Lecture 6 (16.11.2012) Author: Pawel Walczak

MSO_2 logic, randomized FPT algorithms

This lecture will be divided into two parts: in the first one, we will introduce the MSO_2 logic and explain how to use it to find some FPT solutions. In the second part, we will learn about randomized solutions to problems known from previous lectures, which give us better complexity.

1 MSO_2 logic

1.1 motivation

Our intention is to try to specify some logical language, which will operate on graphs, that will be limited enough to find a universal solution for any formula specified in it, with complexity dependent on the formula's size. We will then try to express problems in terms of this language.

1.2 definition

In our logic, we will allow building formulas out of the following elements:

(a) operators:

$$\vee, \wedge, \neg, \Rightarrow, =, \neq$$

(b) quantifiers:

$$\forall, \exists$$

but we allow only quantifying over edges, vertices, sets of edges or sets of vertices.

(c) predicates:

$\mathrm{adj}(u, v)$– verifies if two vertices are connected

$\mathrm{inc}(u, e)$– verifies if a vertex is incident with given edge

(d) symbols:

$$\in, \subseteq$$

1.3 3–colouring

One very good example for building this kind of formulas might be the following problem:

3–COLOURING
Input: A graph G
Question: Determine whether it is possible to assign one of three possible colors to each vertex, so that every edge will connect vertices with different colors assigned

The formula that is true iff the answer to this problem is positive, might look like this:

$$\underset{V_1 \subseteq V}{\exists} \underset{V_2 \subseteq V}{\exists} \underset{V_3 \subseteq V}{\exists} \left(\left(\underset{x \in V}{\forall} x \in V_1 \vee x \in V_2 \vee x \in V_3 \right) \wedge \right.$$
$$\left. \wedge \left(\underset{x,y \in V_1}{\forall} \neg adj(x,y) \right) \wedge \left(\underset{x,y \in V_2}{\forall} \neg adj(x,y) \right) \wedge \left(\underset{x,y \in V_3}{\forall} \neg adj(x,y) \right) \right)$$

1.4 connectivity

CONNECTIVITY **Input:** A graph G **Question:** Determine whether the given graph is connected

The logical expression equivalent to this problem might be:

$$\underset{x,y \in V}{\forall} \neg \left(\underset{V_1 \subseteq V}{\exists} x \in V_1 \wedge y \notin V_1 \wedge \forall_{u,v \in V} adj(u,v) \Rightarrow (u \in V_1 \Leftrightarrow v \in V_1) \right)$$

1.5 solving formulas

Theorem 1. ***(Courcelle)****[1] Given a graph G and it's tree decomposition of width t, and an MSO_2 formula ϕ, it is possible to verify if ϕ is true with complexity $\mathcal{O}(f(t, |\phi|) \cdot n)$.*

The proof of this theorem is constructive, but we will not provide it here.

This theorem gives us a neat way to prove that some problems, which we are able to express in form of a respectively short logical formula, are in fact FPT problems. There is another variation of this method: we might extend our logical language, allowing to check some set's size (for example we could write down the condition $|X| = k$). In such case, it is possible to verify the formula with complexity $\mathcal{O}(f(t, |\phi|) \cdot n^2)$.

Although this method can tell us whether some problem is FPT, the algorithm which we get through the Courcelle theorem is highly unefficient, because of the complexity of the f function. If we want an algorithm with a reasonable dependency on treewidth parameter, we need to use other techniques.

2 Randomized FPT algorithms

In this section, we will show the method of finding randomized FPT algorithms, by following two examples.

2.1 k-path

k-Path
Input: directed graph $G = (V, A)$, value k
Question: Determine whether there is a path in G consisting of k vertices

We want to achieve the complexity $\mathcal{O}(2^k n^{\mathcal{O}(1)})$. At first, we want to reduce the problem to verifying if some polynomial is not equal to zero. We will do it in the following steps:

1) we assume that we have a field F of characteristic 2 (meaning that for any x, $x + x = 0$)

2) we create a set of k labels, $L = \{l_1, l_2, \ldots, l_k\}$

3) we create variables $y(u, v)$ for every $uv \in A$, and x_{v,l_i} for $v \in V, l_i \in L$

4) we establish a starting vertex $s \in V$

Now, the polynomial for our problem will look like this:

$$P = \sum_{\substack{\text{walks of length } k \\ v_1,\ldots,v_k \\ v_1=s}} \sum_{\pi \in S_k} \prod_{1 \le i < k} y(v_i, v_{i+1}) \prod_{1 \le i \le k} x_{v_i, l_{\pi(i)}}$$

Lemma 2. *$P \equiv 0 \Leftrightarrow G$ does not have a path of length k that begins at s.*

Proof.

$\Leftarrow$

Let's assume that there is a k–path in G, and let it be $v_1, \ldots, v_k$. The monomial corresponding to our path will be of a form

$$\prod_{1 \le i < k} y(v_i, v_{i+1}) \prod_{1 \le i \le k} x_{v_i, l_i}$$

Notice that on the path, all the vertices are distinct, and therefore there is only one permutation that might give this monomial. Also the walk corresponding to this monomial must be our path, because there is only one way to go from vertex v_1 to v_2, etc. Therefore, this monomial will be added to our sum exactly once. So the polynomial will not be equal to zero.

$\Rightarrow$

Let's assume that $P \not\equiv 0$. Let's take any non-zero monomial from P. We will now show, that the walk corresponding to that monomial $(v_1, \ldots, v_k)$ is a path.

Let's assume that it is not a path. It means that we visit some vertex twice. We want to take the first of such vertices, say $v_a = v_b$ for $a < b$. Notice, that if the monomial will be added to the sum the even number of times, it will disappear, because of the characteristic of our field. Therefore, our goal is to show, that this monomial is in fact added even number of times. To do so,

we will show a function, that will associate the occurrences of the monomial in pairs. To create a permutation that gives the same monomial out of a given permutation, we could do what follows:

$$\pi'(i) = \begin{cases} \pi(i) & \text{if } i \neq a, i \neq b \\ \pi(b) & \text{if } i = a \\ \pi(a) & \text{if } i = b \end{cases}$$

This allows us to create permutation that gives the same monomial as a given one. Applying this function twice to any permutation will give the same permutation, so it is a good pairing function. Therefore, our monomial will be counted even number of times, so it disappears. We have a contradiction, so the walk was in fact a path. □

2.2 Schwartz-Zippel lemma

Lemma 3. ***(Schwartz-Zippel)*** *Let P be a non–zero polynomial of variables $\{x_1, \ldots, x_n\}$ over the field F. Let it have a degree d. For a randomly chosen set of variables values, it has a value 0 with a probability $\leq \frac{d}{|F|}$.*

We leave this lemma without a proof.

Now, we want to apply this lemma to the $k-path$ problem. We will use the dynamic programming, and the array we will be filling will be $t[v][S]$, where $v \in V$ is the vertex in which our walks end, and $S \subseteq L$ is the set of labels that we allow to use. So, by definition, it will be of the following form:

$$t[v][S] = \sum_{\substack{\text{walks of length } |S| \\ v_1,\ldots,v_{|S|} \\ v_1=s, v_{|S|}=v}} \sum_{f:\{1,\ldots,|S|\} \xrightarrow{1-1} S} \prod_{1 \leq i < |S|} y(v_i, v_{i+1}) \prod_{1 \leq i \leq |S|} x_{v_i, f(i)}$$

We can write the recursive way of calculating these values as follows:

$$t[v][S] = \sum_{(u,v) \in A} \sum_{f_v \in S} t[u][S \backslash \{f_v\}] \cdot y(u,v) \cdot x_{v,f_v}$$

Calculating one field in the array takes polynomial time. Therefore, we can calculate the whole array with complexity $\mathcal{O}(2^k poly(|G|))$. It gives us a randomized algorithm for the $k-path$ problem.

2.3 Hamiltonian cycle

Hamiltonian Cycle
Input: a graph G
Question: verify if G contains a cycle, which passes every vertex of G exactly once

During the previous lecture, we showed how to solve this problem, using dynamic programming, with complexity $\mathcal{O}(c^{tw \log tw} \cdot n^{\mathcal{O}(1)})$, where tw is the treewidth of G. Now, we will try to improve it to $\mathcal{O}(c^{tw} \cdot n^{\mathcal{O}(1)})$.

At first, we will consider a simplification: we will assume, that G contains exactly zero or one Hamiltonian cycles. Therefore, it will be enough to calculate the number of Hamiltonian cycles mod 2.

Definition 4. Let's assume that v_1 is a fixed vertex. We will define a **cut** to be such division of V into subsets V_1, V_2, that $V_1 \dot{\cup} V_2 = V$, and $v_1 \in V_1$.

Definition 5. cycle cover is such subset $C \subseteq E$, that every vertex from V has exactly two edges incident with it in C.

Definition 6. We will say that a cycle cover $C \subseteq E$ is **consistent** with a cut V_1, V_2, iff $C \cap E(V_1, V_2) = \varnothing$.

Lemma 7.

$$|\{(E, (V_1, V_2)) \ : \ E \textit{ is a cycle cover}, (V_1, V_2) \textit{ is a cut, and } E \textit{ is consistent with } (V_1, V_2)\}| \equiv_{\mathrm{mod}\ 2}$$

$$\equiv_{\mathrm{mod}\ 2} \#\textit{Hamiltonian cycles in } G$$

Proof. Notice, that for any cycle cover, it will be included into the sum 2^{a-1} times, where a is the number of cycles it contains. Therefore, every non-Hamiltonian cycle cover will be calculated even number of times. On the other hand, every Hamiltonian cycle will be calculated exactly once. □

Those pairs can be calculated using dynamic programming. The states might be, for each vertex, on which side of a cut it is (2 states), and what is it's current degree (3 possibilities: $0, 1$ or 2). It will give us 6 states in total. However, we can limit them down to 4 states, because when a vertex has a degree of 0 or 2, we don't have to consider which side is it on[2].

Theorem 8. *The parity of the number of Hamiltonian cycles in a given graph can be calculated with complexity* $\mathcal{O}(4^{tw} \cdot n^{\mathcal{O}(1)})$.

2.4 Isolation lemma

Lemma 9. *Let* $F \subseteq 2^U$ *be a non-empty family of sets. If we assign the weights randomly, let's say* $f : U \to \{1, \ldots, N\}$*, then with a probability* $\geq 1 - \frac{|U|}{N}$ *there is exactly one set in* F *with a minimal total weight.*

Proof. For $x \in U$, let's define

$$\alpha(x) = \min_{S \in F, x \notin S} f(S) - \min_{S \in F, x \in S} f(S \backslash \{x\})$$

The values $f(x)$ and $\alpha(x)$ are independent.
If there are two sets in F with the smallest weight, then

$$\bigvee_{x \in S_1 \backslash S_2} \alpha(x) = f(x)$$

After applying union bound, we get the thesis. □

Using this lemma in the Hamiltonian cycle problem, we could modify our algorithm, so that we choose randomly the weights for edges from the set $\{1, \ldots, 2m\}$, and we are looking for the lightest Hamiltonian cycle. With high probability, there will be only one such cycle (if any).

Moreover, the Lemma 7 and Theorem 8 can be easily adapted to solve the problems of Cycle cover and Hamiltonian cycle with weights on edges, because weights have the polynomial size.

References

[1] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[2] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011.