Advanced Graph Algorithms Jan-Apr 2014

Courcelle's Theorem

Speaker: Anantha Padmanabha MS *Date: April 29, 2014*

1 Overview

In this lecture, we shall see **Courcelle's theorem** and its proof in a nutshell. This result was first proved by Bruno Courcelle in 1990 and is considered the archetype of algorithmic meta-theorems. The statement of the theorem is the following :

Theorem 1. *Given a graph $G = (V, E)$ of tree width w, and an MSO sentence ϕ, $G \models \phi$ can be decided in time $O(f(|\phi|, w) \cdot |G|)$.*

2 Background

To prove this theorem, we need to know about Monoidic second order logic and Regular tree languages.

2.1 Monoidic second order logic (MSO)

The syntax of Monoidic second order logic over the Domain D is given as follows :

- $t \to x \in$ variable $|c \in$ constant $|f(t_1, \cdots t_n)$, where $f \in$ functions
- $\phi \to Predicate(t_1, \cdots t_m) \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \exists x\phi_1$, where $x \in$ D $\mid \exists X\phi_1$, where $X \subseteq$ D

Informally, monoidic second order logic has all the powers of the first order logic and in addtion to that, quantification over the subset of the domain is also allowed. As a convention we use captial letters to quantify over subsets and small letters to quantify over elements.

MSO on graphs: In this lecture, we are interested in the graph properties which are expressible in MSO. Let us see a few examples :

1. A Graph is 3-colourable

$$\exists X_1 \exists X_2 \exists X_3 (\forall x (X_1 x \wedge \neg X_2 x \wedge \neg X_3 x) \vee (\neg X_1 x \wedge X_2 x \wedge \neg X_3 x) \vee (\neg X_1 x \wedge \neg X_2 x \wedge X_3 x)) \wedge$$
$$(\forall x \forall y Exy \implies (\neg(X_1 x \wedge X_1 y) \wedge \neg(X_2 x \wedge X_2 y) \wedge \neg(X_3 x \wedge X_3 y))$$

2. A Graph has a dominator size of set k

$$\exists x_1 \exists x_2 \cdots \exists x_k \forall x (\bigvee_{i=1}^{k} x = x_i) \vee \exists y (Exy \wedge \bigvee_{i=1}^{k} y = x_i)$$

2.2 Tree Automaton

A Tree Automaton is a Finite state machine which is used to recognize Tree languages. A Tree language is the set of all trees whose vertices are labelled from a fixed Σ and satisfies some required properties. For our current topic, we restric ourselves only to binary trees.

Suppose $\Sigma = \{a, b\}$ and $\mathcal{T} = \{T \mid T \text{ is a labelled binary tree}\}$, then an example for a tree language is $L = \{T \in \mathcal{T} \mid \text{ Total number of "}a\text{" labelled nodes} \leq 5\}$

A deterministic bottom-up tree automaton is $A = (Q, \Sigma, \delta, F)$ where

- Q is a set of finite states
- Σ is the alphabet used to label the tree nodes
- $F \subseteq Q$ is the set of final states
- $\delta : Q \cup \{\perp\} \times Q \cup \{\perp\} \times \Sigma \to Q$

The Automaton starts associating the nodes of the input tree with some states starting from the leaf and finally the input tree is accepted if the state associated with the root node $\in F$

A Tree Language N is said to be regular if there exists a deterministic tree automaton A such that $L(A) = N$.

Theorem 2. *A tree language L is MSO definable iff L is regular.*

The proof of this theorem is not discussed in this lecture, but we note that the proof for $\Leftarrow$ is done by inductively constructing the tree automaton for an MSO sentence depending on its structure. We shall use this property as a part of our algorithm.

3 Proof of Courcelle's theorem

Let us describe an algorithm which takes a Graph G and an MSO sentence ϕ as inputs and decide whether $G \models \phi$.

Courcelle
Input: Graph $G(V, E)$ of tree width w and an MSO sentence ϕ
Output: whether $G \models \phi$ or not
Step 1: Construct $T_G \leftarrow$ Tree-Decompostion(G)
Step 2: $\mathcal{T} \leftarrow$ Label the vertices of T_G by a suitable Σ
Step 3: Construct MSO sentence ϕ' such that $G \models \phi$ iff $\mathcal{T} \models \phi'$
Step 4: Check if $\mathcal{T} \models \phi'$ and return the $Yes \backslash No$ accordingly.

The correctness of the Algorithm is clear from the construction. We shall prove that the running time of this algorithm is $O(f(|\phi|, w) \cdot |G|)$.

3.1 Step 1 and Step 3

In the first step, we construct the tree decomposition of the given graph. Bodlaender[1] proved that there is an algorithm which given a graph G as input, constructs a tree-decomposition of G of width w in time $O(2^{w^2} \cdot |G|)$. We shall use this algorithm in step 1 to construct the tree Decomposition of G.

For the third step, we use Theorem[2] to construct the tree automaton A for ϕ' and then check if A accepts $\mathcal{T}$. Constructing A takes $O(f(|\phi'|))$ time and checking if A accepts G takes O($|G|$) time.

Thus if we are able to achieve Step 2 and Step 3 in $O(f(|\phi|, w) \cdot |G|)$ then it concludes the proof.

3.2 Step 2

Without loss of generality, we can assume that T_G is a binary tree. Every $S \subseteq V_G$ of size at most $w+1$ can be encoded as a string of length $w+1$ − list the elements of S in some (canonical) order, and repeat some to ensure that the string is of length exactly $w+1$.

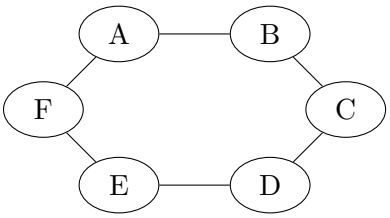
Thus, we have a a tree decomposition $T_G = (V_T, E_T, B_T)$ of width w for graph $G = (V_G, E_G)$ such that $B_T : V_T \to V_G^{w+1}$. Note that $|V_T| = O(|V_G|)$.

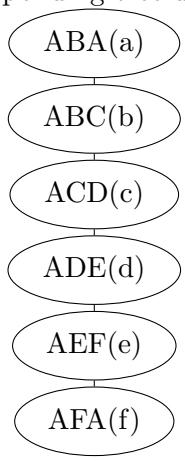
Labelling the vertices of T We could have used B_T itself as the label for the vertices of T. But the problem is that B_T is dependent on the size of $|G|$, but to label the vertices of a tree we need a fixed Σ.

Hence, we construct $\mathcal{T}(V_T, E_T, L_T)$ with a new label for vertices of T using a fixed Σ. L_T is defined as follows : $\forall u \in V_T$, $L(u) = (\lambda_1, \lambda_2, \lambda_3)$ where,

- $\lambda_i \subseteq \{1, 2, \cdots w+1\} \times \{1, 2, \cdots w+1\}$, for i = 1, 2, or 3.
- $\lambda_1 = \{(i, j) |$ i^{th} vertex in $B_T(u)$ and j^{th} vertex in $B_T(u)$ have an edge between them in $G\}$
- $\lambda_2 = \{(i, j) |$ i^{th} vertex in $B_T(u)$ and j^{th} vertex in $B_T(u)$ are the same $\}$
- $\lambda_3 = \{(i, j) |$ i^{th} vertex in $B_T(u)$ and j^{th} vertex in $B_T(v)$ are the same, where v is the parent of u in $T_G\}$

Example Consider the following graph and its corresponding tree-decomposition.





The corresponding labelling would be as follows :

- $L_T(a) = \{(12, 21, 23, 32)(13, 31)()\}$
- $L_T(b) = \{(12, 21, 23, 32)()(11, 13)\}$
- $L_T(c) = \{(23, 32)()(11, 23)\}$
- $L_T(d) = \{(23, 32)()(11, 23)\}$
- $L_T(e) = \{(13, 31, 23, 32)()(11, 23)\}$
- $L_T(f) = \{(12, 21, 23, 32)(13, 31)(11, 31, 23)\}$

Note that the new labelling depends only on w and not on the size of G and hence has a constant Σ. Also, this labelling can be computed in $O(|V_T|) = O(|V_G|) = O(|G|)$ time.

3.3 Step 3

In this step we need to construct ϕ' such that, $G \models \phi$ iff $\mathcal{T} \models \phi'$. To do this, we need to encode the vertices of G as tuples of vertices of $\mathcal{T}$

Any $S \subseteq V_G$ is encoded as $Tree(S) = (U_1, U_2, \cdots, U_{w+1})$, where $U_i \subseteq V_T$ such that $U_i = \{v \in V_T |\ i^{th}$ value in $B_T(v) \in S\}$

For the above example, $Tree(\{A, C, E\}) = (\{a, b, c, d, e, f\}, \{\}, \{a, b, d, f\})$
$Tree(\{A\}) = (\{a, b, c, d, e, f\}, \{\}, \{a, f\})$

It is important to note that, not every tuple of vertices of trees correspond to a vertex in the graph. For instance, in the above example $(\{a\}, \{\}, \{\})$ is not $Tree(S)$ for any S.

Lemma 3. *A tuple of sets of nodes of $\mathcal{T}_G$, $\overline{X} = (X_1, \cdots X_{w+1})$ is the encoding $Tree(S)$ for some S with $|S| \geq 1$ iff*

(a) If $(i,j) \in \lambda_2(u)$ then $u \in X_i$ iff $u \in X_j$.

(b) If $(i,j) \in \lambda_3(u)$ then $u \in X_i$ iff $v \in X_j$, where v is the parent of u.

$\overline{X}$ is $Tree(S)$ for some S with $|S| = 1$ iff (a) and (b) holds and

(c) If $u \in X_i$ and $u \in X_j$ then $(i,j) \in \lambda_2(u)$.

(d) If $u \in X_i$,$v \in X_j$ (where v is parent of u) then $(i,j) \in \lambda_3(u)$.

(e) $\bigcup_{i=1}^{w+1} X_i$ forms a connected component of T_G.

Proof. $\Rightarrow$ is easy to see in both the cases.
We need to prove $\Leftarrow$
Case $|S| \geq 1$
Let $\overline{X} = (X_1, \cdots X_{w+1})$ be a tuple that satisfies (a) and (b).
Define $(Set(\overline{X}) = \{x \in V_G |\ \exists i \exists u, u \in X_i$ and i^{th} position in $B_T(u) = x\}$
Claim : $Tree(Set(\overline{X}) = \overline{X}$
Suppose not, then there are u_1 and u_2 such that i^{th} position in $B_T(u_1) = j^{th}$ position in $B_T(u_2) = x$ such that $u_1 \in X$ and $u_2 \notin X$

- Since $x \in B_T(u_1)$ and $x \in B_T(u_2)$, x appears in the label of every vertex on the path from u_1 to u_2.
- Thus, without loss of generality, we may assume that either $u_1 = u_2$ or u_1 is parent of u_2.
- But then $\overline{X}$ violates (a) if $u_1 = u_2$ and $\overline{X}$ violates (b) if u_1 is the parent of u_2, which contradicts the fact that $\overline{X}$ satisfies both (a) and(b).

Case $|S| = 1$
Let $\overline{X}$ satisfy $(a), (b), (c), (d)$ and (e), once again consider $S = Set(\overline{X})$
Since $\overline{X}$ satisfies (a) and (b), by the previous result $\overline{X}$ is a valid tuple for S. Claim: $|S| = 1$
Suppose not, then $|S| > 1$ and there are vertices u_1 and u_2 and i and j such that $u_1 \in X_1$, $u_2 \in X_j$, and i^{th} position in $B_T(u_1) = j^{th}$ position in $B_T(u_2)$.

- Since $\overline{X}$ satisfies (e), without loss of generality, $u_1 = u_2$ or u_1 is the parent of u_2.
- But then, $\overline{X}$ violates (c) and (d).

□

Once we have proved the validity condition of the tuples, we can encode these conditions in the form of MSO sentence. Let ϕ_a ϕ_b ϕ_c ϕ_d ϕ_e denote the encoding of the conditions (a) (b) (c) (d) (e) respectively.

- $\phi_a(X_1, \cdots X_{w+1}) = \forall x \bigwedge_{i,j \in \{1,2 \cdots w+1\}} \bigwedge_{a:(i,j) \in \lambda_2(a)} (X_i x \iff X_j x)$
- $\phi_b(X_1, \cdots X_{w+1}) = \forall x \forall y \bigwedge_{i,j \in \{1,2 \cdots w+1\}} \bigwedge_{a:(i,j) \in \lambda_3(a)} E_{T_G} xy \implies (X_i x \iff X_j x)$
- $\phi_c(X_1, \cdots X_{w+1}) = \forall x((X_i x \wedge X_j x) \implies \bigvee_{a:(i,j) \in \lambda_2(a)} Q_a x$
- $\phi_d(X_1, \cdots X_{w+1}) = \forall x \forall y((X_i x \wedge X_j y \wedge E_{T_G} yx) \implies \bigvee_{a:(i,j) \in \lambda_3(a)} Q_a x$
- $\phi_2(X_1, \cdots X_{w+1}) = Connected(\bigcup_{i=1}^{w+1} X_i)$ where,

$$Connected(Y) = \forall Y_1 \forall Y_2((Y_1 \cup Y_2 = Y) \wedge (Y_1 \cap Y_2 = \Phi) \wedge Y_1 \neq \Phi \wedge Y_2 \neq \Phi) \implies (\exists x \exists y Y_1 x \wedge Y_2 y \wedge E_{T_G} xy)$$

Now we are ready to construct MSO ϕ' such that given $\phi(x_1, x_2, \cdots x_n, Y_1, Y_2, \cdots Y_m)$ we construct $\phi'(\overline{X_1}, \overline{X_2}, \cdots \overline{X_n}, \overline{Y_1}, \overline{Y_2} \cdots \overline{Y_m}$ such that

$$G \models \phi(x_1, x_2, \cdots x_n, Y_1, Y_2, \cdots Y_m) \iff \mathcal{T}_{\mathcal{G}} \models \phi'(\overline{X_1}, \overline{X_2}, \cdots \overline{X_n}, \overline{Y_1}, \overline{Y_2} \cdots \overline{Y_m})$$

ϕ' is constructed inductively :

Base case

- Case $\phi : (x_1 = x_2)$ then $\phi' : \bigwedge_{i=1}^{w+1} X_{1_i} = X_{2_i}$
- Case $\phi : E_G xy$ then $\phi' : \exists u \bigvee_{a:(i,j) \in \lambda_2(a)} X_{a_i} u \wedge X_{a_j} u \wedge Q_a u$
- Case $\phi : Yx$ then $\phi' : \bigwedge_{i=1}^{w+1} X_i \subseteq Y_i$

Induction step :

- Case $\phi : \phi_1 \wedge \phi_2$ then $\phi' : \phi'_1 \wedge \phi'_2$
- Case $\phi : \phi_1 \vee \phi_2$ then $\phi' : \phi'_1 \vee \phi'_2$
- Case $\phi : \neg \phi_1$ then $\phi' : \neg \phi'_1$
- Case $\phi : \exists Y \phi_1$ then $\phi' : \exists \overline{Y}(\phi_a(\overline{Y}) \wedge \phi_b(\overline{Y}) \wedge \phi'_1)$
- Case $\phi : \exists x \phi_1$ then $\phi' : \exists \overline{X}(\phi_a(\overline{X}) \wedge \phi_b(\overline{X}) \wedge \phi_c(\overline{X}) \wedge \phi_d(\overline{X}) \wedge \phi_e(\overline{X}) \wedge \phi'_1)$

References

[1] Hans L. Bodlaender: A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth.SIAM J. Comput.25(6): 1305-1317,1996