Focused search for arguments from propositional knowledge

Vasiliki Efstathiou and Anthony Hunter

Department of Computer Science
University College London

Contents

- Introduction: Framework for argumentation and motivation for efficient search for arguments
- The connection graph approach, definitions and algorithm demonstration
- Theoretical and experimental results
- Conclusions and further work

Framework for argumentation (Besnard & Hunter 2001)

- We can formalize argumentation using classical logic and adapt it in computational context
- We use $\Delta, \Phi, \ldots$ to denote sets of formulae, $\phi, \psi \ldots$ to denote formulae and $a, b, c \ldots$ to denote the propositional letters each formula consists of.
- In this framework an argument is a pair $\langle \Psi, \phi \rangle$ where Ψ is a set of formulae that minimally and consistently entails a formula ϕ. We call Ψ the **support** of the argument and ϕ the **claim** of the argument

Examples

Some arguments are

- $\langle \{\neg a, (d \vee e) \wedge f\}, \neg a \wedge (d \vee e) \rangle$
- $\langle \{(\neg a \vee b) \wedge c, \neg b \wedge d\}, \neg a \wedge c \rangle$
- $\langle \{\neg a\}, \neg a \rangle$
- $\langle \{\neg b \wedge d\}, d \rangle$

Motivation for efficient algorithms

- We want to automate the construction of arguments.
- This process is computationally expensive.
- Given a knowledgebase Δ, we want to find all the arguments for a formula ϕ.
- We use an automated theorem prover (ATP) to test for entailment and consistency
  - $\Psi \vdash \phi$?
  - $\Psi \nvdash \bot$?

Motivation for efficient algorithms

- We do not know which subsets of Δ to investigate. Testing arbitrary subsets of Δ can be prohibitely expensive. We explore an alternative way for locating the arguments for ϕ
- Our approach is to adapt the idea of connection graphs (R.Kowalski 1975) to reduce the search space for argumentation
- We use this in order to isolate a partition of the knowledgebase that contains the arguments for ϕ

Definitions

We start with a language of disjunctive clauses (disjunctions of 1 or more literals) We define the following relations on clauses

- The Disjuncts relation takes a clause and returns the set of disjuncts in the clause. Disjuncts($\beta_1 \vee .. \vee \beta_n$) $= \{\beta_1, .., \beta_n\}$
- Let ϕ and ψ be clauses. Then, Preattacks(ϕ, ψ) is $\{\beta \mid \beta \in$ Disjuncts(ϕ) and $\neg\beta \in$ Disjuncts(ψ)$\}$
- Let ϕ and ψ be clauses. If Preattacks(ϕ, ψ) $= \{\beta\}$ for some β, then Attacks(ϕ, ψ) $= \beta$ otherwise Attacks(ϕ, ψ) $=$ *null*

Examples

- Preattacks
  - Preattacks$(a \vee \neg b \vee \neg c \vee d, a \vee b \vee \neg d \vee e) = \{\neg b, d\}$
  - Preattacks$(a \vee b \vee \neg d \vee e, a \vee \neg b \vee \neg c \vee d) = \{b, \neg d\}$
  - Preattacks$(a \vee b \vee \neg d, a \vee b \vee c) = \emptyset$
  - Preattacks$(a \vee b \vee \neg d, a \vee b \vee d) = \{\neg d\}$
  - Preattacks$(a \vee b \vee \neg d, e \vee c \vee d) = \{\neg d\}$
- Attacks
  - Attacks$(a \vee \neg b \vee \neg c \vee d, a \vee b \vee \neg d \vee e) = null$
  - Attacks$(a \vee b \vee \neg d \vee e, a \vee \neg b \vee \neg c \vee d) = null$
  - Attacks$(a \vee b \vee \neg d, a \vee b \vee c) = null$
  - Attacks$(a \vee b \vee \neg d, a \vee b \vee d) = \neg d$
  - Attacks$(a \vee b \vee \neg d, e \vee c \vee d) = \neg d$

Connection graphs

- We use Preattacks and Attacks relations on a set of clauses Δ to define different types of graphs
- The nodes of the graphs are elements from Δ
- Arcs exists between nodes which contain contradictory literals
- The number of contradictory literals between pairs of nodes allows for different relations to hold between those nodes, which in turn identify different kinds of graphs

The connection Graph

- The **connection graph** is the graph whose arcs are identified by the Preattacks relation

$$
\begin{array}{ccccccccccc}
\neg b & & \neg c \vee \neg g & & \neg c & \neg h \vee l & — & \neg l \vee \neg k & & n \vee m \vee \neg q & \\
| & & & \diagdown & | & | & & | & | & | & | \\
a \vee b & — & \neg b \vee d & & c \vee g & h \vee \neg l & — & l \vee k & \neg n & \neg m & q \\
| & & | & & | & & & | & & | & \\
\neg a \vee d & — & \neg d & & \neg g & f \vee p & & \neg k & & m &
\end{array}
$$

The attack graph

- The **attack graph** is the graph whose arcs are indentified by the Attacks relation

$$
\begin{array}{cccccccccc}
\neg b & & \neg c \vee \neg g & \neg c & \neg h \vee l & — & \neg l \vee \neg k & \multicolumn{3}{c}{n \vee m \vee \neg q} \\
\mid & & & \mid & & & & \mid & \mid & \mid \\
a \vee b & — & \neg b \vee d & c \vee g & h \vee \neg l & — & l \vee k & \neg n & \neg m & q \\
\mid & & \mid & \mid & & & \mid & & \mid & \\
\neg a \vee d & — & \neg d & \neg g & f \vee p & & \neg k & & m &
\end{array}
$$

The closed graph

- The **closed graph** characterizes the attack graph in terms of connectivity Clauses containing ‘unlinked literals’ are excluded

$$\begin{tikzcd}
\neg b \arrow[d, dash] & & \neg c \arrow[d, dash] & & n \vee m \vee \neg q \arrow[d, dash] \arrow[dl, dash] \arrow[dr, dash] & \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] & \neg n & \neg m \arrow[d, dash] & q \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g & & m &
\end{tikzcd}$$

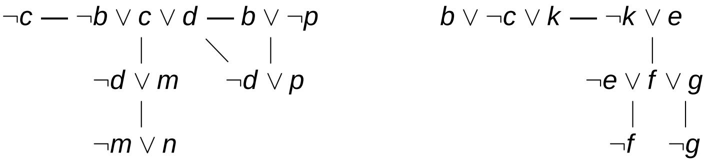
The focal graph

- The **focal graph** is identified by a clause ϕ from Δ, which we call the **epicentre**. The focal graph of ϕ in Δ is the component of the closed graph that contains ϕ
- The following is the focal graph of $\neg b$ in Δ and of $a \vee b$ in Δ and of $\neg b \vee d$ in Δ etc...

$$
\begin{array}{ccc}
\neg b & & \\
| & & \\
a \vee b & — & \neg b \vee d \\
| & & | \\
\neg a \vee d & — & \neg d
\end{array}
$$

Algorithm for the focal graph

- Given a clause ϕ we can find the focal graph of ϕ in Δ by depth-first search of the attack graph for Δ
- The following is the attack graph for a set of clauses Δ. We want to find the focal graph of $\neg c$ in Δ



Algorithm for the focal graph

- Initially all the nodes are considered to be allowed candidates for the focal graph and the unsuitable ones will be rejected while walking over the graph
- First locate $\neg c$ in the attack graph for Δ

$$
\begin{tikzcd}
{\color{blue}\neg c} \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
& \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
& \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
& \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
& \neg f & \neg g
\end{tikzcd}
$$

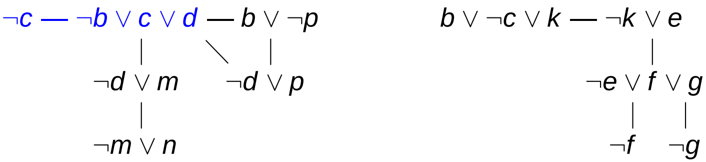
Algorithm for the focal graph

- follow one of the paths that start from $\neg c$

$$
\begin{tikzcd}
\neg c \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
& \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
& \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
& \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
& \neg f & \neg g
\end{tikzcd}
$$

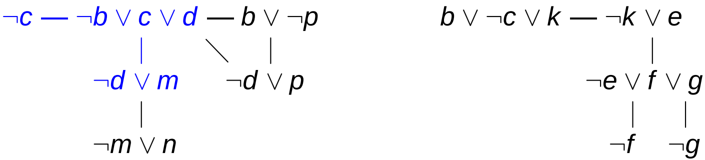
Algorithm for the focal graph

- follow one of the paths that start from $\neg c$
- test if the current node is connected i.e. if all its disjuncts correspond to a link in the graph



Algorithm for the focal graph

- if it is, follow one of the paths that continue from this node



Algorithm for the focal graph

- if it is, follow one of the paths that continue from this node
- test if the current node is connected

$$
\begin{tikzcd}
\neg c \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
 & \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
 & \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
 & \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
 & \neg f & \neg g
\end{tikzcd}
$$

Algorithm for the focal graph

- if it is, follow one of the paths that continue from this node
- continue in the same way for every newly created node

$$
\begin{tikzcd}
\neg c \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
& \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
& \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
& \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
& \neg f & \neg g
\end{tikzcd}
$$

Algorithm for the focal graph

- if a node which is not connected is found then mark it as rejected and backtrack

$$
\begin{tikzcd}
\neg c \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
 & \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
 & \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
 & \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
 & \neg f & \neg g
\end{tikzcd}
$$

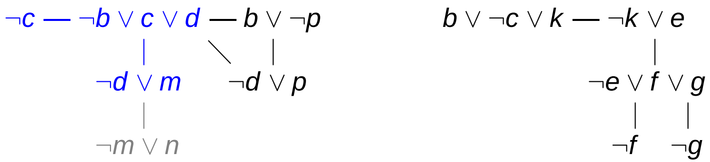
Algorithm for the focal graph

- if a node which is not connected is found then mark it as rejected and backtrack

$$
\begin{tikzcd}[arrows=-]
\neg c \arrow[r] & \neg b \vee c \vee d \arrow[r] \arrow[d] \arrow[dr] & b \vee \neg p \arrow[d] \\
& \neg d \vee m \arrow[d] & \neg d \vee p \\
& \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}[arrows=-]
b \vee \neg c \vee k \arrow[r] & \neg k \vee e \arrow[d] & \\
& \neg e \vee f \vee g \arrow[d] \arrow[dr] & \\
& \neg f & \neg g
\end{tikzcd}
$$

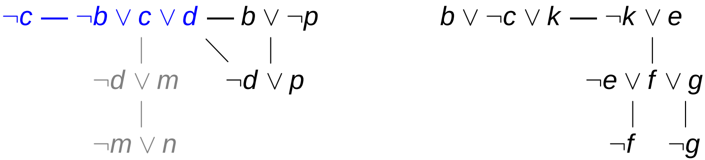
Algorithm for the focal graph

- test if the nodes adjacent to the node rejected last remain connected



Algorithm for the focal graph

- test if the nodes adjacent to the node rejected last remain connected
- if they do not, mark them as rejected and continue backtracking



Algorithm for the focal graph

- test if the nodes adjacent to the node rejected last remain connected

$$
\begin{tikzcd}
\neg c \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
 & \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
 & \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
 & \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
 & \neg f & \neg g
\end{tikzcd}
$$

Algorithm for the focal graph

- test if the nodes adjacent to the node rejected last remain connected
- if they do, continue from that point, by following one of the the paths to the nodes that have not been visited yet

$\neg c$ — $\neg b \vee c \vee d$ — $b \vee \neg p$

$\neg d \vee m$ $\neg d \vee p$

$\neg m \vee n$

$b \vee \neg c \vee k$ — $\neg k \vee e$

$\neg e \vee f \vee g$

$\neg f$ $\neg g$

Algorithm for the focal graph

- and continue in the same way. Only the component of the graph that is linked to $\neg c$ is being searched
- The visited non-rejected nodes of the graph correspond to the focal graph of $\neg c$ in Δ

$$\begin{tikzcd}
\neg c \arrow[r, dash] & \neg b \vee c \vee d \arrow[r, dash] \arrow[d, dash] \arrow[dr, dash] & b \vee \neg p \arrow[d, dash] \\
& \neg d \vee m \arrow[d, dash] & \neg d \vee p \\
& \neg m \vee n &
\end{tikzcd}
\qquad
\begin{tikzcd}
b \vee \neg c \vee k \arrow[r, dash] & \neg k \vee e \arrow[d, dash] & \\
& \neg e \vee f \vee g \arrow[d, dash] \arrow[dr, dash] & \\
& \neg f & \neg g
\end{tikzcd}$$

Why is the focal graph useful?

- The focal graph can be used to reduce the search space for argumentation for knowledgebases and queries in CNF
- Let Conjuncts(ϕ) be the set of a clauses a formula ϕ in CNF consists of
- Let SetConjuncts(Ψ) be the set of clauses all the formulae from Ψ consist of

Why is the focal graph useful?

- Let ϕ be claim for which want to find arguments from Ψ, where Ψ be a set of formulae in CNF
- Let $\overline{\phi} = \phi_1 \wedge \ldots \wedge \phi_n$ be the CNF of the negation of claim ϕ
- The focal graphs of each ϕ_i in SetConjuncts($\Psi \cup \{\overline{\phi}\}$) indicate the part of Ψ which contains the arguments for ϕ and hence help excluding some other which is not relevant
- We call the graph consisisting of these focal graphs the **query graph** of ϕ in Ψ

The query graph

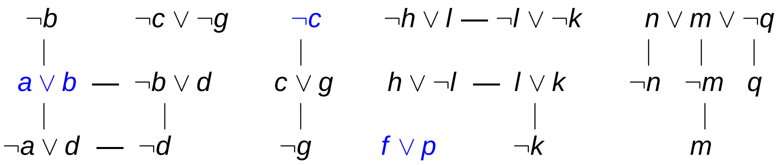
- Let Ψ be set of formulae in CNF

$$\Psi = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), (h \vee \neg l), \neg m \wedge \neg n, m \wedge q\}$$

- Let ϕ be a claim for an argument with $\overline{\phi} = (a \vee b) \wedge (f \vee p) \wedge \neg c$

The query graph

- Then, SetConjuncts$(\Psi \cup \{\overline{\phi}\})$ is Δ from the first example with the following attack graph where the conjuncts of $\overline{\phi} = (a \vee b) \wedge (f \vee p) \wedge \neg c$ are marked



The query graph

- and so the following is the query graph of ϕ in Ψ
- We want to find arguments for ϕ from Ψ and not from SetConjuncts$(\Psi \cup \{\overline{\psi}\})$

$$
\begin{tikzcd}
\neg b \arrow[d, dash] & & \neg c \arrow[d, dash] \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g
\end{tikzcd}
$$

The query graph

- The query graph indicates which subsets of Ψ are useful - find which formula from Ψ each node relates to

$$\Psi = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), q \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d, l \vee k, m \wedge (\neg l \vee \neg k), \neg k \wedge (n \vee m \vee \neg q), (h \vee \neg l), \neg m \wedge \neg n, m \wedge q\}$$

$$\begin{tikzcd}
\neg b \arrow[d, dash] & & \neg c \arrow[d, dash] \\
a \vee b \arrow[r, dash] \arrow[d, dash] & \neg b \vee d \arrow[d, dash] & c \vee g \arrow[d, dash] \\
\neg a \vee d \arrow[r, dash] & \neg d & \neg g
\end{tikzcd}$$

Supportbase

- Use this part of the knowledgebase to look for arguments instead of searching the initial knowledgebase

$$\Psi' = \{(\neg a \vee d) \wedge (\neg c \vee \neg g), \neg d, \neg d \wedge (\neg h \vee l), c \vee g, \neg g, \neg b, \neg b \vee d\}$$

- We call Ψ' the **Supportbase** for Ψ and ϕ. If $\langle \Gamma, \phi \rangle$ is an argument then Γ is a subset of the Supportbase
- Supportbase$(\Psi, \phi) \subseteq \Psi$

Experiment

- We tested the focal graph algorithm for sets of randomly generated clauses
- These sets were of fixed cardinality (600 clauses) and they contained 3-place clauses (**rules**) and 1-place clauses literals (**facts**)
- The evaluation was based on the size of the focal graph of an epicentre ϕ in a set of clauses Δ

Experiment

- 2 dimensions were considered:
  - **clauses-to-variables ratio**
  - **facts-to-rules ratio**
- e.g.knowledgebase with 600 elements:
  - 150 facts + 450 rules, facts-to-rules=1/3
  - constructed with 100 propositional letters: clauses-to-variables ratio = 6 = 600/100
- 1000 repetitions of the algorithm for each fixed clauses-to-variables and facts-to-rules ratio
- Highest average focal graph size of an epicentre ϕ in a set of clauses Δ with 600 distinct elements is $\sim$ 344 (57 % of the initial knowledgebase)

Experimental data

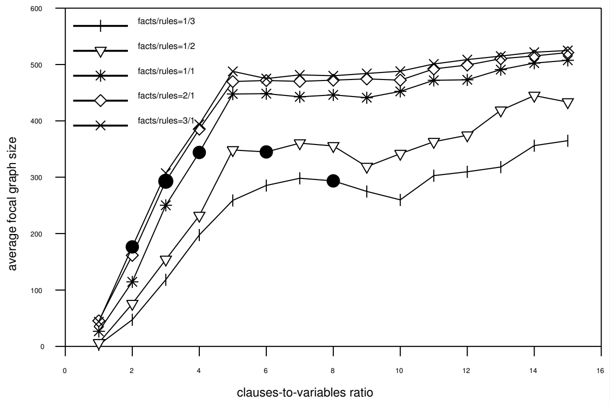


Figure: Focal graph size variation with the clauses-to-variables ratio

Conclusions

- In this talk we presented the theoretical background of algorithms that can make argumentation more effective in terms of computational cost by reducing the search space for arguments
- We presented some empirical results on how this proposal works with random data
- Further work in this framework involves
  - Algorithms for finding arguments with literals for claims and sets of clauses for supports (FOIKS ’08)
  - Generalization to subsets of first order logic
  - Experimenatation with knowledgebases of real data