FAKULTÄT FÜR !NFORMATIK

WWTF

Wiener Wissenschafts-, Forschungs- und Technologiefonds

supported under grant ICT 08-028

Poster Session ACAI'09 Belfast

Argumentation with Bounded Tree-Width

Wolfgang Dvořák

Vienna University of Technology
Institute of Information Systems
Database and Artificial Intelligence Group
Supervisor: Stefan Woltran

Motivation

- Argumentation is a general issue in AI
- Many argumentation problems are in general computationally intractable
- We are interested in tractable fragments
- By Courcelle´s theorem we know that there exists tractable algorithms for bounded tree-width but it doesn't give us practical algorithms.
- Dynamic Programming algorithms are approved for bounded tree-width problems.

Argumentation Framework

An **argumentation framework** (AF) is a pair F=(A,R) where

- A is a set of arguments and
- R ⊆ A × A a set of attacks.

The pair (a,b) ∈R means that a attacks (or defeats) b. A set S⊆ A of arguments *defeats* b (in F), if there is an a∈S, such that (a,b)∈R. An argument a∈A is *defended* by S⊆A (in F) iff, for each b∈A, it holds that, if (b,a)∈R, then S defeats b (in F).

Example: Let F=(A,R) be an AF with A={a,b,c,d,e} and R={(a,b), (c,b), (c,d), (d,c), (d,e), (e,e)}.

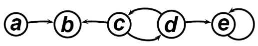


Figure: Argumentation Framework F

Semantics

Let F=(A,R) be an AF. A set S⊆A is

- **conflict-free** (cf), if there are no a, b∈S, such that (a,b)∈ R.
- a **stable extension** of F, if S is conflict-free and each a∈A\S is defeated by S in F.
- an **admissible extension** of F, if S is conflict- free and each a∈S is defended by S in F.
- a **preferred extension** of F, if S is an admissible extension and for each adm. extension T holds S⊄T.

Example: Let be F our example AF then:
stable(F)={{a,d}}
adm(F)= {{a,c},{a,d},{a},{c},{d},∅}
pref(F)= {{a,c},{a,d}}

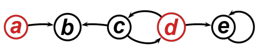


Figure: Stable Extension

Complexity

Reasoning Problems in AFs for e∈{stable,adm,pref}:

- **Cred$_e$**: Given AF F=(A,R) and a∈ A.
  Is a contained in some extension S∈ e(F)?
- **Skept$_e$**: Given AF F=(A,R) and a∈ A.
  Is a contained in each extension S∈ e(F)?

	stable	*adm*	*pref*
Cred$_e$	NP-c	NP-c	NP-c
Skept$_e$	co-NP-c	trivial	Π_2^P-c

So most of these problems are computationally hard.

Fixed Parameter Tractability

As argumentation problems are computationally intractable we are interested in tractable fragments.

- Often the computational complexity primarily depends on some problem parameters rather than on the size of the instance.
- Many hard problems become tractable if some problem parameters are fixed or bounded.
- In the area of graphs tree-width is such a parameter, e.g. there are many hard problems which are tractable for graphs of bounded tree-width.

Tree-Width

Let $\mathcal{G}$=($V_\mathcal{G}$,$E_\mathcal{G}$) be an undirected graph.
A **tree decomposition** of $\mathcal{G}$ is a pair $\langle\mathcal{T},\mathcal{X}\rangle$ where $\mathcal{T}$=($V_\mathcal{T}$,$E_\mathcal{T}$) is a tree and $\mathcal{X}$ =$(X_t)_{t\in V_\mathcal{T}}$ such that

1. $\bigcup_{t\in V_\mathcal{T}} X_t = V_\mathcal{G}$, $\mathcal{X}$ is a cover of $V_\mathcal{G}$
2. For each vertex v∈$V_\mathcal{G}$ the subgraph of $\mathcal{T}$ induced by { t : v ∈ X_t} is connected
3. For each edge {v_i,v_j} ∈ $E_\mathcal{G}$ there exists an X_t with {v_i,v_j} ⊆ X_t

The *width* of such a decomposition is

$$\max\{|X_t| : t \in V_\mathcal{T}\} - 1$$

The **tree-width** of a graph is the minimum width over all tree decompositions.

A **nice tree decomposition** is a tree decomposition where each bag ***t*** is of one of the following types:

- *Leaf*: t is a leaf of $\mathcal{T}$
- *Forget*: t has exact one child t'; $X_t = X_{t'} \dot\cup \{v\}$
- *Insert*: t has exact one child t'; $X_t \dot\cup \{v\} = X_{t'}$
- *Join*: t has exact two children t',t''; $X_t = X_{t'} = X_{t''}$

Given a tree decomposition we can easily compute a nice tree decomposition of the same width.

Example: A nice tree decomposition of our example argumentation framework.

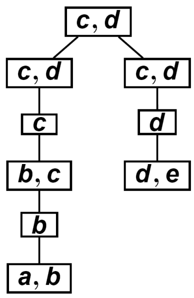


Figure: Nice Tree Decomposition of our example AF

Dynamic Programming (DP)

Given an AF and a nice tree decomposition we can use a dynamic programming algorithm.
Basic Ideas:

- Bottom-up traversal of the tree;
  computing a table for each bag b that
  - assigns values to the arguments in b, encoding if they are in the extension or not
  - stores information about the subtree rooted in b
- The results can be read off the root

A **bag extension** of a bag b is a subset of X_b.
A table for a bag b in the DP algorithm stores bag extensions and assign values $\#_b$ to these extension. These values encode the information about the sub tree rooted in b. Further there are labels on arguments in b that encode information about attacks against the arguments in this subtree.

Counting Stable Extensions

Ideas:

- In each bag b the values $\#_b$ encode the numbers of extensions on the subtree rooted in b, that may be part of a stable extension on the whole subtree.
- We drop out "subtree" extension with a conflict in it - this we can do "locally" as every attack is in at least one bag.
- We drop out extensions having undefeated arguments outside.We can't do this in one bag because we don't know if there already was an attack in previous bags or there will be an attack in future bags.
  ↪ We resolve this problem by labeling arguments that are undefeated - If an argument is marked as undefeated after considering the last incident attack we drop this extension.

Counting Stable Extensions

Following these ideas we get the following **algorithm**:

- **Leaf Nodes:**
  - compute all possible bag extensions
  - label arguments and test if the extension is cf
- **Forget Nodes:**
  - delete extensions where the forgotten argument is undefeated
  - delete the column of the forgotten argument
  - union rows representing the same extension (sum the values $\#_b$)
- **Insert Nodes:**
  - add column for the new argument
  - duplicate each extension - one version including the new argument and one that not
  - update labels and test if the extension is cf
- **Join Nodes:**
  - copy extension which are in both successor tables
  - multiply the values $\#_b$ from the successors
  - update labels, an argument is undefeated only if it is undefeated in both successor extensions
- **Root Node:**
  sum over all bag extensions without undefeated arguments to get the number of stable extensions

Theorem

Given an argumentation framework *F* together with a tree decomposition of width *t*, our DP algorithm computes the number of stable extensions in time *O(f(t)·|F|)* (assuming constant cost for arithmetics).

Example

If we use the DP algorithm to compute the number of stable extensions for our example AF we get the following tables:

c	*d*	$\#_0$
0	0	0
0	1	1
1	0′	0
1	1	0

c	*d*	$\#_1$
0′	0′	$\hat{1}$
0	1	1
1	0	1
1	1	0

c	$\#_2$
0′	$\hat{1}$
1	1

b	*c*	$\#_3$
0	0′	$\hat{1}$
0	1	1
1	0′	0
1	1	0

b	$\#_4$
0	1
1	0

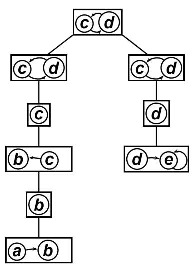
a	*b*	$\#_5$
0′	0′	$\hat{1}$
0′	1	$\hat{1}$
1	0	1
1	1	0

c	*d*	$\#_6$
0	0	0
0	1	1
1	0	0
1	1	0

d	$\#_7$
0	0
1	1

d	*e*	$\#_8$
0′	0′	$\hat{1}$
0	1	0
1	0	1
1	1	0

The sub-frameworks we can access in each bag:



legend:

- **1**: the argument is in the bag extension
- **0**: the argument is not in the bag extension
- **0'**: the argument is not defeated
- ***$\hat{n}$***: there is undefeated attack in the bag extension

Summary

- We have presented a DP-algorithm that counts stable extension in linear time for bounded tree-width.
- With similar techniques we get DP-algorithms for admissible and preferred extensions.
  (and further for stage, semi-stable, ideal)

Future work:

- Implementation of these DP-algorithms
- Evaluate the DP-algorithm's efficiency by comparing it with other systems