Machine Learning with Graphs: Graph Algorithms 2/2 - Hardness, Hard problems

Arlei Silva

Spring 2022

Computational Hardness

Most problems we have discussed in the previous lecture have polynomial-time algorithms—they can be solved exactly in time $O(n^{O(1)})$, where n is the size of the input. Unfortunately, that is rarely the case. Long story short, complexity theory provides tools to identify which problems likely don't have a poly-time algorithm. In such situations, designing *an approximation algorithm* or *heuristic* might be a better use for our time.

The first step in understanding what we mean by a hard problem is the fact that we don't know a good way to show that there isn't a poly-time algorithm for a problem, or, more broadly, the complexity of its (asymptotically) fastest algorithm. All we know is that some problems seem to be harder than others. This is formalized using the notion of *complexity class*—i.e. two problems are in the same class if solving them requires a *similar* amount of resources. A complexity class also depends on a *model of computation*, which gives the cost of the fundamental operations required to produce the output.

During this course, we will only discuss two complexity classes, *P* and *NP*. These classes are defined for *decision problems*—i.e. the output is *YES/NO*. Formally, a problem is in P if it can be solved in poly-time using a *deterministic Turing Machine*. Informally, problems in *P* can be solved efficiently. A problem is in *NP* (Non-deterministic poly-time) if it can be verified in poly-time by a deterministic Turing Machine.[1] Notice that, by definition, P is a subset of NP, but we don't know whether P is a *proper subset* of NP.

A problem is said to be *NP-hard* if it is as hard as any problem in *NP*. More formally, any problem in *NP* can be reduced to an *NP-hard* problem in poly-time. Notice that *NP-hard* problems are not necessarily in *NP* and are not constrained to be decision problems. Problems that are both *NP* and *NP-hard* are called *NP-complete*.

Most machine learning problems are not decision problems but instead *optimization problems* (e.g., maximizing the likelihood of a model given the data, minimizing a loss function). An optimization problem can be written as:

[1] Or it can be solved in poly-time by a *non-deterministic Turing Machine*

$$\max f(x) \text{ st. } x \in \mathcal{X}$$

As we will see along this course, *NP-hardness* often separates the ideal solution from what is practical in machine learning.

Another notion of hardness that will be mentioned during this course is defined for *counting problems*. A counting problem asks how many solutions exist given an input. The classes *#P*, *#P-complete*, and *#P-hard*, are counting counterparts for *NP*, *NP-complete*, and *NP-hard*. In particular, a *#P-complete* problem is at least as hard as any *NP-complete* problem. Counting complexity is useful for analyzing the *sampling complexity*—i.e. how many samples are required—of machine learning algorithms.

The First NP-complete Problems

The first problem shown to be *NP-complete* was the *Satisifiability Problem* (SAT). Given a CNF (Conjunctive Normal Form) expression $\phi = C_1 \wedge C_2 \wedge C_m$, where $C_i = (l_{i1} \vee l_{i2} \vee \ldots l_{ik})$ is a clause with $l_{ij} \in \{x_1, \ldots x_n, \overline{x}_1, \ldots \overline{x}_n\}$, the problem consists of whether there is an assignment to boolean variables $x_1, \ldots_x n$, $x_i \in \{0, 1\}$, such that ϕ is satisfied (true). In particular, SAT is hard even if we constrain each clause to have only 3 literals (e.g. $C_i = (x_1 \vee \overline{x}_2 \vee x_3)$), which is known as 3-SAT.

After SAT was shown to be *NP-complete*, several problems followed. The first comprehensive list of *NP-complete* problems–including max-cut, set-cover, and k-clique—was published by Karp in 1972 [2].

Some NP-hard Graph Problems

Many graph problems that we will see along this course are NP-hard. Here, we discuss a few representative examples.

Vertex cover

A *vertex cover* for a graph $G = (V, E)$ is a set of vertices $V' \subset V$ such that for every $(u, v) \in E$, $u \in V'$ or $v \in V'$. In *vertex cover problem*, we ask whether a graph $G = (V, E)$ contains a cover of size k.

We will show that vertex cover is *NP-complete*. First, it is easy to show that vertex cover is in NP, i.e. there exists a poly-time algorithm to verify if a solution is correct. Any candidate cover can be checked in time $O(|E|)$. To show that vertex cover is *NP-hard*, we will use a reduction from 3-SAT (see previous section). For each variable x_i create vertices v_i and $\overline{v}_i$, each pair connected by an edge. Moreover, for each clause C_i, create a triangle in G, with a vertex associated to each literal l_{ij} in C_i. Each node corresponding to l_{ij} is connected to v_i if $l_{ij} = x_i$ of $\overline{v}_i$ if $l_{ij} = \overline{x}_i$. We clam ϕ is satisfiable iff G has a cover with $k = 2n + m$ vertices.

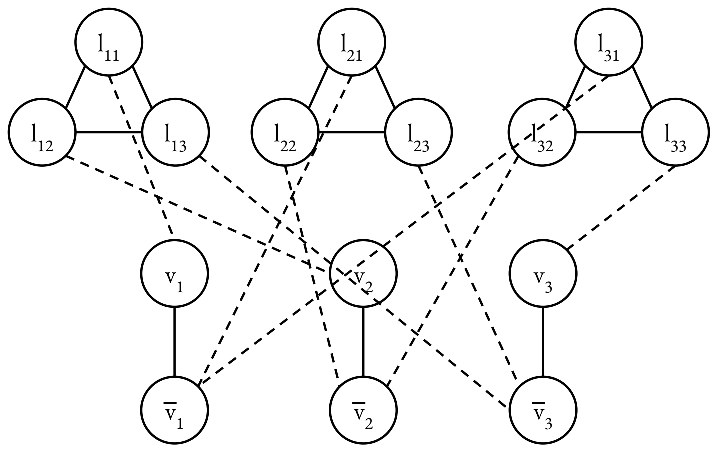

Figure 1: Example of reduction from 3-SAT instance $(x_1 \vee x_2 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3)$ to a vertex cover instance.

Figure 1 shows an example of the reduction for an instance of 3-SAT.

Subgraph isomorphism

Two graphs, $G(V, E)$ and $G'(V', E')$, are *isomorphic* if there is a *bijection* $f : V \to V'$ such that $(u, v) \in E$ iff $(f(u), f(v)) \in E'$. We say that G' is subgraph isomorphic to G if G' is isomorphic to a subgraph of G.

The *subgraph isomorphism problem* asks whether a graph G' is subgraph isomorphic to another graph G (decision). We will show that subgraph isomorphism is NP-complete. First, we have to show that the problem is in *NP*. A solution here is a mapping from $f' : V'' \to V'$, where $V'' \subseteq V$ and we want to check if f' is a bijection. This can be easily done using the definition of bijection for the edges induced by V'' in E.

We want to show that subgraph isomorphism is *NP-hard*. We will use a reduction from *clique problem*, which, given a graph H and a constant k, asks whether H has a *complete subgraph* with k vertices. Clique is known to be *NP-complete* [2]. Let $G = H$ and let G' be a complete graph with k vertices. Then, G' is subgraph isomorphic to G iff H contains a clique of size k.

Sparsest cut

A graph cut $(S, \overline{S})$ divides a weighted graph $G = (V, E, W)$ into partitions $S \subseteq V$ and $\overline{S} = V - S$. Let $|(S, \overline{S})| = |\{(u, v) \in E | u \in S \wedge v \in \overline{S}\}|$ and $|S|$ be the size of the partition S. The *edge expansion* of S is defined as follows:

$$\sigma(S, \overline{S}) = \frac{|(S, \overline{S}))|}{\min\{|S|, |\overline{S}|\}}$$

Given a graph G, the *sparsest cut problem* asks for a cut $(S, \overline{S})$ with minimum value of $\sigma(S, \overline{S})$. Notice that sparsest cut is an optimization problem. We will show that this problem is NP-hard using a reduction from *max-cut*, wich is also NP-hard [2]. Given a graph $H = (V', E')$, max-cut asks for a cut $(T, \overline{T})$ of H with maximum $|(T, \overline{T})|$. We build a graph $G = (V, E)$ such that $V = V' \cup U$ and $|U| = |V'|$. Moreover, $E = \{(u, v) | u, v \in V \wedge u \neq v\} - E'$.

Let $S = T \cup T'$, where $T \subseteq V'$, $T' \subseteq W$, and $|S| \leq |V|/2$. Then, $|(S, \overline{S})| = |S||V| - |S|^2 - |(T, \overline{T})|$. Moreover, the edge expansion of S is as follows:

$$\sigma(S, \overline{S}) = \frac{|S||V| - |S|^2 - |(T, \overline{T})|}{|S|}$$

Now, we claim that $\sigma(S, \overline{S})$ is minimum for $|S| = |V|/2$, which gives:

$$\sigma(S, \overline{S}) = \frac{|V|}{2} - 2\frac{|(T, \overline{T})|}{|V|}$$

Finally, because $|V|$ is constant:

$$\min \sigma(S, \overline{S}) = \frac{|V|}{2} - 2\frac{\max |(T, \overline{T})|}{|V|}$$

References

[1] Volker Kaibel. On the expansion of graphs of 0/1-polytopes. In *The Sharpest Cut: The Impact of Manfred Padberg and His Work*, pages 199–216. SIAM, 2004.

[2] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[3] Ingo Wegener. *Complexity theory: exploring the limits of efficient algorithms*. Springer Science & Business Media, 2005.