Research overview

Lyndon Drake

lyndon@cs.york.ac.uk

University of York, United Kingdom

Outline

- Neighbour resolution
- Future work
- Swan - my SAT solver

Example: neighbour resolution 1

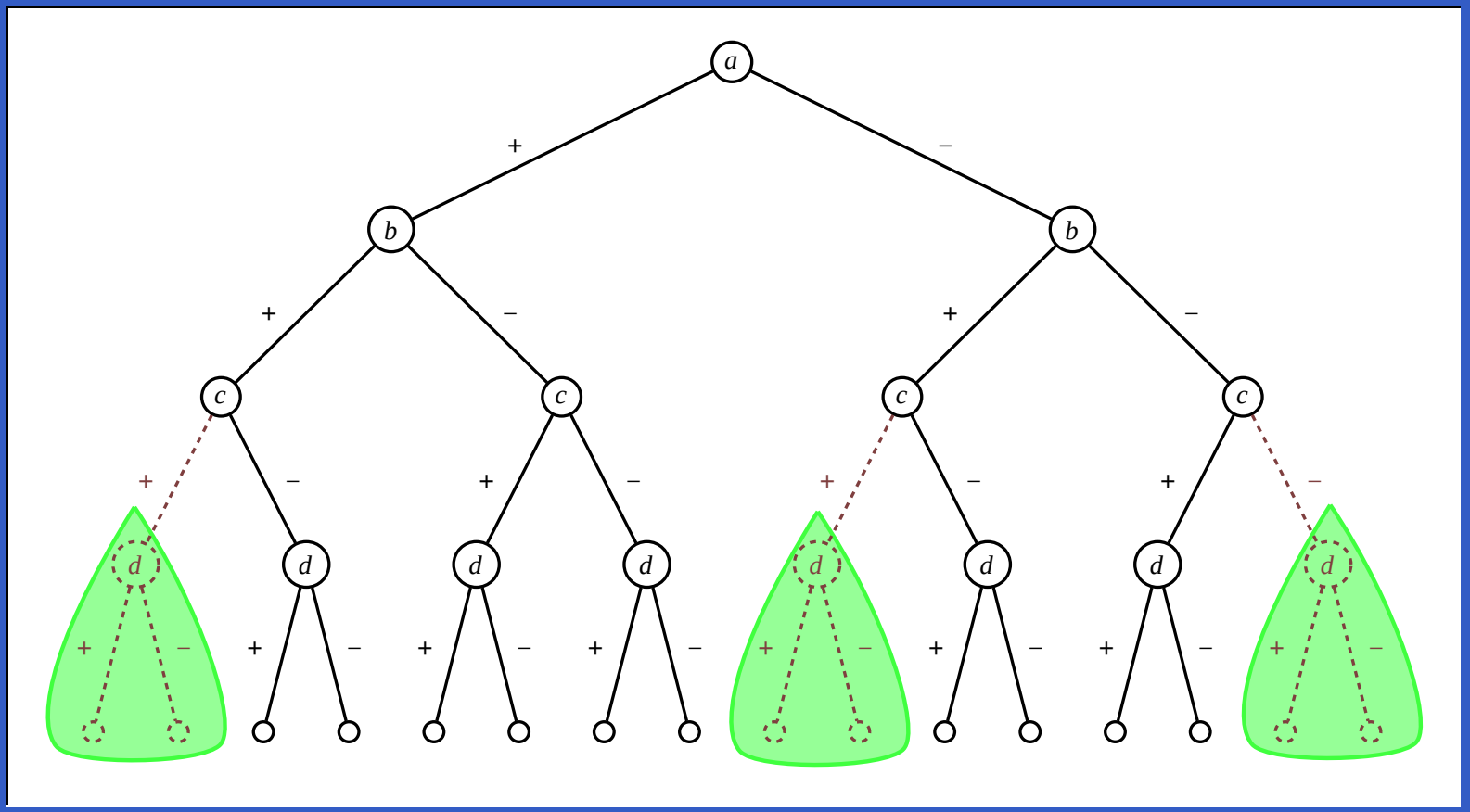
$$\frac{\neg b \vee d \qquad \neg b \vee \neg d}{\neg b}$$

Example: Unsatisfiable SAT instance

$$
\begin{array}{lllll}
\neg a & \vee & \neg b & \vee & \neg c \\
a & \vee & \neg b & \vee & \neg c \\
a & \vee & b & \vee & c \\
\neg b & \vee & d & & \\
\neg b & \vee & \neg d & & \\
a & \vee & c & \vee & d \\
\neg b & \vee & c & \vee & \neg d
\end{array}
$$

$$
\begin{array}{lllll}
a & \vee & b & \vee & d \\
a & \vee & \neg c & \vee & \neg d \\
\neg a & \vee & b & \vee & d \\
b & \vee & \neg c & \vee & \neg d \\
\neg a & \vee & c & \vee & d \\
b & \vee & c & \vee & \neg d
\end{array}
$$

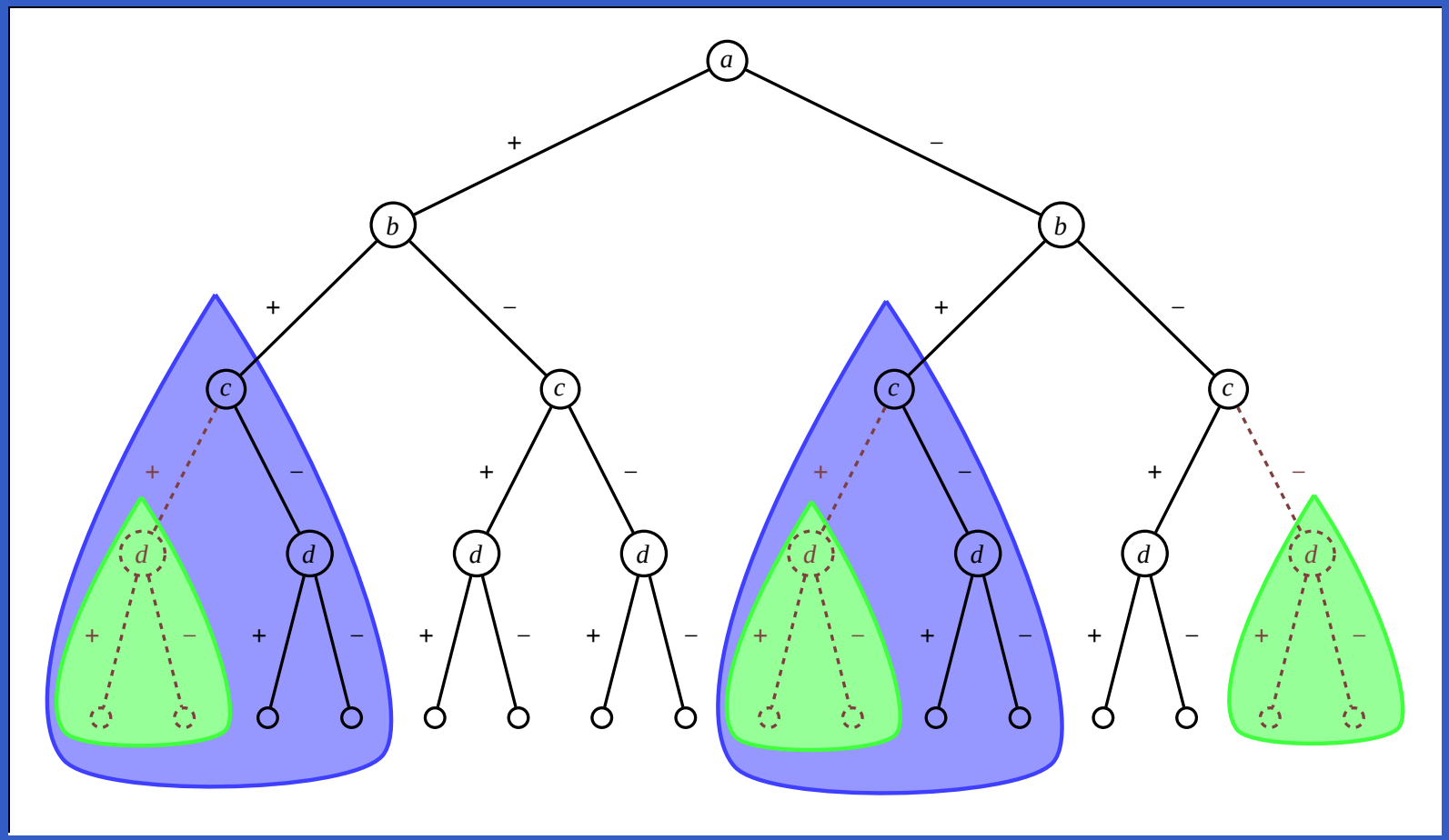
Example: search tree



Example: neighbour resolution 2

$$
\begin{array}{rlllll}
 & \neg a & \vee & \neg b & \vee & \neg c \\
 & a & \vee & \neg b & \vee & \neg c \\
 & a & \vee & b & \vee & c \\
\Longrightarrow & \neg b & & & & \\
 & a & \vee & c & \vee & d \\
 & \neg b & \vee & c & \vee & \neg d
\end{array}
\qquad
\begin{array}{llllll}
a & \vee & b & \vee & d \\
a & \vee & \neg c & \vee & \neg d \\
\neg a & \vee & b & \vee & d \\
b & \vee & \neg c & \vee & \neg d \\
\neg a & \vee & c & \vee & d \\
b & \vee & c & \vee & \neg d
\end{array}
$$

Example: pruned search tree



Motivation

- SAT is the archetypal NP-complete problem, and therefore interesting in its own right
- Many other problems can be usefully mapped to SAT, including quasigroup completion problems and model checking
- Adding inference to search can guarantee pruning, but is often too expensive

Previous work

Combining resolution and search:

- Rish and Dechter. Resolution versus search: two strategies for SAT. In SAT2000, IOS Press, 2000.
- van Gelder. Satisfiability testing with more reasoning and less guessing. In Second DIMACS implementation challenge, 1995.
- Cha and Iwama. Adding new clauses for faster local search. In Proc AAAI-96, 1996.

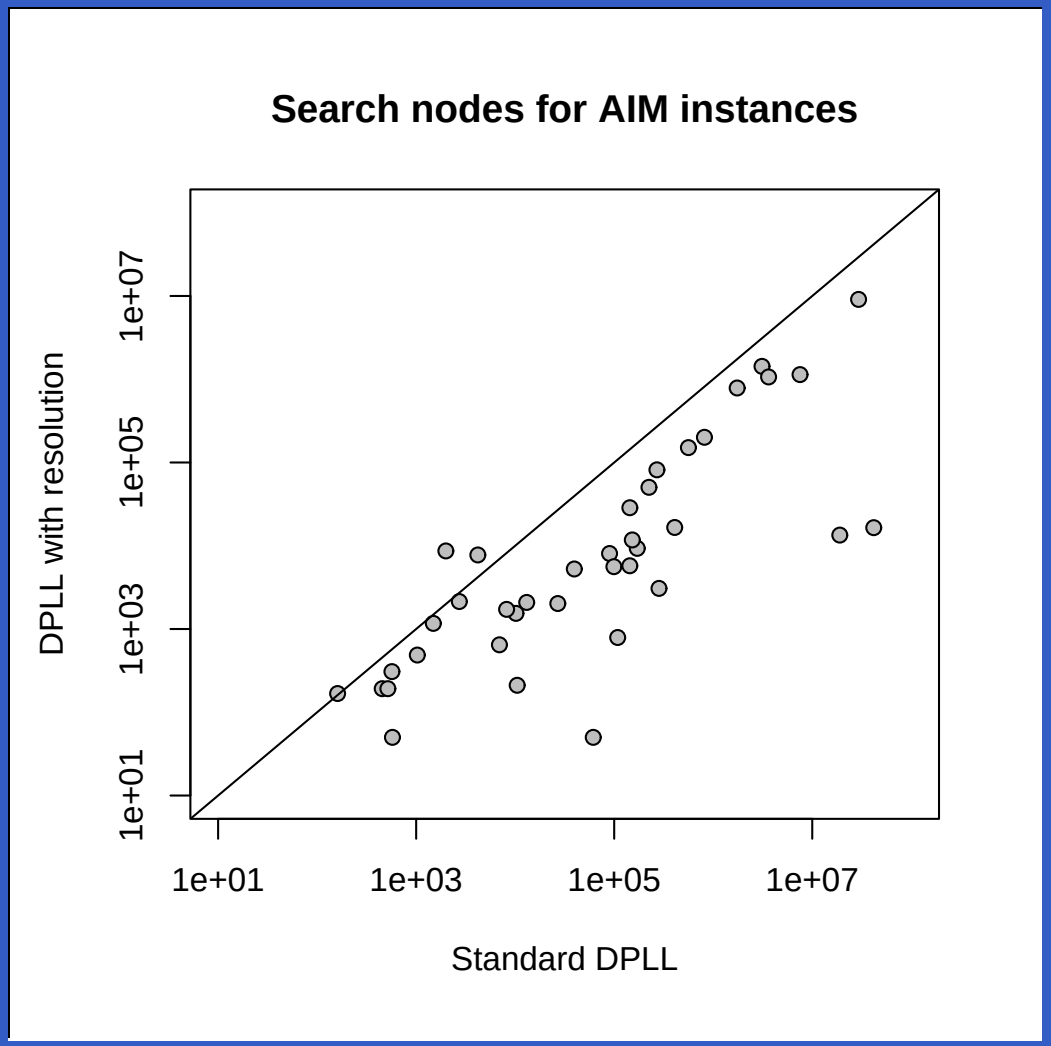
First attempt: during search

- At each branching node, we identified all current neighbours and resolved them
- Far too expensive in time for practical use, but confirmed the potential value of the technique

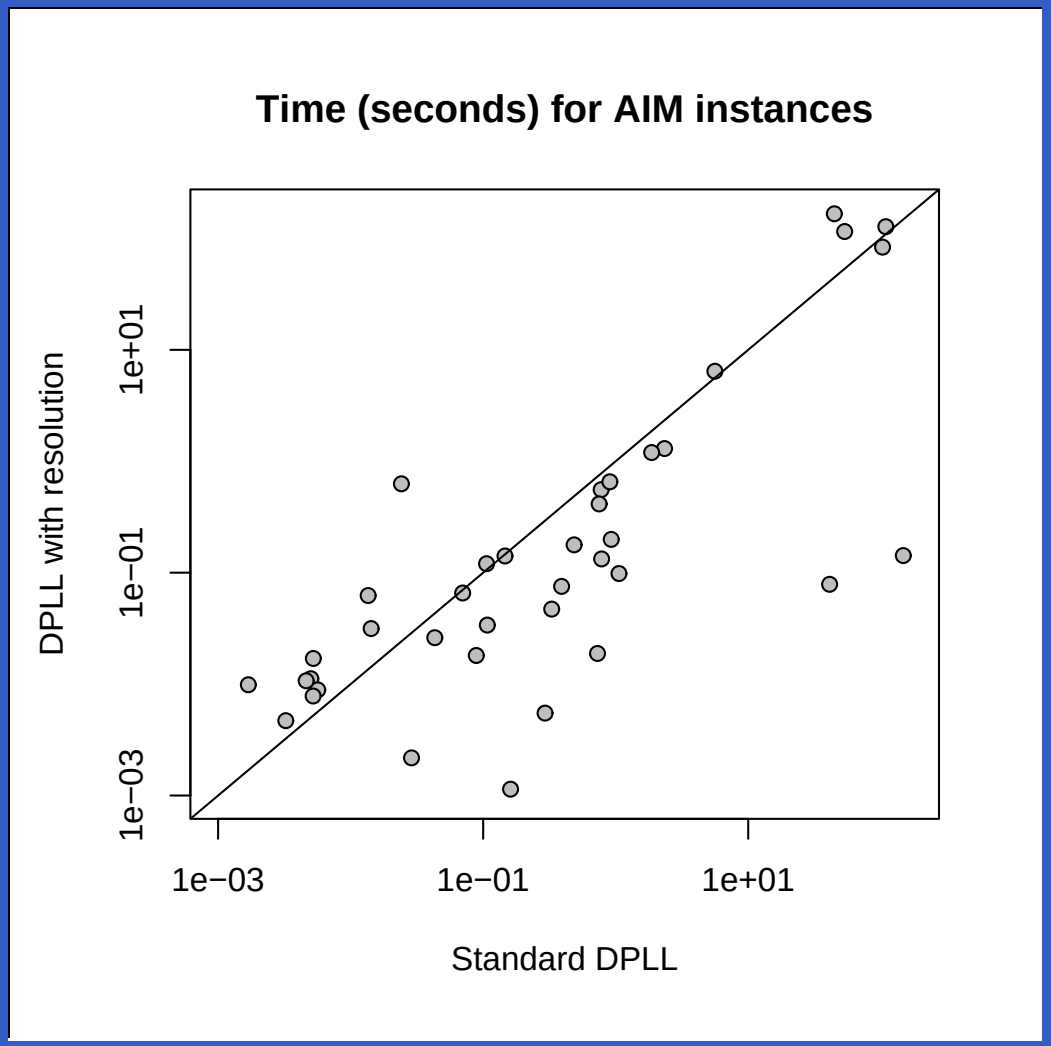
Second attempt: preprocessing

- Doing all binary resolutions at the root of the search tree is equivalent to doing neighbour resolutions at each branching node
- A single search for neighbours saves time, but we still potentially have a large number of new clauses
- By only doing a subset of the possible resolutions for each variable, we can limit the number of added clauses

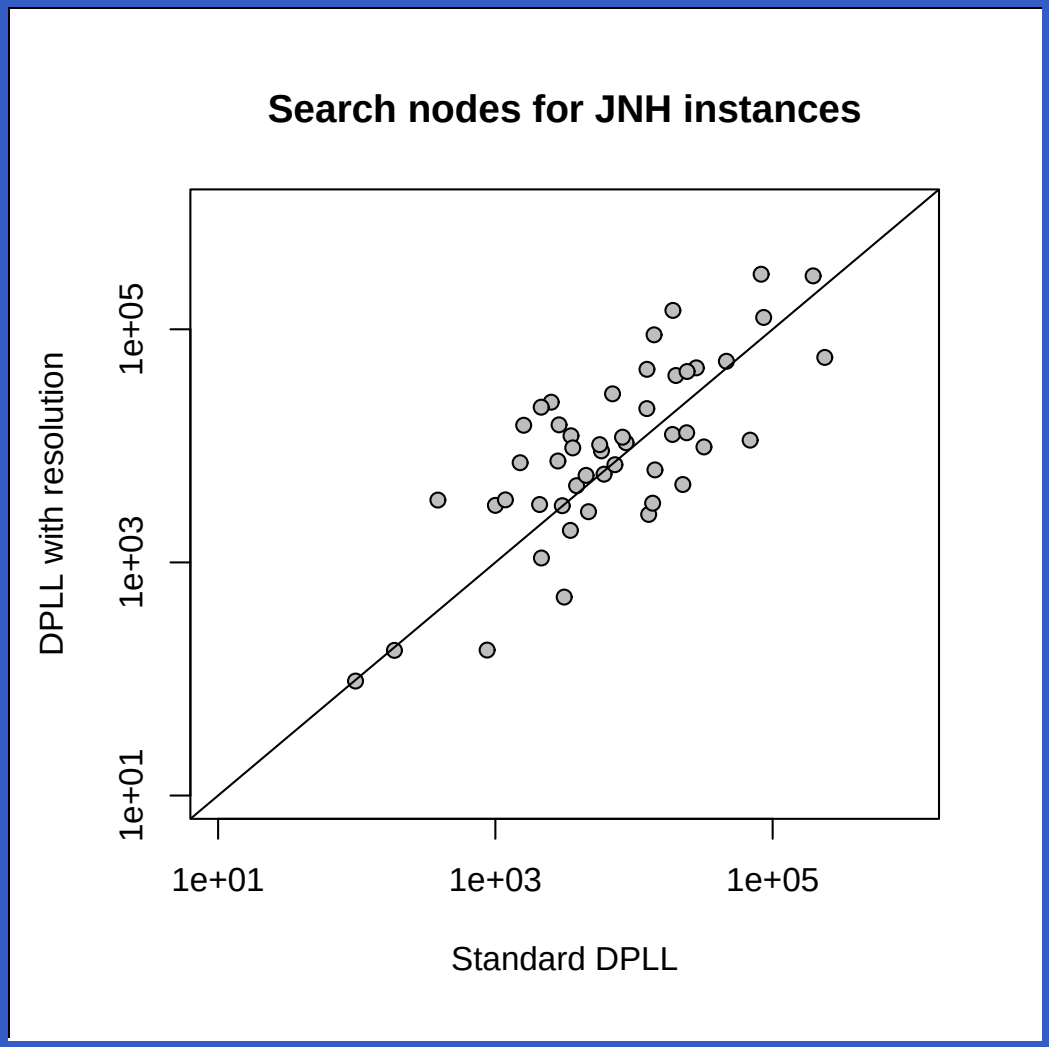
Results: search nodes on AIM instances



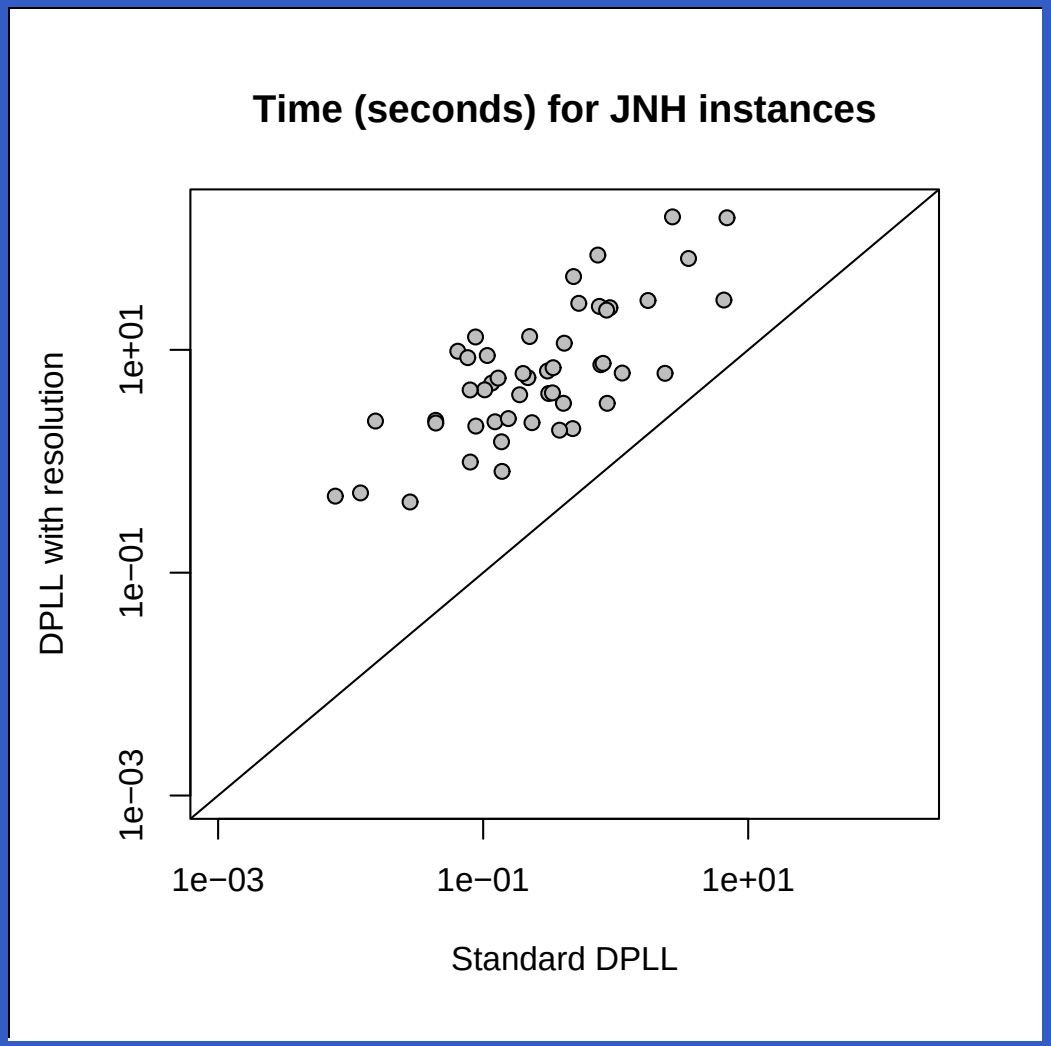
Results: search time on AIM instances



Results: search nodes on JNH instances



Results: search time on JNH instances



Future work

- Making the preprocessing more closely simulate neighbour resolution:
  - choice of resolutions
  - subsumption of parents
- Investigate interactions with other techniques for generating implied clauses, and if possible devise a theoretical framework.
- Look at interactions with variable ordering heuristics: initially static orderings, but perhaps extending it to dynamic ones

Motivation for writing Swan

Existing SAT solvers are:

- difficult to understand
- difficult to modify

Swan

Swan is:

- written in C
- designed to be easy to modify (though perhaps I should have chosen a different language)
- counter based
- currently being rewritten

Future work on Swan

- Simpler code: more abstraction and modularity
- Lazy data structures
- Clause recording
- Restarts

Summary

- We have taken neighbour resolution and applied it to a complete SAT solver
- Using neighbour resolution as a preprocessing step shows some promise