Solving disjunctive fuzzy answer set programs

LPNMR 2015

M Mushthofa, S Schockaert, M De Cock

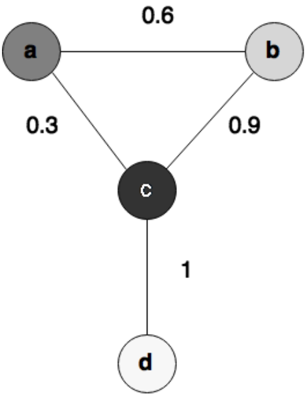
Fuzzy Answer Set Programming

- Fuzzy ASP = Fuzzy logic + ASP
- Allows graded truth values of atoms (usually in [0, 1])
- Extends the operators $\wedge$, $\vee$, **not** and $\leftarrow$ to fuzzy domain, e.g. using Łukasiewicz semantics
- Interpretations are functions $I : \mathcal{B}_P \mapsto [0, 1]$ extended to expressions as follows:
  - $I(a \otimes b) = \max(I(a) + I(b) - 1, 0)$
  - $I(a \oplus b) = \min(I(a) + I(b), 1)$
  - $I(\textbf{not } a) = 1 - I(a)$
  - $I(a \leftarrow b) = 1$ iff $I(a) \geq I(b)$

FASP Semantics

- I is a model of P iff $I(r) = 1, \forall r \in P$
- I is an answer set of positive P iff it is a minimal model of P
- **[Extended Gelfond-Lifschitz reduct]**: P^I is a positive program obtained by replacing every expression **not** a with the constant $\overline{I(\textbf{not } a)}$
- I is an answer set of P iff it is a minimal model of P^I

Fuzzy graph coloring



FASP Encoding:

$$node(a).\dots node(d).$$
$$edge(a,b) \leftarrow \overline{0.6}$$
$$edge(a,c) \leftarrow \overline{0.3}$$
$$\vdots$$
$$black(X) \oplus white(X) \leftarrow node(X)$$
$$edge(X,Y) \leftarrow edge(Y,X)$$
$$\overline{0} \leftarrow black(X) \otimes black(Y) \otimes edge(X,Y)$$
$$\overline{0} \leftarrow white(X) \otimes white(Y) \otimes edge(X,Y)$$

Answer set(s):

$$\{black(a)[0.8], black(b)[0.6], \dots\}$$

A solver for FASP

While many solvers exist for classical ASP, e.g.:

- **clasp**
- **DLV**
- **LP2SAT**
- **WASP**

not many prototype systems exist for FASP:

- (Alviano & Peñaloza, 2014) proposed a method for FASP evaluation using answer set approximation operators
- (Mushthofa, Schockaert & De Cock, 2014) developed a FASP solver using a translation to classical ASP
- **Problem**: cannot handle **disjunctive** programs correctly!

Disjunctive rules

Classical ASP:

$$a \vee b \leftarrow c$$
$$c \leftarrow$$

Answer sets: $\{a, c\}, \{b, c\}$

Fuzzy ASP:

$$a \oplus b \leftarrow c$$
$$c \leftarrow \overline{0.8}$$

Answer sets, e.g.:
$\{a[0.7], b[0.1], c[0.8]\}$
$\{a[0.5], b[0.3], c[0.8]\}$

Disjunction in (F)ASP can:

- increase the expressivity of the language (from NP-Complete to Σ_2^P)
- allow for more intuitive encoding of many classes of problems

Shifting method for classical ASP

The disjunctive classical ASP program:

$$a \vee b \leftarrow c$$
$$c \leftarrow$$

can be rewritten into the **non-disjunctive** (normal) program:

$$a \leftarrow c \wedge \textbf{not } b$$
$$b \leftarrow c \wedge \textbf{not } a$$
$$c \leftarrow$$

using the so-called **shift operation**

Head cycle free programs

Shifting only preserves semantics for head cycle free (HCF) programs, i.e., programs where there are no cycle of positive dependencies between head propositions. For example:

$$
\begin{aligned}
a \vee b &\leftarrow c \\
a &\leftarrow b \\
b &\leftarrow a \\
c &\leftarrow
\end{aligned}
$$

has only one answer set $\{a, b, c\}$ and is **not** equivalent to its shifted version

How about Fuzzy ASP?

Some non-HCF FASP programs **can** be shifted to obtain an equivalent normal program. For example:

$$a \oplus b \leftarrow \overline{1} \qquad a \leftarrow b \qquad b \leftarrow a$$

is equivalent to

$$a \leftarrow \textbf{not } b \qquad\qquad a \leftarrow b$$
$$b \leftarrow \textbf{not } a \qquad\qquad b \leftarrow a$$

and both have one answer set, namely $\{a[0.5], b[0.5]\}$

Motivating questions

- How do we characterize the class of FASP programs that can be shifted to obtain normal programs (and allow for a more efficient evaluation)?
- How can we evaluate disjunctive FASP programs that cannot be shifted?

A simple example

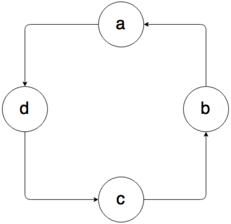
- The following FASP program cannot be shifted to obtain an equivalent normal program

$$a \oplus b \leftarrow \overline{1} \qquad a \leftarrow a \oplus a$$
$$a \leftarrow b \qquad b \leftarrow a$$

- The rule $a \leftarrow a \oplus a$ causes the truth value of a to "saturate" (Booleanized)

Self-Reinforcing Cycles

- **Self-Reinforcing Cycles**: a cycle of positive dependencies between propositions s.t. there is a rule involved in the cycle containing a disjunction in the body



- Potentially causing a saturation to the propositions involved

$$a \leftarrow b \otimes p \qquad b \leftarrow c$$
$$c \leftarrow d \oplus q \qquad d \leftarrow a \otimes r$$

Self Reinforcing Cycle Free (SRCF) programs: no self-reinforcing cycles involving propositions occurring in a disjunction in the head of a rule.

Theorem

Let $\mathcal{P}_1 = \mathcal{P} \cup \{a \oplus b \leftarrow c\}$ *be any SRCF program. Then, an interpretation* I *is an answer set of* $\mathcal{P}_1$ *iff it is also an answer set of* $\mathcal{P}_2 = \mathcal{P} \cup \{a \leftarrow c \otimes \textbf{not } b, b \leftarrow c \otimes \textbf{not } a\}$*.*

- All strict FASP (no disjunctions in the body) can be *shifted* to normal programs
- *HCF* $\subset$ *SRCF*: a large class of disjunctive FASP programs can be rewritten into normal programs

Non-SRCF programs need extra minimality checks

- The solver developed in [Mushthofa *et al*, ECAI2014] can generate *candidate/potential* answer set(s) for disjunctive FASP programs, e.g.:

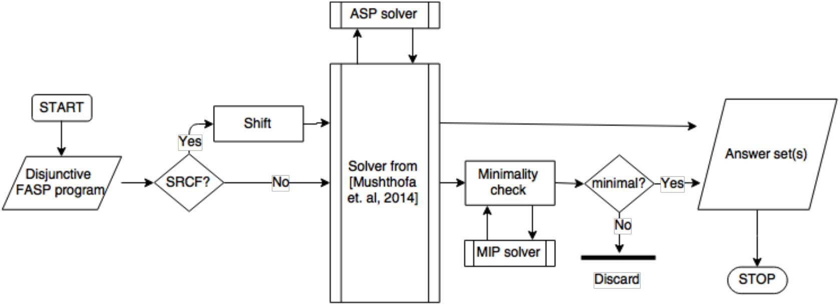
$$a \oplus b \leftarrow \overline{1} \qquad a \leftarrow b \qquad b \leftarrow a$$

  - For $k = 1$, we get candidate answer set $\{a[1], b[1]\}$
  - For $k = 2$, we get candidate answer set $\{a[0.5], b[0.5]\}$
- Only **minimal** models are considered as answer sets: check minimality!

Minimality check using Mixed Integer Programming

- **Problem**: Given a program $\mathcal{P}$ and a possible answer set I, check whether I is a minimal model of $\mathcal{P}^I$
- Express the problem as a Mixed Integer Programming (MIP) optimization problem:
  - Express the program $\mathcal{P}^I$ as MIP constraints
  - Set objective function $=$ the sum of the truth values of the propositions
  - If the solution returned $= I$, then I is minimal

Overall framework

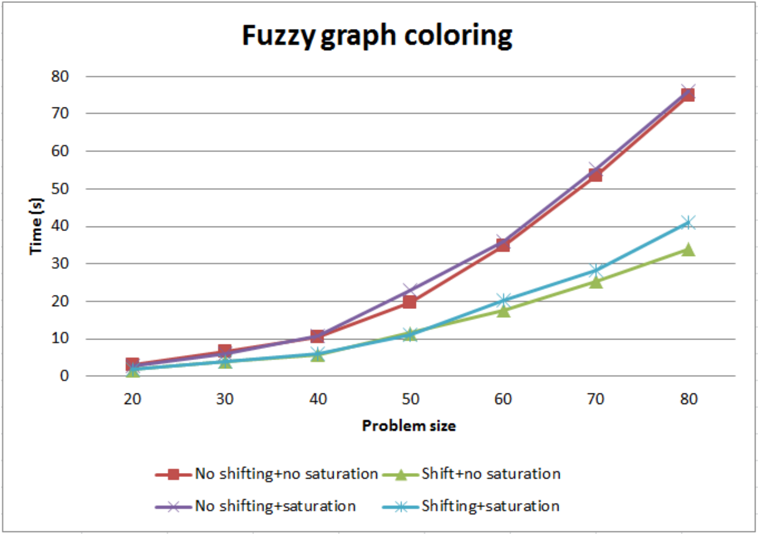


Implementation

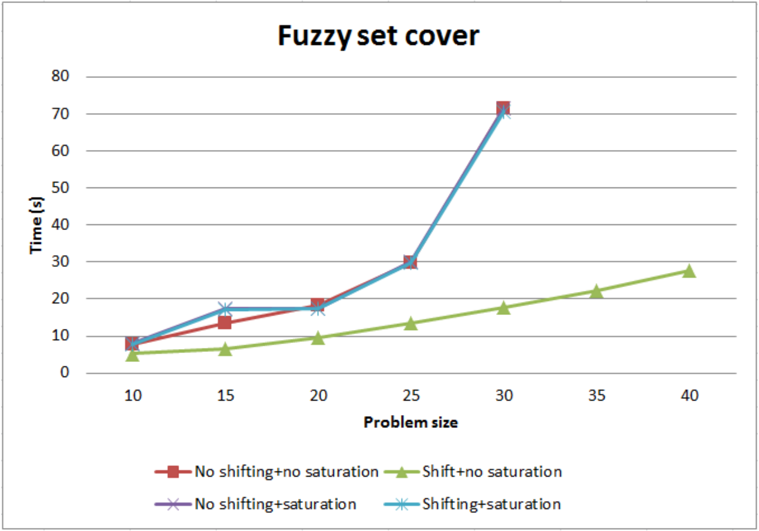
- Written on top of the previous solver [Mushthofa *et al*, ECAI2014]
- Uses clasp as external ASP solver and Cbc+GLPK as MIP solver
- Perform program modularity analysis and decomposition to further increase efficiency
- Available at `https://github.com/mushthofa/ffasp`

Benchmark

- Compare the performance of the solver when SRCF detection and shifting is applied vs not applied
- Benchmark problems: fuzzy graph coloring and fuzzy set covering
- Generate random instances (with varying sizes), with and without random saturation rules
- Measure running times


Fuzzy graph coloring
Time (s)
0
10
20
30
40
50
60
70
80
20
30
40
50
60
70
80
Problem size
No shifting+no saturation
Shift+no saturation
No shifting+saturation
Shifting+saturation

Implementation & Benchmark



Conclusions

- We identified a large class of disjunctive FASP programs (called SRCF programs) that can be rewritten into normal programs (for efficient evaluation) via shifting operation
- We devised a mechanism to handle evaluation non-SRCF programs (via minimality checks using MIP)
- We implemented the method on top of our previous solver to allow evaluation of disjunctive FASP programs
- We performed a benchmark of our method/implementation on simple benchmark problems

Thank you

Mushthofa
Computational Web Intelligence
Dept. of Applied Mathematics, Statistics and Informatics
Ghent University, Belgium

Supported by: UGent MRP project Nucleotides2Network

Mushthofa.Mushthofa@UGent.be