Automata based graph algorithms for logically defined problems

Bruno Courcelle
(includes current work with Irène Durand)

Bordeaux-1 University, LaBRI (CNRS laboratory)
References:


Topics of lectures

*Algorithmic meta-theorems*: existence and construction of (relatively) efficient graph algorithms from logical descriptions of the problems.

*These lectures*: meta-theorems based:
- on problem descriptions in (extensions of) MSO (Monadic Second-Order) logic,
- on hierarchical decompositions of graphs
- and on automata, possibly with *infinitely many states*.

A kind of *theory of dynamic programming*.
Summary of 3 lectures

Part 1

First example: construction of a finite automaton for the 2-colorability of series-parallel graphs.

Graph decompositions expressed by algebraic terms: tree-width and clique-width, parameters for FPT and XP graph algorithms.

Automata based algorithms: the general scheme.

Difficulty: the size of automata; the example of connectedness.

Fly-automata: definitions.
3 types of fly-automata: P, FPT and XP.

**Part 2**

Monadic Second-Order logic: definitions, examples.

The main construction: from MSO formulas to automata (accepting clique-width terms).

Existential quantifications and nondeterministic automata.

Example: colorability problems.
Part 3 (Recent work)

Beyond MSO logic for graph properties and functions.

A fly-automaton for regularity of graphs (not an MSO property).

Boolean and first-order constructions of properties and functions, and their interpretations in terms of fly-automata.

Monadic-second order constructions; spectra.

Implementation (in AUTOGRAPH) and tests.

Conclusions and call for interesting problems to handle in this way.
2-colorability of Series-Parallel (SP) graphs

Graphs with distinguished vertices marked 1 and 2, generated from
\( e = 1 \rightarrow 2 \) by the operations of parallel-composition // and series-composition •

\[
\begin{align*}
S &= S // S \cup S \cdot S \cup \{ e \}
\end{align*}
\]
**Inductive computation**: Test of 2-colorability for SP graphs

Not all series-parallel graphs are 2-colorable (see $K_3$).

G and H 2-colorable does not imply that $G//H$ is 2-colorable (see $K_3= P_3//e$).

One can check 2-colorability with 2 auxiliary properties:

- $\text{Same}(G) = G$ is 2-colorable with sources of the same color,
- $\text{Diff}(G) = G$ is 2-colorable with sources of different colors

by using the rules:

- $\text{Diff}(e) = \text{True} ; \text{Same}(e) = \text{False}$
- $\text{Same}(G//H) \iff \text{Same}(G) \land \text{Same}(H)$
- $\text{Diff}(G//H) \iff \text{Diff}(G) \land \text{Diff}(H)$

- $\text{Same}(G\cdot H) \iff (\text{Same}(G) \land \text{Same}(H)) \lor (\text{Diff}(G) \land \text{Diff}(H))$
- $\text{Diff}(G\cdot H) \iff (\text{Same}(G) \land \text{Diff}(H)) \lor (\text{Diff}(G) \land \text{Same}(H))$
Application: An algorithm based on a finite bottom-up automaton

For every term $t$, we can compute, by running a finite deterministic bottom-up automaton on $t$, the pair of Boolean values $(\text{Same}(G(t)), \text{Diff}(G(t)))$, where $G(t)$ is the graph value of $t$. We get the answer whether $G(t)$ is 2-colorable.

Example: $\sigma$ at node $u$ means that $\text{Same}(G(t/u))$ is true, $\overline{\sigma}$ that it is false, $\delta$ that $\text{Diff} \ (G(t/u))$ is true, etc... Computation is done bottom-up with the rules of previous page.

Answer: the graph is not 2-colorable.
Algebraic view of tree-decompositions

Graph G

Tree T

Tree-decomposition of G

Dotted lines --- link copies of a same vertex.

Width = max. size of a box -1. Tree-width = minimal width of a tree-decomposition
Graph operations and terms for tree-decompositions

Graphs have distinguished vertices called sources, (or terminals or boundary vertices) pointed to by source labels from \{a, b, c, ..., d\}.

**Binary operation**: *Parallel composition*

G // H is the disjoint union of G and H and sources with same label are fused.

(If G and H are not disjoint, one first makes a copy of H disjoint from G).
Unary operations:

*Forget a source label*

$\text{Forget}_a(G)$ is $G$ without any $a$-source: the source is no longer distinguished (it is made "internal").

Source renaming:

$\text{Ren}_{a \leftrightarrow b}(G)$ exchanges source labels $a$ and $b$

(replaces $a$ by $b$ if $b$ is not the label of any source)

Nullary operations denote basic graphs: edge graphs, isolated vertices.

Terms over these operations define (or denote) graphs (with or without sources)
Example: Trees

Constructed with two source labels, \( r \) (root) and \( n \) (new root).

Fusion of two trees at their roots:

Extension of a tree by parallel composition with a new edge, forgetting the old root, making the "new root" as current root:

\[
\text{Extension} \quad e = r \cdot \cdots \cdot n
\]

\[
\text{Renr} \quad r \ (\text{Forget} r \ (G // e))
\]

Defining equation: \( T = T // T \cup \text{extension}(T) \cup r \)
Series-parallel graphs have tree-width 2.

*Proposition:* A graph has tree-width \( \leq k \) if and only if it can be constructed from edges by using the operations \( // \), \( Ren_{a \leftarrow b} \) and \( Forget_{a} \) with \( \leq k+1 \) labels \( a, b, \ldots \).

*Consequences:*
- Representation of tree-decompositions by terms.
- Algebraic characterization of tree-width.
- Terms as inputs to graph algorithms
From an algebraic expression to a tree-decomposition

Example: \( cd \leftrightarrow_{\text{Ren}_a}^c (ab \leftrightarrow_{\text{Forget}_b}(ab \leftrightarrow bc)) \)  
\( \text{(ab denotes an edge from a to b)} \)
Graph operations for defining clique-width

Graphs are simple, directed or not, and labelled by $a, b, c, \ldots$.
A vertex labelled by $a$ is called an $a$-vertex.

One binary operation: disjoint union : $\oplus$

Unary operations: edge addition denoted by $\text{Add}_{a,b}$

$\text{Add}_{a,b}(G)$ is $G$ augmented with directed or undirected edges from every $a$-vertex to every $b$-vertex. The number of added edges depends on the argument graph.

$H = \text{Add}_{a,b}(G)$; only 5 new edges added
vertex relabellings:

\[ \text{Relab}_a \rightarrow_b (G) \text{ is } G \text{ with every } a\text{-vertex is made into a } b\text{-vertex} \]

Basic graphs: those with a single vertex.

**Definition:** A graph \( G \) has **clique-width** \( \leq k \) if and only if \( G = G(t) \) is defined by a term \( t \) using \( \leq k \) labels.

**Example:** Cliques have clique-width 2. 
\( K_n \) is defined by \( t_n \) where \( t_{n+1} = \text{Relab}_b \rightarrow_a ( \text{Add}_{a,b} (t_n \oplus b) ) \)
Tree-width and clique-width

Proposition: (1) Bounded tree-width implies bounded clique-width \((\text{cwd}(G) \leq 2^{2\text{twd}(G)} + 1\) for \(G\) directed), but not conversely.

(2) Unlike tree-width, clique-width is sensible to edge directions: Cliques have clique-width 2, tournaments have unbounded clique-width.

Classes of **unbounded tree-width** and **bounded clique-width**:
- Distance hereditary graphs (3),
- Graphs without \(\{P_5, 1 \otimes P_4\}\) (5), or \(\{1 \oplus P_4, 1 \otimes P_4\}\) (16) as induced subgraphs.

Classes of **unbounded clique-width**:
- Planar graphs of degree 3, Tournaments, Interval graphs.
- Graphs without induced \(P_5\). \((P_n = \text{path with } n \text{ vertices})\)
Exercises

1) Complete the proof of the proposition page 14: transform a tree-decomposition of width $k$ into a term that defines the same graph and uses $k+1$ source labels.

2) Prove that this proposition holds without the source renaming operations.

3) What is the maximal clique-width of a SP graph?

4) Give upper-bounds to the tree-width and the clique-width of the rectangular $n \times m$ grids.

5) Give an upper bound to the clique-width of a graph whose biconnected components have clique-width at most $k$. 
The parsing problem: construction of decompositions

Automata take terms as inputs, not graphs: the parsing must be done before. (Graph automata do not exist in a satisfactory way).

A difficult problem: deciding \( \text{twd}(G) \leq k \) and \( \text{cwd}(G) \leq k \) (for input \((G,k)\)) are NP-complete problems.
There are FPT approximation algorithms, taking time $f(k).n^a$, that output the following for given $k$ and $G$ with $n$ vertices:

(i) either the answer that $wd(G) > k$,

(ii) or a term witnessing that $wd(G) \leq g(k)$.

Hence from an algorithm taking as input a term $t$ in $T(F_k)$ ($F_k$ : the operations for terms of width $\leq k$) and whose computation time is $h(k).n^b$, we get (by trying $k = 1, 2, \ldots$ until we reach Case (ii) ) an FPT algorithm for given $G$ with computation time $\leq m(wd(G)).n^{\max(a,b)}$.

For clique-width: approximation algorithms based on articles by Oum, Seymour, Hlineny, Kanté, 2005-2013).

However, graphs arising from concrete problems are not random. They may have “natural” hierarchical decompositions from which terms of small tree-width or clique-width are not hard to find.

Compilation: flow-graphs of structured programs have tree-width ≤ 6.

In linguistics and chemistry: graphs of tree-width ≤ 3.
Algorithmic meta-theorems through automata: the general scheme

\[ A(\varphi, k) \]

Steps are done “once for all”, independently of \( G \)

\( A(\varphi, k) \): finite automaton on terms \( t \)

\( wd = \) tree-width or clique-width or equivalent notion.
Automata on terms that check graph properties

Terms are seen as labelled trees. We want to check a property \( P(G) \), for \( G = G(t) \), \( t \) in \( T(F) \).

For each *labelled* graph \( G \), we define some piece of information \( q(G) \) consisting of properties of \( G \) and of values attached to \( G \), with:

(i) inductive behaviour of \( q \): for \( f \) in \( F \) and graphs \( G,H \):

\[
q(f(G,H)) = f^q (q(G), q(H))
\]

for some computable function \( f^q \).

(ii) \( P(G) \) can be decided from \( q(G) \).

Recall the 2-colorability of SP graphs, page 8.
Then $q(G(t/u))$ is computed bottom-up in a term $t$, for each node $u$. This information is relative to the graph $G(t/u)$ defined by the subterm $t/u$ of $t$ issued from $u$.

$q(G(t/u))$ is a state of a finite or infinite deterministic bottom-up automaton.

These automata formalize some form of dynamic programming.

In the sequel we only consider clique-width: the automata are simpler to build and they can be adapted to bounded tree-width as bounded tree-width implies bounded clique-width.

Now an example.
The deterministic automaton for connectedness.

The state at node $u$ is the set of types (sets of labels) of the connected components of the graph $G(t/u)$. For $k$ labels ($k = \text{bound on clique-width}$), the set of states has size $\leq 2^{2^k}$.

Proved lower bound: $2^{2^k/2}$.

\[ \rightarrow \text{Impossible to “compile” the automaton (i.e., to list the transitions).} \]

Example of a state: $q = \{ \{a\}, \{a,b\}, \{b,c,d\}, \{b,d,f\} \}$, ($a,b,c,d,f : \text{labels}$).

Some transitions:

$Add_{a,c}: \quad q \rightarrow \{ \{a,b,c,d\}, \{b,d,f\} \}$,

$Relab_{a \rightarrow b}: \quad q \rightarrow \{ \{b\}, \{b,c,d\}, \{b,d,f\} \}$

Transitions for $\oplus$: union of sets of types.

Note: Also state $(p,p)$ if $G(t/u)$ has $\geq 2$ connected components, all of type $p$. 

26
In a fly-automaton: the states and transitions are computed and not tabulated.

We allow fly-automata with infinitely many states and with outputs: numbers, finite sets of tuples of numbers, etc.

**Example continued:** For computing the number of connected components, we use states such as:

\[ q = \{ (\{a\}, 4), (\{a,b\}, 2), (\{b,c,d\}, 2), (\{b,d,f\}, 3) \} \]

where 4, 2, 2, 3 are the numbers of connected components of respective types \{a\}, \{a,b\}, \{b,c,d\}, \{b,d,f\}. 

Fly-automaton (FA)

**Definition**: \( A = \langle F, Q, \delta, \text{Out} \rangle \)

- **F**: finite or countable (effective) signature (set of operations),
- **Q**: finite or countable (effective) set of states (integers, pairs of integers, finite sets of integers: states can be encoded as finite words, integers in binary),
- **Out**: \( Q \rightarrow D \) (an effective domain, i.e., set of finite words), computable.
- **\( \delta \)**: computable (bottom-up) transition function

**Nondeterministic case**: \( \delta \) is *finitely multi-valued*. 
This automaton defines a **computable function** : $T(F) \rightarrow D$

(or : $T(F) \rightarrow P(D)$ if it is not deterministic)

If $D = \{ \text{True, False} \}$, it defines a **decidable property**, equivalently,

a **decidable subset** of $T(F)$.

**Deterministic** computation of a nondeterministic FA :

bottom-up computation of *finite* sets of states (classical simulation of the determinized automaton): these states are the useful ones of the **determinized automaton**; these sets are *finite* because the transition function is finitely multivalued.

Fly-automata are “implicitly determinized” and they run deterministically
Computation time of a fly-automaton

F : all graph operations,  \( F_k \) : those using \( k \) labels.

On term \( t \in T(F_k) \) defining \( G(t) \) with \( n \) vertices, if a fly-automaton takes time bounded by :

\[
(k + n)^c \rightarrow \text{it is a P-FA (a polynomial-time FA)},
\]

\[
f(k).n^c \rightarrow \text{it is an FPT-FA},
\]

\[
a.n^{g(k)} \rightarrow \text{it is an XP-FA}.
\]

The associated algorithm is polynomial-time, FPT or XP for clique-width as parameter.
**Proposition**: Every polynomial-time computable function \( T(F) \rightarrow D \) is computable by a fly-automaton whose computation time is polynomial.

Nothing new! : Our concern is to have easy and uniform constructions of FA’s from logical and combinatorial descriptions of functions and properties.

**Theorem** : Every graph property expressible in monadic second-order (MS) logic can be checked by a fly-automaton whose restriction to each subsignature \( F_k \) has finitely many states.

Hence, it is a linear FPT-FA.

Linear : its computation-time is bounded by \( f(k).n \)