02-712 Week 2

Biological Modeling and Simulation

Aidan Jan

September 4, 2025

More (Intractible) Graph Problems

Matching Problems

- Input: $G = (V, E)$, $w : E \to \mathbb{R}$
- Output: $E' \subseteq E$ such that $\nexists u, v, w \in V$ such that $(u, v) \in E'$ and $(u, w) \in E'$
- Objective: $\max \sum_{e \in E} w(e)$

Basically, a matching is a set of edges that do not share any vertices. The question is, what is the highest weighted matching on the graph?

- Interestingly, this problem is intractible for general graphs. However, it is solvable in polynomial time for bipartite graphs.

Vertex Cover

- Input: Undirected $G = (V, E)$, $B \in \mathbb{N}$.
- Objective: Does there exist $V' \subseteq V$ such that $\forall (u, v) \in E$, $u \in V'$, $v \in V'$, $|V'| \leq B$?

Basically, a vertex cover is a set of vertices such that every edge on the graph is conneccted to at least one of the vertices in the set. Find the minimum number of vertices required to create a vertex cover.

Independent Set

- Input: $G = (V, E)$, $B \in \mathbb{N}$.
- Objective: Does there exist $V' \subseteq V$ such that $\nexists (u, v) \in E$ where $u \in V'$, $v \in V'$ such that $|V'| \geq B$?

Find the largest set such that no two vertices in the set are connected by an edge.

Max Clique Problem

- Input: $G = (V, E)$, $B \in \mathbb{N}$.
- Objective: Does there exist $V' \subseteq V$ such that $\forall v_1, v_2 \in V'$, $(v_1, v_2) \in E$, and $|V'| \leq B$?

A clique is another name for a complete subgraph. (e.g., a subgraph where every vertex in it is connected to every other vertex in it via an edge.) What is the largest clique in graph G?

Graph Coloring

- Input: $G = (V, E)$, $k \in \mathbb{N}$.
- Objective: Does there exist $e : V \to \{1, \ldots, k\}$ such that $\forall (u, v) \in E$, $c(u) \neq c(v)$?

Each edge is 'colored' with a number less than k. Is it possible to color the entire graph such that no neighboring vertices are colored the same?

Max Subgraph / Max Induced Subgraph with Property Π

- Input: $G = (V, E)$, $B \in \mathbb{N}$.
- Objective (variant 1): Does there exist $G' = (V, E')$, where $E' \subseteq E$ such that G has property Π if $|E'| \geq B$?
- Objective (variant 2): Does there exist $G' = (V', E')$, where $V' \subseteq V$ and $(u, v) \in E'$ and $u \in V' \wedge v \in V'$ if $|V'| \geq B$?

Variant 1: Does the graph have property Π?
Variant 2: Does the graph contain any subgraphs with property Π?
Property Π is a nontrivial, inheritable property of a graph. (Defined separately.) Some examples of possibilities of Π:

- Bipartiteness
- Independence
- Planarity

Nontrivial = are there an infinite number of graphs that do and do not have property Π?
Inheritable = can adding edges or vertices change the result? If not, it is inheritable.

String Problems

Longest Common Subsequence

This is a problem that is solvable (runnable in $O(n^2)$), but not tractible. Usually genomes are way too big for this to be practical at all.

- Given a multiple sequences of letters (or nucleotides):
  - ABCABCABC
  - CABBCCBB
  - ABCABABC
  - etc.
- What is the longest sequence of letters that is in each sequence? The letters don't need to be consecutive. ($O(|S||T|)$)
- This is a variation of the longest common substring, where the characters must be consecutive. ($O(|S| + |T|)$)

Formal definition for longest common subsequence:

- Input: sequences $w_1, w_2, \ldots, w_k$, $B \subseteq \mathbb{N}$
- Objective: Does there exist a sequence s of $w_1, \ldots, w_k$, where $|s| \geq B$?

This problem is intractible in k. This means that if k is known (for example, you KNOW you are aligning 3 genomes), or treated as a constant, it is tractible. If k is treated as a variable, then this problem is intractible.

Shortest Common Superstring

Given a sequence of short strings:

- ACGGA
- GGACT
- ACTCCA
- CAGG

What is the shortest string that contains all of the short strings?

- For this example: ACGGACTCCAGG

This is an intractible problem.

Shortest Common Supersequence

Given a sequence of short strings:

- GCGCA
- CGATA
- ACGAAA

What is the shortest string that contains all of the sequences? (The characters don't have to be consecutive.)

- For this example: GACGCATAA

Set Problems

Minimum Test Set

- Input: set $S = \{s_1, \ldots, s_n\}$, collection $C = \{c_1, \ldots, c_m\}$, where each $c_i \subseteq S$, $B \in \mathbb{N}$
- Objective: Does there exist a $C' \subseteq C$, such that $\forall s_i, s_j \in S \exists c \in C'$ such that $|\{s_i, s_j\} \cap C| = 1$, $|C'| \leq B$?

For example, consider

- $s_1 =$ ATACC
- $s_2 =$ CTGTC
- $s_3 =$ CGGTG
- $s_4 =$ ATGTG
- $s_5 =$ AGGCC
- $c_1 = \{s_1, s_3, s_5\}$
- $c_2 = \{s_1, s_2, s_4\}$
- $c_3 = \{s_2, s_3, s_4, s_5\}$

The question is, what is the minimum of subset of collections you need to be able to narrow down a to a single set that is in all the collections?

- In this example, c_1 is the collection of sets with A as the first letter. c_2 is the set with T as the second letter. c_3 is the set with G as the third letter.
- Pretend we have sets for the forth and fifth letters too.
- Suppose you extract a bacterial genome and it matches one of the sets, but you don't know which. What is the minimum number of positions in the genome you need to look at to determine the set?

Minimum Set Cover Problem

- Input: Set S, Collection C, Bound B.
- Objective: Does there exist $C' \subseteq C$ such that $\forall s \in S \exists c \in C'$ such that $s \in C, |C'| \leq B$?

Practical explanation: suppose you have a list of antibiotics, and each one is able to kill a set of bacteria $c \in C$. You have a pool of bacteria (S) that you want to kill. What is the minimum number of antibiotics needed to kill all the bacteria?

Interspecies Phylogenetics

Suppose you sampled some DNA from some individuals of the same species.

- ACGA
- CCGA
- CTGA
- CTGC
- CTAA

The tree may look something like:

```
CTGG - CTGA - CCGA - ACGA
         |
       CTAA
```

How do we build this tree? One way to do it:

- Make all the sequences nodes on a fully connected graph.
- Weigh all of the edges as the hamming distance (or edit distance) between two nodes.
- Find the MST to build the tree!
  - Input: $G = (V, E)$, $d : V \times V \to \mathbb{N}$
  - Output: Tree $T = (V, E')$, $E' \leq E$
  - Objective: $\min_{(u,v) \in E} \sum d(u, v)$

In this case, one way to model this problem would be to reduce it to a minimum spanning tree problem. Now, what assumptions did we make by using this model?

1. Data is not noisy.
2. Mutations effectively have equal likelihood.
3. We can accept any optimal solution.
4. There is no missing data.

Assumptions tend to lead to inaccuracies, so what can we do about them?

1. We can live with the assumption.
2. Change our preprocessing. (Maybe we can't assume mutations are equal likelihood, so instead of using a basic hamming distance, we may add more weights on it based on which parts of the DNA the section is from.)
3. Multiple optimal solutions. (Does which solution matter? For example, if all we want to know which species are linked, but not which is an ancestor of another, any MST would suffice.)

- Another solution to this problem is to return the union of all the MSTs - a minimum spanning network (MSN). This tells us that an optimal tree can be found in that set of edges.

4. If there is missing data, there may be a better answer. For example, we may be missing a DNA sample, which may actually lower the total MST cost. However, filling these in with software is intractible; it is similar to the minimum steiner tree problem.

NP-Completeness Proofs

Usually, researchers will try to solve a problem given to them in polynomial time, but after some time of trying and not finding a solution, it is common to switch to methods to solve NP-hard problems, and assume the problem is intractible. The first step tends to be to prove that the problem is hard in the first place. We would do this using a **reduction**.

- If we have problem A, and don't know if A is hard, we find a similar problem B that is known to be hard.
- We come up with a polynomial time algorithm to convert some given input for problem B into an input for problem A.
- Additionally, we need a process to convert a solution to problem A to a solution for problem B.
- If we can come up with the reduction from input B to A and output A to B, we can then infer that if a solution for problem A exists (an oracle), then it makes problem B solvable, which is a contradiction. Therefore, problem A is also hard.

For example, consider the max clique and max independent set problems.

- We know independent set is hard.
- For a max clique problem, flip all the edges. As in, if there is an edge between two nodes, remove it, and if there is not an edge between two nodes, add it. This turns it into an independent set problem.
  - (This is a polynomial time reduction, obviously)
- From the independent set problem, copy the solution, and that will also be the solution to the max clique problem.
  - (This is a O(1) reduction, obviously.)
- Therefore, we can infer that if a solution to max clique exists, then independent set also becomes tractible. This is a contradiction. Therefore, max clique must also intractible.

What do we do After Proving Hard?

We have a few options. An optimal, fast algorithm is proven to be impossible.

- Brute force solution. Guess every possibility, and hopefully you will find a good one in a finite amount of time.
- Approximation algorithms. This may not be applicable to all problems, but for some may work
  - For example, the vertex cover problem. We know that finding an optimal solution is intractible. However, we can approximate.
  - First, pick an edge and check if it is a matching. Repeat with random edges until you cannot add more matchings.
  - This isn't a *maximum* matching, but it is a *maximal* matching. In fact, this solution is a 2-approximation, or in other words, is never more than twice the size of the theoretical optimal vertex cover.

Approximation Example: Traveling Salesperson Problem

The TSP problem is a known hard problem. However, if all the nodes satisfy the triangle equality (e.g., the sum of the weights of two edges is always greater than the third edge), the problem becomes tractible. This can be used for approximation purposes.

Branch and Bound Optimization

Branch and bound is a method of brute force that may be more efficient than blind guessing. This is also known as the search tree method.

- Consider the vertex cover problem again. Consider a binary tree, where each layer basically asks whether a given node is present in the vertex cover. Following the tree, use DPS. Once a contradiction is found, there is no more need going down that branch, which can eliminate very large sections of the search tree.
- We use a **heuristic** to help decide when a solution does not work or is suboptimal, and therefore should ascend the tree instead of continue searching down the same branch. These heuristics tend to be approximations.

Transformation

ILP (Integer linear program):

- Input: $\vec{a} = a_1, \ldots, a_n$, $C = [n \times n \text{ matrix}]$, $\vec{b} = b_1, \ldots, b_n$
- Output: $x_1, \ldots, x_n \in [0, 1]$ such that $C_{11}x_1 + C_{12}x_2 + \cdots + C_{n1}x_n \leq b_1$, $C_{21}x_1 + C_{22}x_2 + \cdots + C_{n2}x_n \leq b_2$, etc. . . , basically, $C\vec{x} \leq \vec{b}$
- Objective: $\min a_1x_1 + a_2x_2 + \cdots + a_nx_n$
- Vertex cover: $G = (V, E), v_1, \ldots, v_n, x_1, \ldots, x_n \forall (v_i, v_j) \in E \rightarrow x_i + x_j \geq 1$, $\min \sum_{\forall i} x_i$

Basically, we convert our problem into a matrix optimization problem, and plug it into a solver. The existing solvers can do a pretty good job at approximating a solution.

Satisfiability (SAT)

- Input: $x_i \in [True, False]$
- Boolean Formula (BF): $\phi = (x_1 \vee x_2 \vee x_3) \wedge \ldots$, basically some boolean formula, or series of boolean formulas.
- Question: Does there exist $x_1, \ldots, x_n$ such that $\phi(\vec{x})$ is true?

Heuristics

- Hill climbing: Keep doing moves until you can't do any more moves that improves your solution.
- Simulated Annealing: Keep doing moves until you can't do any more moves, even if it makes the solution worse temporarily.
- Genetic Algorithm: Come up with a solution, and apply mutations to it and hope it gets better.
- Coordinated Descent: Divide and conquer, basically. Divide the big problem into smaller problems, and solve them individually.
- Kitchen Sink: Use every variable, feature, and resource in an attempt to find a solution.
- Give up: If you can't find an acceptable solution, maybe you need a new model!