Lecture 3 (26.10.2012) Author: Michal Kawiak

Dynamic programming, Inclusion-Exclusion Principle and FSC (Fast subseteq Convolution)

During this lecture we will solve two problems defined below using three methods (which gives us 6 solutions in total).

1 Dynamic Programming

1.1 Set Cover

SET COVER
Input: A set F of subsets of universum U
Question: Find minimal subset $G \subseteq F$ such as $\bigcup_{f \in G} f = U$

Theorem 1. SET COVER *can be solved in* $\mathcal{O}^\star(2^{|U|})$ *time.*

We'll build easy dynamic algorithm working in above-mentioned time. Assuming that $F = \{S_1, S_2, \ldots, S_m\}$ let's create a dynamic array that meets the following constraint:

$$T[i][X] \text{ is equal to minimal number of sets out of } \{S_1, S_2, \ldots, S_i\} \text{ that covers } X$$

for every $X \subseteq U$ and $i \in \{1, 2, \ldots, m\}$.

If we are able to keep this constraint we will find the solution clearly in $T[m][U]$, since this field contains the minimal number of sets out of $\{S_1, S_2, \ldots, \S_m\}$ that covers the whole universum U.

To fill the array we can use the following recursion:

$$T[0][\emptyset] = 0$$
$$T[0][X \neq \emptyset] = \infty$$
$$T[i][X] = \min(1 + T[i-1][X \backslash S_i], T[i-1][X])$$

The reasoning is quite straighforward: in the latest equation we can either use or not use set S_i. In case we decide to use it, we still have to cover set $X \backslash S_i$ using $S_1, S_2, \ldots, S_{i-1}$. But we have already used S_i, so we have to add 1 to the overall result.
On the other hand we may decide not to use S_i. In this case we have to cover the entire set X with remaining sets. Clearly we take minimum out these two values.

1.2 Chromatic Number

CHROMATIC NUMBER
Input: A grph $G = (V, E)$
Question: Find minimal k such as G can be colored using at most k colors.

Theorem 2. CHROMATIC NUMBER *can be solved in* $\mathcal{O}^\star(3^{|V|})$ *time and* $\mathcal{O}^\star(2^{|V|})$ *memory.*

To create algorithm solving this part let's note first that CHROMATIC NUMBER is special case of SET COVER problem. We want to highlight sets of independent vertices in graph G by coloring them to different colors. We can simply assume that family F of sets contains all independent sets.

However this approach leads us to algorithm working in time $\mathcal{O}^\star(4^n)$, since F can be expotentially big (e.g. when none vertices are connected). To obtain better complexity ($\mathcal{O}^\star(3^n)$) we have to do something better. In this solution we will also use dynamic algorithm.

Let's create an array that meets the following constraint:

$$T[X] \text{ is equal to CHROMATIC NUMBER of } G[X]$$

for all $X \subseteq V$. Obviously the solution would be stored in $T[V]$.

We can fill array T using following recursion rules:

$$\begin{aligned} T[\emptyset] &= 0 \\ T[X \neq \emptyset] &= \min_{\substack{Y \subseteq X \\ Y-independent}} (1 + T[X \backslash Y]) \end{aligned}$$

The justification is as simple as in previous problem: We can set color for an independent set Y, and paint the remaining $X \backslash Y$ part using other colors. Note that that will generate proper coloring (since Y is independent).

If we would count the complexity naively, we would get the result $\mathcal{O}^\star(4^n)$: We have (at most) 2^n independent sets, we iterate through all independent subsets, which gives us $\mathcal{O}^\star(4^n)$ operations. However, if we take a deeper look into this algorithm we'll note that the maximum number of family of independent sets of graph with i vertices is 2^i. That observation leads us to equation

$$\sum_{i=1}^{n} \binom{n}{i} \cdot 2^i = 3^n \tag{1}$$

since for a set of size i we will use at most 2^i time. The factor $\binom{n}{i}$ represents the number of subsets of size i taken from set of size n.

We can limit memory usage only to array T, which gives us $\mathcal{O}^\star(2^n)$ memory complexity.

Remark 3. (Moon-Moser) In any graph there is at most $3^{n/3}$ maximal independent sets (due to inclusion).

Let $v \in V(G)$ be the vertex of minimal degree. Let $d = deg(v)$. We know that either v or one of its neighbours belongs to maximal independent set. (Otherwise we could point bigger independent set that contains either v or one of its neighbours). We know also that max. independent set can contain at most one vertex out of v and its neighbours. This leads us to branching algorithm working in time $T(n)$ that we can estimate with

$$T(n) \leq (d+1)T(n-d-1)$$

It turns out that $T(n)$ is the biggest when $d = 2$, which gives us estimation $T(n) \leq 3^{n/3}$.

Note that in graph composed of independent triangles we hit the estimation, so it cannot be any better.

Corollary 4. CHROMATIC NUMBER *can be found in time* $\mathcal{O}^\star((1 + \sqrt[3]{3})^n) \approx \mathcal{O}^\star(2.44^n)$

We can use the same algorithm, but during the recursion we can interate only through maximal independent sets. That would lead us to estimation

$$\sum_{i=1}^{n} \binom{n}{i} \sqrt[3]{3}^i = (1 + \sqrt[3]{3})^n \approx 2.44^n$$

2 Inclusion-Exclusion Principle

The Inclusion-Exclusion Principle tells us how to count $\bigcup A_i$ using only intersections of some of the sets. Namely it gives us the following formula:

Theorem 5. $\bigcup\limits_{i=1\ldots m} A_i = \sum\limits_{\substack{I \subseteq \{1\ldots m\} \\ I \neq \emptyset}} (-1)^{|I|+1} \bigcap\limits_{i \in I} A_i$

To prove this theorem let's take $x \in \bigcup\limits_{i=1\ldots m} A_i$. We want to check how many times we count it on the right side of the equation.

Let's name I_0 the set of all indexes i such that $x \in A_i$. Note that $I_0 \neq \emptyset$.
Note that we can count x on the right side iff I_0 is superset of I. So we count x exactly

$$\sum_{i=1}^{|I_0|} \binom{|I_0|}{i} (-1)^i = 1 + \left(\sum_{i=0}^{|I+0|} \binom{|I_0|}{i} (-1)^i \right) = 1 + (1-1)^{|I_0|} = 1 \tag{2}$$

which proves the theorem.

The main idea of this setion is to show, that we can use Inclusion-Exclusion Principle if it is easier for us to count intersection than sum of sets.

2.1 Set Cover

Let's translate SET COVER problem into language that we can use in Include-Exclude Principle. Now we are concidering decisive variant of the problem: is it possible to cover U using exactly k sets from F?

Let's take:

- Ω - k-tuples of sets of F
- A_x - k-tuples containint x in one of its sets. (Defined for all $x \in U$)

Our goal is to count $\bigcap\limits_{x\in U} A_x$. Note that:

$$|\bigcap_{x\in U} A_x| = |\Omega| - |\bigcup_{x\in U} \bar{A_x}| = |\Omega| - \sum_{\substack{I\subseteq U\\ I\neq\emptyset}} (-1)^{|I|+1} |\bigcap_{x\in I} \bar{A_x}|$$

where $\bar{A_x} = \Omega \backslash A_x$.

Remember that in our case it is difficult to count $|\bigcap_{x\in I} A_x|$, but we can easily count $|\bigcap_{x\in I} \bar{A_x}|$! We can say that $|\bigcap_{x\in I} \bar{A_x}|$ is the number of k-tuples that does not contain any element of I. So if we call $F_I \subseteq F$ family of sets disjoint with I, we can count that

$$|\bigcup_{x\in I} \bar{A_x}| = |F_I|^k$$

Obviously, we can eaisly count size of Ω too.

Corollary 6. *We can solve problem* Set Cover *in time* $\mathcal{O}^\star(2^n)$ *and polynomial memory.*

2.2 Chromatic Number

As before we will translate Chromatic Number problem to Set Cover.

Let's call:

- Ω -k-tuples of independent sets in graph G
- A_v - k-tuples containing v in one of its sets. (Defined for all $v \in V$).

Our goal this time is also to count $\bigcap\limits_{v\in V} A_v$ (or rather if it's size is positive).

As before we use Inclusion-Exclusion Principle to write:

$$|\bigcap_{v\in V} A_v| = |\Omega| - |\bigcup_{I\subseteq V} A'_I| = |\Omega| - \sum_{I\subseteq V} (-1)^{|I|+1} |\bigcap_{I\subseteq V} A'_I| \tag{3}$$

And with analogy to Set Cover we can observe that the interesting values $|\bigcap\limits_{I\subseteq V} A_I|$ are equal to the number of independent sets in graph $G[I]$.

Theorem 7. *Number of independent sets in graph* H *can be found in time* $\mathcal{O}^\star(1.246^n)$ *and polynomial memory.*

No proof. [stated in [3]]

Corollary 8. Chromatic Number *can be found in time* $\mathcal{O}^\star(2.246^n)$ *and polynomial memory.*

Open Question: Is it possible to do better than $\mathcal{O}^\star(2.246^n)$ with preserving polynomial memory complexity? Target is $\mathcal{O}^\star(2^n)$.

Remark 9. Chromatic Number problem can be solved in time and memory $\mathcal{O}^\star(2^n)$

We can create dynamic array T that follows such a constraint: $T[X]$ is a number of independent sets in $G[X]$ (for all $X \subseteq V$).
To fill the array one can use recursion:

$$T[X] = T[X \backslash \{v\}] + T[X \backslash N[v]]$$

where $v \in X$ is an arbitrary vertex.

After aplying this result to (3) we obtain above mentioned complexity.

Question: Can we also get colouring?
Answer: Most of algorithms that we consider is self-reductible. Which means in this case that we are able to find coloring by executing polynomially many times the above-described algorithm.

It is also worth to take a look to [2].

3 Fast Subset Convolution

For functions $f, g : 2^U \to \mathbb{Z}$ and $X \subseteq U$ we define *convolution* in the following way:

$$(f *_c g)(X) = \sum_{\substack{A,B \subseteq X \\ A \cup B = X}} f(A) \cdot g(B) \tag{4}$$

and *disjoint convolution*:

$$(f * g)(X) = \sum_{\substack{A,B \subseteq X \\ A \cup B = X \\ A \cap B = \emptyset}} f(A) \cdot g(B) \tag{5}$$

We will usually call both functions just *convolution* and exact meaning will follow from sign used: either $*_c$ or $*$.

Our goal is to count all values of $*_c$ and $*$ efficiently (meaning in $\mathcal{O}^\star(2^{|U|})$ time) for all $X \subseteq U$. We take also an assumption that counting values of functions f and g for any argument is *fast*, which means it takes $\mathcal{O}(1)$ time and memory.

Before we are able to write an algorithm counting $*_c$ and $*$ we need to define some more functions:

For function $f : 2^U \to \mathbb{Z}$ and for $X \subseteq U$ we call ζ*-transform* the following value:

$$(\zeta f)(X) = \sum_{Y \subseteq X} f(Y) \tag{6}$$

and μ*-transform*:

$$(\mu f)(X) = \sum_{Y \subseteq X} (-1)^{|X \backslash Y|} f(Y) \tag{7}$$

Two above mentioned transforms will help us in counting convolutions. To do this efficiently we have to know how to count ζ-tramsform and μ-transform fast. For now we will skip this problem taking temporary assumption that we can count all values of ζ-transform and μ-transform in time $\mathcal{O}^\star(2^{|U|})$. The question is: how to use these transforms in order to count convolutions?

We need two lemmas to do so (first we focus only on $*_c$):

Lemma 10.

$$\zeta(f *_c g) = (\zeta f) \cdot (\zeta g) \tag{8}$$

Lemma 11.

$$\mu\zeta f = f \tag{9}$$

Having these two lemmas proven we could count $*_c$ as follows:

$$f *_c g = \mu\Big((\zeta f) \cdot (\zeta g)\Big) \tag{10}$$

which will work in the same time complexity as worse of ζ or μ transforms.

To prove equation (8) note that:

$$\begin{aligned}\zeta(F *_c g)(X) = \sum_{Y\subseteq X}(f *_c g)(Y) = \sum_{Y\subseteq X}\sum_{\substack{A,B\subseteq Y\\ A\cup B=Y}} f(A)\cdot g(B) = \sum_{A,B\subseteq X} f(A)\cdot g(B) = \\ = \Big(\sum_{Y\subset X} f(Y)\Big)\Big(\sum_{Y\subseteq X} g(Y)\Big) = (\zeta f)\,(\zeta g)\,(X)\end{aligned}$$

which proves lemma 10.

To prove lemma 11 observe that:

$$\begin{aligned}\mu(\zeta f)(X) &= \sum_{Y\subseteq X}(-1)^{|X\setminus Y|}(\zeta f)(X) = \\ &= \sum_{Y\subseteq X}(-1)^{|X\setminus Y|}\sum_{Z\subseteq Y} f(Z) = \\ &= \sum_{Z\subseteq X} f(Z)\cdot\sum_{Z\subseteq Y\subseteq X}(-1)^{|X\setminus Y|} = \\ &= f(X) + \sum_{Z\subsetneq X} f(Z)\Big(\sum_{Y_1\subseteq X\setminus Z}(-1)^{|Y_1|}\Big) = \dots\end{aligned}$$

But since in the last equation we sum over *strict* subsets of X we know that $\Big(\sum_{Y_1\subseteq X\setminus Z}(-1)^{|Y_1|}\Big) = 0$, so

$$\dots = f(X)$$

which is what we wanted to prove.

Now let's come back to convolution $*$. We are now ready to give an algorithm counting $*$ having given all values of ζ, μ and $*_c$. Recall that

$$(f * g)(X) = \sum_{\substack{A,B \subseteq X \\ A \cup B = X \\ A \cap B = \emptyset}} f(A)g(B)$$

so we can "split" the sum on the right side of the equation according to sizes of sets A and B:

$$(f * g) = \sum_{c=0}^{n} \sum_{a=0}^{c} \rho_c \Big((\rho_a f) *_c (\rho_{c-a} g) \Big)$$

where ρ_i is "projection" operator that cuts off all sets of size different than i:

$$(\rho_i f)(X) = \begin{cases} f(X) & \text{if } |X| = i \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

This algorithm gives us complexity bigger by factor $\mathcal{O}(n^2)$, but we don't care about polynomial complexity that much.

At this moment it's about time to tell how we can count values of μ and ζ. Let's focus on ζ first. Note that we can think about this transform not as $(\zeta f) : 2^n \to \mathbb{Z}$, but rather $(\zeta f) : \{0,1\}^n \to \mathbb{Z}$ to not to think about subsets but rather n-bit sequences. Obviously the definition is equivalent, it's just a different representation.

For $i = 1, \ldots, n+1$ let's define

$$f_i(a_1, a_2, \ldots, a_n) = \sum_{\substack{b_i, b_{i+1}, \ldots, b_n \\ b_i \leq a_i \\ b_{i+1} \leq a_{i+1} \\ \ldots \\ b_n \leq a_n}} f(a_1, a_2, \ldots, a_{i-1}, b_i, b_{i+1}, \ldots, b_n) \tag{12}$$

so we fix i first bits and sum all values on latter $n - i$ positions provided that corresponding bit is set.

Note that $(\zeta f) = f_1$ and $f = f_{n+1}$. Moreover we can compute value of f_i from the following recursion:

$$f_i(a_1, a_2, \ldots, a_n) = \begin{cases} f_{i+1}(a_1, a_2, \ldots, a_n) & \text{if } a_i = 0 \\ f_{i+1}(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n) + f_{i+1}(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n) & \text{if } a_i = 1 \end{cases}$$

Observe that this recursion gives us a time complexity of $\mathcal{O}^\star(2^n)$.

Computing values of μ is very analogous, the only difference is the factor $(-1)^{|X \setminus Y|}$. In this case we have to modify slightly f_i:

$$f_i(a_1, a_2, \ldots, a_n) = \sum_{\substack{b_i, b_{i+1}, \ldots, b_n \\ b_i \leq a_i \\ b_{i+1} \leq a_{i+1} \\ \ldots \\ b_n \leq a_n}} f(a_1, a_2, \ldots, a_{i-1}, b_i, b_{i+1}, \ldots, b_n) \cdot (-1)^{\sum_{j=i}^{n} a_j - b_j} \tag{13}$$

and all other comments are valid also in this case.

Finally, after this long introduction we can solve Set Cover and Chromatic Number problems using convolution techniques.

3.1 Set Cover

Let's define function $g : 2^U \to \{0, 1\}$ telling us whether argument belongs to the family F or not:

$$g(X) = [X \in F]$$

Now the value

$$\Big(\underbrace{g *_c g *_c g *_c \ldots *_c g}_{\text{k times}}\Big)(U)$$

will be equal to the amount of (ordered) coverages of U. If the value is positive it implies that this is possible to cover U using exactly k highlighted sets.

The complexity is still $\mathcal{O}^\star(2^n)$, so there is no gain using this method over Inclusion-Exclusion Principle.

3.2 Chromatic Number

The idea is roughly the same: define $h(X) = [X \text{is independent in } G]$. Again we check if

$$\Big(\underbrace{h *_c h *_c h *_c \ldots *_c h}_{\text{k times}}\Big)(G)$$

is positive.

However using this method gives us complexity $\mathcal{O}^\star(2^n)$, which is better than dynamic algorithm and the same as Inclusion-Exclusion Principle (with expotential memory).

For further reading it is good to take a look to [1].

References

[1] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *STOC*, pages 67–74, 2007.

[2] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[3] Martin Fürer and Shiva Prasad Kasiviswanathan. Algorithms for counting 2-satsolutions and colorings with applications. In *AAIM*, pages 47–57, 2007.