AUTOMATICALLY IMPROVING CONSTRAINT MODELS IN SAVILE ROW THROUGH ASSOCIATIVE-COMMUTATIVE COMMON SUBEXPRESSION ELIMINATION

Peter Nightingale
Özgür Akgün
Ian P Gent
Chris Jefferson
Ian Miguel

MOTIVATION – THE LONG-TERM GOAL

Automated Reformulation – Given a naïve model, automatically improve it via a sequence of reformulations

We can take inspiration from two main sources:

- Compilers
- Automating techniques used by expert modellers

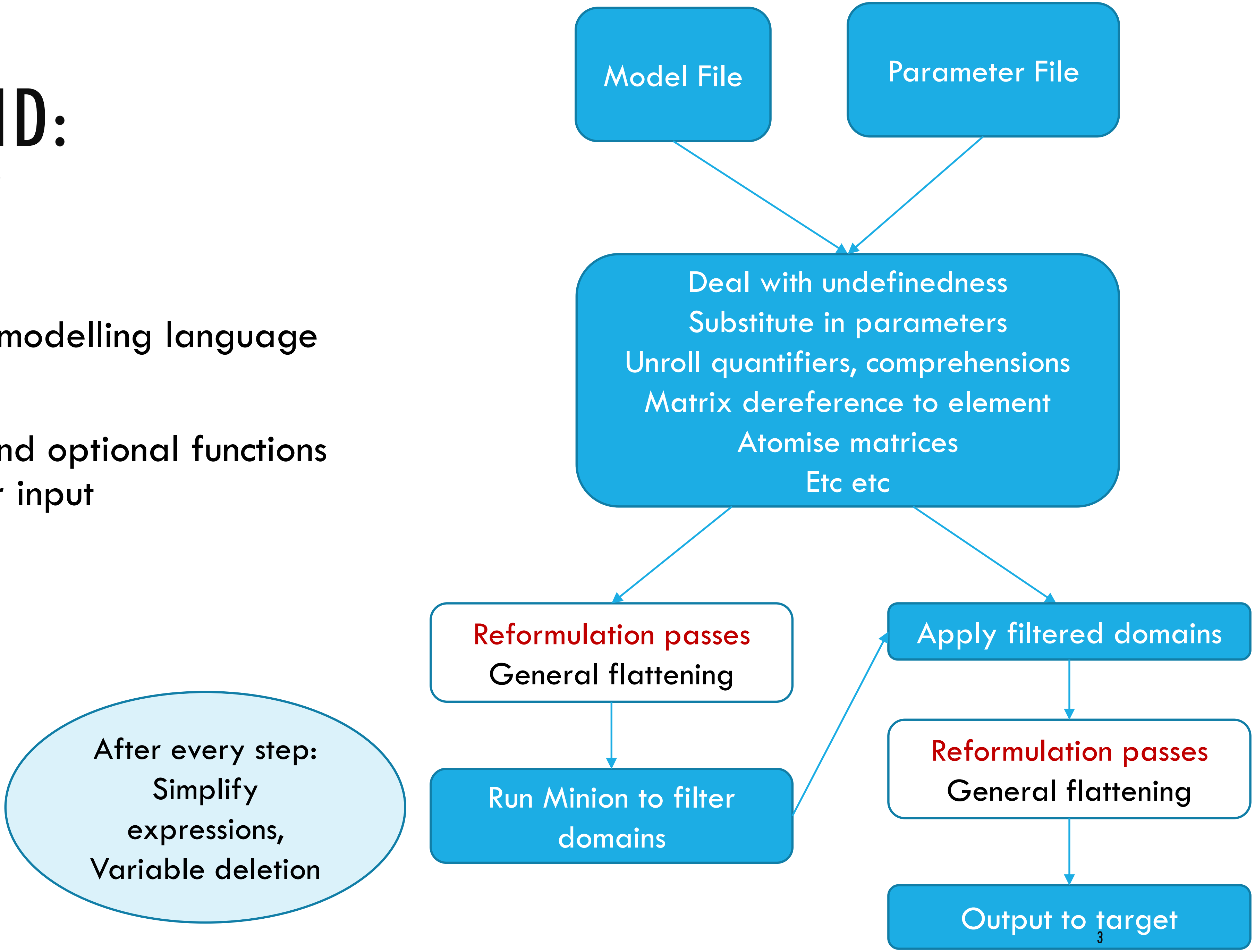
Typically stronger reformulations can be done on problem instances

This paper describes an algorithm for an instance-level reformulation – AC-CSE

BACKGROUND: SAVILE ROW

A system that takes modelling language Essence'

Performs essential and optional functions to translate to solver input



BACKGROUND: IDENTICAL COMMON SUB-EXPRESSION ELIMINATION

The most basic Common Sub-Expression Elimination (CSE) extracts identical expressions

$$x+y \leq 4, \quad x+y=z$$

$$\text{where } x,y \text{ in } \{0..5\}, z \text{ in } \{0..10\}$$

Extract x+y and replace with new *auxiliary variable*

$$x+y=aux, aux \leq 4, aux=z$$

This allows the constraint solver to see that $z \leq 4$

Without CSE, the solver can discover that $z \leq 8$ (via $x \leq 4$ and $y \leq 4$)

CSE connects together overlapping constraints via a new variable

ASSOCIATIVE-COMMUTATIVE COMMON SUBEXPRESSION ELIMINATION

Associative-Commutative Common Subexpression Elimination (AC-CSE) uses the fact that sum, product, conjunction, disjunction are associative and commutative

In practice treat sums, products, conjunctions and disjunctions as unordered sets

Extract common subsets from sets of expressions

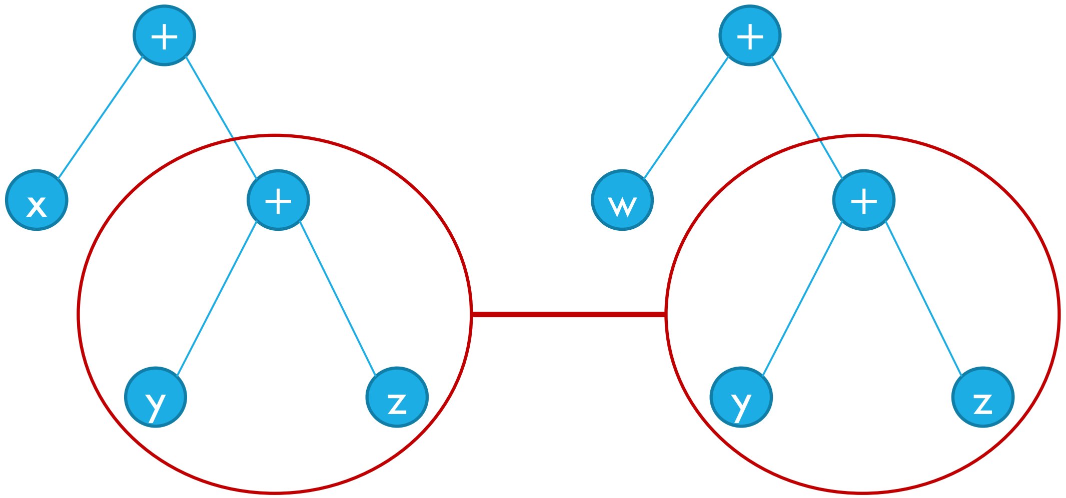
ASSOCIATIVE-COMMUTATIVE CSE

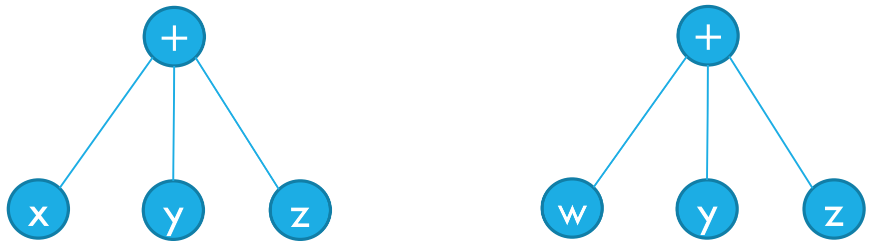
We have already sorted associative-commutative expressions (normalisation)

- x+y+z matches z+x+y with Identical CSE

But we cannot yet match arbitrary overlaps

- A binary tree representation would allow matching prefixes (left-branching) or postfixes (right-branching) in the sorted order… GNU C++ compiler does this
- …but Savile Row represents AC operators using non-binary trees



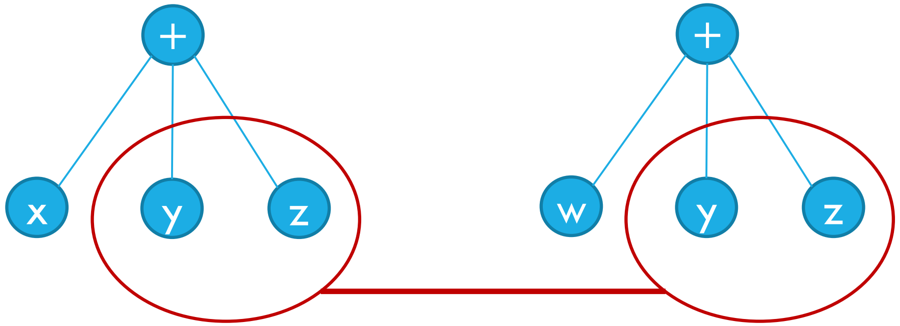


ASSOCIATIVE-COMMUTATIVE CSE

Treat an AC expression as a set of terms

Find a subset common to two or more AC expressions

Extract the common subset everywhere and replace with an auxiliary variable



Can improve propagation dramatically

With some sensible assumptions, never reduces propagation

ASSOCIATIVE-COMMUTATIVE CSE

Extracting one AC-CS may block others

X-CSE (our proposed algorithm) uses heuristic ordering

- Extracts AC-CS with most occurrences first
- Never copies original expressions – can be more efficient in finite-domain context

I-CSE [Araya et al, CP 2008] extracts all AC-CSs between two expressions

- Makes copies of original expressions – potential big slowdown
- Context: Numerical CSP

Genuine choice – difficult to know right answer

Our experiments suggest X-CSE better in finite-domain context

EXPERIMENTS

Comparing total time (solving plus Savile Row time)

With AC-CSE (implemented by X-CSE algorithm) vs same config without AC-CSE

Many other options switched on: domain filtering, variable deletion, etc

Minion as the solver

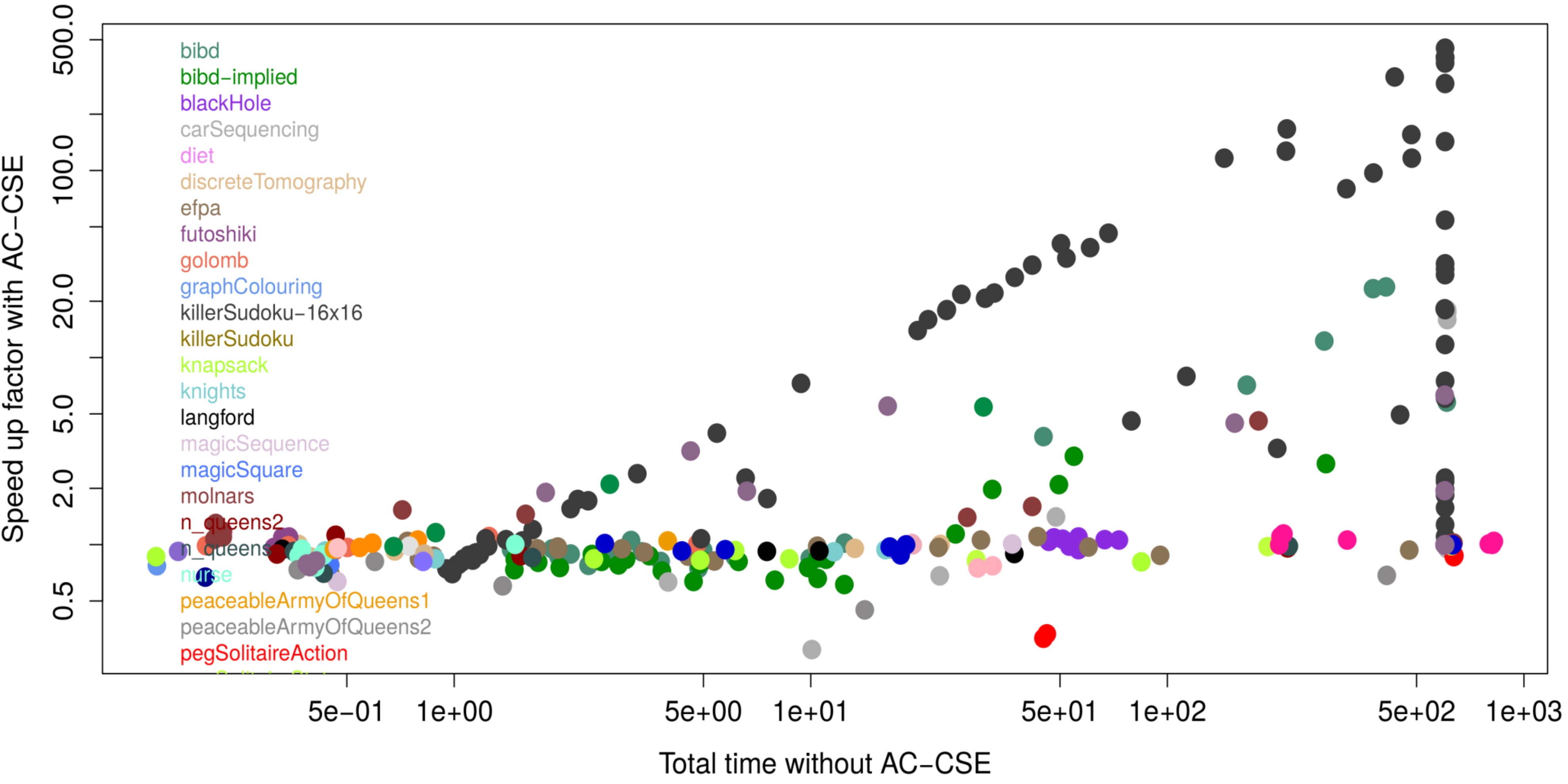
- Static variable, value orderings

AC-CSE never increases search

- But can increase Minion time, Savile Row time.

Results on following slides are from a slightly newer Savile Row than paper

EXPERIMENTAL RESULTS

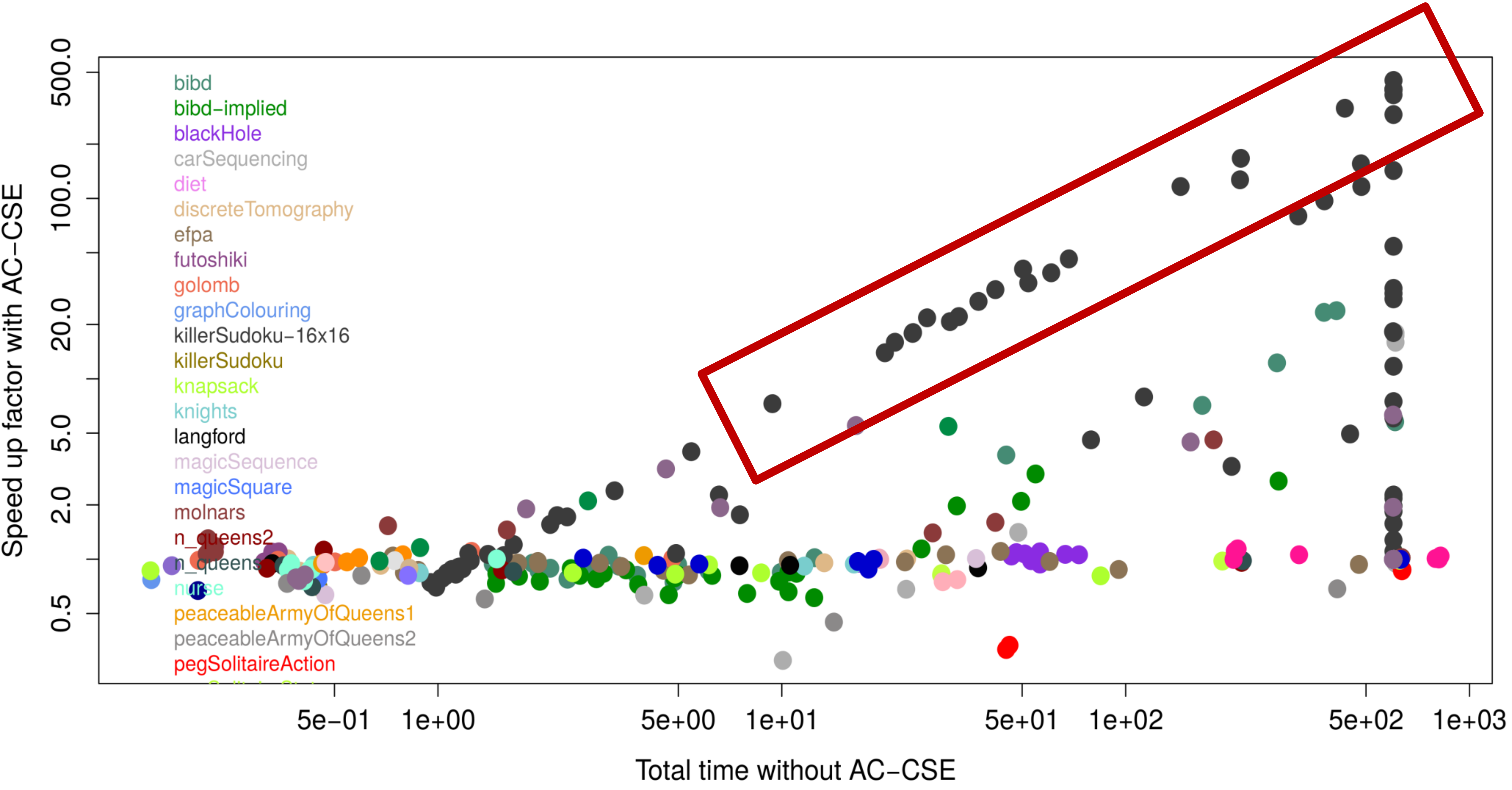


X-axis : total time without AC-CSE

Y-axis: speed-up quotient by applying AC-CSE

Y-axis on log scale

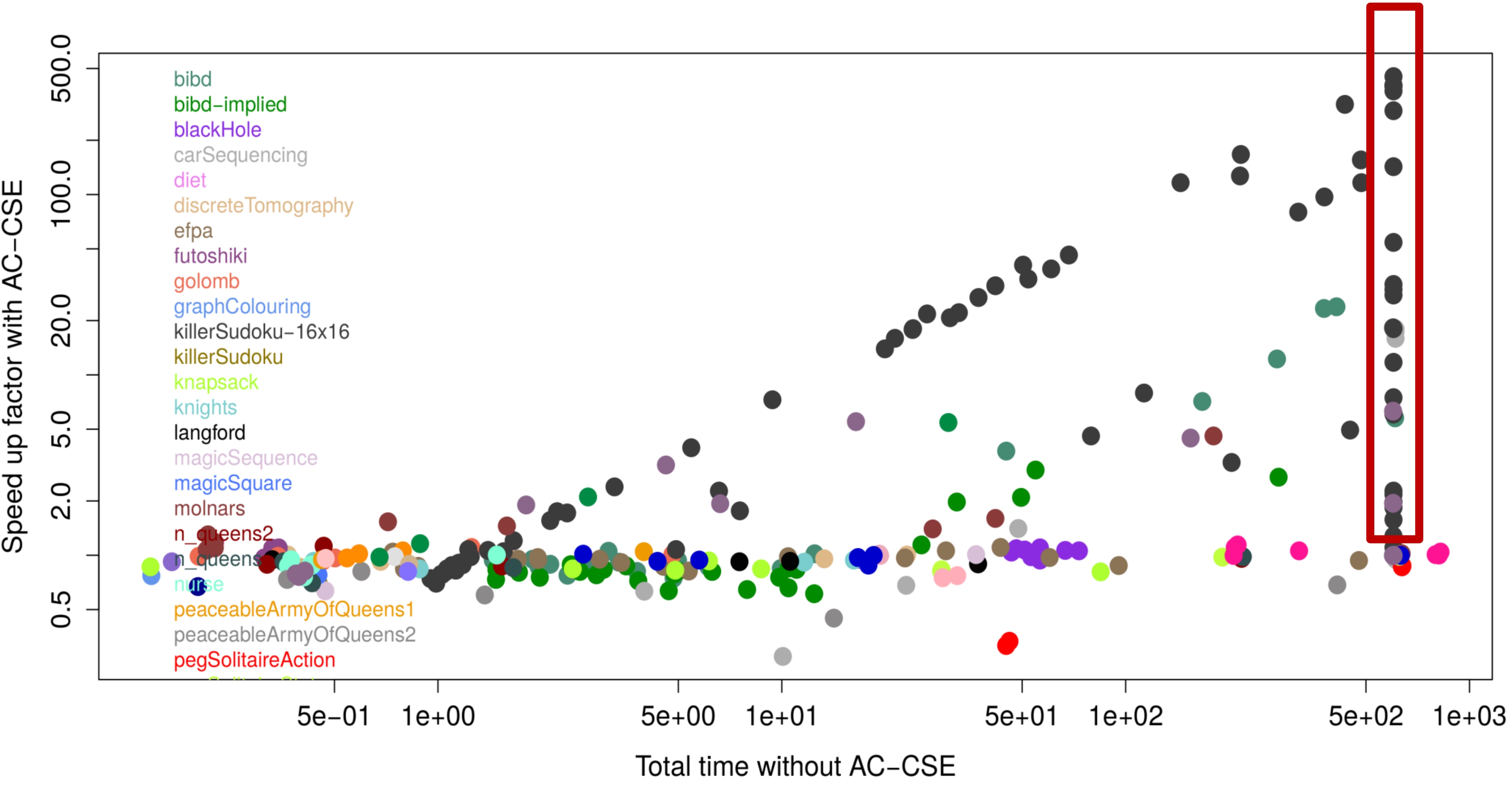
EXPERIMENTAL RESULTS



Some Killer Sudoku instances are rendered almost trivial by AC-CSE

Seems to have exponential growth in speed-up quotient

EXPERIMENTAL RESULTS



Some instances solve within 10 minutes with AC-CSE, time-out without it

Killer Sudoku (24)

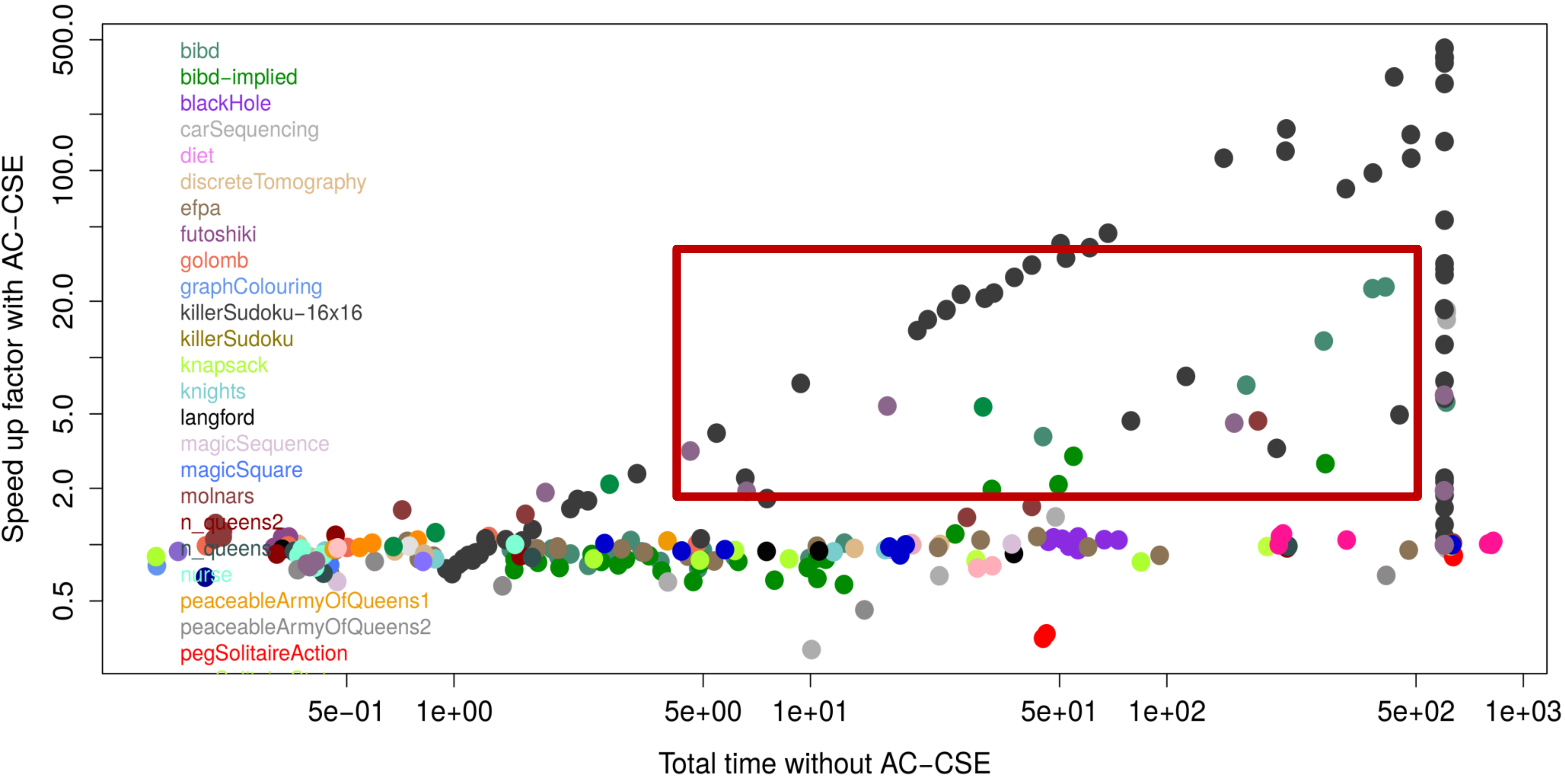
Car Sequencing (3)

SONET (2)

BIBD (2)

BIBD-implied (1)

EXPERIMENTAL RESULTS



The rest above 2x

BIBD (5)

BIBD-implied (3)

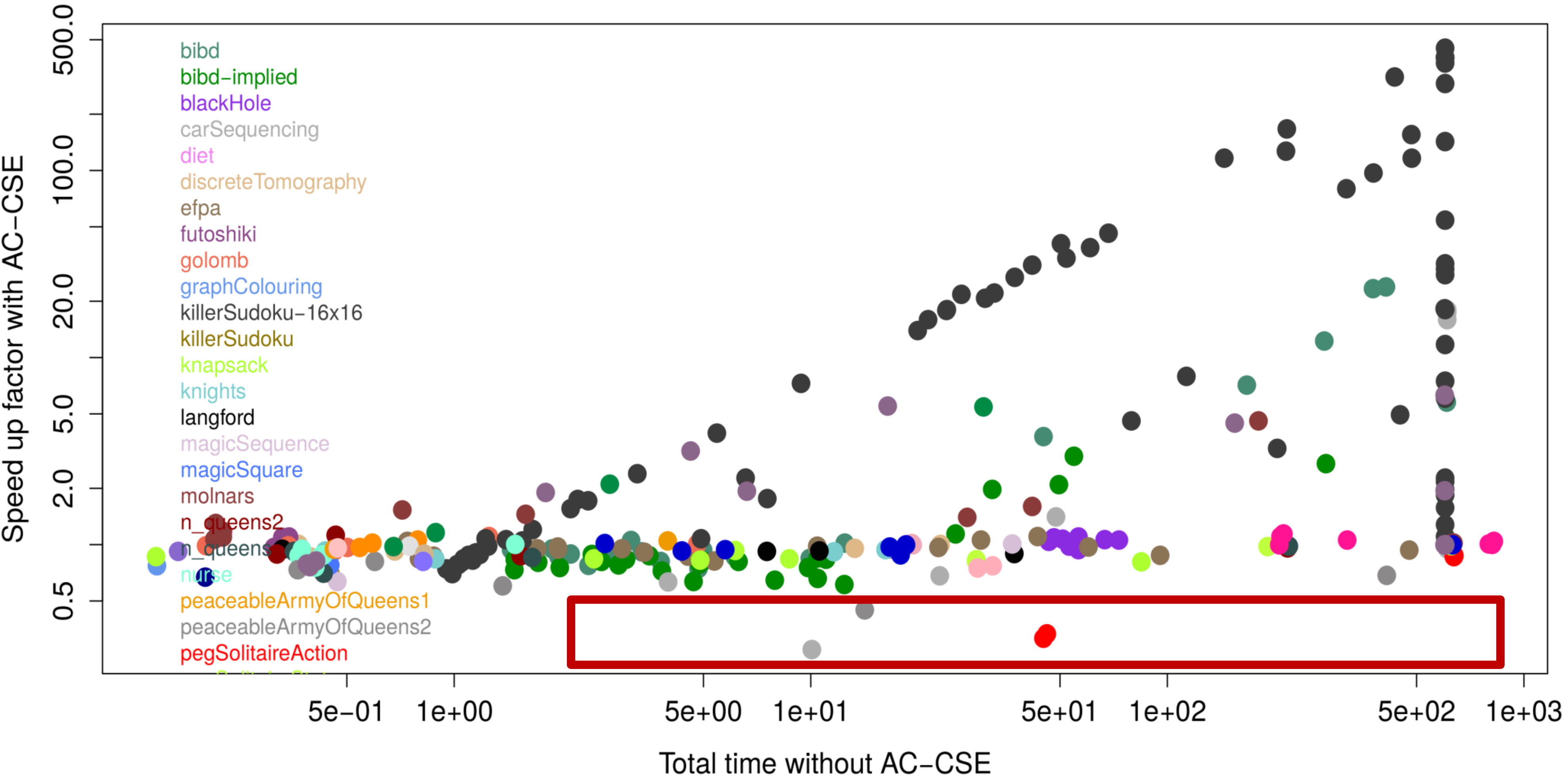
Killer Sudoku (5)

SONET (3)

Molnars (1)

Waterbucket (2)

EXPERIMENTAL RESULTS



Slow down by 2x or more

Car sequencing! (1)

- One very easy instance: X-CSE takes a long time, saves no search

peaceableArmyQs2

pegSolitaireAction (2)

KILLER SUDOKU

9x9 grids too easy, we did 16x16

16x16 matrix, each cell takes value in {1..16}

Rows, columns and 4x4 subsquares: allDifferent

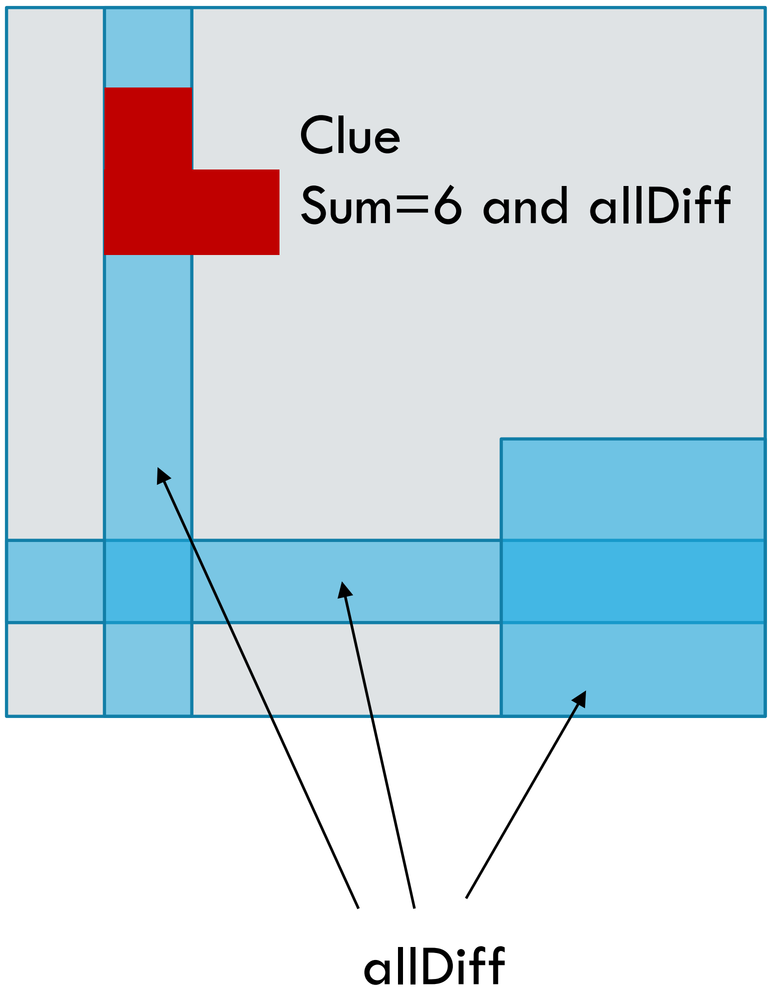
Clues are contiguous sets of cells

- The sum is given as part of the clue
- Cells within the clue are allDifferent

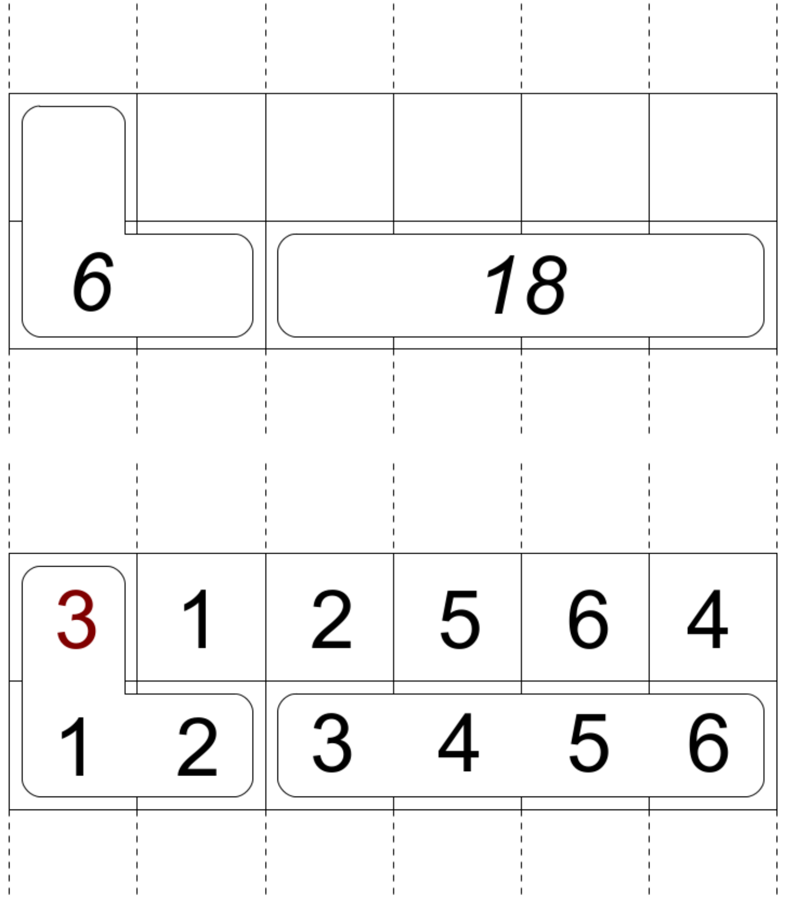
In the example (right) the clue must contain values 1,2,3

Entire matrix covered by non-overlapping clues

Model is exactly the above constraints



KILLER SUDOKU



Rows/columns/subsquares are a permutation

Introduce sum constraints from AllDifferent

For each row, column and subsquare X:

sum(X) = 136 (for 16x16 case)

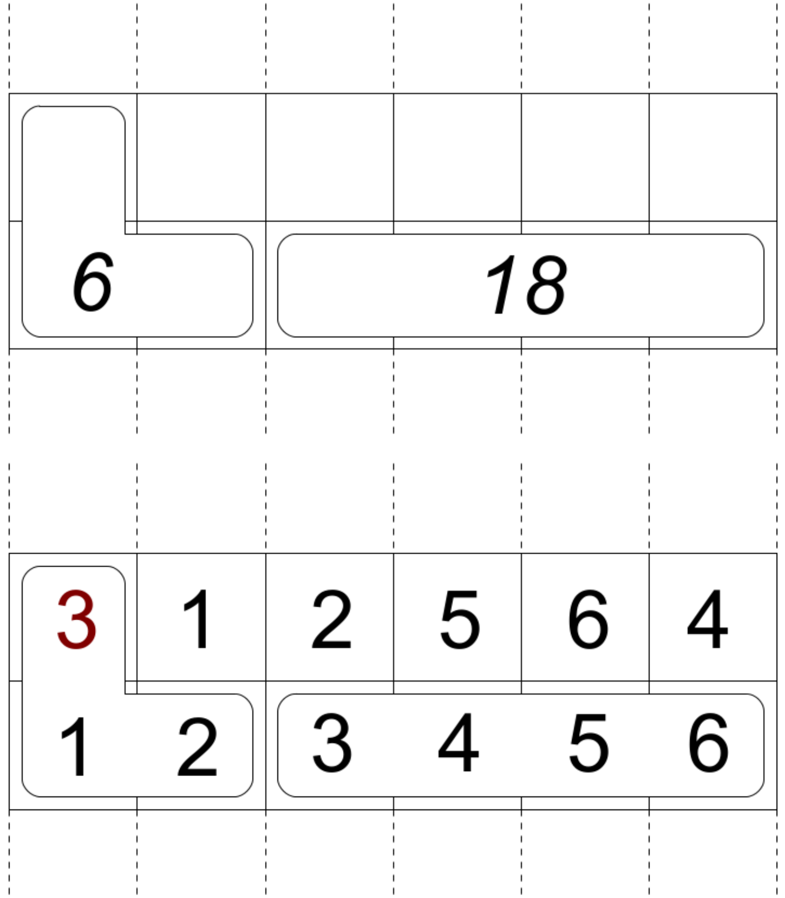
Suppose we had 6x6 Killer Sudoku (left)

sum(X) = 21

For each clue, we also get useless sum≤a and sum≥b

- Removed by Identical CSE followed by simplifiers

KILLER SUDOKU



New sums on rows/columns/subsquares intersect with clues

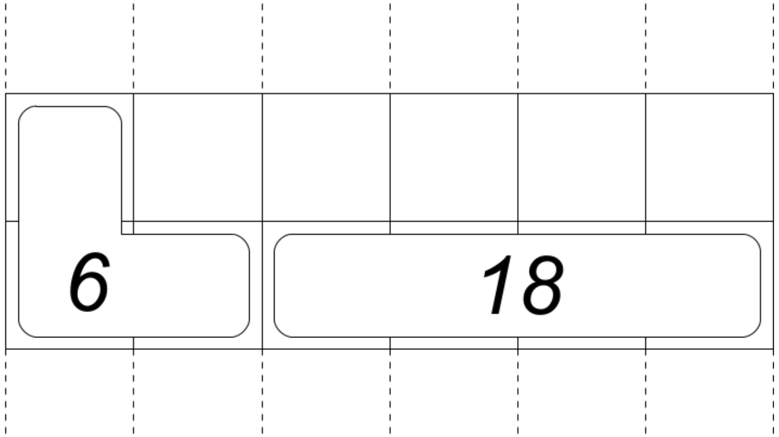
In example (left), suppose two rows are k[1,..] and k[2,..]

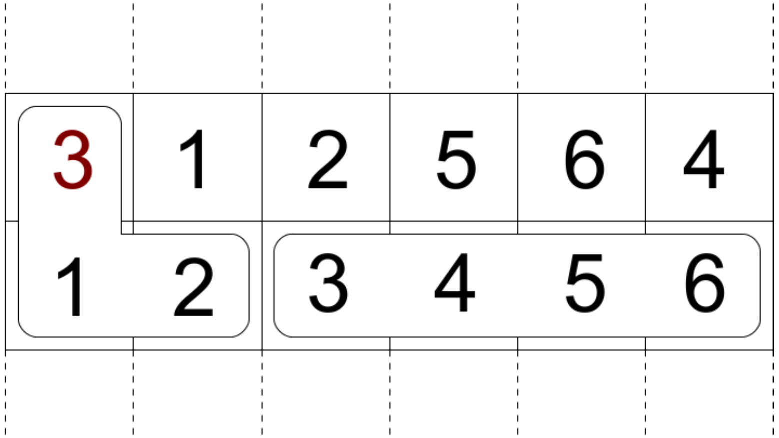
AC-CSE connects clues to rows

k[2,3] + … + k[2,6] is common to the 18 clue and the row sum

k[2,1] + k[2,2] is common to the 6 clue and the row sum

KILLER SUDOKU





k[2,3] + … + k[2,6] = aux1

k[2,1]+k[2,2] = aux2

aux1=18, aux2+k[1,1]=6

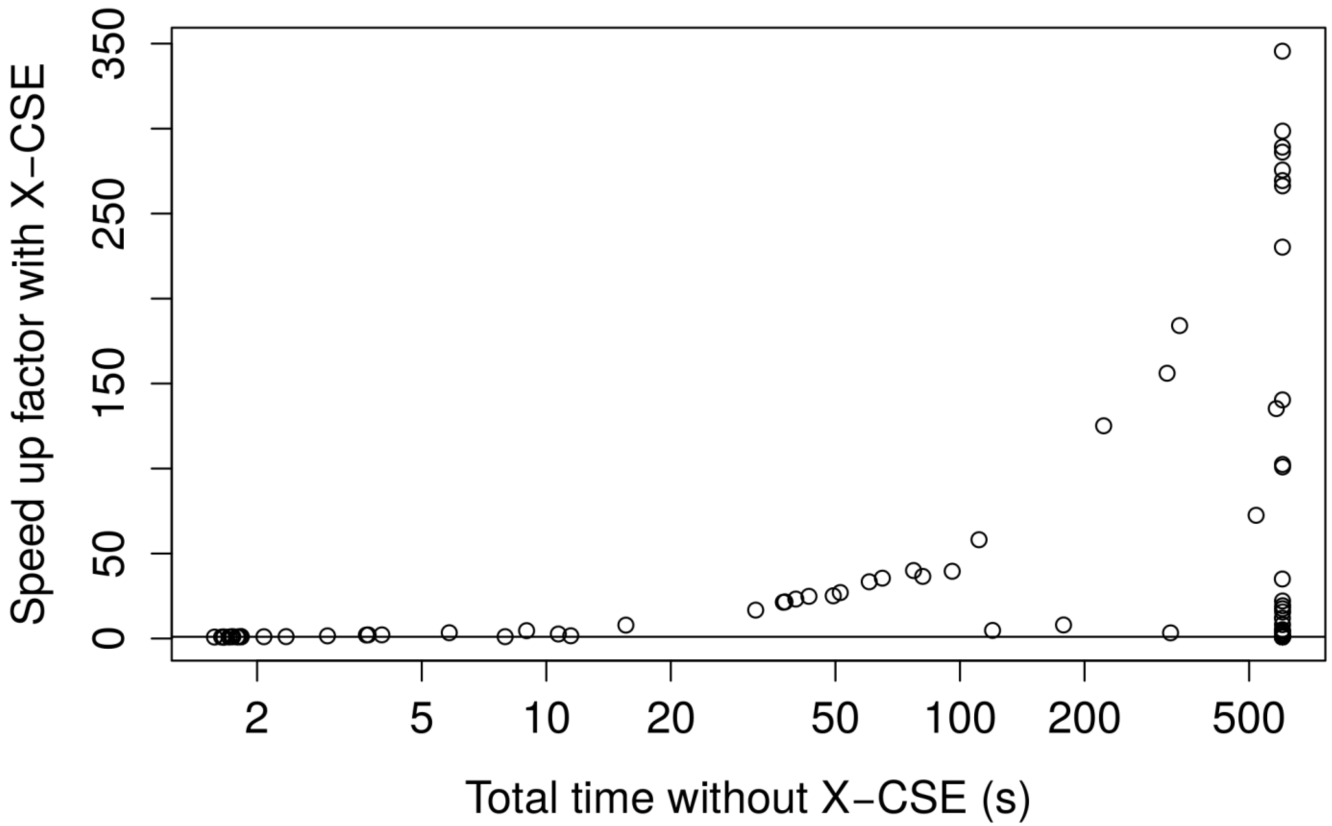
aux1+aux2=21

aux1 replaced with 18 (variable deletion)

aux2 becomes 3 (simplifier, then var deletion)

k[1,1] becomes 3 (simplifier, then var deletion)

KILLER SUDOKU – RESULTS



Some hard problems made almost trivial

Peak instance:

Without X-CSE Savile Row took 2.26s

Minion timed out at 600s

2,774,028 nodes

With X-CSE Savile Row took 1.62s

Minion took 0.13s, 2 nodes

savilerow –O3 killer.eprime …

KILLER SUDOKU – SUMMARY

We need to do these steps:

1. Add implied sum to all AllDifferent constraints
2. Apply AC-CSE
3. Variable deletion (interleaved with simplifiers)

FUTURE WORK

Scalability

Both X-CSE and I-CSE can fail if…

- Very large number of sum constraints
- Very long sums

Many possible ways of improving scalability

- Anytime algorithm with time quota

Obtain more implied sum (or product) constraints from globals

THANK YOU

AC-CSE is implemented in Savile Row:

http://savilerow.cs.st-andrews.ac.uk/

Any questions?