~~Playing with AVATAR~~
How to play with AVATAR

Giles Reger, Martin Suda and Andrei Voronkov

School of Computer Science, University of Manchester

The 1st Vampire Workshop

Overview

1. Introduction
2. Reviewing AVATAR
3. The variables
4. How to evaluate
5. Results
6. Conclusion

Introduction

In this talk we will:

- Briefly recall what the AVATAR architecture is
- List the parameters that control its behaviour
  - (and what effects they have)
- Discuss how we should evaluate these kinds of frameworks
- Present results of our experimental evaluation

Work in progress!

Overview

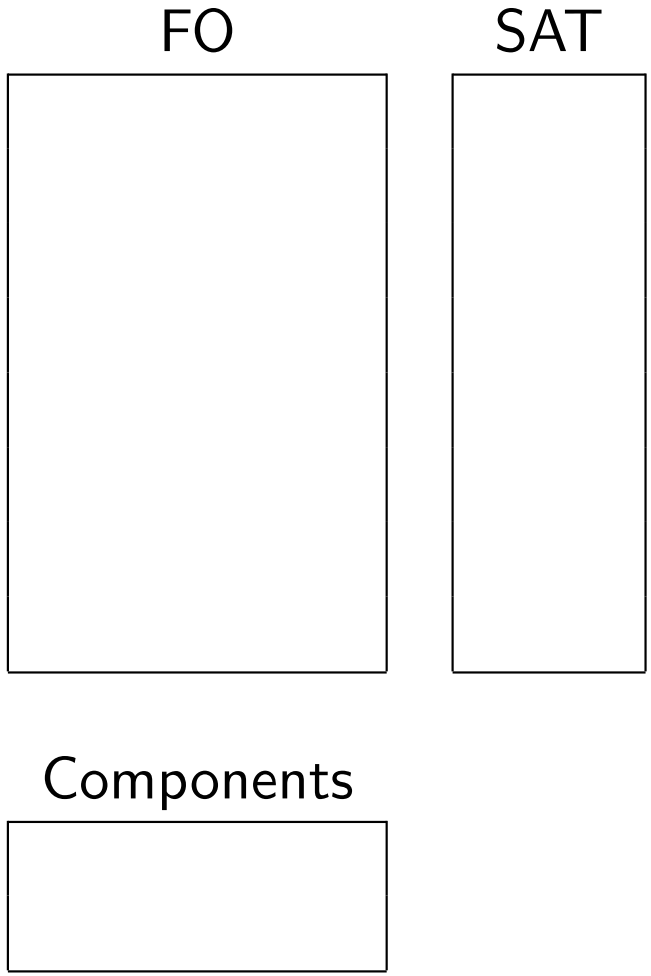
1. Introduction
2. Reviewing AVATAR
3. The variables
4. How to evaluate
5. Results
6. Conclusion

AVATAR

- Input:

  $$p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

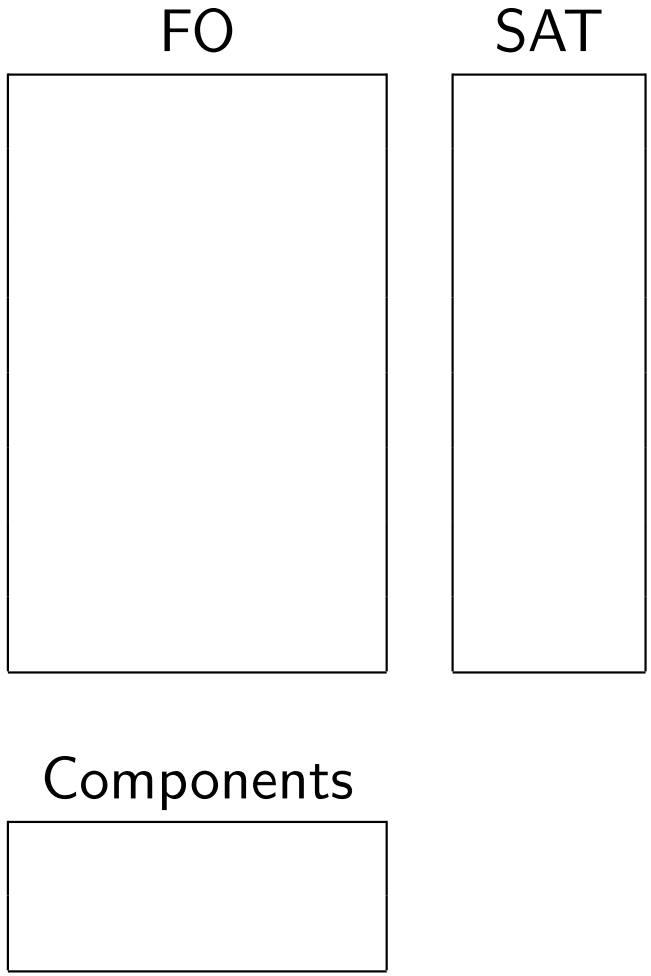


AVATAR

- Input:

  $$p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

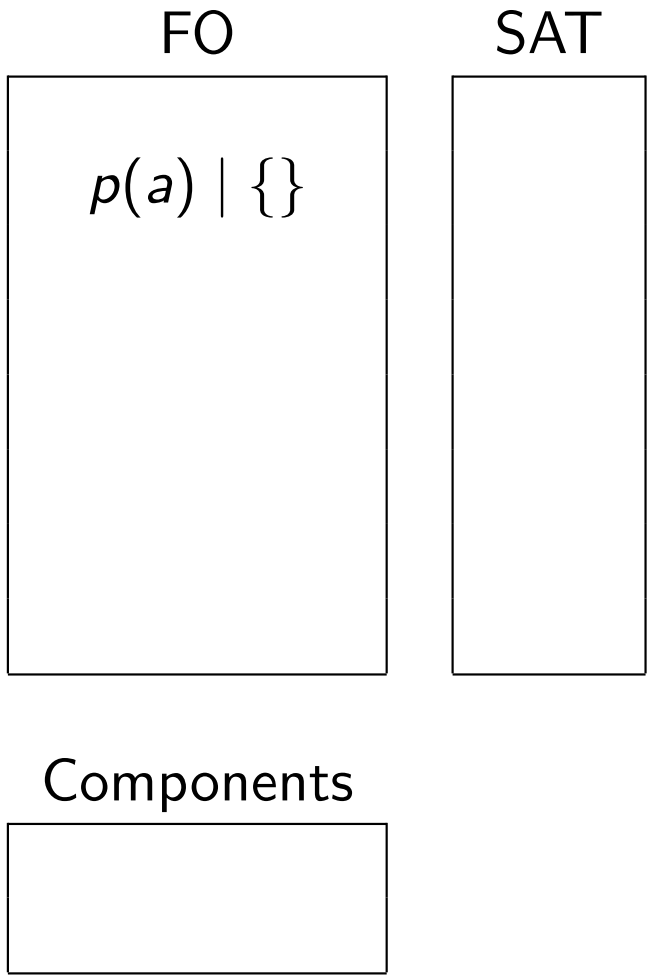


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

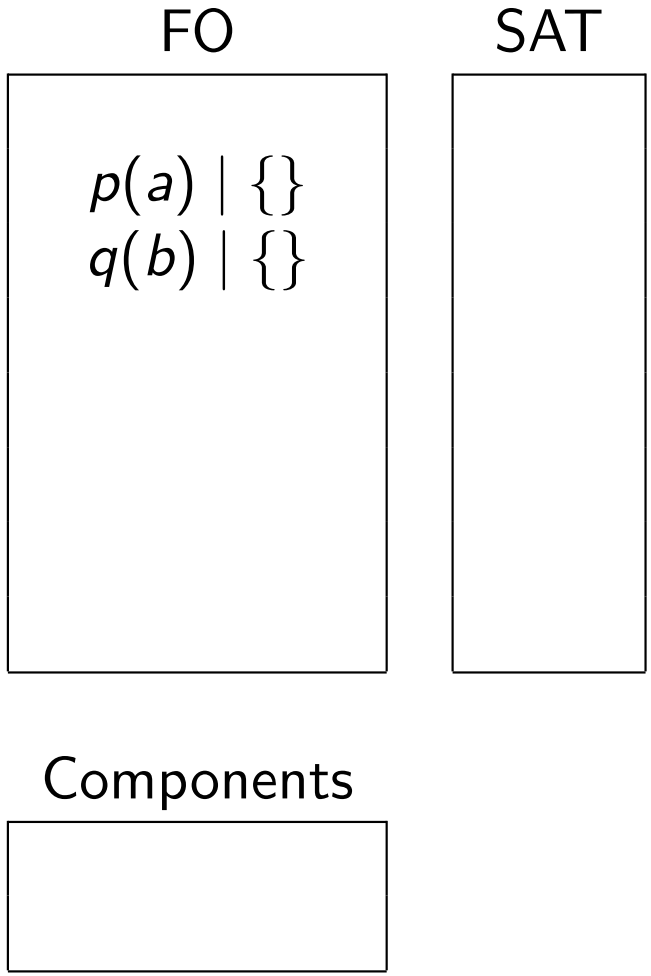


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

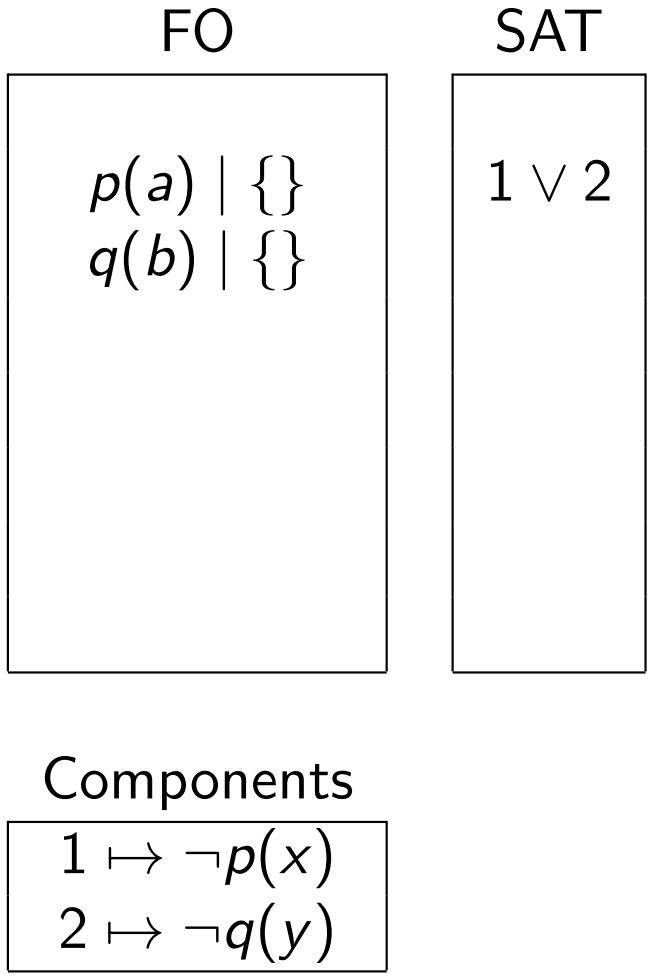


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

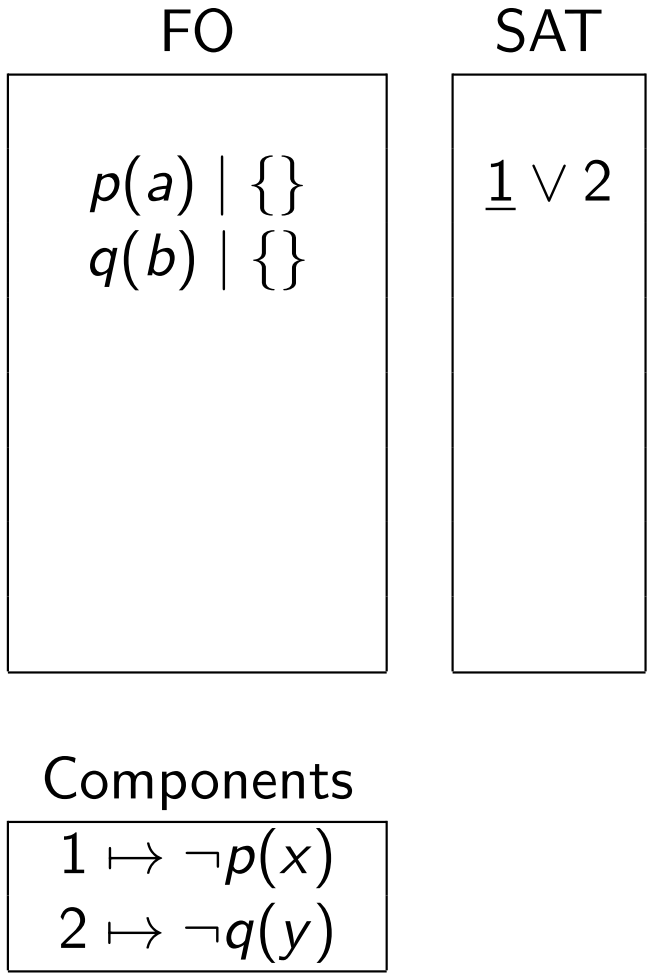


AVATAR

- Input:

  $$p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

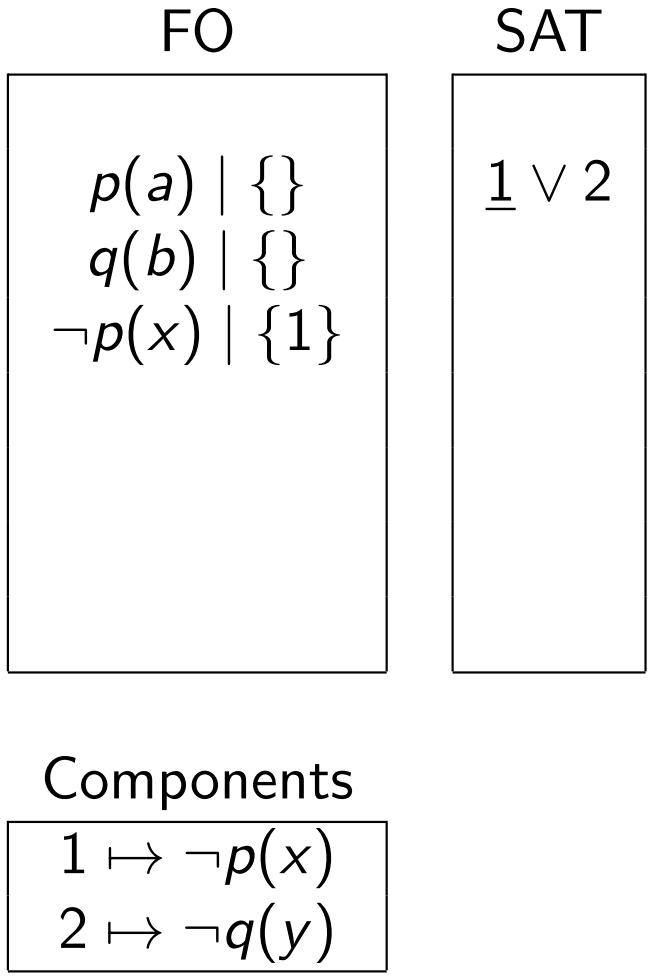


AVATAR

- Input:

  $$p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

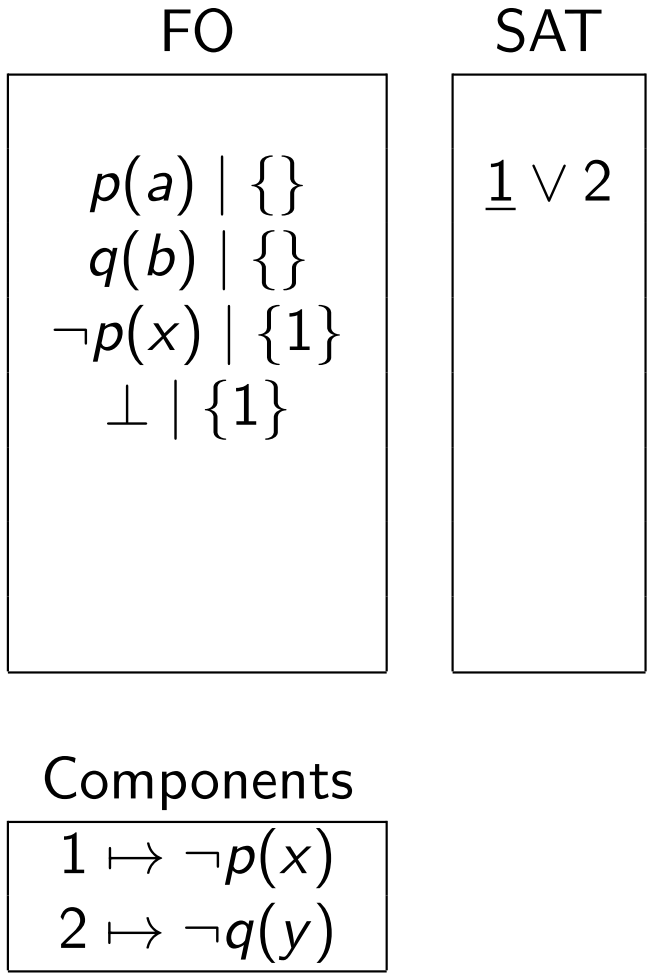


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

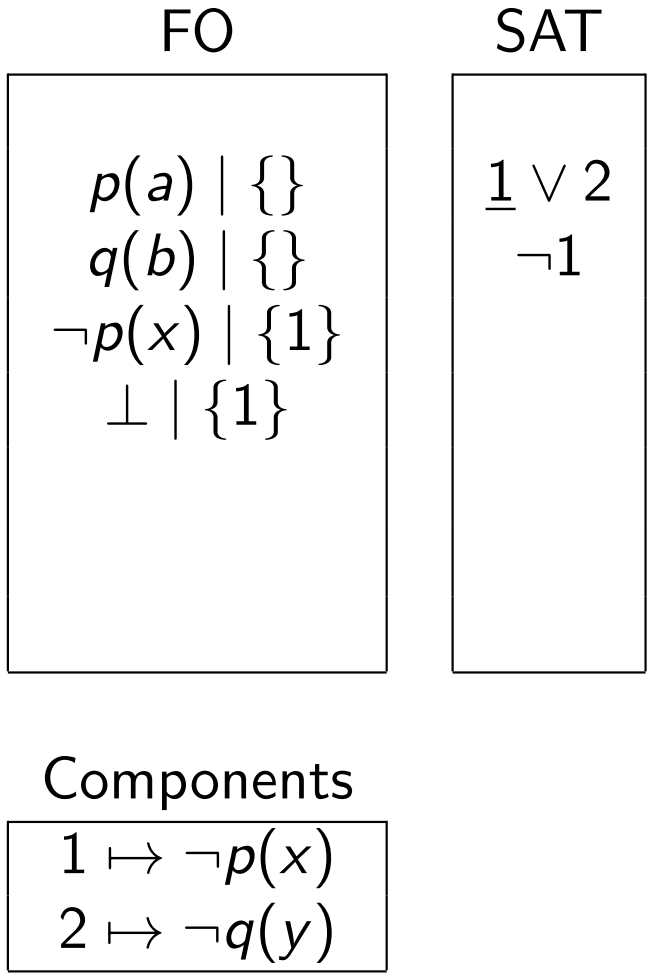


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

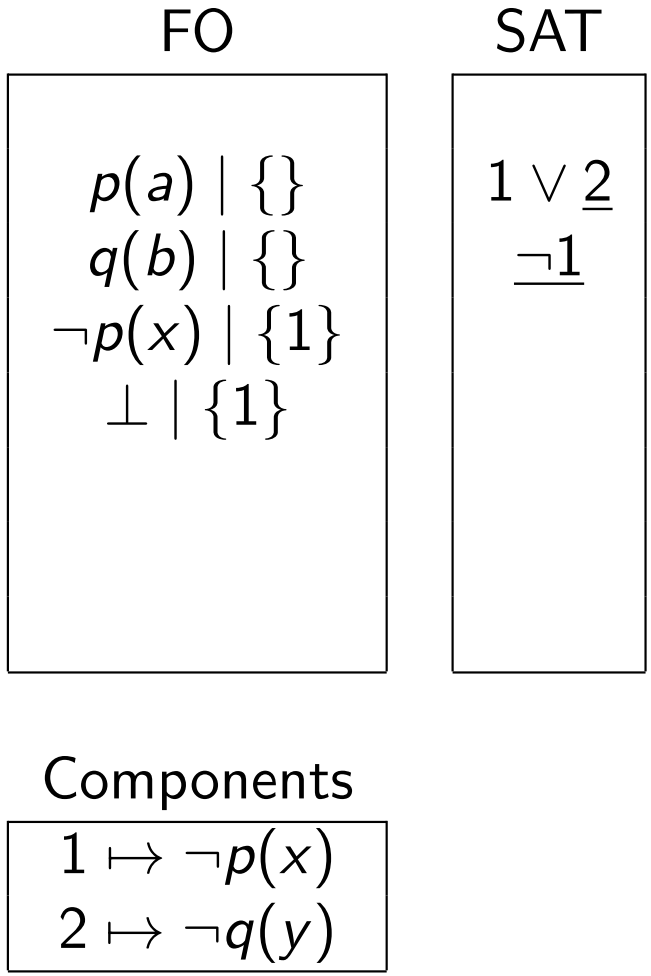


AVATAR

- Input:

  $$p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

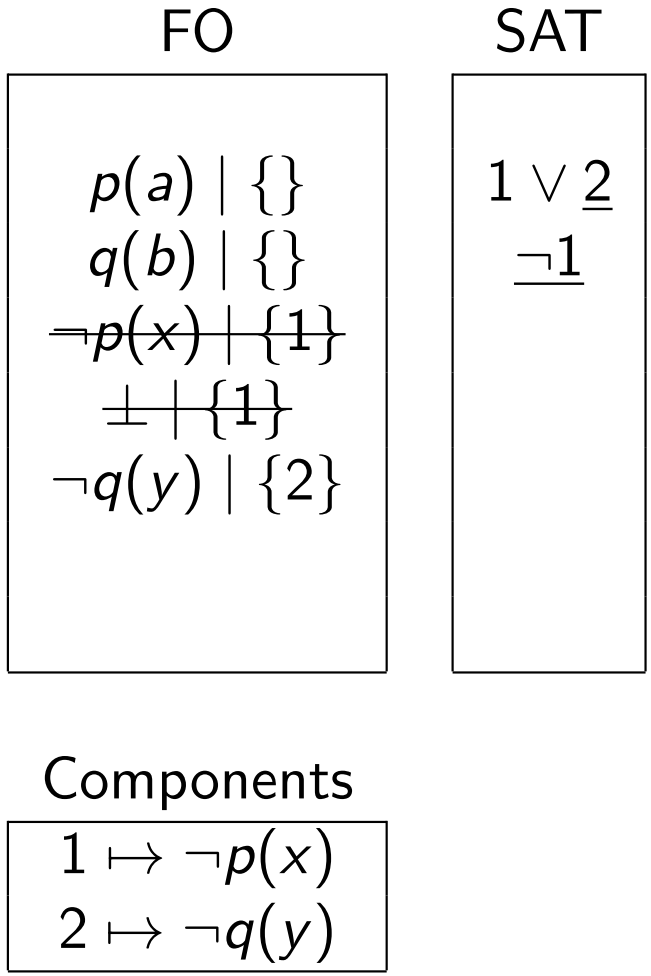


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

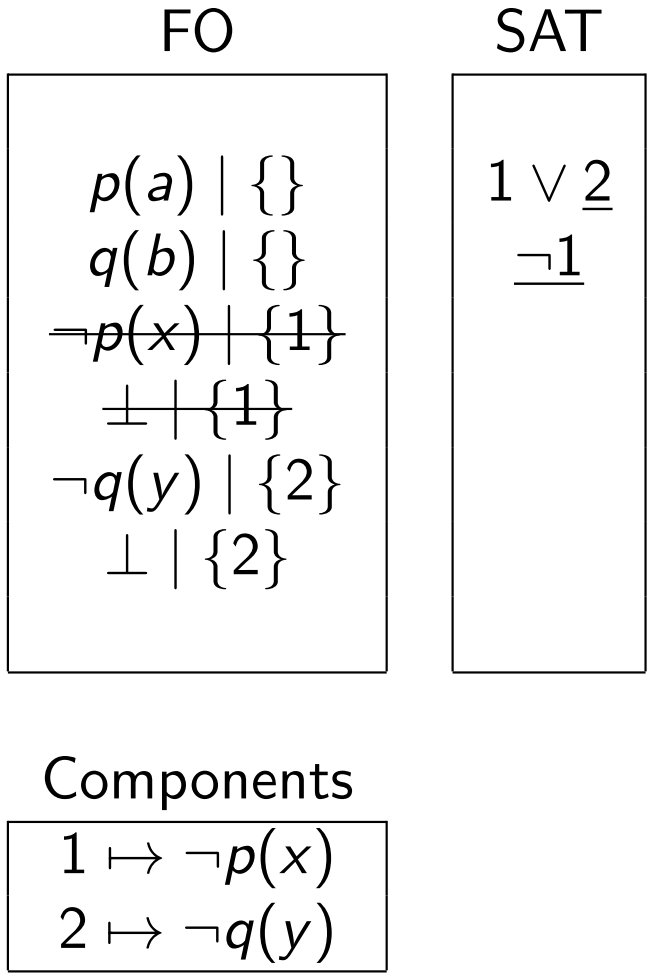


AVATAR

- Input:

  $$p(a),\quad q(b),\ \neg p(x) \vee \neg q(y)$$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

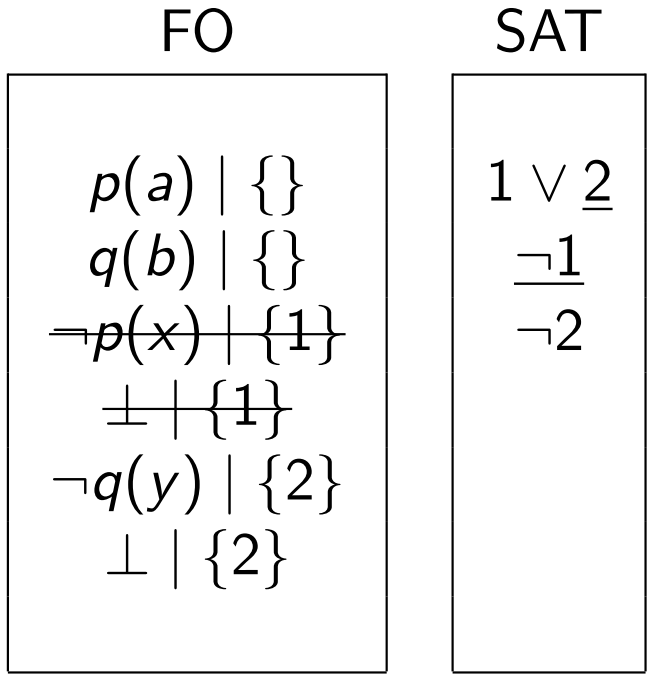


AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation



Components

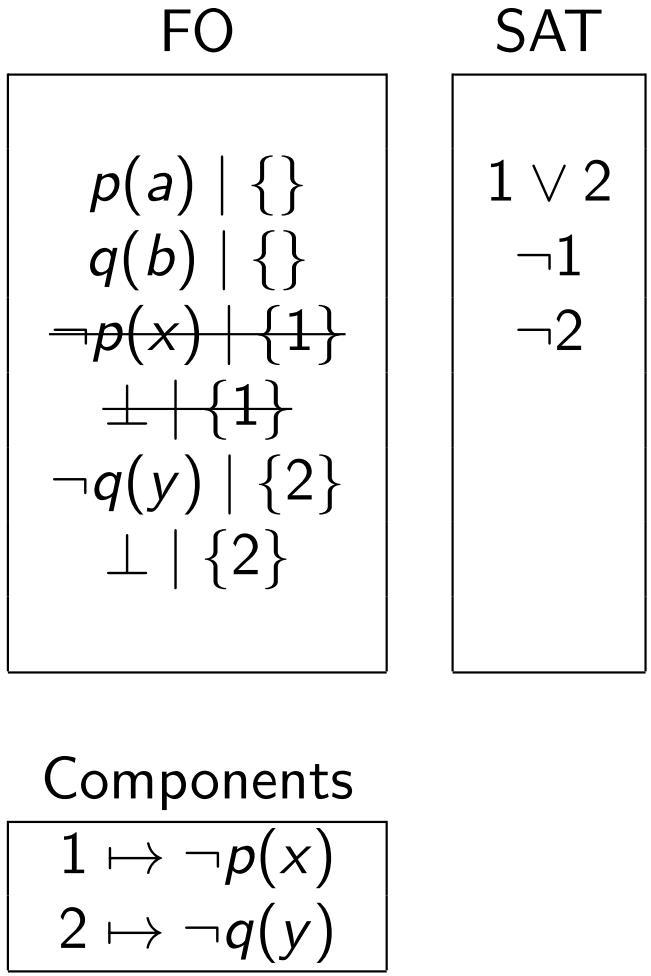
$1 \mapsto \neg p(x)$
$2 \mapsto \neg q(y)$

AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation



AVATAR

- Input:

  $p(a),\ \ q(b),\ \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
  - FO: Do FO proving
    - ★ Process refutation

- Refutation
  - From the SAT solver as we cannot construct a model

FO

$p(a) \mid \{\}$
$q(b) \mid \{\}$
~~$\neg p(x) \mid \{1\}$~~
~~$\bot \mid \{1\}$~~
$\neg q(y) \mid \{2\}$
$\bot \mid \{2\}$

SAT

$1 \vee 2$
$\neg 1$
$\neg 2$

Components

$1 \mapsto \neg p(x)$
$2 \mapsto \neg q(y)$

Important points

- Components are always named consistently (up to variants)
- An inference between two clauses with assertions takes the union of those assertions:

$$\frac{c_1 \mid a_1 \qquad c_2 \mid a_2}{d \mid (a_1 \cup a_2)}$$

- Removal of redundant clauses is conditional in general:
  - assume that c_2 is subsumed by c_1 for clauses $c_1 \mid a_1$ and $c_2 \mid a_2$
  - If $a_1 \subseteq a_2$
    - ★ Then whenever $c_1 \mid a_1$ is backtracked, then $c_2 \mid a_2$ must be also, as an assertion in a_1 is retracted, which must also be in a_2
    - ★ Therefore, we can remove $c_2 \mid a_2$
  - otherwise ($a_1 \not\subseteq a_2$)
    - ★ Later, if an assertion in a_2/a_1 is retracted then $c_1 \mid a_1$ would be backtracked, but $c_2 \mid a_2$ would not be
    - ★ Therefore, we conditionally remove (freeze) $c_2 \mid a_2$
    - ★ Then, if $c_1 \mid a_1$ is later removed we must add (unfreeze) $c_2 \mid a_2$

Overview

1. Introduction
2. Reviewing AVATAR
3. The variables
4. How to evaluate
5. Results
6. Conclusion

Adding components (nonsplittable clauses)

- If we cannot split a clause into components what do we do?
  - Just add it anyway - it might be useful later!
  - Only add it as a component if it has assertions (dependencies) i.e.
    - ★ If we derive $q(x) \vee p(x)|\{2,4\}$ we would add $\neg 2 \vee \neg 4 \vee 8$ (for fresh 8)
    - ★ Helps if 8 is derived again later
  - Only add it as a component if it is a known component i.e.
    - ★ We previously added $2 \vee 4$ for $r(y) \mapsto 2$ and $q(x) \vee p(x) \mapsto 4$
    - ★ We then derive $q(x) \vee p(x)$ and add 4
    - ★ The SAT solver must always choose 4 - simplifying $2 \vee 4$
  - Don't add it

Adding components (ground components)

- If a component is ground it is safe to introduce a name for its negation (not safe for non-ground)
- If we have $p(x) \vee q(a)$ and $\neg p(x) \vee \neg q(a)$ we can add

$$1 \vee 2 \text{ and } 3 \vee 4$$

but it is better to add

$$1 \vee 2 \text{ and } 3 \vee \neg 2$$

- This is something we do not play with, as previous experiments showed that it was consistently a good idea
- Note that a ground component will be a literal

Constructing a model

- In AVATAR the SAT solver is a black box that is allowed to construct any valid model. There are two things we can consider
  - How quickly a model can be constructed
  - What model is constructed
- It is obvious that the model produced has a very large effect on the exploration of the search space.

- We consider two SAT solvers:
  - A native (two watched literals) solver
  - lingeling (with relatively default options)

- We also consider a buffering optimisation that buffers a clause if, either
  - it contains a fresh variable that can be made true, or
  - it is already true in the model

  This may lead to fewer calls to the SAT solver, but will also lead to a different model

Using a model

- As mentioned above, we do not need the whole model
- If we use a partial model we
  - Have to pay to minimise the model
  - But, we potentially add fewer FO clauses and do less freezing/unfreezing
- Choices:
  - Total model
  - Minimised model - a partial model that satisfies all added clauses
  - Minimised model for split clauses - satisfy split clauses only
- Note - partial model is a sub-model of the total one
- If a component was previously asserted, but is now don't care (not in the partial model) we can either
  - eagerly remove it, or
  - leave it there... it might be asserted again later

An overview of the relevant options

- Adding components
  - `ssplitting_nonsplittable_components`
    - When to add a component that is not splittable
    - `known`, `all`, `all_dependent`, `none`
- Constructing a model
  - `sat_solver`
    - Which sat solver is used to construct the model
    - `lingeling` or `vampire`, with `buffering` or not
- Using a model
  - `ssplitting_model`
    - We can minimise the model to reduce the number of components asserted in the FO part
    - `total`, `min_all`, `min_sco`
  - `ssplitting_eager_removal`
    - When using a non-total model we can eagerly remove components no longer mentioned by the model
    - `on`, `off`

Overview

1. Introduction
2. Reviewing AVATAR
3. The variables
4. How to evaluate
5. Results
6. Conclusion

How should we evaluate?

- CASC mode makes use of 47 different (still valid) options
- Many of these have multiple values (some are continuous)
- If we stick only to values selected in CASC mode we have 493,748,224 possible combinations (some of which will not be valid)

- TPTP v6.0.0 has 16,004 FOF and CNF problems

- Giving one minute per experiment that takes 1,500 millennia per value we want to compare
  - That's 144,000 millennia for the experiments here...
  - To finish now we should have started at the end of the Jurassic period

- We need to consider what we are looking for...

Directly comparing options

- If we want to generally compare different values for an option we need to systematically run through the same experiments for each value.

- Massive search space requires us to select a subset of options or problems
  - Select subset of options
    - ⋆ May miss the best strategies
  - Select subset of problems
    - ⋆ May miss the easy/hard problems
  - Probably need to do both to have a reasonable search space

- Alternatively, we could use the CASC-mode approach that attempts multiple strategies, but
  - This suffers from similar restrictions i.e. the results are not generalisable from the chosen strategies.
  - Additionally it is biased as the default values for all of these options were included in the CASC-mode training... so are more likely to be successful.

Searching for improvements

- Observation: A CASC-mode-like approach makes use of many strategies. Therefore, if a strategy can be shown to perform well for some problems, its performance on other problems does not matter.
- If our aim is to solve new problems or solve problems faster then we want to identify cases where new options lead to these interesting cases.
- We can randomly select a strategy, a problem and an option to experiment with. We then vary the values for this option and check whether the result is interesting.
- However, our results are not generalisable.

Overview

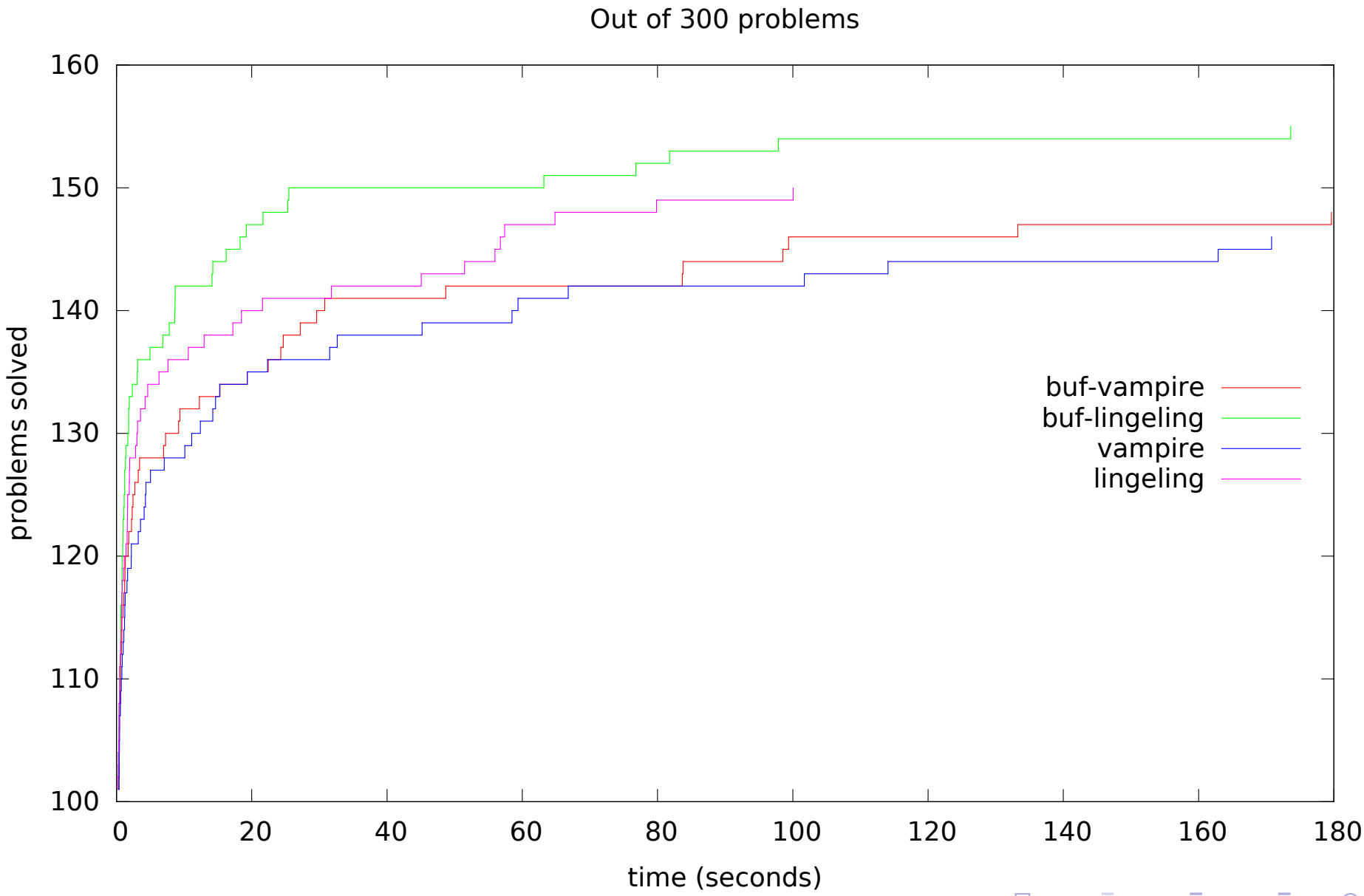
1. Introduction
2. Reviewing AVATAR
3. The variables
4. How to evaluate
5. Results
6. Conclusion

Our experiments

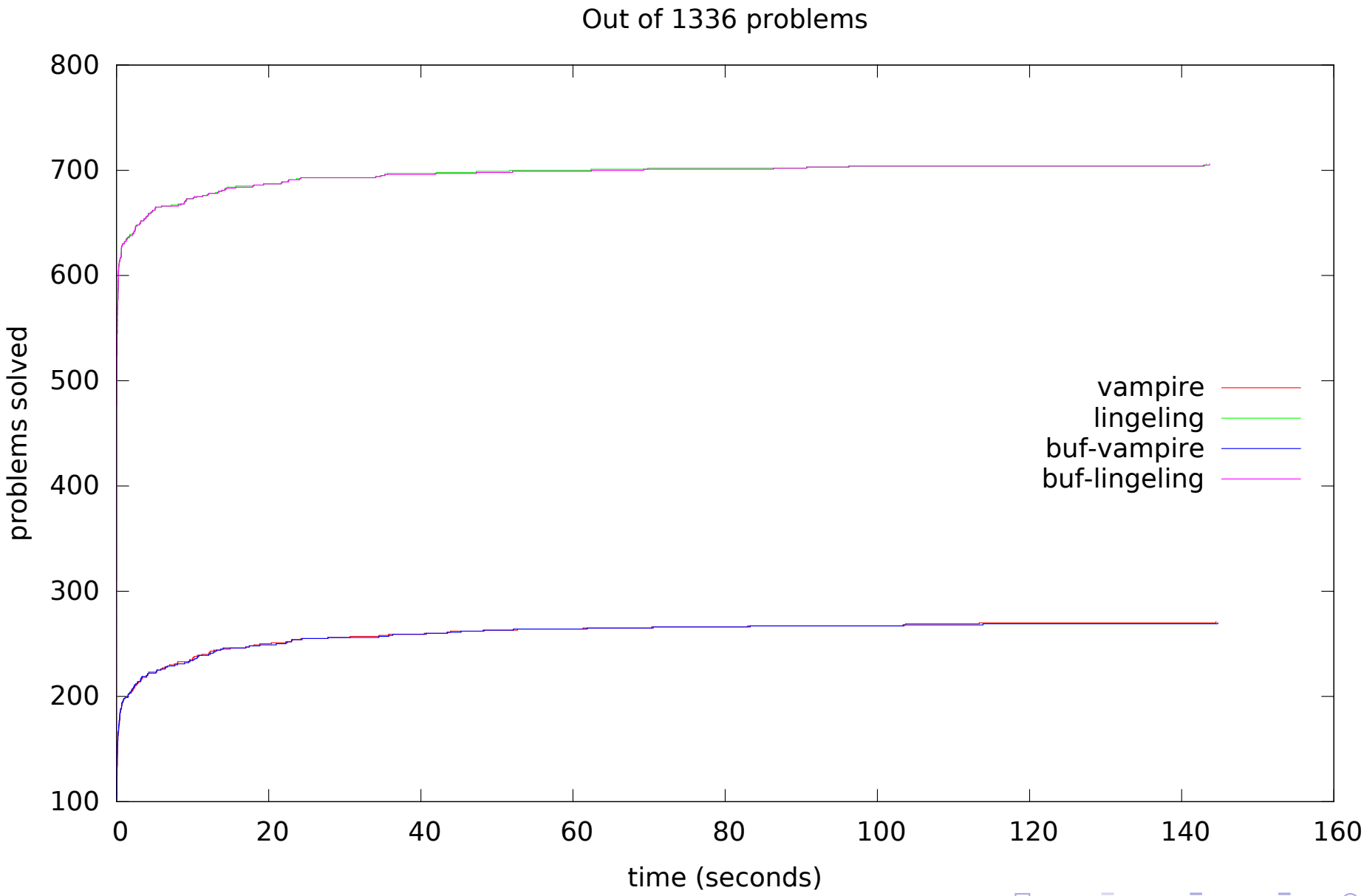
- Systematic
  - Use CASC13 problems
  - Use default options
- Random
  - Construct an experiment by randomly selecting
    - ★ A problem
    - ★ A set of options
    - ★ An experimental option
  - Vary the value for the experimental option
  - However - currently keep other experimental options as default

- These results
  - are not complete
  - can only be generalised within a certain context
  - are not very exciting

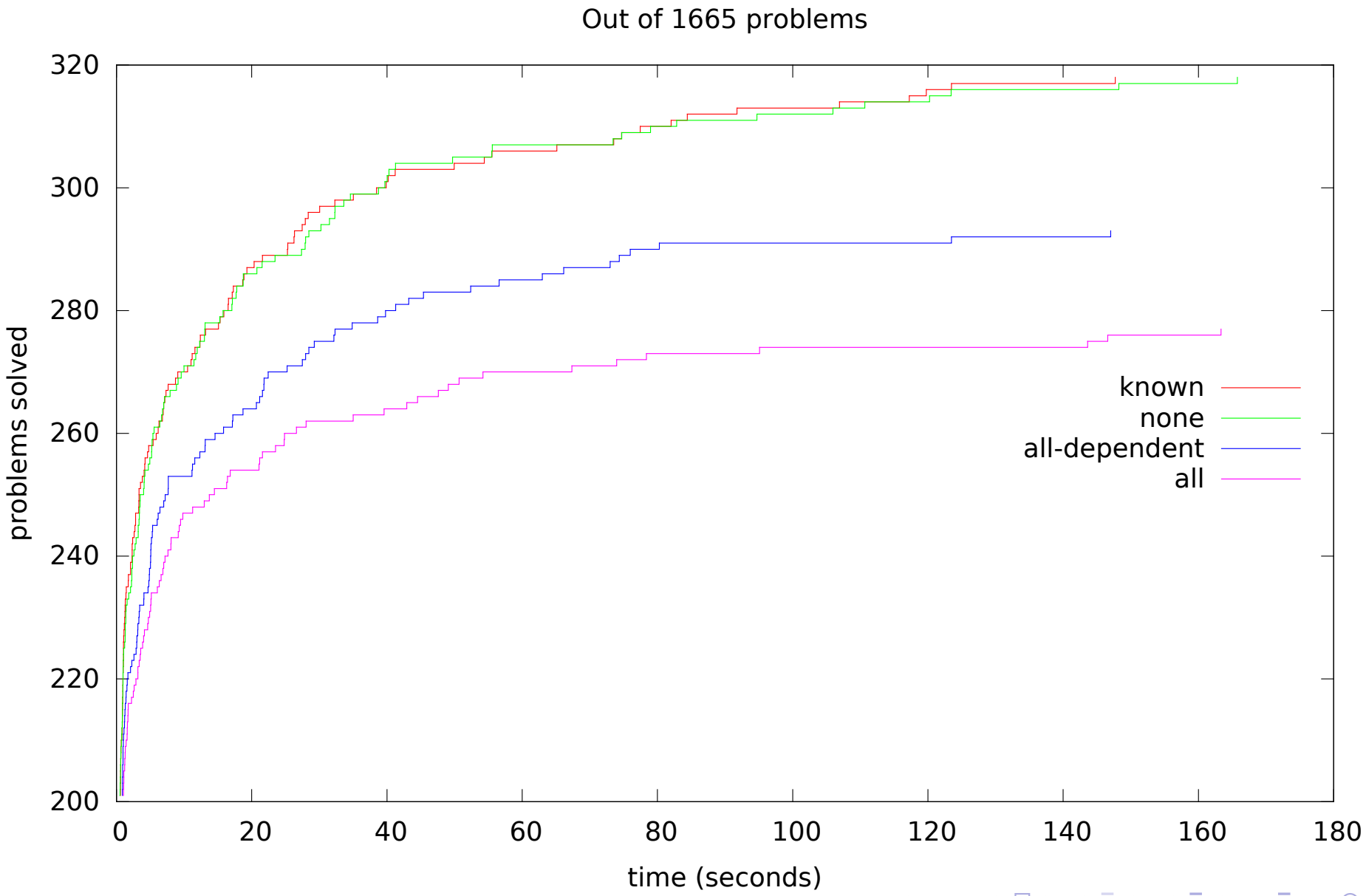
SAT solver


Out of 300 problems
problems solved
time (seconds)
100
110
120
130
140
150
160
0
20
40
60
80
100
120
140
160
180
buf-vampire
buf-lingeling
vampire
lingeling

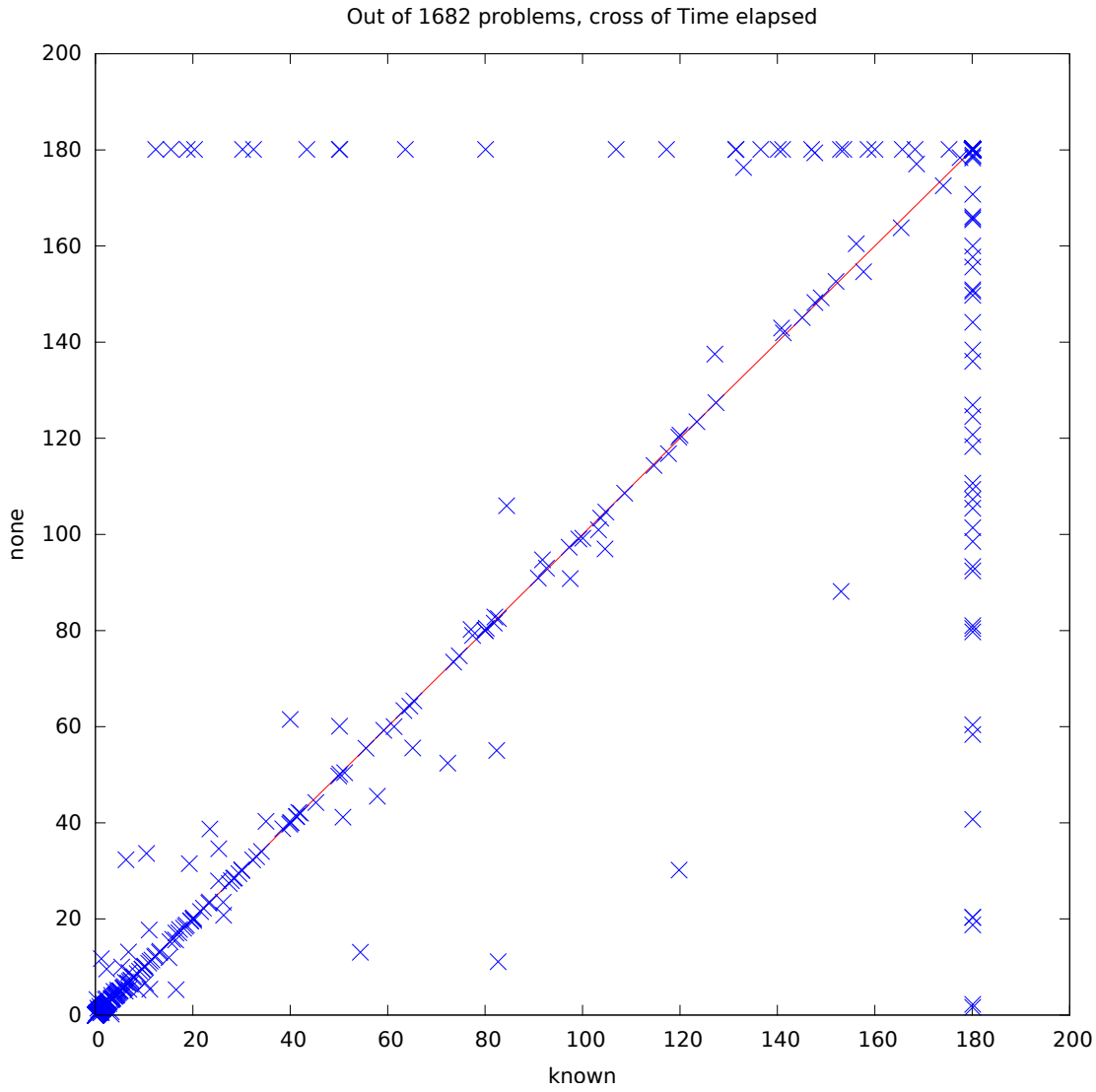
SAT solver


Out of 1336 problems
problems solved
time (seconds)
100
200
300
400
500
600
700
800
0
20
40
60
80
100
120
140
160
vampire
lingeling
buf-vampire
buf-lingeling

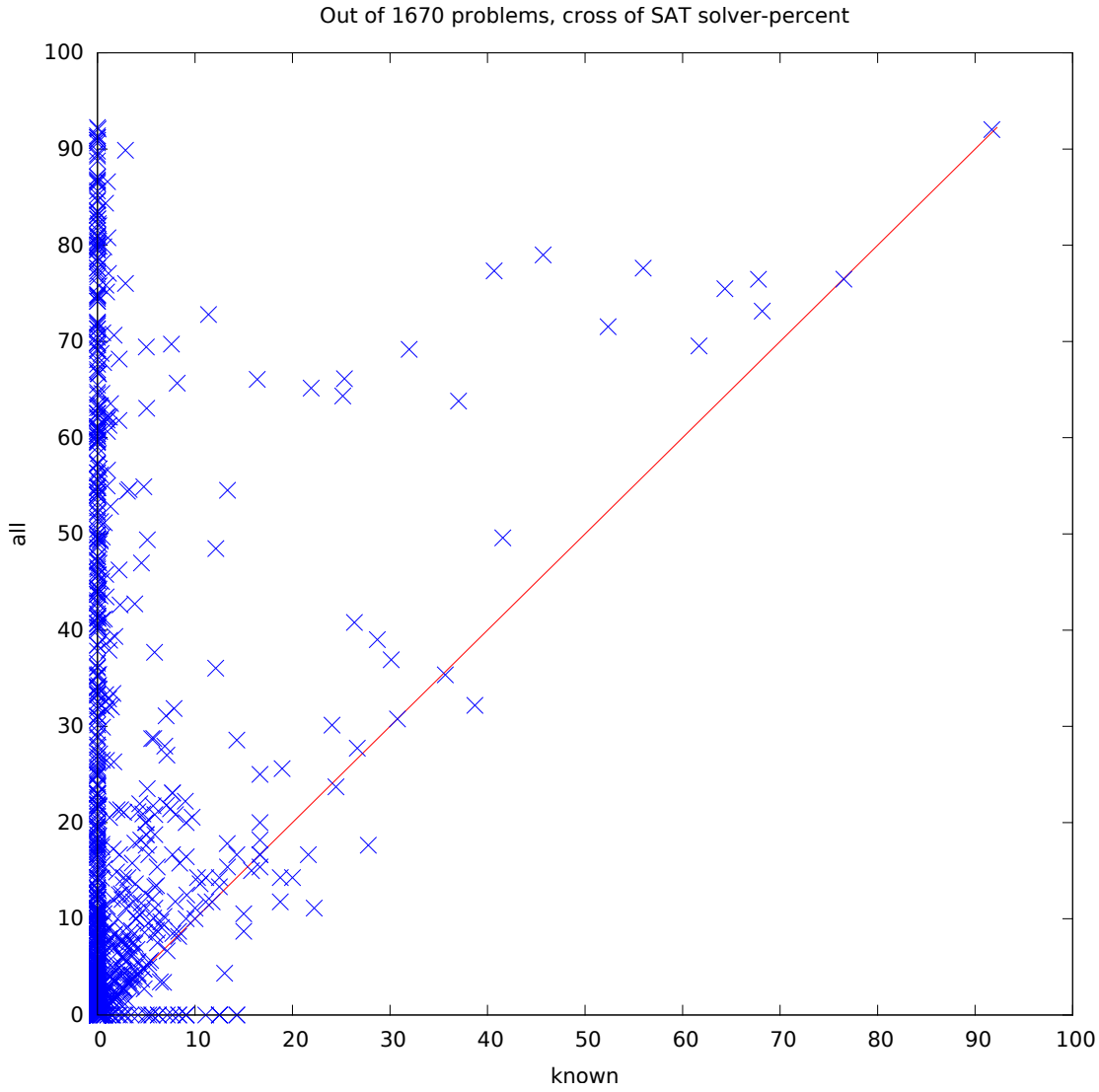
Nonsplittable Components



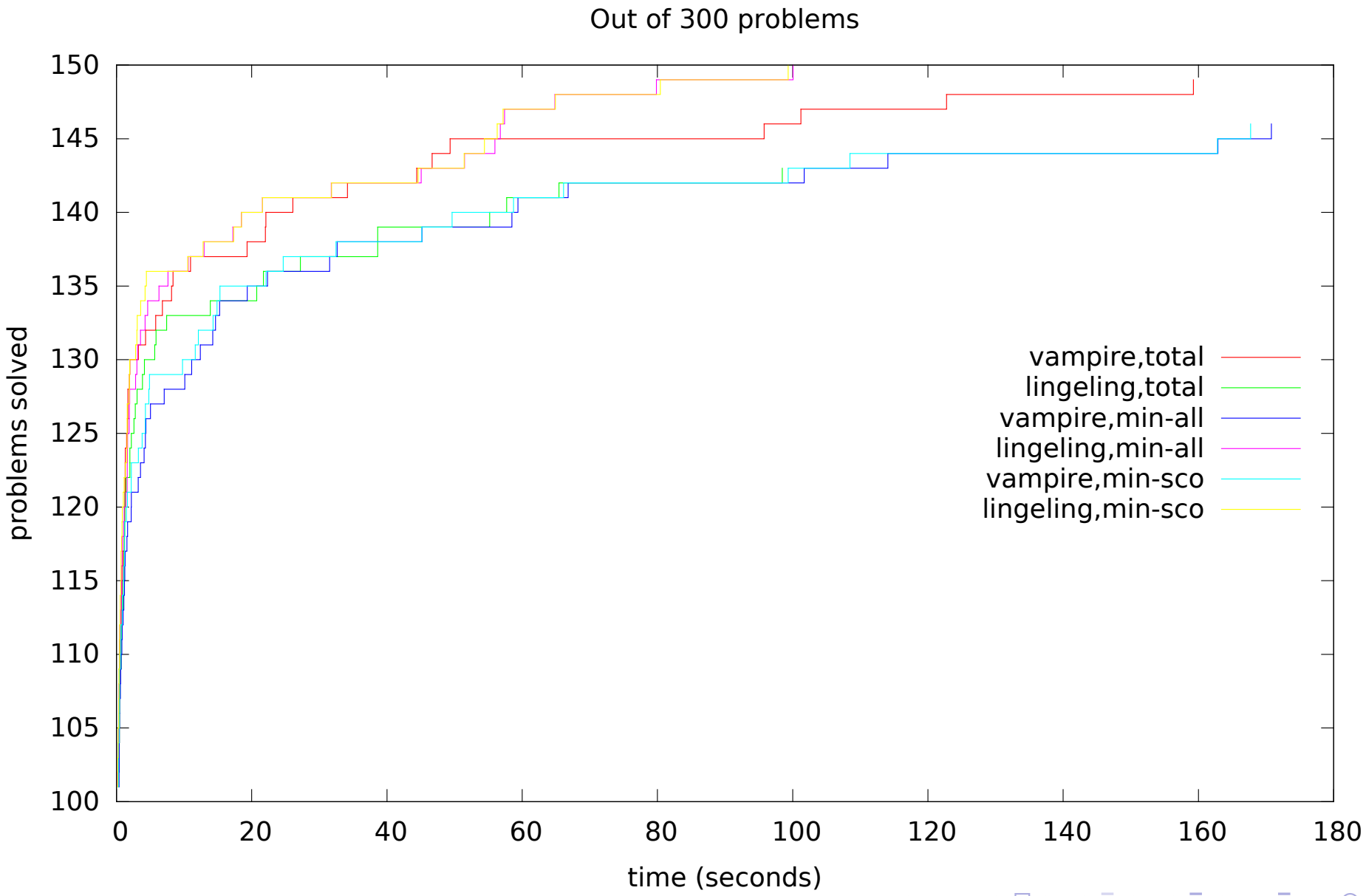
Nonsplittable Components



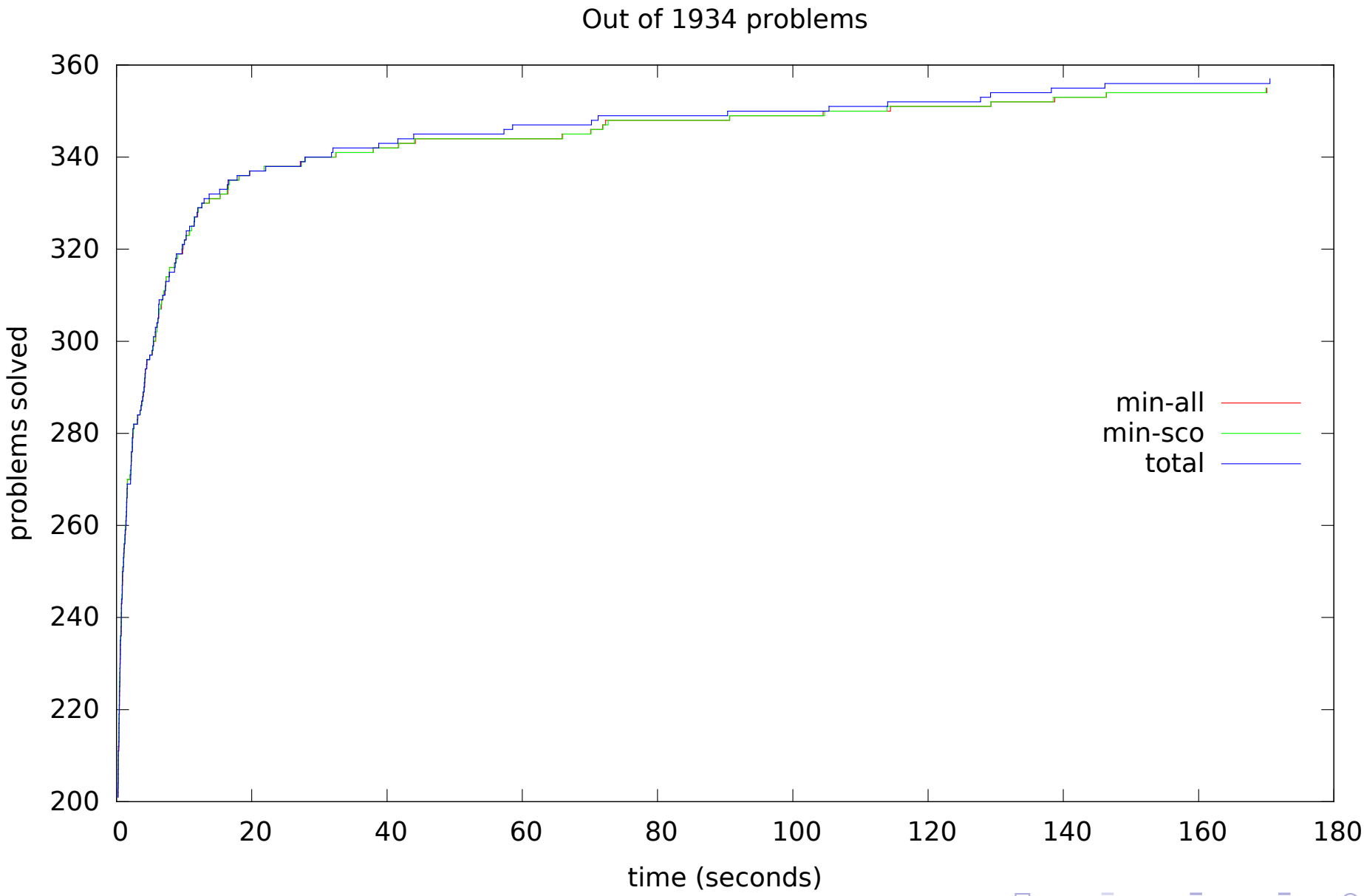
Nonsplittable Components



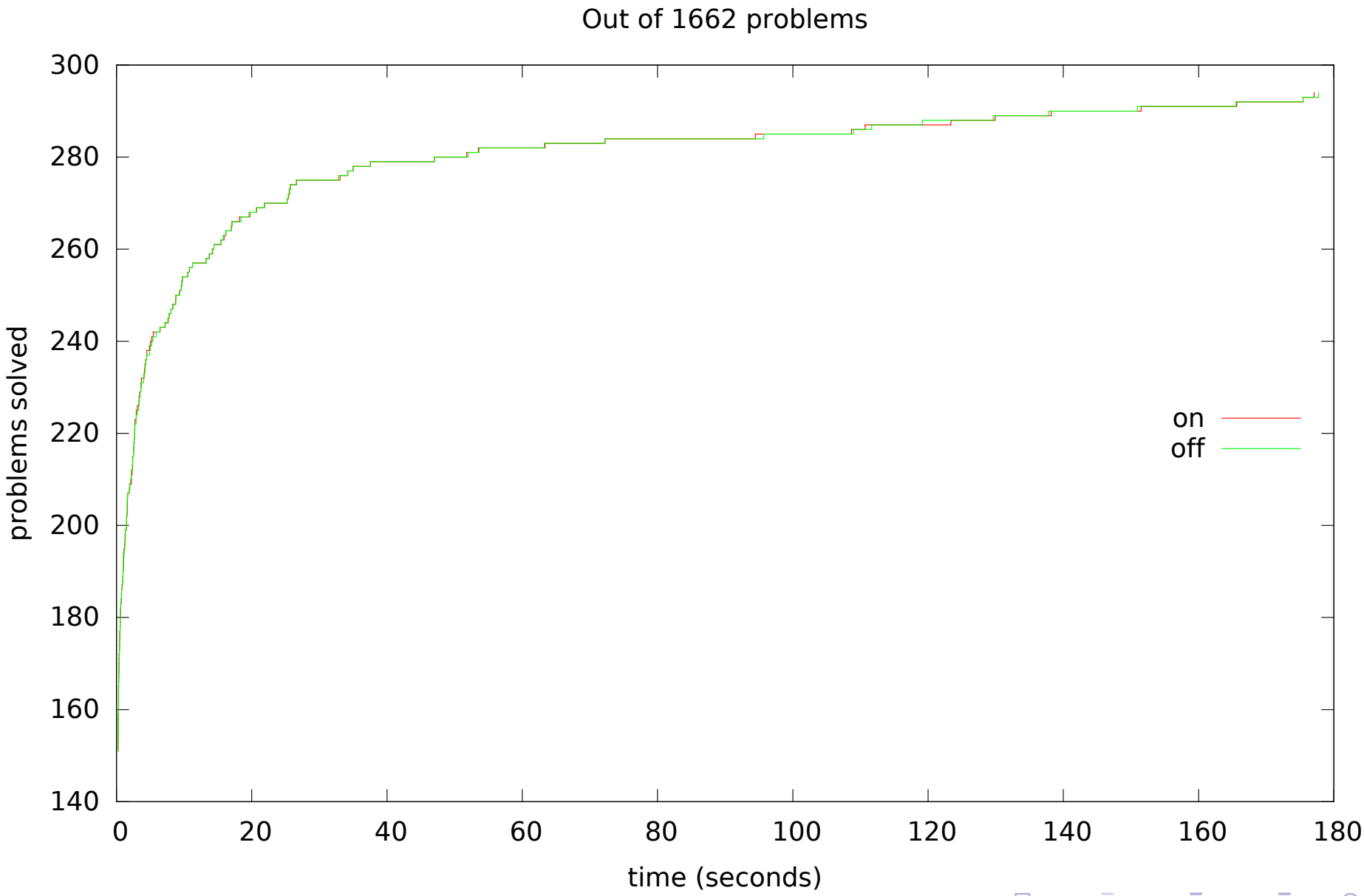
Model minimisation



Model minimisation



Eager removal



Overview

1. Introduction
2. Reviewing AVATAR
3. The variables
4. How to evaluate
5. Results
6. Conclusion

Unanswered questions

- Can we encourage the SAT solver to construct a model that leads to 'nice' clauses being added to the FO part?
  - i.e. light, small clauses rather than heavy, long ones
- What makes a nice model?
  - How constrained is the model (can we make any difference?)
  - How does the constructed model interact with selection?
- Can we encourage the SAT solver to construct a model with a minimal difference from the previous model?
  - Beyond phase saving and Vampire's backtrack-to-last-valid-choice
- Would giving the SAT solver more information help?
  - i.e. add a clause if one component subsumes another
- Can we do more from a refutation with assumptions?
  - i.e. minimise them, collect multiple refutations in one FO run

Conclusions

- AVATAR is fun
- There are lots of things we can tweak
- Running experiments is difficult
- Our results were not interesting - maybe we asked the wrong questions