Learning Software Language DFAs with SAT solvers

Marijn Heule[1], Sicco Verwer[2]

1. Delft University of Technology
2. KU Leuven

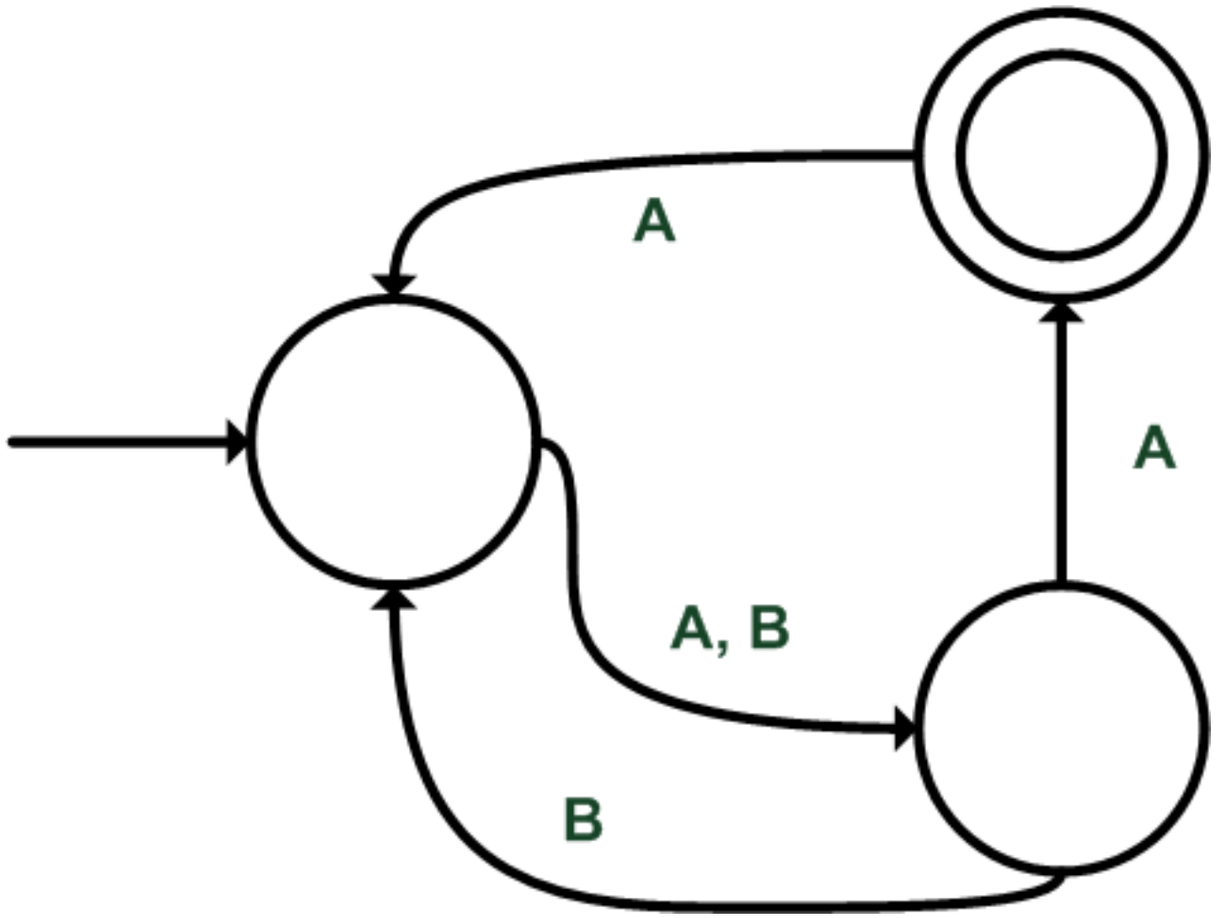
Overview

- Why use satisfiability to solve DFA learning?
- Learning DFAs as graph coloring
- Encoding DFA learning into satisfiability
- Results
- Conclusions

Learning DFA

- Given data (a set of positive and negative strings)
- Find the smallest DFA that is consistent with the data (that can produce these positive and negative stings)

Example

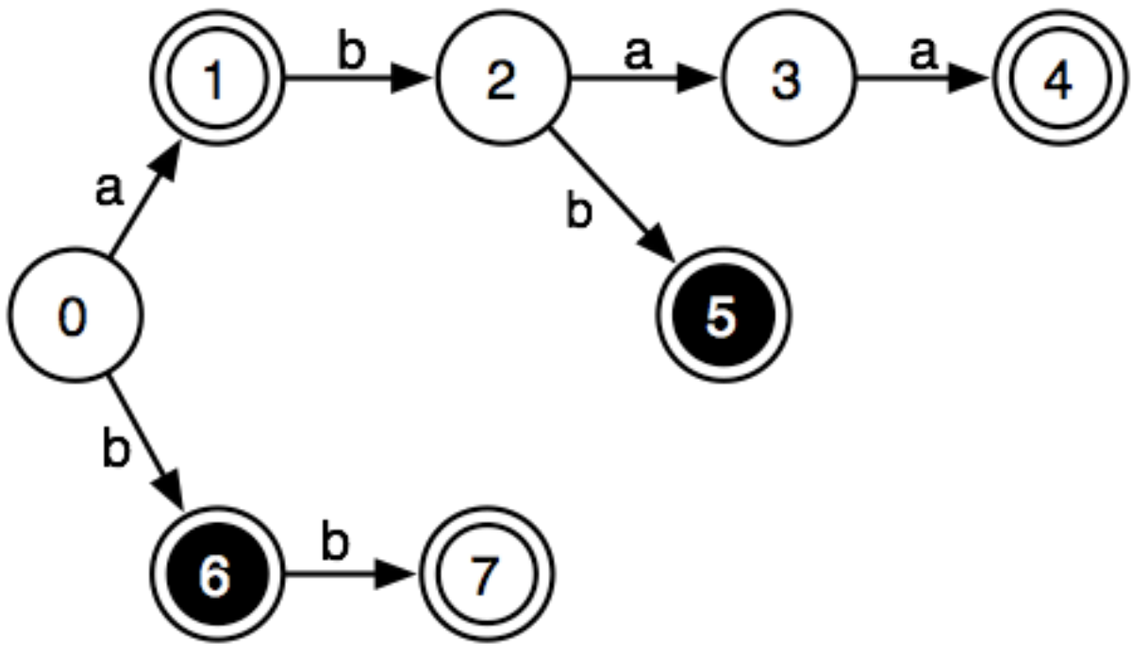


Accepts: AA,BA,ABBA,...
Rejects: A,B,AAA,ABA,...

Why use satisfiability?

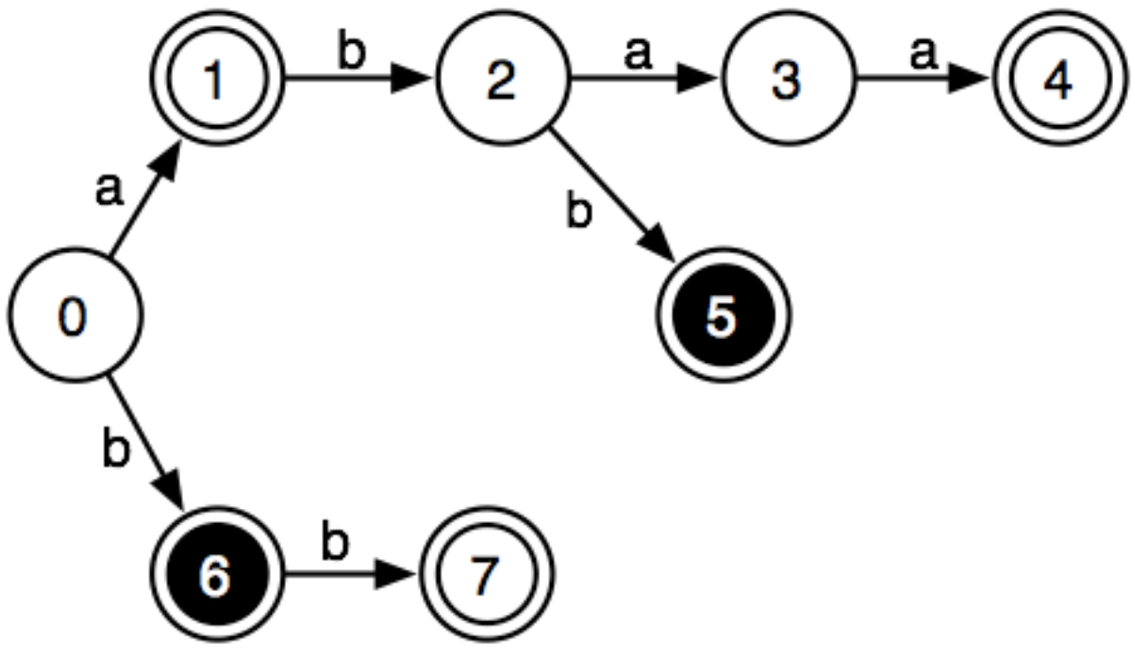
- Satisfiability is one of the best studied problems in computer science
- SAT-solvers are highly optimized and use advanced search techniques
- We can make direct use of these advanced techniques by encoding DFA learning to satisfiability and running a SAT-solver
- Such an approach has been shown to be competitive for several well-known problems

Prefix trees



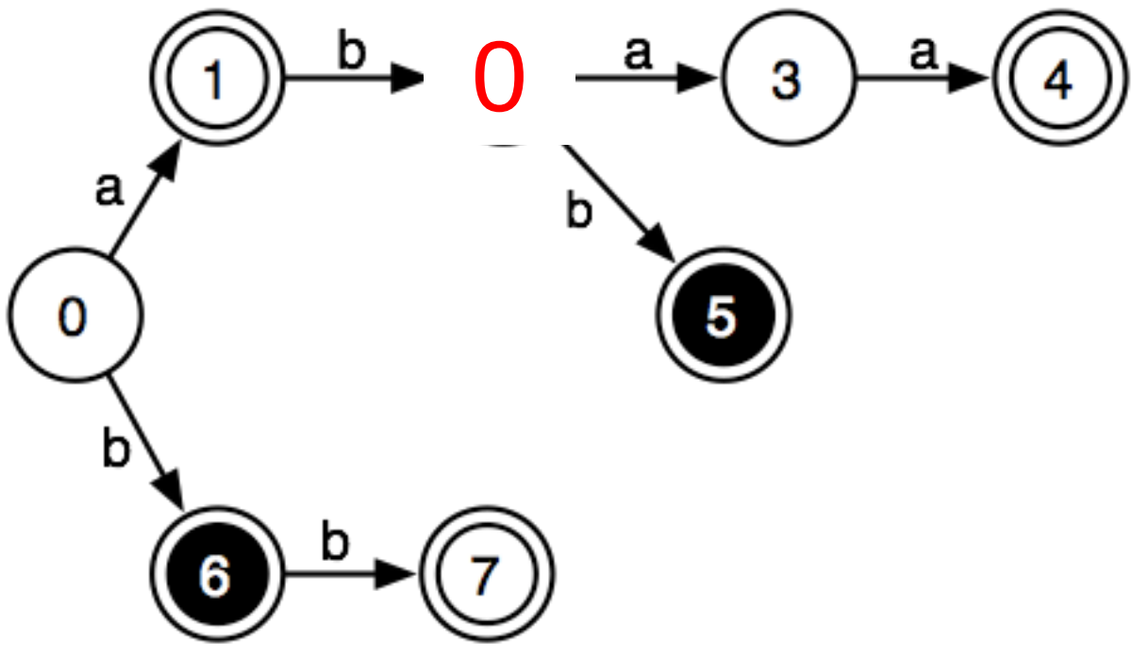
Representation of a labeled data set:
S+ = {a, abaa, bb}, S- = {abb,b}

Merging



Find sets of equivalent states:
If state x and y are equivalent then x and y model the same state in a DFA

Merging

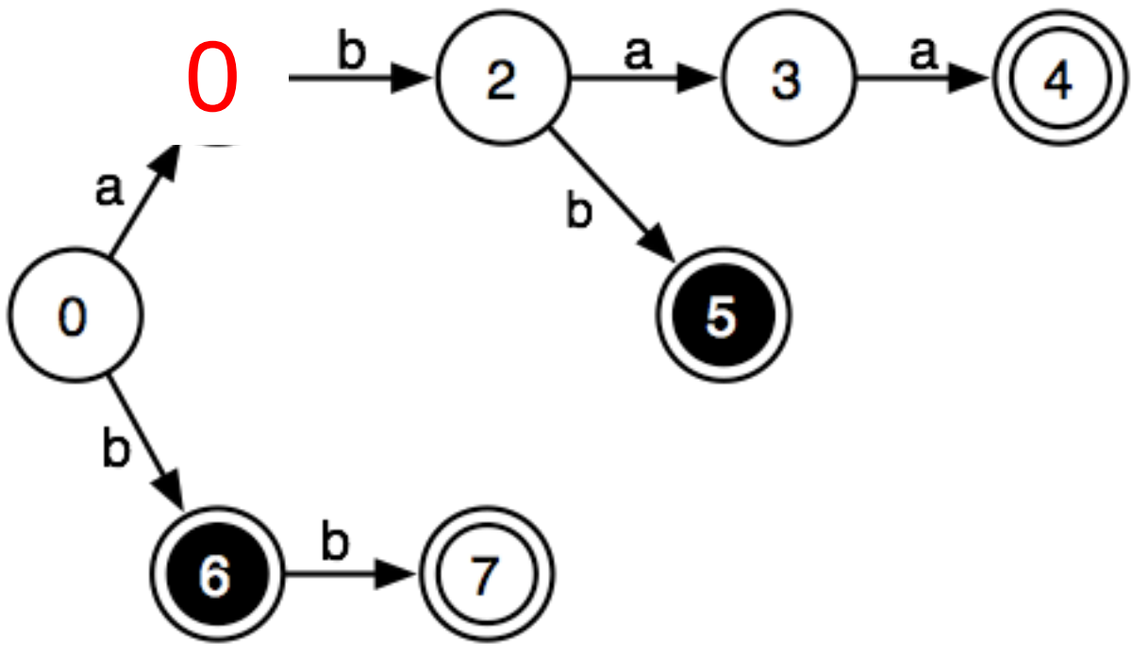


If 0 and 2 are equivalent then after reading ab, the DFA is back in state 0:

L+ = {a, abaa, bb, aba,ababaa,abbb,ababa,…},

L- = {abb,b,ababb,abb,abababb,…}

Merging



0 and 1 cannot be equivalent since:

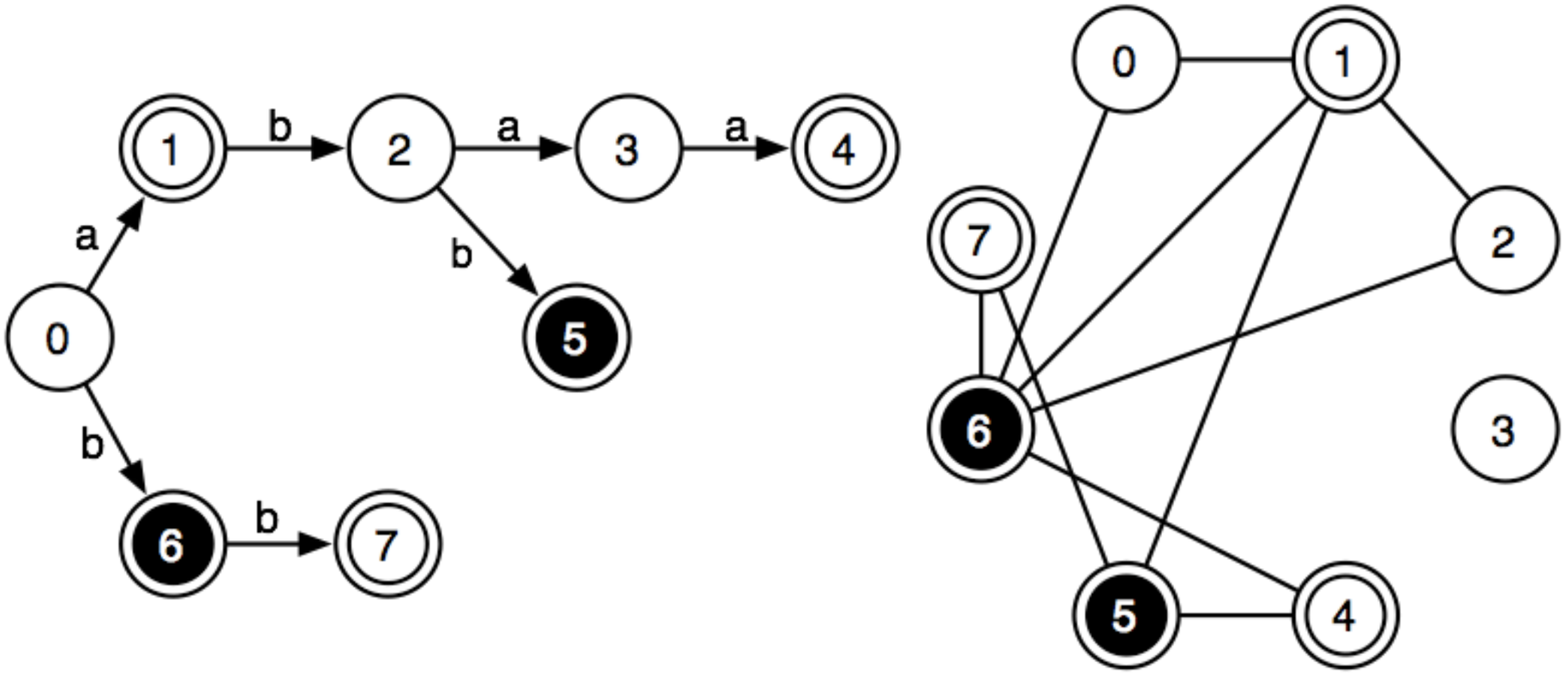
L+ = {λ, a, abaa, bb, aa,aabaa,abb,aaa,…},

L- = {abb,b,aabb,ab,aaabb,…}

Graph coloring

- Given a graph G = (V, E) and a value k
- Is G colorable using k colors such that no two adjacent nodes have the same color?

from DFAs to GC



Start with a prefix tree

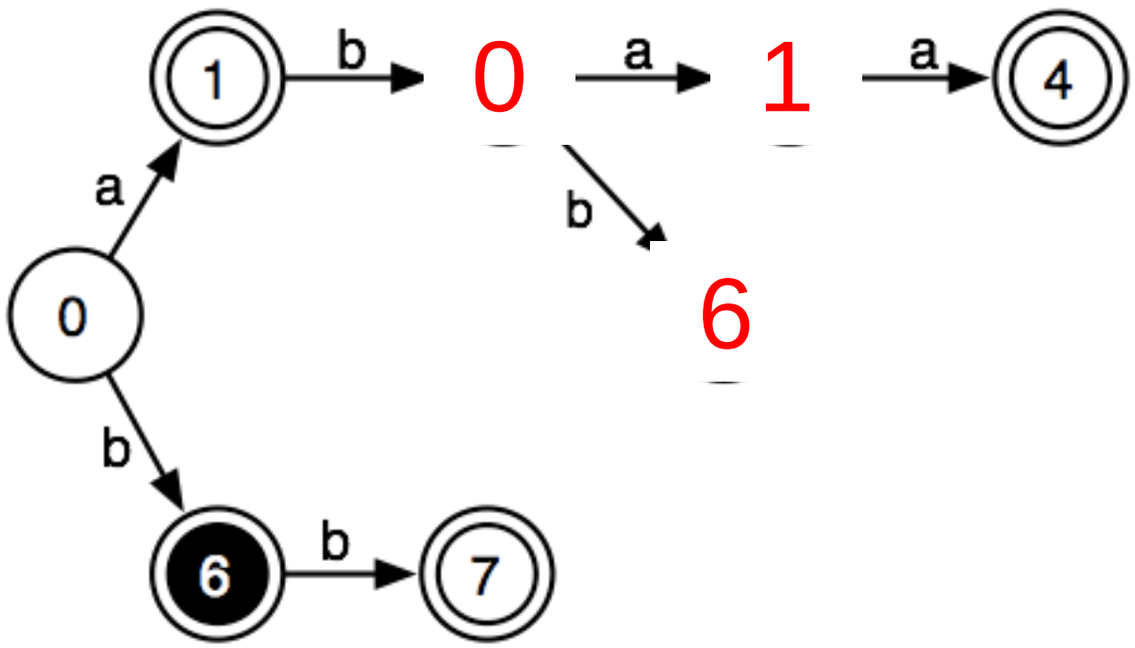
Create one node for every state

connect two nodes if the states cannot be equivalent

from DFAs to GC

- This encodes inequality constraints:
  - restricts when two nodes cannot have the same color
- We also require equality constraints:
  - two nodes can be colored with the same color only if their children reached by the same label have the same color…

Equality



0 and 2 can be equivalent only if 1 and 3 are equivalent, and 5 and 6 are equivalent

Such a constraint is difficult to implement in graph coloring

Satisfiability

- Given a formula *F* in propositional logic
- Does there exist an assignment of truth values to the variables of *F* such that each clause in *F* is satisfied?

Standard GC to SAT

- Variable $x_{v,i}$ denotes that vertex v has color i
- The following then encodes graph coloring:

$(\neg x_{v,i} \vee \neg x_{w,i})$ for each edge

$(x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,n})$ for each node

- Where v and w are adjacent vertexes, i.e., states that cannot be merged, this encodes inequality
- Requires $O(|C| \times |E|)$ clauses

Simple equality in SAT

- If *v* and *w* have the same incoming label this then encodes the equality constraints:

$$x_{p(v)} = x_{p(w)} = i \rightarrow x_v = x_w = j$$

  - $(\neg x_{p(v),i} \vee \neg x_{p(w),i} \vee x_{v,j} \vee \neg x_{w,j}) \wedge$
  - $(\neg x_{p(v),i} \vee \neg x_{p(w),i} \vee \neg x_{v,j} \vee x_{w,j})$

- Requires $O(|C|^2 \times |V|^2)$ clauses
- Unfortunately, this is too large for state-of-the-art SAT solvers

More efficient equality

- Additional variables:
  - $y_{a,i,j}$ denotes that for all vertexes with color *i*, the child reached by label *a* has color *j*
- This then encodes the equality constraints:
  - $(\neg x_{p(v),i} \vee \neg x_{v,j} \vee y_{l(v),i,j})$
  - $(\neg y_{a,i,j} \vee \neg y_{a,i,k})$
- Requires $O(|C|^2 \times |V|)$ clauses

More efficient inequality

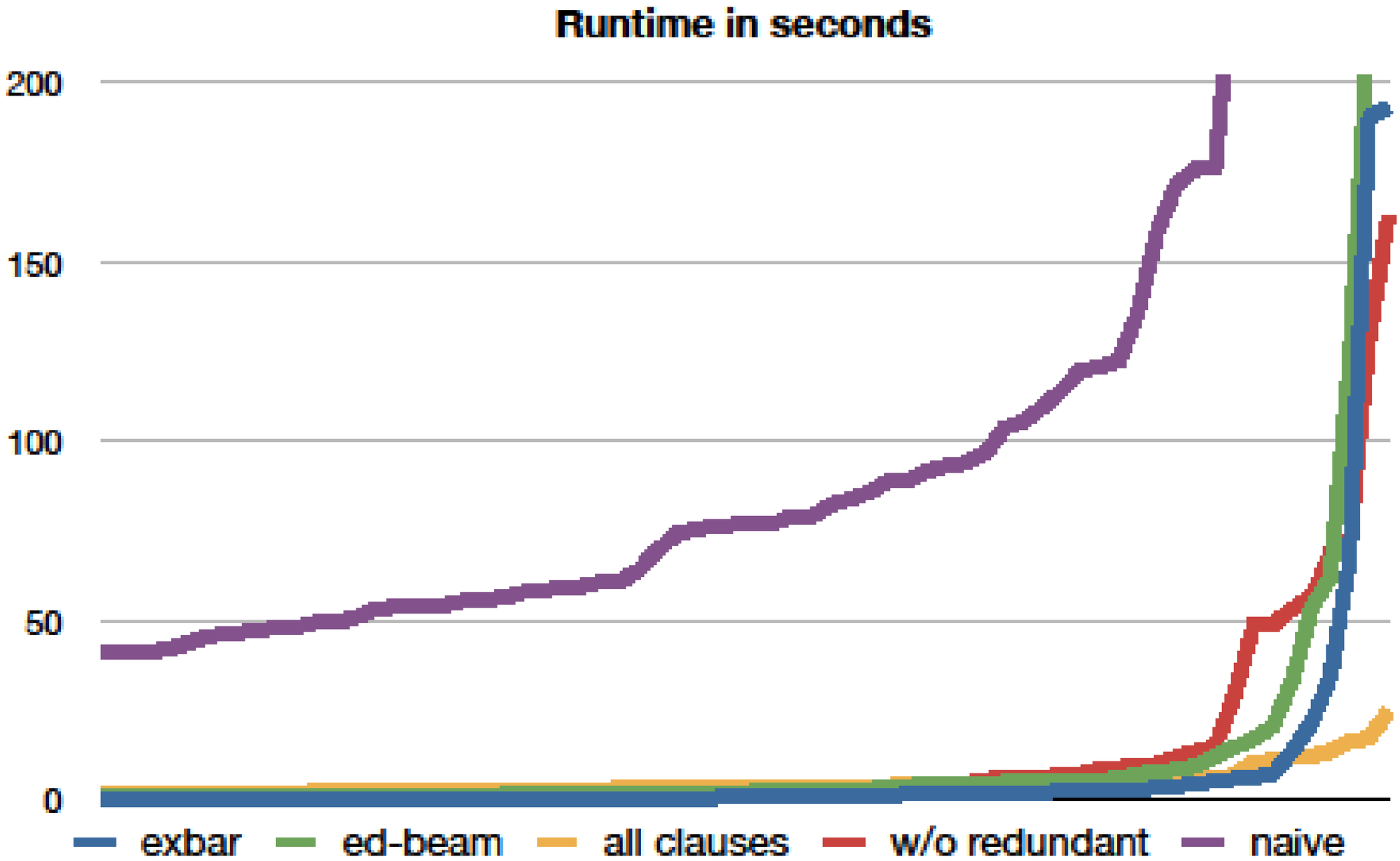
- Additional variables:
  - z_i denotes that color i is used for accepting states
- This then encodes the inequality constraints:
  - $(\neg x_{v,i} \vee z_i) \wedge (\neg x_{w,i} \vee \neg z_i)$
  - Where node v is accepting and node w is rejecting

- Requires $O(|C| \times |V|)$ clauses
- The original GC constraints are now redundant

Learning automata by SAT

1. Generate the prefix-tree T
2. Compute a large clique C in T
3. Initialize the set of color L, where $|L| = |C|$
4. Construct a SAT formula F using T, C and L
5. Solve F using a SAT solver
6. If F is unsatisfiable add a color to L, goto 4
7. Return the DFA found in step 5

Results


Runtime in seconds
200
150
100
50
0
exbar
ed-beam
all clauses
w/o redundant
naive

Using a SAT subroutine

- This translation does not require a prefix tree as initial DFA
- Any partially constructed DFA can be used
- Thus we can:
  - Apply a few steps of a greedy DFA learner (EDSM)
  - Apply our translation to SAT

- Using this approach we were able to solve win the STAMING software language (DFA) learning competition

Learning software languages

- Learn DFAs of size approx 50, with alphabet of size 5,10,20,50

- Approach:
  - Modify the heuristic of EDSM to fit software languages
  - Run EDSM until 45 states
  - Check with SAT whether a solution of size 50 exists
  - Return the solution

- Running only the greedy algorithm nearly always results in over 100 states when a solution of size 50 exists!

Conclusions

- We have constructed an efficient translation from DFA learning to SAT
- The translation is based on graph coloring
- We include symmetry breaking predicates and redundant clauses to help the SAT solver
- By first running a greedy algorithm and then applying the translation we obtain a very powerful algorithm.
- It is competitive with the current state-of-the-art