CS 0368-4246: Combinatorial Methods in Algorithms (Spring 2025) May 12, 2025

Lecture 7: Inclusion–Exclusion Principle in Algorithm

Instructor: *Or Zamir* Scribes: *Noam Licht*

1 Introduction

Today's lecture is about the applications of the inclusion-exclusion principle in algorithm. In particular, we will see some $\mathcal{NP}$-complete problems, and show how to improve the best known algorithms using this basic yet powerful principle.

2 Finding Hamiltonian Path

Hamiltonian path is a path that visits each vertex of the graph exactly once.

It is known to be $\mathcal{NP}$-complete.

Algorithm 1. *Naive solution: go over every order of the vertexes and check if it is a valid path - takes* $O^*(n!)$ *time*[1].

But we already solved a similar problem when we dealt with color-coding - finding a path with visit every color exactly once. We solved in $O^*(2^k)$ time, so in ours case where $k = n$ we get a solution in $O(2^n)$ time. For completeness we will describe the solution again:

Algorithm 2.

We will solve with dynamic programming. For each $v \in V, V' \subseteq V$, let

$dp(v, V') =$ Is there a path which visits every vertex of V' exactly once (and only them), and ends in v

then

$$dp(v, V') = \bigwedge_{\substack{u \in V' \\ (u,v) \in E}} [dp(u, V' \backslash \{v\})]$$

Thus we can compute the whole table in $O^*(2^n)$ time.

Issue 3. *We used* $O^*(2^n)$ *space. We would like to use less space.*

Can we solve this in $O^*(2^n)$ but with polynomial space?

Reminder 4. *Inclusion-Exclusion Principle*

[1] The symbol $O^*(T)$ is $O(T \cdot n^c)$ for some c, meaning up to polynomial factor

Let $A_1, A_2 \dots A_k \subseteq B$. Then the principle states that:

$$\underline{\text{Union:}} \quad \left|\bigcup_{i=1}^{k} A_i\right| = \sum_{\emptyset \neq I \subseteq [k]} (-1)^{|I|+1} \left|\bigcap_{i \in I} A_i\right|$$

$$\underline{\text{Intersection:}} \quad \left|\bigcap_{i=1}^{k} A_i\right| = \sum_{I \subseteq [k]} (-1)^{|I|} \left|\bigcap_{i \in I} A_i^c\right| = \sum_{I \subseteq [k]} (-1)^{|I|} \left|B \backslash \bigcup_{i \in I} A_i\right|$$

Proof. (of the union case, the intersection case is similar)

Let $x \in B$ and a set $I \subseteq [k]$. When does the coefficient of x will be:

- 0? $x \notin \bigcap_{i \in I} A_i$, meaning there exists $i \in I$ such that $x \notin A_i$
- $+1$? $x \in \bigcap_{i \in I} A_i$ and $|I|$ is odd
- -1? $x \in \bigcap_{i \in I} A_i$ and $|I|$ is even

Note as J the set of indices of groups A_i which contains x. Meaning we get 0 coefficient if $I \not\subseteq J$, and otherwise $-1^{|I|+1}$

Lemma 5. *Let J be a non-empty set. Then the number of subsets of J of even size is equal to the number of subsets of odd size.*

Proof. Take some $i \in J$. We create a pairing of 2^J in which every subset $I \subseteq J \backslash \{i\}$ is paired to the subset $I \cup \{i\}$. It is a valid pairing and each pair contains one even and one odd subset. □

We will finish the proof. In the case $x \in \bigcup A_i \to J \neq \emptyset$, we got that the sum over all I is:

$$\sum_{\emptyset \neq I \subseteq J} -1^{|I|+1} = 0 - (-1^{|\emptyset|+1}) = 1$$

In the case $x \notin \bigcup A_i \to J = \emptyset$, we got that the sum over all I is:

$$\sum_{\emptyset \neq I \subseteq J} -1^{|I|+1} = 0$$

□

Reminder 6. *We know how to compute the number of paths (not necessarily simple) of length k between every 2 vertex in polynomial time*

Observation 7. *The number of Hamiltonian paths is equal to the number of length n paths that visit every vertex*

Note as B the set of all length n paths in the graph. Note as A_v the set of length n paths which visits v. From the observation it is sufficient to know wether $\left|\bigcap_{v \in V} A_v\right| > 0$. From the inclusion-exclusion principle:

$$\left|\bigcap_{v\in V} A_v\right| = \sum_{V'\subseteq V} (-1)^{|V'|} \cdot \left|\bigcap_{v\in V'} A_v^c\right|$$

Let us look at $\bigcap_{v\in V'} A_v^c$ - it is the set of length n paths which does not visit any vertex of V'. In other words, it is the set of length n paths in the graph $G[V\backslash V']$. Thus, the size of this set can be computed in polynomial time.

To sum up, in order to compute the number of Hamiltonian paths in our graph, it is sufficient to compute a sum with 2^n elements - each can be computed in polynomial time (and space).

Conclusion 8. *The number of Hamiltonian paths in a graph can be computed in $O^*(2^n)$ time and polynomial space.*

3 Set partitioning via Inclusion–Exclusion (Björklund, Husfeldt, Koivisto [1])

Problem 9. *Set Partition/Cover Problems*

Given a set V (of size n) and a family $\mathcal{F}$ of subsets of V, there are a few interesting questions:

- Can we partition V into k subsets from $\mathcal{F}$?
- Can we cover V with k subsets from $\mathcal{F}$?
- How many [covers/partitions] there are of V with k subsets from $\mathcal{F}$?
- Weighted Case (every subset from $\mathcal{F}$ has a weight): What is the [sum of all/max value of] the [covers/partitions] of V with k subsets from $\mathcal{F}$?

Observation 10. *Graph G is k-colorable $\iff$ V can be covered by k sets from the family independent set of G*

How fast can we check if G is k-colorable?

Algorithm 11. *Naive: $O^*(k^n)$*

Exercise 12. *Solve in $O^*(3^n)$ time (using the observation and dynamic programming)*

Our goal is to solve all the set-partition problems in $O^*(2^n)$ time (for general $\mathcal{F}$ it is the best we can hope for as it is the size of the input).

We might think of $\mathcal{F}$ as a function $f : 2^V \to \{0, 1\}$

Definition 13. *Zeta Transform* [2]

Given a function $f : 2^V \to \{0, 1\}$ define $\hat{f} : 2^V \to \{0, 1\}$ as follow:

$$\hat{f}(U) = \sum_{U'\subseteq U} f(U')$$

How to compute it?

Algorithm 14. *Naive: Just compute it by definition*

To compute the whole function it will take

$$\sum_{U\subseteq V}\sum_{U'\subseteq U}1=\sum_{i=0}^{n}\binom{n}{i}2^i=\sum_{i=0}^{n}\binom{n}{i}2^i1^{n-i}=(2+1)^n=3^n$$

The first equality follows from grouping together every $|U|=i$. The third equality is the binomial theorem.

Algorithm 15. *$\hat{f}$ can be computed in $O^*(2^n)$ time* (Yates [3])

We will maintain a series of functions $f=f_0,f_1\dots f_{n-1},f_n=\hat{f}$. For every $i\in[n],U\subseteq V\backslash\{v_i\}$: [2]

$$f_i(U)=f_{i-1}(U)$$
$$f_i(U\cup\{v_i\})=f_{i-1}(U\cup\{v_i\})+f_{i-1}(U)$$

Claim 16. (by induction over i) $f_i(U)$ sums f over all the subsets of U whose $\{v_{i+1}\dots v_n\}$ part is the same as in U

Meaning that indeed $f_n=\hat{f}$. Overall the algorithm took $O(n2^n)=O^*(2^n)$ time (and space).

Theorem 17. *We can compute the number of k-covers of V in $O^*(2^n)$ time*

Proof. Note as B the set of all k-tuples of sets from $\mathcal{F}$.

Cover of size $k\iff$ such k-tuple in which every element of V is in at least one of the subsets.

Note as $A_v\subseteq B$, all the k-tuples in which v appear at least in one of the sets.

Thus the problem is to count $\left|\bigcap_{v\in V}A_v\right|$. Again from the inclusion-exclusion principle:

$$\left|\bigcap_{v\in V}A_v\right|=\sum_{V'\subseteq V}(-1)^{|V'|}\cdot\left|\bigcap_{v\in V'}A_v^c\right|$$

But what is $\bigcap_{v\in V'}A_v^c$? It is the set of all k-tuples which does not contain any element from V'. In other words, every subset in the k-tuple does not contain any element from V'.

We claim that $\left|\bigcap_{v\in V'}A_v^c\right|=\hat{f}(V\backslash V')^k$. Because every element in the k-tuple can be every element of $\mathcal{F}$ which is also a subset of $V\backslash V'$, which is exactly denoted as $\hat{f}(V\backslash V')$.

Thus we finished our proof, as we can compute $\hat{f}$ in $O^*(2^n)$ time, and then compute the final sum in $O^*(2^n)$ time as well. □

We only solved covers, not partitioning. Define $f_i(U)=f(U)\cdot 1_{|U|=i}$. And notice that $\hat{f}_i(U)$ note the number of subset of U of size exactly i.

Observation 18. *A cover is also a partition $\iff$ The sum of the subsets sizes is n*

Conclusion 19. *Define B and A_v as such k-tuple like before, but with sum of sizes equal to n. Then the number of partitions is:*

[2] v_i notes the i'th element in V

$$\left|\bigcap_{v\in V} A_v\right| = \sum_{V'\subseteq V} (-1)^{|V'|}\cdot\left|\bigcap_{v\in V'} A_v^c\right|$$
$$= \sum_{V'\subseteq V} (-1)^{|V'|} \sum_{i_1+i_2+\ldots i_k=n} \prod_{j=1}^{k} \hat{f}_{i_j}(V\backslash V')$$

Because now $\bigcap_{v\in V'} A_v^c$ means we have to choose a k-tuple with sum of sizes n.

Thus it is sufficient to compute $\sum_{i_1+i_2+\ldots i_k=n} \prod_{j=1}^{k} \hat{f}_{i_j}(V\backslash V')$ for every V'. We will compute this value using dynamic programming. Define the solution with parameters n', k' as $dp(n', k')$. Thus

$$dp(n,k) = \sum_{i_1+i_2+\ldots i_k=n} \prod_{j=1}^{k} \hat{f}_{i_j}(V\backslash V')$$
$$= \sum_{i_k=0}^{n} \hat{f}_{i_k}(V\backslash V') \cdot \left[\sum_{i_1+i_2+\ldots i_{k-1}=n-i_k} \prod_{j=1}^{k-1} \hat{f}_{i_j}(V\backslash V')\right]$$
$$= \sum_{i_k=0}^{n} \hat{f}_{i_k}(V\backslash V') \cdot dp(n-i_k, k-1)$$

Thus given $\hat{f}$ we can compute the entire dp table in $O(n^2k)$ time, and $dp(n,k)$ is the value we needed.

Summing up we can compute the number of partition of size k in $O^*(2^n)$ time.

Exercise 20.

- Show how to count the number of [covers/partitions] if instead of $\mathcal{F}$ there are k families of sets $\mathcal{F}_1, \mathcal{F}_2 \ldots \mathcal{F}_k$, and a [cover/partition] takes one set from every family.
- Show how to sum the weight of all [covers/partitions]. Meaning instead of f, we have w which is a general function (not only to $\{0,1\}$), and the weight of a [cover/partition] is $\prod_{i=0}^{k} w(v_i)$.

Note 21. *All the algorithm we presented works with really big numbers ($\approx 2^{nk}$). But it is fine because the length of there representation is $O(nk)$ and arithmetic operation are polynomial in the length of the representation - yielding $O^*(1)$*

Claim 22. Reduction: Algorithm which computes the sum of weights of [covers/partitions] $\rightarrow$ Algorithm which computes the maximum weight of a [cover/partition] (of a somewhat limited function w)

Proof. Assume that w takes only integer values. Define M as the maximum absolute value w takes over 2^V. We will take a very large β, and define a new function $w'(u) = \beta^{w(u)}$. We run the sum-algorithm on w', and we gets

$$\sum_{\substack{V_1,\ldots V_k \\ \text{covers/partitions}}} \prod_{i=1}^{k} w'(V_i) = \sum_{\substack{V_1,\ldots V_k \\ \text{covers/partitions}}} \beta^{\sum_{i=1}^{k} w(V_i)}$$

We take $\beta > (2^n)^k$ (which is the maximum number of [covers/partitions]). Meaning this number in base β tells us exactly how many [covers/partitions] there are in any weight - so we could easily find the maximum [cover/partition] weight.

Note that in w' the maximum weight is $\beta^M = 2^{nkM}$, which means that this reduction works only for polynomial M (in order to keep the representation length of the numbers polynomial). □

4 Summery

We solved all variants of the set partition/cover problem, which generalizes many known problems. A partial list of such problems we now can solve:

- Computing χ
- Computing the number of k-coloring
- Checking if a graph is colorable if every node have a list of possible colors
- max k-CUT: partition of V into k groups with maximal sum of edges weights between the groups

What about memory? The computation of $\hat{f}$ took $O(2^n)$ memory
Open: Is it possible to compute χ in $O^*(2^n)$ time and polynomial memory?

In general, there are problems in which we can efficiently compute $\hat{f}$ without computing all the values beforehand - which yields the desired solution.

References

[1] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion–exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. Paper id:: DOI: 10.1137/070683933.

[2] Gian-Carlo Rota. On the foundations of combinatorial theory. i. theory of möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2:340–368, 1964.

[3] F. Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, Harpenden, 1937. Technical Communication No. 35.