02-512/02-712 Computational Methods for Biological Modeling and Simulation
Lecturers: Russell Schwartz and Oana Carja
Tuesdays and Thursdays 2:00pm-3:20pm HH B131

Problem Set 1 Solutions

Assigned 9/7/23, Due 9/21/23

Problem 1 (20 points): The goal of this problem is for you to take informal descriptions of biological questions and cast them as well-known optimization problems we have seen in class. For each description, identify a classic computational problem we have studied that would solve the informal biological problem and describe how to convert the informal problem into a formal statement of inputs, outputs, and objective function.

To give an example, here is a similar problem:

> Suppose we are trying to develop a new small molecule drug to prevent a virus from binding to a cellular receptor protein by attaching to some binding pockets on the receptor protein. There are a set of chemical groups on the molecule, each of which has some known binding energy for each binding pocket. We want to know the most stable (minimum energy) way of assigning chemical groups to binding pockets. Assume that each group can be bound to only one pocket and each pocket to only one group.

A reasonable solution to this problem would be:

> We can pose this as a maximum bipartite matching problem. We can create one part of the graph with a node for each chemical group and the other part with a node for each pocket. We will place one edge between any group/pocket pair capable of binding, weighted by minus the energy of binding (minus because this is a maximization problem and we want to minimize energy). The maximum matching will then provide a minimum energy assignment of chemical groups to binding pockets.

The systems to address are the following:

a. Scientists are studying the distribution of various plant species in a forest ecosystem. They collect data on the presence or absence of each plant species in different plots within the forest. Given this data, they want to identify the distinct ecological niches, characterized by distinct preferred distributions of plant species, that exist in the forest.

Solution: This can be framed as a clustering problem, which in turn can be cast as a maximum k-cut problem. Each plant species can be seen as a dimension, and the abundance of each species in a plot can be represented as distance in one dimension to represent each plot as a vector or point in plant space. A maximum cut would group the plants into similar subsets that could correspond to distinct clusters (ecological niches) that best explain the data.

b. Suppose we have characterized patterns of connectivity in a neural network, identifying synapses between neurons by which they may communicate and a frequency with which each a given synapse is used. We then want to interpret experimental data showing how a stimulus applied in one part of the brain triggers a response elsewhere in the brain. We would like to identify the most likely

sequences of neuron-neuron signalling to trigger each response in a distant part of the network to a given stimulus.

Solution: This can be framed as a single source shortest path problem in a weighted graph. Here, neurons are nodes, synapses are edges with weights corresponding to frequencies of use. We can then pose the problem of finding most likely chains of connectivity by finding a weighted shortest path using weights derived from the -log frequencies of each synapse.

c. We are running an automated science laboratory, where we would like as much as possible to have robotic equipment running laboratory experiments according to directions supplied by remote users. A user has requested an experiment that requires a robot to visit various supply stations to collect reagents before returning to a workbench to perform the experiment. We want to determine the most efficient way to pick up all of the necessary items from the supply stations while spending as little time as possible traveling around the facility.

Solution: We can model this as a traveling salesperson problem (TSP). The various stations the robot must visit are nodes in the graph and the distances between them define weights for edges connecting stations. The TSP solution then defines a most efficient path for the robot to collect all of the reagents it needs for its experiments.

d. Suppose we have a set of people who seem to suffering from a shared genetic disease. We suspect there may be a few different mutations that might cause the same disease phenotype but do not know which ones. We sequence likely pathogenic mutations in potential risk genes for each person in a study. We would like to identify the smallest set of mutations we can to explain the disease, under the assumption that everyone with the disease has at least one of the mutations in our set.

Solution: This can be posed as a minimum set cover problem. Here, each patient is a member of our set and the collections correspond to the subsets of patients with each observed mutation. A minimum set cover is a minimal set of mutations such that each patient has at least one of them, as desired for our output here.

e. We have a collection of medical records for a group of people suffering from a common progressive illness, describing clinicians' reports of their medical exams. We suspect that there might be early symptoms of the disease that are found in the medical records but have not been connected to the disease. We decide to look at medical records from a year before the patients were diagnosed and identify the longest phrases in common across the medical reports. Our theory is that these long phrases will describe symptoms we might previously have missed that tell us who is at risk of developing the disease in the next year. You can assume here that we are looking for exact phrases.

Solution: This can be posed as a longest common substring problem. Each medical record is an input string. The longest common substring is the longest phrase in common across all of the reports that we hope will identify some new symptom.

(Note: I used ChatGPT to suggest some problems here. It offered some good ideas, although all of them needed some work to be usable.)

Problem 2 (15 points): In this problem, we will consider some algorithmic options for working with a model of an intractable optimization problem. Suppose we are seeking to optimize operations in an automated lab. We have a bunch of experiments submitted to our lab and we want to run

them all as efficiently as possible. To keep things simple, we will assume that each experiment requires a single unit of time during which it will run on some subset of the equipment in our lab. Some experiments need to use a common piece of equipment, though, and those experiments cannot be run at the same time. We would like to group our experiments into batches of experiments that can be run simultaneously and do so in a way that completes all of the experiments in as few batches as possible. We realize we can accomplish this by representing it as a graph coloring problem.

a. Describe how we can reduce this problem to a graph coloring model.

Solution: Each experiment can be represented as a node in our graph. If any two experiments share a piece of equipment, then there is an edge between the corresponding nodes. A graph coloring will then identify groups of experiments (colors) that can be run in a common batch. The chromatic number of the graph is then the minimum number of batches of experiments we need to get through all of them.

b. Suppose we come up with the following strategy for solving the graph coloring:

```
while any node is uncolored
   pick any uncolored node u uniformly at random
   assign u the smallest positive integer not assigned to any neighbor of u
```

We assert that this will produce a coloring of the graph.

Will this be a good strategy if we have a small number of experiments and a lot of common equipment among them? How about if we have a large number of experiments and small amounts of shared equipment among them? Explain your answers in a sentence or two.

Solution: This will be a reasonably good strategy for large graphs because it is fast (polynomial in graph size). Since this is an NP-hard problem, though, we can be confident that a provably efficient algorithm like this is not provably optimal. It will therefore be a poor choice for small graphs where we can likely get optimal or near optimal solutions in a reasonable time by other methods.

c. Suppose we come up with an alternative strategy: if we have n experiments, then we will enumerate over all sequences of the form $s = \{1, 2, \ldots, n\}^n$. For each such sequence, we will assign experiment i to the batch corresponding to element s_i, the ith element of the sequence s. If s assigns any two incompatible experiments to the same batch, then we will reject it; otherwise we will store it as a possible solution. Once we have run through every sequence s, we will keep the solution that required the fewest distinct batches.

Will this be a good strategy if we have a large number of experiments and must produce some solution even if it is sub-optimal? Will it be a good strategy for small numbers of experiments where we want to be as efficient as possible in our use of the lab's time? Explain your answers in a sentence or two.

Solution: This is a brute-force search, so it will guarantee an optimal solution, but will require exponential time in problem size to do that. It would therefore be a poor choice for large numbers of experiments, but a good choice for a small number experiments when we really want to get as close to optimal as possible.

Problem 3 (20 points): In this problem, we will examine a different variant of sequence assembly problem similar to those we saw in class. We will assume here that rather than having a single genome from which all of our sequence reads are drawn, we are trying to assemble fragments from RNA sequencing, where fragments may come from any of many different genes. We assume we have a set of sequence reads R and two particular pairs of reads s and t that we believe to be the 5' UTR (beginning) and 3' UTR (end) of a particular gene. We assume that for any pair of reads $r_1, r_2 \in R$, we know whether a suffix of r_1 matches a prefix of r_2 with enough overlap that we can confidently conclude they might be consecutive in the same RNA. Given a minimum number of reads k, we would like to ask the question whether there is a chain of overlaps through a sequence of reads $s, r_1, r_2, \ldots, r_{k-1}, t$ connecting s to t. If so, we will interpret that as the sequence of an RNA containing all of those reads.

a. Formally pose this as a decision problem, specifying an input and question. (Hint: Model it as a graph problem.)

Solution: We will model this problem by assuming our reads define a graph G with vertex set $|V| = |R|$ and an edge set E where $(v_i, v_j) \in E$ iff there is a sufficiently large overlap of a suffix of r_i and a prefix of r_j. Then we can model the problem as follows:

Input: directed graph $G = (V, E)$, nodes $s, t \in V$, length $k \in \mathcal{N}$
Question: Does there exist a path $s, v_1, v_2, \ldots, v_{k-1}, t$ such that $(s, v_1) \in E$, $(v_i, v_{i+1}) \in E$ $\forall i \in \{1, k-2\}$, $(v_{k-1}, t) \in E$?

b. We want to try to prove that this problem is NP-complete. To show that, first show that it is in NP. We can show that be demonstrating that if there exists such a chain of overlapping reads answering the question in the affirmative then there is some polynomial size set of evidence we could check in polynomial time to verify that there is a valid solution.

Solution: We could use the list of overlapping fragments as our evidence. It is trivial to verify in linear time that each consecutive pair of fragments corresponds to an edge in our graph, as well as that the path is a valid length, which will check that the provided solution is correct.

c. We will now show the problem hard through a reduction from a problem we already know is hard. A good choice of problem for this reduction is the Hamiltonian path problem. Specify a mapping of problem inputs to inputs between the two problems for this reduction.

Solution: We need to reduce the Hamiltonian path problem to our problem. Recall that the Hamiltonian Path problem has as input just a directed graph $G = (v, E)$. We will need to turn that into an instance of our new problem. We can start by adding a new s and t to G and creating edges from s to every $v \in V$ and from every $v \in V$ to t. More formally, we say $G' = G \cup \{s, t\}$ and $E' = E \bigcup_{v \in V}(s, v) \bigcup_{v \in V}(v, t)$. Then the input to our genome problem will be $G' = (V', E'), s, t, |V| + 2$.

d. Now show how to map the answer to the questions between the Hamiltonian path problem and the genome problem to complete the reduction.

Solution: The answers are the same. A path of length at least $|V| + 2$ exists from s to t in G' if and only if there is a Hamiltonian path in G.

e. Use your answers to parts c and d to argue that this RNA assembly problem in NP-complete.

Solution: We have shown the RNA assembly problem is polynomially reducible from Hamiltonian path, proving that an efficient solution to the RNA problem would imply an efficient solution to the Hamiltonian path problem, showing that the RNA assembly problem is NP hard. We previously showed the RNA assembly problem is polynomially checkable, showing it is in NP. Therefore, it is NP complete.

f. What would be a suitable strategy for this problem if we needed to be able to solve this problem for large read sets (billions of reads) and were willing to tolerate some errors to do that.

Solution: We need a fast method, which means it will be inexact. We might look for an approximation algorithm to the equivalent optimization problem, perhaps potentially not finding the longest sequences. Alternatively, we could try a fast heuristic, such as greedily connecting the longest overlapping fragments from s until we read t or hit a dead end.

Problem 4 (25 points): In this programming problem, we will look at some of the tradeoffs between a couple of different approaches to a hard optimization problem. Let us assume we are interested in a problem motivated by genomics. We have a set R of sequence reads each identifying a short piece of genomic sequence. We also have a set of whole genomes from different strains of a microorganism, G, each of which contain some of the reads. The genomes are related, so there may be several genomes containing the same read. We would like to use the reads to identify which strains are present in a sample. We will model this as the problem of finding the smallest set of genomes $G' \subseteq G$ such that each read $r \in R$ is contained in at least one genome $g \in G'$.

a. The problem as described is equivalent to a common intractable problem we have seen in class. Which problem is it? Provide a mapping from the genomics problem described here to inputs and outputs of the classic problem we saw in class.

Solution: This is essentially the Set Cover problem. Here the reads R correspond to the set S in the set cover and each genome g corresponds to a collection $c \in C$, where $s_i \in c_j$ iff read r_i is found in genome g_j.

b. We will first consider a heuristic approach to the problem. In this heuristic, we will greedily assign genomes by repeatedly picking the genome that contains the most reads still to be accounted for, until we have a set of genomes that contain all of the reads. Provide pseudocode for this greedy algorithm.

Solution: See Algorithm 1 (thanks to ChatGPT):

c. We will next consider a brute force solution to the problem. In this solution, we will try every subset of genomes and test whether that subset of the genomes contains all of the reads. At the end we will return a smallest subset of the genomes containing all of the reads. Provide pseudocode for this brute-force solver.

Solution: See Algorithm 2 (thanks to ChatGPT)

d. Write code implementing both of your methods. Each routine should take as input the following parameters:

n

Algorithm 1 GreedySetCover(R, G)

```
G' ← {}                                          ▷ Initialize the set of selected genomes
while R is not empty do                          ▷ Not all reads are covered
    max_coverage ← 0
    best_genome ← None
    for genome in G do
        covered_reads ← {rforrinRifrisingenome}
        coverage ← lengthofcovered_reads
        if coverage > max_coverage then
            max_coverage ← coverage
            best_genome ← genome
        end if
    end for
    G'.add(best_genome)                          ▷ Select genome with maximum coverage
    R ← R − {readsinbest_genome}                 ▷ Remove covered reads
end while
return G'
```

Algorithm 2 BruteForceSetCover(R, G)

```
min_genomes ← None
min_size ← ∞
for subset in power_set(G) do                    ▷ Generate all possible subsets of genomes
    covered_reads ← {}
    for genome in subset do
        covered_reads ← covered_reads ∪ {rforrinRifrisingenome}
    end for
    if lengthofcovered_reads = lengthofR and lengthofsubset < min_size then
        min_genomes ← subset
        min_size ← lengthofsubset
    end if
end for
return min_genomes
```

$$
\begin{array}{l}
m \\
g_{11}\ g_{12}\ \cdots\ g_{1n} \\
g_{21}\ g_{22}\ \cdots\ g_{2n} \\
\vdots \\
g_{m1}\ g_{m2}\ \cdots\ g_{mn}
\end{array}
$$

where n is the number of reads, m is the number of genomes, and g_{ij} is 1 if genome i contains read j and 0 otherwise.

e. Test your methods by creating a wrapper program that takes as input m, n, and a number of iterations k. For each iteration, it should generate a random problem instance by filling in each element of g_{ij} to be 0 or 1 with 50% probability. It should record the average solution size and the average time to derive a solution for each method over all iterations. Run your program for

$k = 100$ and $m = n = 1$, $m = n = 2$, $m = n = 3$, etc., until one of the methods becomes too slow to be practical. Return a plots of the average solution size and the average time required for each methods as a function of n.

f. We conjecture that the greedy algorithm is a 2-approximation algorithm for the exact problem. Can you prove or disprove the conjecture with your plots? Why or why not?

Solution: The greedy algorithm is in fact a $log(n)$ approximation. It is impossible to prove the conjecture correct, but you might be able to prove it incorrect if you get to a large enough number of iterations to show that the average solution size is worse than 2 in practice.

***Problem 5 (20 points):** In this problem, we will build on the automated science problem we used as a model for problem 2, in this case by considering integer linear programming (ILP) as a solution.

a. We will first start by using the problem as posed in problem 2, as a graph coloring problem. We will want to develop an ILP to solve for the graph coloring model. We will assume our input for n experiments is encoded as an $n \times n$ matrix X where each element x_{ij} is 1 if experiments i and j need to use a common piece of equipment and 0 if they do not. Our ILP will aim to output an $n \times n$ matrix Y where y_{ij} is 1 if experiment i is in batch j and 0 otherwise. In terms of this model, define a set of constraints that says that each experiment is assigned to exactly one batch.

Solution:

$$\sum_{j=1}^{n} y_{ij} = 1 \quad \forall i \in \{1, n\}$$

b. Now design a set of constraints that say no two experiments can be assigned to the same batch if they share a common piece of equipment. (Hint: We'll need $O(n^3)$ constraints, one per batch per pair of experiments.)

Solution: To assert that experiments i and j can only both be in batch k if they do not share a common piece of equipment, we can say:

$$y_{ik} + y_{jk} \leq 2 - x_{ij}$$

This effectively says that the two experiments can only both be in batch k if $x_{ij} = 0$, i.e., if they do not share common equipment. Then we can get the full set of constraints by saying this applies for all experiments and batches:

$$y_{ik} + y_{jk} \leq 2 - x_{ij} \quad \forall i, j, k \in \{1, n\}$$

c. The way we have posed this so far makes it difficult to evaluate the objective function in terms of X and Y. To help that along, we will create a length n vector of auxiliary variables $\vec{z}$, where $z_i = 1$ if some experiment is assigned to batch i and 0 otherwise. Provide the family of constraints needed to ensure no experiment is assigned to batch i unless $z_i = 1$.

Solution: We can constrain this as follows:

$$z_i \geq y_{ij} \quad \forall j \in [1, n]$$

This will ensure z_i must be 1 if any $y_{ij} is 1$.

d. Complete the program by defining the objective function in terms of X, Y, and/or $\vec{z}$. Solution: We want to minimize the number of batches, which has a simple interpretation in terms of the auxiliary variables:

$$\min \sum_{i=1}^{n} z_i$$

e. Let us propose that we want to modify the problem a bit to make it slightly more realistic. We will assume that each experiment i has a run time t_i and we will define the time needed for a batch of experiments to be the time of the slowest experiment in the batch. We then want to organize the experiments into batches so that the total time needed for all batches is minimized. Provide a sketch of a proof for why the problem is still NP-hard.

Solution: If we have a graph coloring problem, we can reduce it to an instance of the new problem by simply defining each node as an experiment and each edge as an overlap in equipment, as before, and further defining $t_i = 1$ for all i. Then the solution to our new problem will solve the graph coloring problem, providing a polynomial reduction from graph coloring to our new problem that proves the new problem is still hard.

f. Modify the ILP you defined above to solve optimally for the modified problem statement of part e.

Solution: One way we could handle this is to add some extra auxiliary variables w_i to be the time needed for batch i. We could enforce this with a constraint set

$$w_j \geq t_i y_{ij} \quad \forall i, j \in \{1, n\}$$

(Remember that x_{ij} is an input so a constraint can be non-linear in x_{ij}'s and t_i's but not in y_{ij}'s or z_i's. Then we can change the objective to be

$$\min \sum_{i=1}^{n} w_i$$

*Recall that the starred problems are only for the 02-712 students.