles as shown above. Also, due to the restarts, we need to re-estimate the total number of iterations of our greedy program as approximately equal to $10 \times 350 = 3500$ (still smaller than the lowest observed run time of simulated annealing as we shall see later).

Solution 2(b). The total run time of the greedy algorithm can be determined as follows:

- cost of determining vertices in initial solution (`lines:50-55 in greedy.m`) is: $O(n)$, where n is the size of the solution vector (100)
- cost of computing initial objective value (`lines:60-67 in greedy.m`) is: $(n_v - 1) + (n_v - 2) + (n_v - 3) + 1 = n_v + n_v * (n_v + 1) / 2 = O(n_v^2 - n_v)$, where n_v is the number of vertices in solution (usually ≈ 14)
- cost of updating objective value and vertices vector (`updateParams.m`) is: $2 * O(n_v)$ as shown in Solution 2(a)
- The previous function evaluation is carried out until the system is trapped, ≈ 350 iterations as observed from the output results

Hence the total run time of the algorithm is $O(n) + O(n_v^2 - n_v) + 350 * (2 * O(n_v))$.

Solution 2(c). A key limitation of our greedy method is that the final solution obtained is not always an independent set (see examples discussed in Solution 2(a)). This is because the objective function, which is defined as (`No. of vertices - No. of edges`) in `solution subgraph`, does not eliminate the effect of edges altogether, rather it just penalises them i.e. -1 for every edge. As a result of this, a better solution, one that has a higher numerical objective value (even a positive value), *can* have edges as well. The solution vector represents an IS of the graph only if its objective value **equals** the number of vertices.

In order to overcome this limitation, the objective function can be made more strict by setting its value to 0 whenever any edge is found in the solution subgraph. However, a drawback of this approach is that the greedy search now becomes too narrow, since the distance (in terms of objective value) between neighboring solutions is drastically reduced (0 in most cases). Hence, only a small fragment of the solution space is traversed (rejecting most neighbors) and unless your lucky with a great starting point, very poor results are obtained at the end.

Another alternative is to increase the penalty for edges (currently 1) in the solution subgraph i.e. set error penalty (p) for edges as 2, 3, 4.. etc., where the new objective function is now defined as (`No. of vertices - p * No. of edges`) in `solution subgraph`. Higher the penalty, greater the chance that the final solution represents an IS, but narrower the search space and lesser the chance of finding a global maxima.

Solution 3(a). The MATLAB code to solve the MIS problem for the given graph using Simulated Annealing is attached (`annealing.m`). The single file **includes all three annealing strategies** as specified in the problem statement, and the strategy to test can be selected using the global integer variable *strategy*. Also, the file uses the same function `updateParams.m` as seen in the previous solution, to compute the objective value of the next intermediate solution and determine vertices vector and number of edges in the new solution.

Again, two sample runs for each strategy are presented in order to analyze the solutions better.

Strategy I: Poor Annealing (*static objective function, linear cooling*)

Sample Run 1

IS of graph found
Best Solution contains 11 vertices and 0 edge(s)
Columns 1 through 11
9 13 25 26 29 44 55 67 71 73 76
No. of iterations: 9001

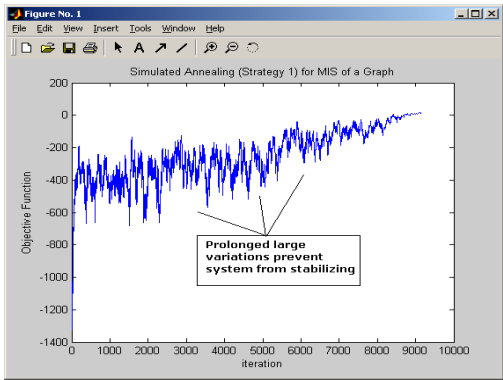


Figure 4: Objective Func. for **Sample Run 1** using **Annealing Strategy I** (with comments in boxes)

Sample Run 2

IS of graph found
Best Solution contains 13 vertices and 0 edge(s)
Columns 1 through 13
1 2 4 10 14 21 29 31 37 46 52 93 94
No. of iterations: 9001

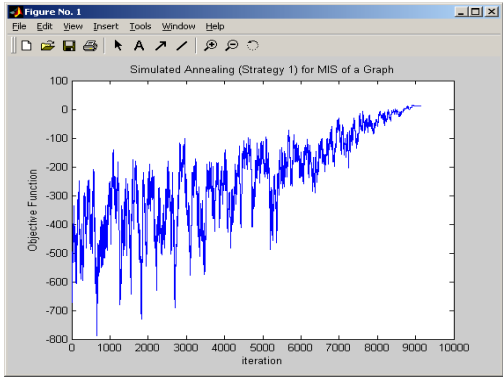


Figure 5: Objective Func. for **Sample Run 2** using **Annealing Strategy I**

Strategy II: Good Annealing (*dynamic objective function, exponential cooling*)

Sample Run 1

IS of graph found

Best Solution contains 16 vertices and 0 edge(s)

Columns 1 through 16

3 4 9 11 31 38 40 49 52 55 77 79 84 87 95 98

No. of iterations: 8792

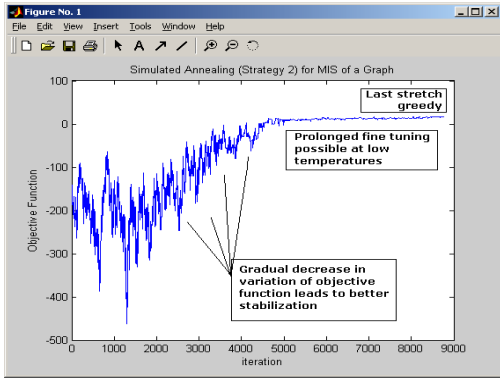


Figure 6: Objective Func. for **Sample Run 1** using **Annealing Strategy II** (with comments in boxes)

Sample Run 2

IS of graph found

Best Solution contains 17 vertices and 0 edge(s)

Columns 1 through 17

5 9 14 16 29 31 32 38 39 43 44 52 57 77 90 93 95

No. of iterations: 9310

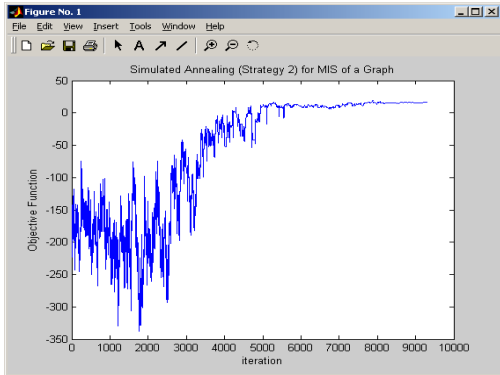


Figure 7: Objective Func. for **Sample Run 2** using **Annealing Strategy II**

Strategy III: Efficient Annealing (*better coverage of solution space, dynamic normalized objective function, logarithmic cooling with dynamic schedule*)

Sample Run 1

IS of graph found

Best Solution contains 20 vertices and 0 edge(s)

Columns 1 through 20

1 4 10 14 21 25 28 33 41 49 52 63 69 71 81 86 88 90 93 94

No. of iterations: 6851

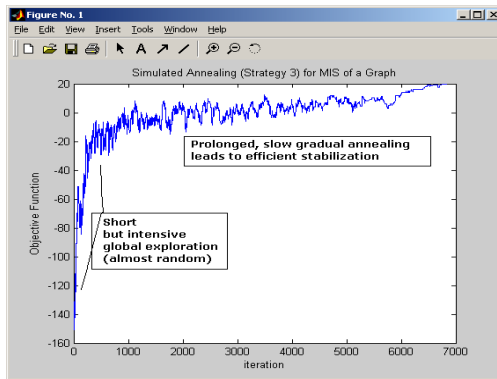


Figure 8: Objective Func. for **Sample Run 1** using **Annealing Str. III** (with comments in boxes)

Sample Run 2

IS of graph found

Best Solution contains 19 vertices and 0 edge(s)

Columns 1 through 19

4 9 14 26 31 36 38 40 43 49 55 57 60 77 79 81 84 90 95

No. of iterations: 7850

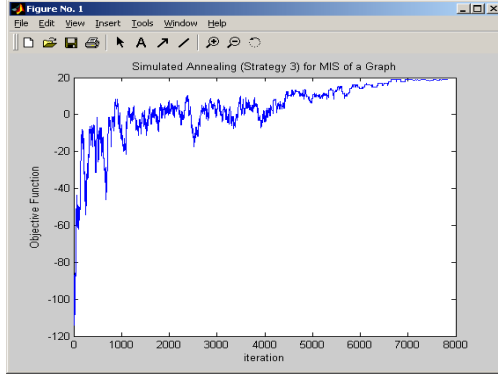


Figure 9: Objective Func. for **Sample Run 2** using **Annealing Str. III**

Note: In an effort to reach the global optima in a lesser number of iterations, Strategy III risks not returning an IS **always** as its end solution (a few cases have 1-4 edges as explained in solution 3(b)). However, the compromise is justified given that the overall observed performance of this strategy is superior to that of the previous two as shown below.¹

Results Summary:

Annealing Strategy	Best Output Solution (over 10 cycles)	No of Iterations
I	14 vertices, 0 edges	9001
II	17 vertices, 0 edges	8863
III	20 vertices, 0 edges	7545

Hence, in terms of quality of results and running time:

Strategy III (**best**) > Strategy II > Strategy I (**worst**)

Solution 3(b).

1. Effect of objective function and edge penalty:

We already discussed some of the tradeoffs in selecting an objective function and edge penalty (p) in solution 2(c), where we found that setting $p > 1$ helps to eliminate edges in our final solution, increasing the probability of the end result being an IS. Additionally, defining a strict objective function with a small distance between neighboring solutions results in poor (narrow) coverage of the solution space as most of the transitions are rejected. Hence, ideally, we need an objective function which allows us to traverse the solution space flexibly, *gradually* driving out edges from our solution.

In **Strategy I**, we use an objective function with a constant edge penalty (p) of 4. The latter ensures that final solutions have little or no edges in most cases. However, the objective function is static i.e. edges are penalized **equally** throughout the search process, resulting in a narrower search. Ideally, we would like to allow *more* edges in our solution at the

¹For a detailed analysis of the outputs, see the attached files `strategy1_output.txt`, `strategy2_output.txt` and `strategy3_output.txt`.

start of the search process to explore the solution space effectively, and *less* towards the end during fine tuning.

In **Strategy II**, we vary the edge penalty **with temperature**, initially setting it to 1 and gradually increasing it as the temperature drops (the ratio T_0/T_k increases as T_k decreases, for a constant initial temperature T_0). This gives us a more flexible objective function with greater solution coverage. On the downside, edge penalty variation is not normalized i.e. since $T_0 = 45$ (given) and $T_k \approx 0$ (when the system is near the end of the annealing phase), edge penalties at low temperatures are disproportionately high resulting in poor fine tuning and less optimal end results.

In **Strategy III**, we increase the edge penalty *carefully* with temperature, normalizing the ratio T_0/T_k such that the edge penalty remains in the range of 0.75 – 1.2. Hence, edge penalties in the middle and latter stages of the annealing cycle are not too high allowing better fine tuning of the solution to take place. **Note:** It is a function of the efficient cooling schedule for this strategy that ensures the final solution has no edges (**or in few cases, 1 – 4 edges**) even though the final edge penalty is not too high.

2. Effect of transition mechanism between intermediate solutions

In **Strategies I and II**, a single bit is flipped between intermediate solutions implying that only neighbors of the current solution are considered in the next iteration. This has the effect of moving *slowly* within the solution space. In order to speed up this process, we can allow transitions *beyond* neighbors. However, the tradeoff here is that large jumps between intermediate solutions makes the search process appear random. Hence, ideally, we need a dynamic transition mechanism that supports large transitions between solutions during the exploration phase (initially), and smaller transitions during the stabilizing and fine tuning phase (towards the middle and end of the annealing cycle). This is achieved in **Strategy III** by making the number of flip-bits **temperature dependent** ($= \min(\text{round}(T_k/5) + 1, 8)$), with an upper limit of 8 to prevent completely ad hoc transitions. Since $T_0 = 45$, initially as many as 8 bits are flipped between intermediate solutions (almost random search) but as the temperature drops, fewer bits are flipped until finally only neighbors of the current solution are considered (as in the other strategies). Note from the plots of objective function that the initial variations are much larger in Strategy III than in Strategies I and II.

3. Effect of the cooling schedule

This is a critical piece of the simulated annealing process and we discuss the effect of two of its key parameters here, in conjunction with the other variables discussed earlier such as objective function, edge penalty and transition mechanism:

- (a) temperature decay function (dT)
- (b) number of iterations for which temperature remains constant (dn)

Sidenote: In all three strategies, we select the same initial temperature $T_0 = 45$. The reason for this is the following: Kirkpatrick [1] suggested that a suitable initial temperature is one that results in an average bad-case acceptance probability of about 0.8. Now the empirically determined initial average decrease in objective function value was found to be 10. Since, our probability acceptance criteria is $P = \exp(-\delta Obj/T)$, we can determine the initial temperature as follows:
 $P_0 = \exp(-\delta Obj_0/T_0) = 0.8$ (suggested)
Hence, $T_0 = -\log(\delta Obj_0)/P_0 = -\log(10)/0.8 = 45$.

In **Strategy I**, dT is a linear decay function as shown in the plot below. Also, dn is set to a constant value of 250. The effect of this is a *equi-proportional* cooling schedule where the search is distributed across all temperatures equally. However, this strategy coupled with a static cost function and fixed transition mechanism results in poor end results due to a narrow exploration of the search space and no gradual stabilization of the system i.e. large variation in objective function as seen in its plot in solution 3(a).

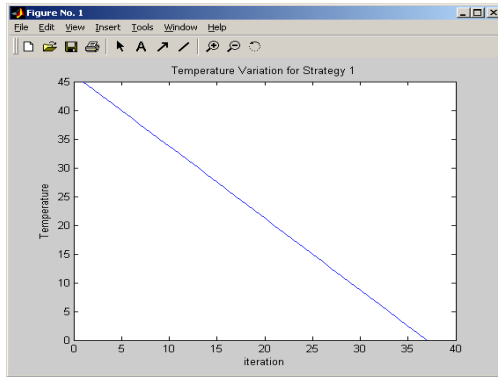


Figure 10: Temperature Decay in **Annealing Strategy I: Linear**

In **Strategy II**, dT is an exponential decay function as shown in the plot below while dn remains constant at 250. This is one of the most commonly used cooling schedules (in practice) because the gradual temperature decay allows the system to stabilize efficiently. Also, due to the exponential nature of the decay function, the final temperature never reaches 0. Hence, fine-tuning of the system can be continued for as long as required until its state is frozen i.e. no improvement in solution is found for a consecutive set of iterations. This cooling schedule coupled with an objective function that drives out edges with a decrease in temperature results in better end results than Strategy I.

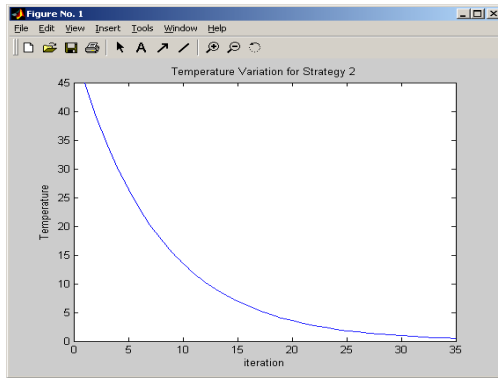


Figure 11: Temperature Decay in **Annealing Strategy II: Exponential**

In **Strategy III**, dT is primarily a logarithmic decay function (final stretch being a linear decay down to 0 to allow better fine tuning at very low temperatures) as shown in the plot below. Also dn is dynamic, being dependent on temperature ($dn = 250/T_k$). This cooling schedule coupled with a dynamic objective function and a dynamic transition mechanism has the following effects:

- (a) At high temperatures the decay rate is very fast (almost sudden drop in T) and the number of iterations per temperature is less resulting in a controlled (/restrained) random search (when transitions are large and edge penalty is small).
- (b) As the temperature drops below a certain low threshold (< 5), the decay rate is reduced and the number of iterations is increased resulting in prolonged gradual annealing (when transitions are smaller and edge penalty is larger).

Thus, Strategy III gives us the best overall results.

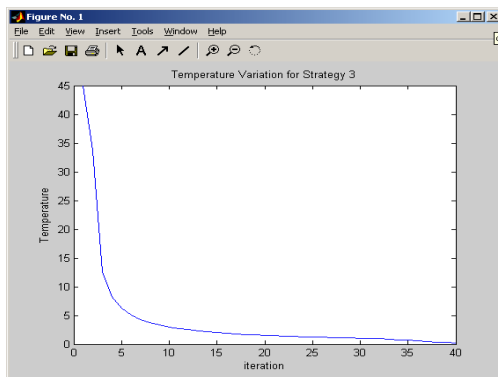


Figure 12: Temperature Decay in **Annealing Strategy III: Logarithmic**

Solution 4. Comparing the final solutions of the **greedy** approach versus the **best simulated annealing strategy (III)**:

- *Quality of results:* The best solution computed over 10 cycles for the annealing strategy (20 vertices, 0 edges) is clearly better than the greedy approach (15 vertices, 0 edges). Moreover, for a single cycle, the average-case annealing (18 vertices, 0 edges) works **significantly** better than the greedy method (14 vertices, 3 edges), which is obvious given the poor search space/direction of the greedy algorithm. Hence, random restarts *do* help the greedy algorithm but still tend to produce worse results than optimal annealing.
- *Number of iterations:* A single greedy cycle converges to a solution faster, taking nearly 20 times lesser iterations than the optimal annealing algorithm (annealing takes longer since it needs to stabilize by gradually eliminating bad transitions). Hence, a greedy approach *can* be combined with random restarts without compromising on speed altogether.

References:

[1] Kirkpatrick, S., Optimization by Simulated Annealing - Quantitative Studies, J. Stat. Phys. 34, 975-986, 1984.