1 How do we cope with intractability?

- Heuristics are often used in practice.
- Approximation algorithms.
  - Efficient, but suboptimal.
  - Answer may be off by a multiplicative factor $\alpha > 1$.
- New lens on tractability.
  - Polynomial time vs. non-polynoimal time is a coarse lens.
  - E.g. Hamilton cycle can be solved by brute force in $O(n!)$ time, but a more clever algorithm runs in $O(2^n)$ time. While neither is polynomial-time, the latter is much better.

This course will focus primarily on the last option.

1.1 Fine-Grained Complexity

- c^n time for c as small as possible.
  - Any improvement in c is an exponential improvement in runtime.
  - Subexponential, e.g. $2^{\sqrt{n}}$ can be much better in practice.
- Analyze runtime in finer detail to capture dependence on one or more parameter.
  - Usually one parameter, denoted k.
  - E.g. $T(n,k) = 2^{\text{max-degree(G)}}n^2$ or $T(n,k) = k^k n^3$.
- Hardness results in fine-grained complexity.
  - Reductions must respect the granularity of runtimes.
  - For example, if we assume 3-SAT has no $O(1.2^n)$ time exact algorithm, and reduce a 3-SAT instance to a 3-COL instance with n^3 vertices, then we can only assume that 3-COL has no $O(1.2^{\sqrt[3]{n}})$ time algorithm.

1.2 Why this course? Why now?

- Practical considerations may demand an exact answer. Even if it cannot be done in polynomial time, we want it to be "fast enough."
- Explosion of interest in the last ~ 10 years.
  - Beautiful algorithmic ideas.
  - Emerging algorithmic paradigms. "A theory is building."
  - Ideas that influence complexity more broadly
  - Combines with other topics, e.g. approximation algorithms.

2 Introduction by Example – Vertex Cover

Definition 1.1. A *vertex cover* in a graph $G = (V, E)$ is a set $U \subseteq V$ such that every $e \in E$ has a vertex in U, namely $e \cap U \neq \emptyset$.

Vertex cover (VC) is a classic NP-hard problem, so there is no $\text{poly}(n, k)$-time algorithm unless $\mathbf{P} = \mathbf{NP}$.

A maximal matching is a 2-approximate vertex cover and can be found in $O(n + m)$ time. In fact, this simple algorithm is believed to be essentially optimal. There is likely[1] no $(2-\varepsilon)$-approximation algorithm that runs in $\text{poly}(n, k)$-time [**?**].

But what if we want an exact algorithm? Naively, we can check all $\binom{n}{k}$ subsets of k vertices by brute force. But $\binom{n}{k} \approx n^k$, and we'd like a much faster algorithm.

2.1 Kernelization algorithm

A first algorithm is motivated by the following observation: given a vertex cover S and vertex v, either $v \in S$ or S contains all $u \sim v$. Therefore, any vertex cover of size k must contain all vertices v such that $\deg_G(v) \geq k + 1$. The high-level algorithm is simple: add a high-degree vertex to the vertex cover, delete it, repeat until there are no high-degree vertices, then use brute force.

Algorithm 1 Kernelization-VC(G, k)

```
1:  S ← ∅
2:  H ← G
3:  k' ← k
4:  while Δ(G) > k' do
5:      v ← any vertex s.t. deg_H(v) > k'
6:      S ← S ∪ {v}
7:      H ← H − v
8:      k' ← k' − 1
9:  end while
10: if |S| > k then return NO
11: else if E(H) = ∅ then return YES
12: else
13:     if |E(H)| > (k')^2 then return NO
14:     else
15:         Remove isolated vertices from H.
16:         S' ← Minimum vertex cover of H                ▷ Use brute force
17:         if |S| + |S'| ≤ k then return YES
18:         else return NO
19:         end if
20:     end if
21: end if
```

2.1.1 Analysis

Correctness. When the while loop in lines $4 - 9$ terminates, any vertex cover T such that $|T| \leq k$ contains S. If $|S| > k$, there can be no such T. If $E(H) = \emptyset$ and $|S| \leq k$, then S itself is a vertex

[1] I.e. unless the Unique Games Conjecture is false.

cover of size $\leq k$.

At line 12, H is nonempty and has max-degree at most k'. Each vertex can only cover k' edges, so a vertex cover of size at most k' is only possible if $|E(H)| \leq (k')^2$. In this case, brute force guarantees that we find the minimum vertex cover of H. If $|S| + |S'| <\leq k$, then $S \cup S'$ is a vertex cover of size k. Otherwise, there can be no vertex cover of size at most k.. Indded, since any vertex cover of size at most k must contain S and cover H, its size must be at least $|S| + |S'| > k$. □

Runtime. Each iteration of the while loop can be executed in $O(n + m)$ time, so the enitre while loop takes at most $O(k(n + m))$ time. If $|E(H)| \leq (k')^2$, then

$$|V(H)| \leq 2|E(H)| \leq 2(k')^2 \leq 2k^2.$$

In line 15, there are $\binom{2k^2}{k} \leq k^{O(k)}$ candidate vertex covers of size k to consider. Each candidate can be verified as a vertex cover or not in $O(k^2)$ time.

The total runtime is bounded by $k^{O(k)} + O(k(n + m))$. □

Note. This algorithm used compression (Kernelization) in that it made a poly(n, k)-time reduction

$$\langle G, k\rangle \to \langle H, k'\rangle,$$

where H has size independent of n, $k' < k$ and $\langle G, k\rangle \in VC$ if and only if $\langle H, k'\rangle \in VC$.

2.2 Branching Algorithm

A second algorithm is motivated by the observation that given an edge uv and vertex cover S, either $u \in S$ or $v \in S$. We can branch on these two choices and iterate this process. In doing so, we create a search tree for a minimal vertex cover. The minimal depth of a leaf in this tree is the size of a minimum vertex cover.

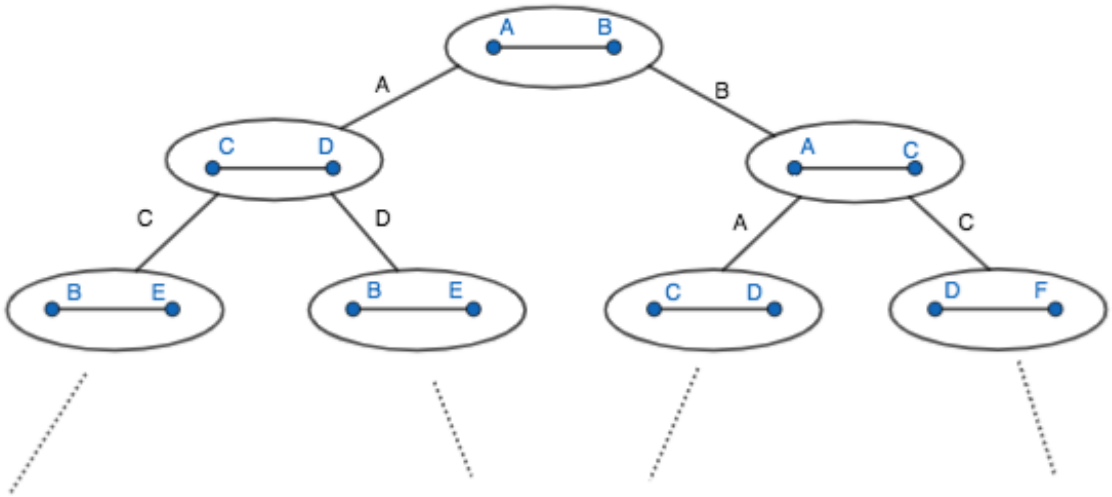

Figure 1.1: Search tree for Branching Algorithm

This can be formalized a recursive algorithm.

Algorithm 2 Branching-VC(G, k)

```
if E(G) = ∅ then return YES
else if k = 0 then return NO
end if
e ← arbitrary uv ∈ E(G)
G_u ← G − u; G_v ← G − v
ans_u ←Branching-VC(G_u, k − 1); ans_v ← Branching-VC(G_v, k − 1)
return (ans_u OR ans_v)
```

2.3 Analysis

Correctness. If G has no vertex cover of size at most k, then after deleting any k vertices, the graph will remain nonempty and the algorithm will always return NO.

If G has a vertex cover S of size at most k, then at each step in the recursion, (at least) one of the two choices will be a vertex in S. After $|S| \leq k$ steps, some path in the recursion will have chosen every vertex in S, and therefore will return YES.

Runtime. Branching-VC(G, k) is called at most 2^k times, each call takes $O(m+n)$ time, for a total of $O(2^k(m+n))$.

3 Complexity Classes

3.1 Remarks on Complexity

- Note that in both algorithms, for a fixed k, the runtime is linear.
- Combinatorial explosion is restricted to the parameter k.
- In general, we consider two possibilities.
  - Runtime $f(k) \cdot n^{g(k)}$ is *slice-wise polynomial* (**XP**).
  - Runtime $f(k) \cdot n^c$ for some fixed c independent of n, k is *fixed-parameter tractable* (**FPT**). Such an algorithm is called a *fixed parameter algorithm.*
  - Note that $\mathbf{P} \subseteq \mathbf{FPT} \subseteq \mathbf{XP}$.

To finish, we finally give a few formal definitions.

Definition 1.2. A *parametrized problem* is $L \subseteq \Sigma^* \times \mathbb{N}$. An *instance* is $(x, k) \in \Sigma^* \times \mathbb{N}$, and k is a *parameter.* $L \in \mathbf{FPT}$ if there is are

1. An algorithm $\mathcal{A}$,
2. A computable (non-decreasing) function $f : \mathbb{N} \to \mathbb{N}$,
3. A constant $c \in \mathbb{R}$,

such that given $(x, k) \in \Sigma^* \times \mathbb{N}$, $\mathcal{A}$ runs in time $f(k) \cdot |(x, k)|^c$ and correctly tells whether $(x, k) \in L$.

Lastly, a few examples.

1. $VC \in \mathbf{FPT}$.

2. Let $\textsc{Coloring}(G, k)$ be the problem of determining whether the input graph G is k-colorable. The natural parameter here is k, the number of colors. The fact that $3\textsc{Coloring} \in \mathbf{NP}$ implies that $\textsc{Coloring} \notin \mathbf{FPT}$ with parameter k (unless $\mathbf{P}{=}\mathbf{NP}$). Indeed, if it were, we would have an algorithm with runtime $f(k)n^c$, and plugging in $k = 3$ would give us a poly-time algorithm for $3\textsc{Coloring}$. Indeed, this shows that $3\textsc{Coloring}$ is not in $\mathbf{XP}$, either.

3. Let $\textsc{Clique}(G, k)$ be the problem of determining whether the input graph G has a k-clique, i.e., a set of k vertices which are all connected to each other. Again the parameter is k, the size of the clique. Then $\textsc{Clique} \in \mathbf{XP}$, since we can easily check if any set of k vertices induces a clique in $\binom{n}{k} n^2$ time. But, as we will see later in the course, it is likely that $\textsc{Clique} \notin \mathbf{FPT}$, unless the Exponential Time Hypothesis is false.