MSSC 6000 - Final exam

Henri Medeiros Dos Reis

May 9, 2023

1. (a) The search space for this problem is continuous. In fact the search space is all $d_1 \in \mathbb{R}$, such that $0.0625 \leq d_1 \leq 1$, all $d_2 \in \mathbb{R}$, such that $0.0625 \leq d_2 \leq 1$, all $r \in \mathbb{R}$ such that $r \geq 10$, and all $L \in \mathbb{R}$ such that $L \leq 200$. Given that all the variables are also constrained by

$$\begin{aligned} d_1 &\geq 0.0193r, \\ d_2 &\geq 0.00954r, \\ \pi r^2 L + \frac{4}{3}\pi r^3 &\geq 1296000, \\ L &\leq 240 \end{aligned}$$

(b) I believe that hill climbing with random restarts would perform better in this type of problem.

Since it is not possible to see how the "landscape" of our search space is, I believe that we definitely need random restarts to guarantee some exploration of the search space. Since the hill climbing part of the algorithm is going to take care of the exploitation.

In order to solve this problem, we will use a tweak function that will change each variable by adding or subtracting a small number (relative to it's bounds). We will do 300 restarts, the stopping condition at every restart will be that the tweaks did not give a better solution for the last consecutive 5000 iterations

(c) Only code

(d) Particle swarm optimization is a meta heuristic that tracks N solutions that move in the search space. A particle next move is influenced by inertia (where it is currently going), the best personal solution it has ever seen, and the best solution any of the N particles has ever seen.

Since this is a continuous problem, all we need to define is the search space, and the constraints. Then we create the particles and use the formula to update each particle's movement. The formula is defined as

$$x(t+1) = x_i(t) + v_i(t+1)$$

Where $x(t)$ denotes the position of the particle at time t, and $v(t)$ is the velocity at time t and $v(t+1)$ is defined as:

$$v(t+1) = \alpha v_i(t) + \beta r_1(b(t) - x_i(t)) + \gamma r_2(B(t) - x_i(t))$$

Where α is the scaling factor for inertia, β is the scaling factor for personal best, γ is the scaling factor for global best, $b(t)$ is the position of the best solution this particle has seen, $B(t)$ is the position of the best solution any particle has seen, and r_1, r_2 are random vectors in $[0, 1]$.

The main difference between those two approaches is that PSO is less likely to get stuck in a bad hill, since it has more exploration, and the particles have no problem going downhill in order to find better hills. While HC requires the start point to be a good start point for it to get a good solution. In very spiky landscapes it is unlikely that the start point is the best hill.

2. (a) Employees' shifts at work. Consider a graph where each vertex corresponds to a worker and each edge to a scheduling issue between two workers, and we want to avoid two people working the same shift. Therefore, in this graph, an independent set would be a group of workers that can be scheduled to work the shift alone without encountering any scheduling problems.

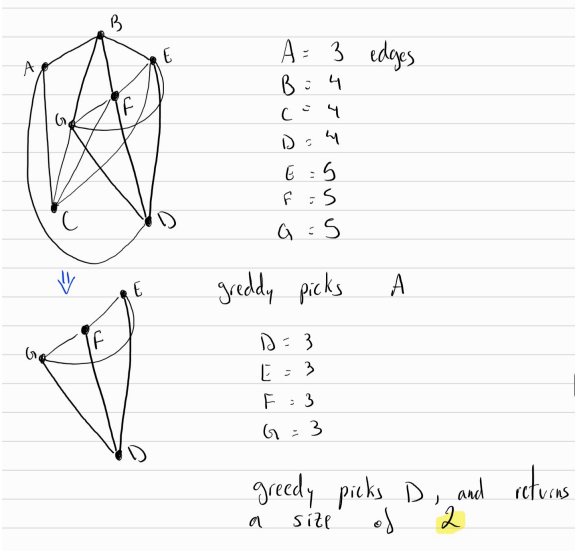
The objective in this scenario would be to schedule as many staff as possible while avoiding people working the same shift. The set of vertices in the maximum independent set corresponds to the maximum number of employees who may be scheduled without any conflicts in this independent set maximization problem. Which could be used to minimize how much an employer needs to pay for all employees.

(b) The number of possible subsets of vertices in a graph with n vertices is 2^n.We must filter out the dependent subsets because not all subsets are independent sets. We must determine whether any vertices in a particular subset of vertices are adjacent to one another, which will take k iterations for k edges. Therefore the search space size is $k * 2^n$

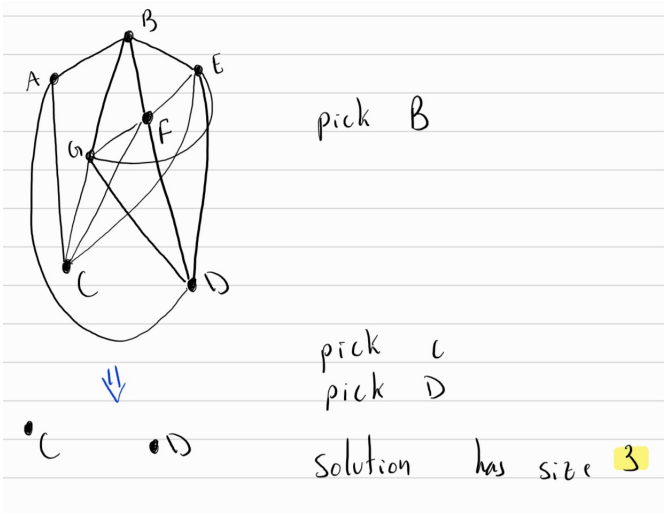
(c)
```
s = empty list
while are are veritices remaining in the graph
    v = the vertex with the smallest # of edges
    s += v
    remove v abd all vertices that have an edge with
       that vertice
return s
```

The greedy algorithm is not optimal. See pictures:

Greedy :



Better solution :



(d) For small inputs, we can get the algorithm to print what the graph looks like after each iteration, then draw the graph by hand and see if it makes sense.

This is help full to check if the logic works and fix bugs.

Now, for large inputs, we could use some library in python that can draw graphs. But if it works on small inputs, and we try to check some edge cases, it should work for larger inputs.

(e)

```
Start with the solution being the empty set
and a list of all vertices

For each vertex v in the graph
   #either it v is in or out of the solution
```

```
    add v to the solution
    remove all vertices that have an edge with v
    solve on what's left of the graph
    calculate the bound b for the partial sol
    is b smaller than the solution?
        prune this branch
     recursevely solve the problem from the remainig
        graph
return solution
```

For the bound function, we can use the total number of vertices left after removing that specific vertex, and all vertices that had an edge with it.

(f) The meta heuristic that I will use is simulated annealing. In order to use SA to solve this problem we will need a tweak function, a way to generate random solutions, a scoring function, decide the initial temperature of the system, and final temperature.

This approach is appropriate, since it is a discrete problem, which SA can handle without problems, the algorithm can handle large search spaces and find good solutions quickly.

For the tweak function, I went with creating a whole new random solution and get two elements from the previous solution, if they are in the random solution, take them out, if they are not, add them. This is not a good tweak function, but it was the one I was able to implement. A Better choice of tweak function is, randomly deciding weather remove or add a vertex to the solution, If the addition of that vertex makes the solution to not be valid, remove all vertices that have edges with the newly added vertex.

In order to generate random solutions I used the same algorithm as the greedy one, except, instead of picking the vertex with the smallest number of edges, just pick a random one.

Scoring function is trivial, it is just the number of vertices in the solution. The initial and final temperature are decided from the algorithm taught in class.

(g) Only code