Adding resolution to the DPLL procedure for Boolean satisfiability

Lyndon Drake, Alan Frisch and Toby Walsh

lyndon@cs.york.ac.uk, frisch@cs.york.ac.uk and tw@4c.ucc.ie

University of York, United Kingdom and 4C, UCC, Ireland

Introduction

- Combining inference and search can be a useful technique for solving SAT problems
- We present two techniques that combine inference and search:
  - A restriction of resolution, called neighbour resolution, applied during search
  - Resolution during preprocessing that generates the same clauses as neighbour resolution during search

Neighbour resolution examples

$$\frac{\begin{array}{ccc} d & \vee & \neg b \\ \neg d & \vee & \neg b \end{array}}{\neg b}$$

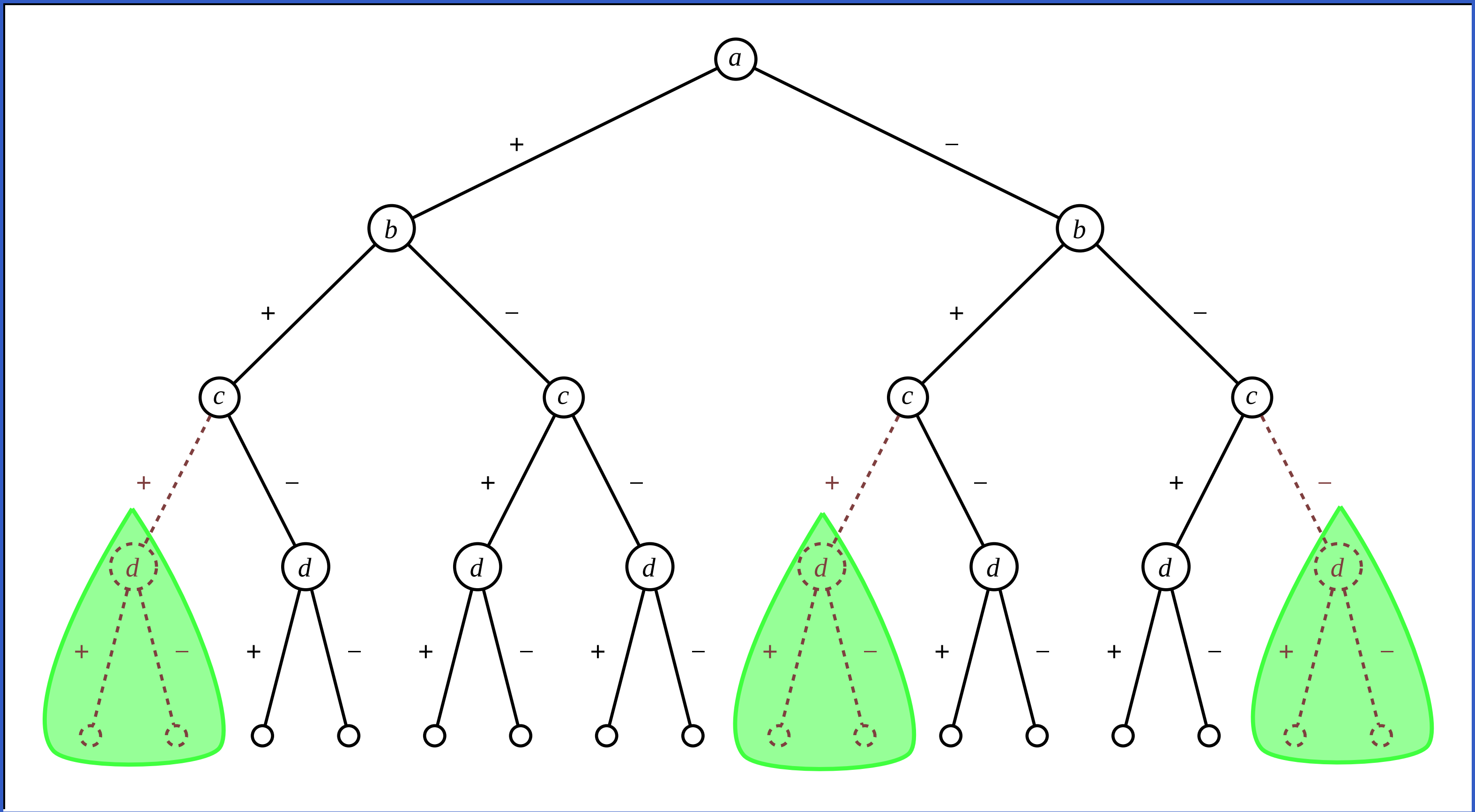
$$\frac{\begin{array}{ccccc} d & \vee & \neg b & \vee & a \\ \neg d & \vee & \neg b & \vee & a \end{array}}{\neg b \vee a}$$

Unsatisfiable SAT instance

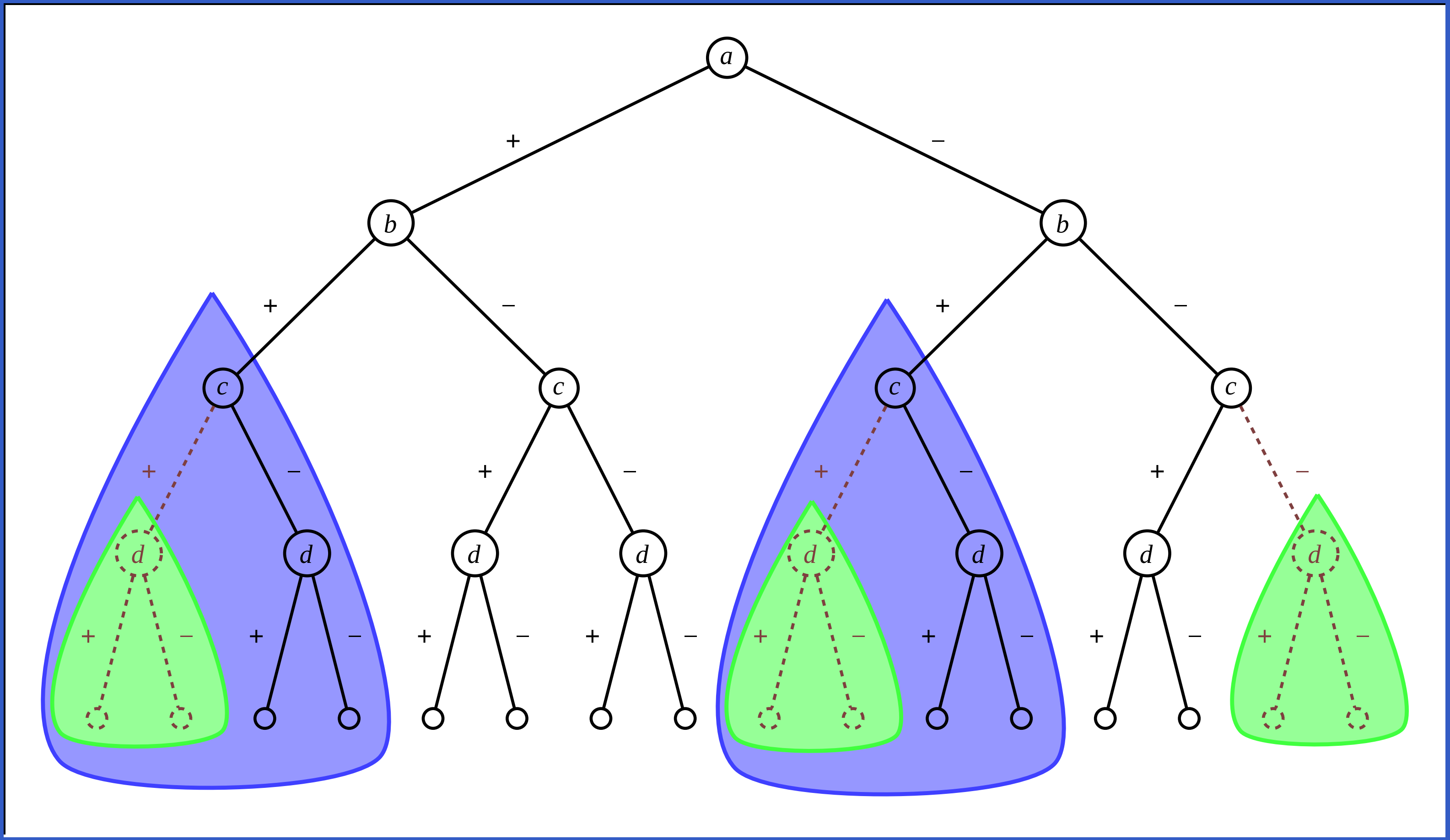
$$\neg a \vee \neg b \vee \neg c$$
$$a \vee \neg b \vee \neg c$$
$$a \vee b \vee c$$
$$\neg b \vee d$$
$$\neg b \vee \neg d$$
$$a \vee c \vee d$$
$$\neg b \vee c \vee \neg d$$

$$a \vee b \vee d$$
$$a \vee \neg c \vee \neg d$$
$$\neg a \vee b \vee d$$
$$b \vee \neg c \vee \neg d$$
$$\neg a \vee c \vee d$$
$$b \vee c \vee \neg d$$

Search tree for DPLL



After neighbour resolution



Inference and search

- DPLL includes some inference (unit propagation)
- A tradeoff exists between inference and search in the DPLL algorithm
- Adding inference to search prunes the search tree
- Inference is expensive, though, so the benefit from the inference must be great than the cost

Related 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.

Neighbour resolution defined

$$\frac{\begin{array}{rcl} w & \vee & X \\ \neg w & \vee & X \end{array}}{X}$$

- The resolvent of a neighbour resolution always subsumes both parents
- Observationally, real-world SAT instances contain very few neighbour resolutions at the root of the search tree
- However, clauses become neighbours during search as literals are deleted

First method: during search

Neighbour resolution during search involves two steps before each branch:

- Adding all neighbour resolvents
- Deleting all the subsumed parents

The effect is to add some implied clauses and at the same time remove some redundant clauses.

Second method: before search

- The binary resolvent of $(a \vee b \vee c)$ and $(\neg a \vee b \vee d)$ is $(b \vee c \vee d)$
- If during the search the clauses become neighbours: $(a \vee b)$ and $(\neg a \vee b)$, then the same assignment will make the resolvent into (b), which is the neighbour resolvent
- Binary resolution before search can simulate neighbour resolution during search
- The current simple implementation adds some extra unnecessary clauses and does not delete subsumed clauses

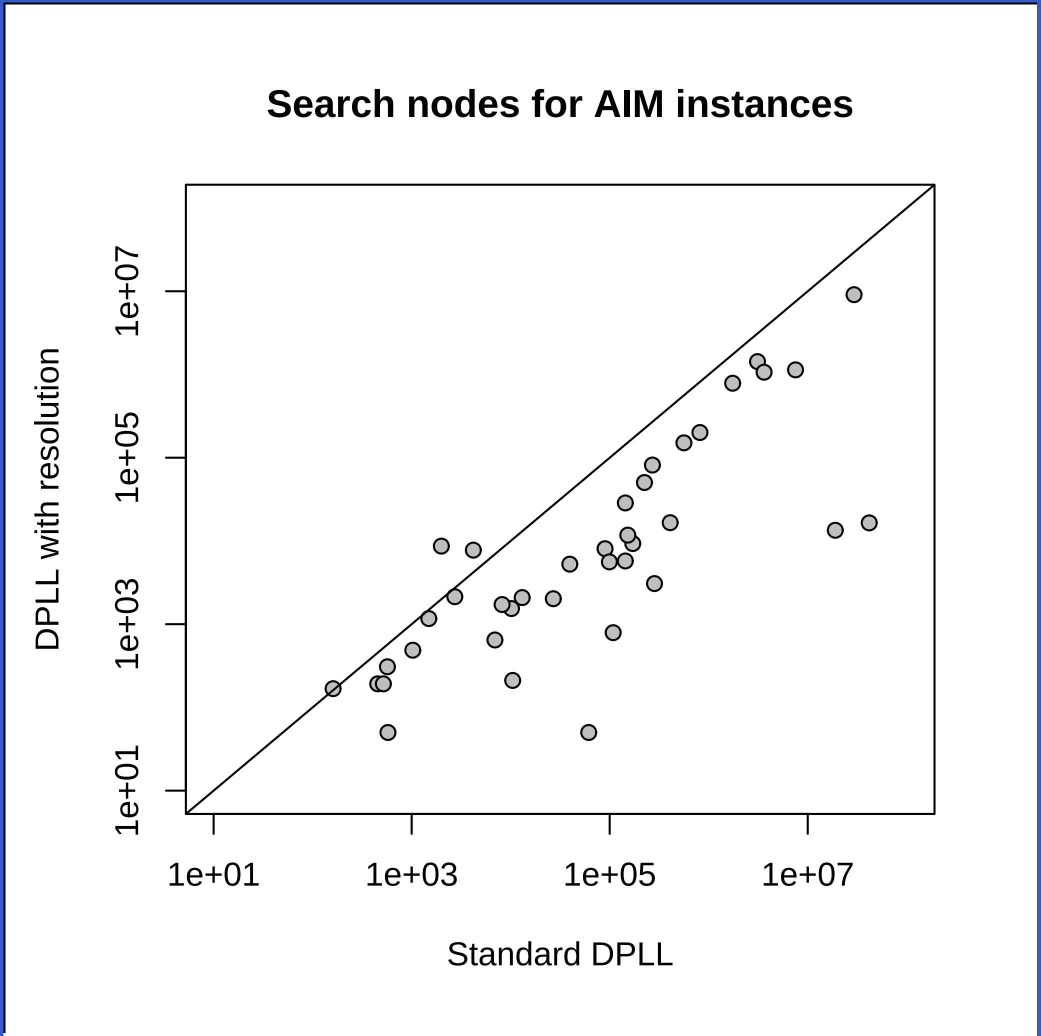
Experimental results: during search

- On observed instances, neighbour resolution during search generally significantly prunes the search tree
- The main problem is that the process of identifying neighbouring clauses is very slow
- The speed cost generally outweighs the pruning benefits, so that neighbour resolution during search is typically not practical
- However, on a few instances, the pruning is enough to outweigh the cost

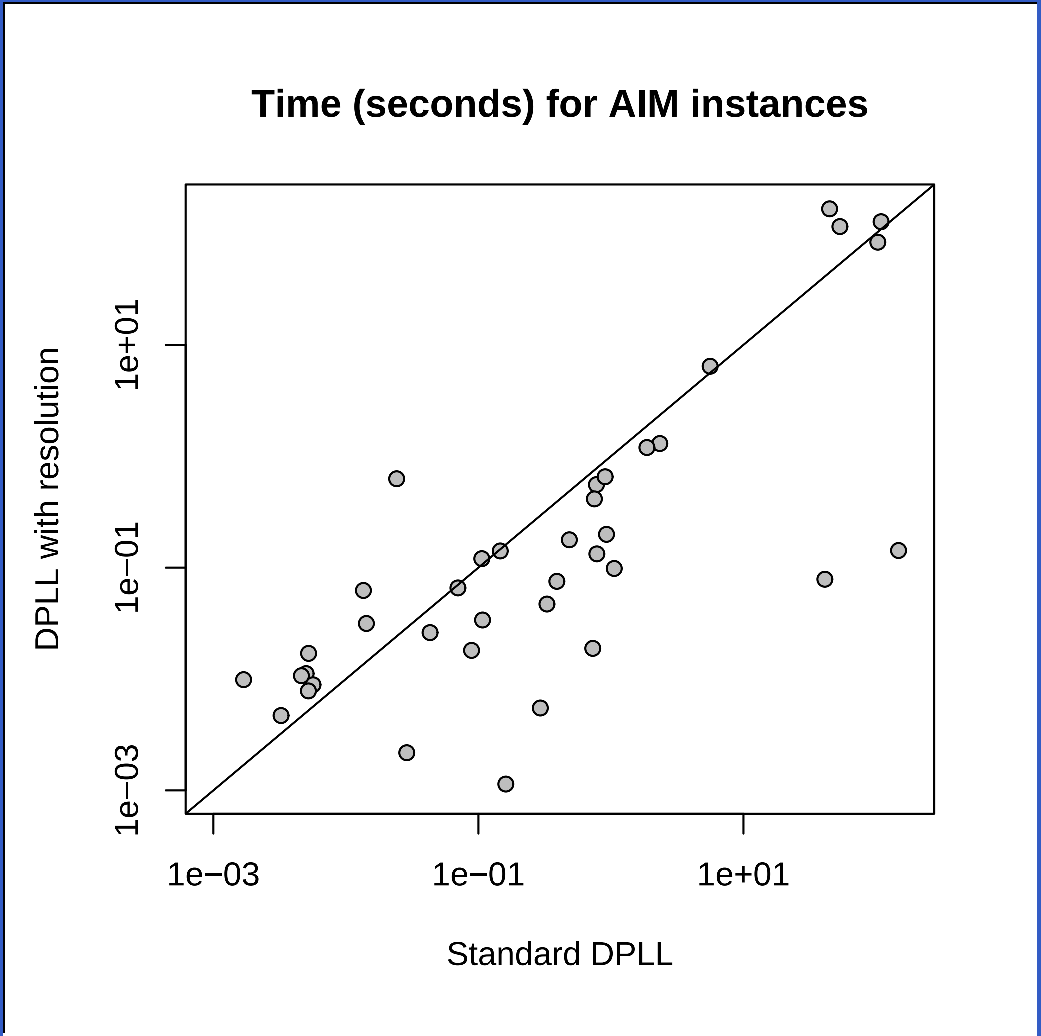
Experimental results: before search

- On observed instances, consistently reduces the number of nodes in the search tree
  - Occasionally increases the size of the search tree, when the added clauses cause the branching heuristic to make poorer decisions
- The cost is low enough that the speed of the solver is normally improved
- Main exception is that the JNH SATLIB instances perform worse; note that these instances are random

Experimental results: search nodes



Experimental results: search time



Future work: improved simulation

In our current implementation of simulated neighbour resolution:

- Subsumption during search is ignored
  - We can mark resolvent clauses and cheaply apply subsumption to just those clauses during search
- Extra resolvents (not corresponding to actual neighbour resolvents) are added
  - We can use a static branching heuristic to predict which resolvents are unlikely to correspond to actual neighbour resolvents

Future work: improved implementation

- Neighbour resolution during search is slow because identifying neighbouring clauses is expensive
  - We have an improved algorithm for neighbour identification which we plan to implement
- It is not worth applying resolution to some problem classes (e.g. the JNH SATLIB instances)
  - We are developing syntactic methods for identifying some such problem classes

Future work: investigation

- If implied clauses are visible to the branching heuristic, the search tree may actually be grow instead of being pruned
  - We plan to investigate the effect of including the implied clauses, but making the branching heuristic ignore them

Conclusion

- Using preprocessing to simulate resolution during search can be an effective technique
- As a general research question, we want to know which inferences are useful in practice; how can we identify such inferences?