

ENIGMA: Efficient learNing-based Inference Guiding MACHine

Jan Jakubův¹ Josef Urban¹

¹Czech Technical University in Prague

TU Innsbruck seminar, 6th Dec 2017
(presenting CICM'17 ENIGMA paper)

Outline

- 1 Motivation
- 2 Automated Theorem Proving
- 3 Enigma Models
- 4 Conclusion

Using Automated Theorem Provers

- Solve problems in First-Order Logic
- Built-in automated strategy selection
- E Prover:

```
$ eprover --auto-schedule problem.tptp
```

- Vampire:

```
$ vampire --mode casc problem.tptp
```

No Success?

```
$ eprover --auto-schedule problem.tptp
...
# Failure: Resource limit exceeded (time)
# SZS status ResourceOut
eprover: CPU time limit exceeded, terminating
```

Try your own strategy!

```
$ eprover --definitional-cnf=24 --oriented-simul-paramod \  
--forward-context-sr --destructive-er-aggressive \  
--destructive-er --prefer-initial-clauses -tAuto \  
-Garity -F1 -WSelectMaxLComplexAvoidPosPred \  
-H(1*ConjectureRelativeTermWeight(PreferProcessed,1,1,1,...), \  
  1*ConjectureTermPrefixWeight(SimulateSOS,1,3,0.5,10,...), \  
  34*ConjectureRelativeTermWeight(DeferSOS,1,3,0.2,10,...)) \  
problem.tptp  
  
# Proof found!  
# SZS status Theorem
```

Our Task

- Invent targeted strategies for E
- ... specific for a given benchmark set
- ... using machine learning methods (BliStrTune).
- (Recently also for Vampire – EmpireTune)

Outline

- 1 Motivation
- 2 Automated Theorem Proving**
- 3 Enigma Models
- 4 Conclusion

Given Clause Loop Paradigm

Problem representation

- first order clauses (ex. " $x = 0 \vee \neg P(f(x, x))$ ")
- posed for proof by contradiction

Given an initial set C of clauses and a set of inference rules, find a derivation of the *empty clause* (for example, by the resolution of clauses with conflicting literals L and $\neg L$).

Basic loop

```
Proc = {}
Unproc = all available clauses
while (no proof found)
{
    select a given clause C from Unproc
    move C from Unproc to Proc
    apply inference rules to C and Proc
    put inferred clauses to Unproc
}
```

Clause Selection Methods

Selection mechanisms

- symbol count (weighting)
- user-defined weighting patterns
- attribute-based restrictions (e.g., set of support)
- model-based selection (semantic guidance)
- subsumption-based selection (e.g., hints)
- selection by learned classifier

User-defined rules or scripts can be used to specify combinations of these mechanisms.

Outline


- 1 Motivation
- 2 Automated Theorem Proving
- 3 Enigma Models**
- 4 Conclusion

Enigma Models

Idea: Lets use fast linear classifier to guide given clause selection!

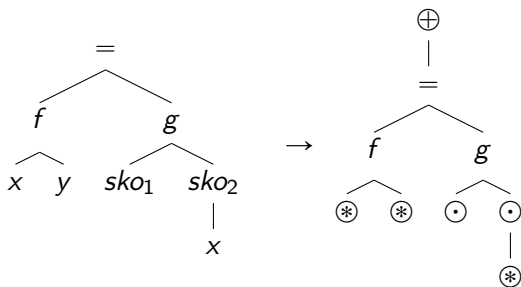
ENIGMA: Efficient learNing-based Inference Guiding MACHine

- LIBLINEAR: open source library¹
- **input:** positive and negative examples (float vectors)
- **output:** model (\sim a vector of weights)
- **evaluation** of a generic vector: dot product with the model

¹<http://www.csie.ntu.edu.tw/~cjlin/liblinear/> 

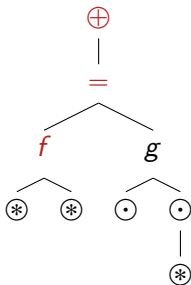
Clauses as Feature Vectors

Consider the literal as a tree:



Clauses as Feature Vectors

Count descending paths of length 3 (our features):



$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

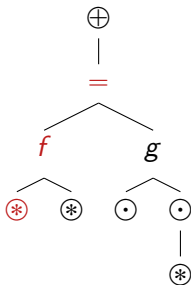
$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

Clauses as Feature Vectors

Count descending paths of length 3 (our features):



$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

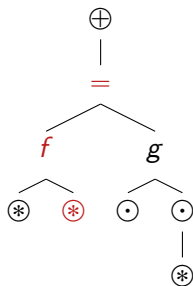
$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

Clauses as Feature Vectors

Count descending paths of length 3 (our features):



$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

Clauses as Feature Vectors

Number features and construct a (sparse) vector:
 (each feature has a fixed position, $|features| = |\Sigma|^3$)

$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

$$(0, \dots, 1, \dots, 1, \dots, 2, \dots, \\ 2, \dots, 1, \dots, 0, \dots)$$

Clauses as Feature Vectors

Number features and construct a (sparse) vector:
 (each feature has a fixed position, $|features| = |\Sigma|^3$)

$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

$$(0, \dots, 1, \dots, 1, \dots, 2, \dots, \\ 2, \dots, 1, \dots, 0, \dots)$$

Clauses as Feature Vectors

Number features and construct a (sparse) vector:
 (each feature has a fixed position, $|features| = |\Sigma|^3$)

$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

$$(=, f, \odot) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, \ast) \mapsto 1$$

$$(0, \dots, 1, \dots, 1, \dots, 2, \dots, \\ 2, \dots, 1, \dots, 0, \dots)$$

Clauses as Feature Vectors

Number features and construct a (sparse) vector:
 (each feature has a fixed position, $|features| = |\Sigma|^3$)

$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

$$(0, \dots, 1, \dots, 1, \dots, 2, \dots, \\ 2, \dots, 1, \dots, 0, \dots)$$

Clauses as Feature Vectors

Number features and construct a (sparse) vector:
 (each feature has a fixed position, $|features| = |\Sigma|^3$)

$$(\oplus, =, f) \mapsto 1$$

$$(\oplus, =, g) \mapsto 1$$

$$(=, f, *) \mapsto 2$$

$$(=, g, \odot) \mapsto 2$$

$$(g, \odot, *) \mapsto 1$$

$$(0, \dots, 1, \dots, 1, \dots, 2, \dots, \\ 2, \dots, 1, \dots, 0, \dots)$$

ENIGMA inside E Prover

- E Prover uses clause weight functions
- ... the clause with the smallest weight is selected
- Implemented new weight function `Enigma`
- ... parametrized by a given model (argument)
- `Enigma` can be composed with other weight functions:

```
(100*Enigma(modelX, args)
 5*OtherWeight(args))
```

Experiments Setting

- AIM problems from CASC (LTB category)
- ... 1020 training problems, 200 testing problems
- advantages (simplifications):
- ... different conjectures in the same theory
- ... small number of symbols (8+4)
- ... symbols are used consistently

Manual Evaluation


- 1 Use single E strategy S to solve 239 training problems (in 30s)
- 2 Extract training examples from solved problems:
positive: processed clauses used in proofs
negative: processed but not used in proofs
- 3 Create Enigma model M
- 4 Combine M with S in different ways (S_1, \dots, S_{17})
(M alone or plugged into S ; different frequencies)

Manual Evaluation Results

- Original S (without Enigma) solves 22 problems in 180s
- Best S_i (with Enigma) solves 41 problems
- Other S_i 's? ($3 \times < 22$, $5 \times \geq 40$)
- All S_i 's together solves 52 (in $17 * 180$ s)
- Only 3 S_i 's covers 52 solved problems
- (for comparison: Vampire solves 47 in $3 * 180$ s)

Automated Strategy Development

- BliStrTune²: System for developing E strategies
 - provide set of benchmarks
 - run it for few days (7) on few cores (32)
 - collect new E Prover strategies
- EnigmaTune = BliStrTune + Enigma
 - develop Enigma models on the way
 - incorporate Enigma models into other strategies

²<http://github.com/ai4reason/BliStrTune> 

Preliminary Results (work in progress)

- Select strategies best-performing on training problems
- Run them in sequence diving the time limit
- Compare with E, Vampire, Prover9 in 300 seconds

prover	solved	SOTA+
E 1.9.1 (EnigmaTune)	77	+37%
E 1.9.1 (BliStrTune)	74	+32%
Prover9 (Bob's mode)	56	+0%
Vampire 4.0 (CASC)	45	-19%
E 1.9.1 (auto-schedule)	31	-44%

Outline

- 1 Motivation
- 2 Automated Theorem Proving
- 3 Enigma Models
- 4 Conclusion**

Future Work and Challenges

- Better ways to incorporate BliStrTune with Enigma
- Generalize to different theories
- Need training on “similar” problems
- Deal with large theory signatures
- Deal with different term orderings, etc.
- Deal with inconsistent symbols (e.g. TPTP)

Thank you

Questions?