Learning Reasoning Components

AKA Learning Theorem Proving Components

Karel Chvalovský[1] Jan Jakubův[1,2] Miroslav Olšák[2] Josef Urban[1]

[1]Czech Technical University in Prague [2]University of Innsbruck

Introduction

Aim improve automated theorem proving in classical FOL with equality.

- The state-of-the-art provers, like Vampire and E, are mostly based on the superposition calculus.
- Proof search is usually controlled by the saturation loop.
- In this loop we want to select the right clause from all unprocessed clauses.

Means improve this selection process.

Superposition prover

the problem of provability in classical FOL with equality

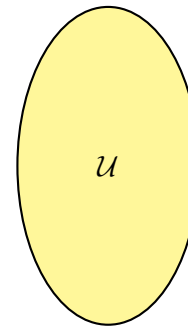
the problem of deriving the contradiction from a set of clauses

The superposition calculus restricts the space of derived clauses
(and still maintains the refutational completeness).

Saturation loop (a simplified picture)

A way how to organize proof search. We split the derived clauses into two sets

- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.

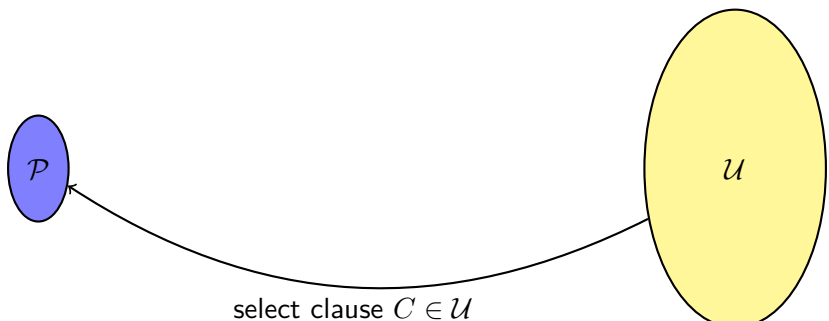


We start with the input clauses in $\mathcal{U}$ and $\mathcal{P} = \emptyset$.

Saturation loop (a simplified picture)

A way how to organize proof search. We split the derived clauses into two sets

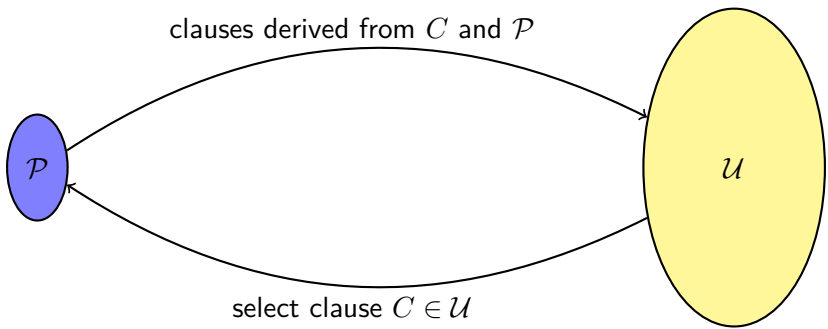
- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.



Saturation loop (a simplified picture)

A way how to organize proof search. We split the derived clauses into two sets

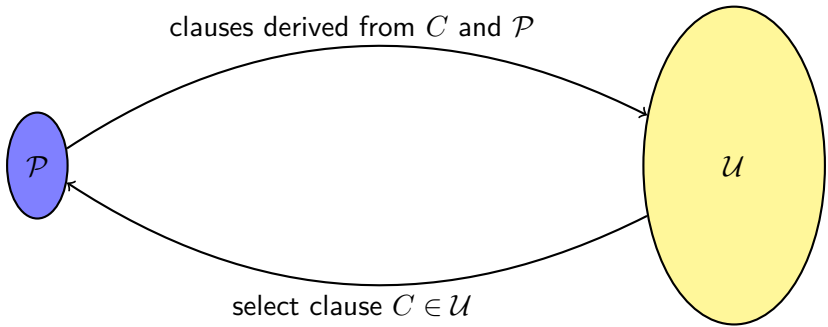
- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.



Saturation loop (a simplified picture)

A way how to organize proof search. We split the derived clauses into two sets

- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.



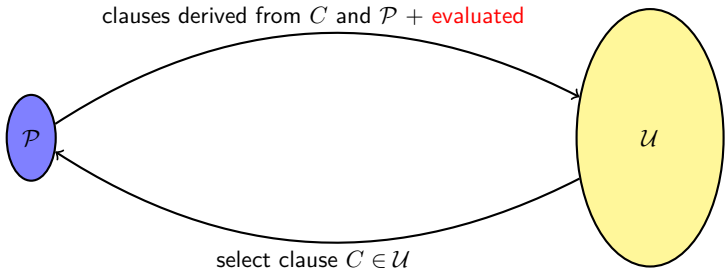
It holds that "all" the important consequences of clauses in $\mathcal{P}$ are in $\mathcal{P} \cup \mathcal{U}$.

How to select a (given) clause? Traditional approach

For example, all clauses are evaluated when added to $\mathcal{U}$ and we pick clauses, in a given ratio, say 1:10, by

age prefer clauses with a smaller derivational depth, and

weight prefer (shorter) clauses containing fewer symbols.



Humans do not scale very well when it comes to developing new heuristics, but there are more and more proofs available... $\Rightarrow$ Use statistical methods!

ENIGMA: Machine Learning Clause Selection

- We can extract training examples from successful proofs

  positive examples clauses from $\mathcal{P}$ that are used in the proof, and
  negative examples clauses from $\mathcal{P}$ that are not used in the proof.
- We can train a machine learning model based on them and use it to evaluate clauses from $\mathcal{U}$.

the theorem prover E
+
ML clause selection
=
Efficient learNing-based Inference Guiding MAchine (ENIGMA)

ENIGMA variants

There are many ENIGMA variants available, for example,

- it is possible to extract many types of features from clauses
  - age, weight, ...
  - descending paths of length 3,
  - ⋮
- it is possible to use various machine learning models
  - decision trees,
  - neural networks,
  - ⋮

Note that we want to be clever, but we have to be also fast, because $\mathcal{U}$ can be huge.

A problem with older ENIGMAs

The evaluation of a clause C only depends on

- the clause C itself (and the features extracted from it) and
- the conjecture we want to prove.

Advantage

- It is fast and easy to train and evaluate.

Disadvantage

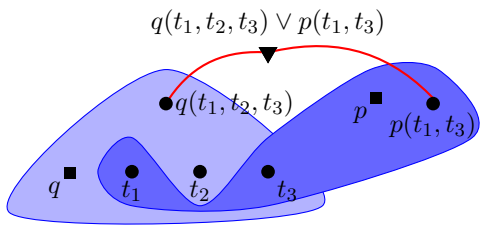
- The evaluation does not depend on the context.

A set of clauses is structured data. $\Rightarrow$ Use Graph Neural Networks (GNNs).

Graph Neural Network (GNN) used in ENIGMA

A hypergraph is produced for a set of clauses with

- three types of vertices and
- two types of (hyper)edges.



Interpretation

node embedding

(hyper)edge message-passing between nodes

GNN-ENIGMA

- The evaluation is invariant under the renaming of symbols; only connections matter not the actual names.
- The evaluation of a clause C depends on
  - the clause C,
  - the conjecture we want to prove, and
  - the context of other clauses.

Context

We usually use

- the first n clauses in $\mathcal{P}$ as the context,
- moreover, we do not evaluate individual clauses, but evaluate them in batches.

Mizar benchmark

- 57880 problems extracted from the Mizar Mathematical Library (1148 articles)
- ATPs can solve
  - 25.86% by a single good strategy from E (in 10s),
  - 48.10% by Vampire (in 300s),
  - 50.32% by various ENIGMAs (in 2019),
  - 65.65% by various ENIGMAs (in 2020), and
  - 75.53% combining various approaches, details available at

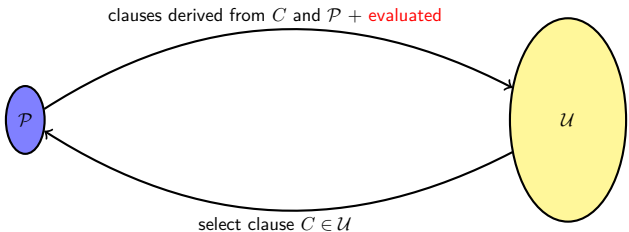
`https://github.com/ai4reason/ATP_Proofs`

A problem with the saturation loop

Observation

The evaluation of a clause is heavily influenced by the context. Hence it should change as new clauses are derived.

Moreover, there is, for example, no way how to get rid of a poorly selected clause.



A simple workaround—analyze generated clauses, change the input, run again...

Leapfrogging

- Run the solver for a while and generate a large set of clauses $\mathcal{L}$,
  - limit solver by abstract or real time,
- select only "good" clauses from $\mathcal{L}$ and produce a smaller set $\mathcal{S} \subseteq \mathcal{L}$,
  - how to produce $\mathcal{L}$—use only processed or all generated clauses,
  - how to produce $\mathcal{S}$—a premise selection problem,
    - "Surprisingly", we can use the whole $\mathcal{L}$ as $\mathcal{S}$.
    - Moreover, GNN-ENIGMA does an implicit premise selection.
- run the solver on $\mathcal{S}$ for a bit longer...

⋮

Leapfrogging experiment

Data 28k hard Mizar problems.

- Run as follows
  1. use the input and stop after 300 processed clauses,
  2. use 300 processed clauses as the input and stop after 500 processed clauses,
  3. use 500 processed clauses as the input and run for 60s.

GNN-strategy	original (60s)	leapfrogging (300-500-60s)	union	added-by-lfrg
G_1	2711	2218	3370	659
G_2	2516	2426	3393	877
G_3	2655	2463	3512	857
G_4	2477	2268	3276	799
$\bigcup G_i$	4271	3958	5146	875

Leapfrogging experiment II

Data hard Mizar problems.

- Run as follows
  1. use the input and stop after 800 processed clauses,
  2. use 800 processed clauses and stop after 1600 processed clauses,
  3. use 1600 processed clauses, and run for 240s.

Problem types	original (240s)	leapfrogging (800-1600-240s)	union	added-by-lfrg
hard	181	211	257	76
minimized	841	940	1197	356

Another problem with the saturation loop

Observation

The saturation loop "suggests" that all the processed clauses should heavily interact.

Reality

In many cases there are likely reasonably defined components that

- enable different types of reasoning and
- may be solved separately and then (easily) combined.

Example

There are problems with computational and reasoning parts. For example, in numerical calculations, computing derivatives and integrals, algebraic rewriting, ...

Aim

We want to identify such components.

Identifying components

We train a GNN classifier c based on successful proofs—a pair of clauses (C_i, C_j) is

positive if a clause derived from C_i and C_j is used in the proof,

negative otherwise.

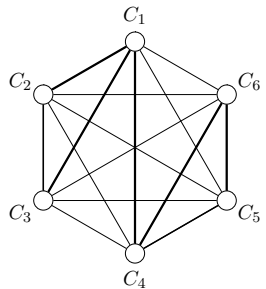
For unsuccessful proof attempts,

1. we evaluate all the pairs (C_i, C_j) in $\mathcal{P}$ using the classifier c,
   - we obtain the likelihood of inferring a useful clause from the pair,
2. we produce components based on the evaluation,
   - use directly the computed likelihoods, or
   - use clause embeddings produced by the classifier as a byproduct.

Interactions between clauses as a graph

We start with a weighted complete graph

- vertices are clauses and
- weights say how likely the pairs produce a clause that will end up in a proof.

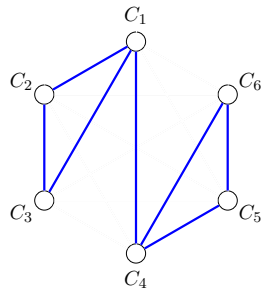


However, we train on positive/negative examples not on likelihoods.

Interactions between clauses as a graph

We start with a weighted complete graph

- vertices are clauses and
- weights say how likely the pairs produce a clause that will end up in a proof.

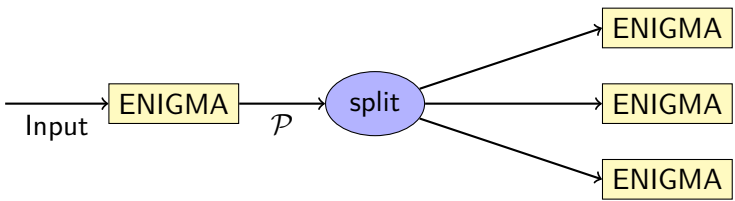


However, we train on positive/negative examples not on likelihoods.

We obtain a graph after pruning using a threshold.

Splitting into components experiment

- We train a classifier c on 20k solved problems ($|\mathcal{P}| \leq 1000$).
- For unsolved problems, we use c and clustering to split $\mathcal{P}$ into components.



ENIGMA timeout is $|\mathcal{P}| \leq 1000$.

Producing components (clustering)

Method	#clusters	Newly solved problems (out of 3000)
k-means	2	67
k-means	3	78
soft k-means	2	63
soft k-means	3	93
Graphviz	≤ 4	111

k-means vectors into k clusters s.t. their within-cluster variance is minimal,

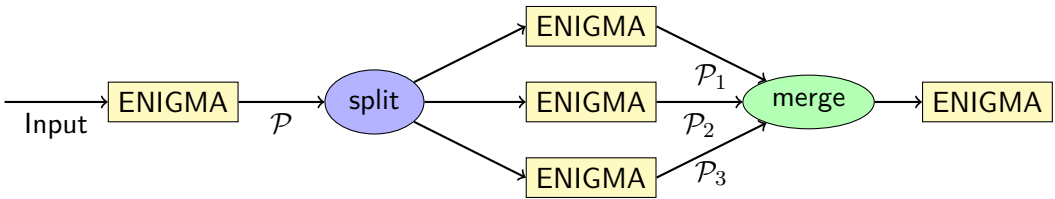
soft k-means a generalization of k-means that allows overlapping clusters,

Graphviz graphs into clusters based on the modularity measure
(we also add highly connected clauses into all clusters).

Merging components

Experiment

We merge the generated sets of processed clauses and run a premise selection on them.



ENIGMA timeout is $|\mathcal{P}| \leq 1000$.

Using this setup, 51 (out of 2889) new problems are proved.

Conclusions

- We discussed several methods that make the clause selection in the saturation loop more dependent on the proof state.
- We presented some initial experiments showing encouraging results of these methods.
- However, the presented methods are rather modest, but we should keep in mind that these methods have to reasonably fast.

Thank you!

Interesting Mizar proofs produced by ATPs are available at

`https://github.com/ai4reason/ATP_Proofs`