Saving Checks does Not Always Save Time*

M.R.C. van Dongen
Cork Constraint Computation Centre
Computer Science Department, UCC

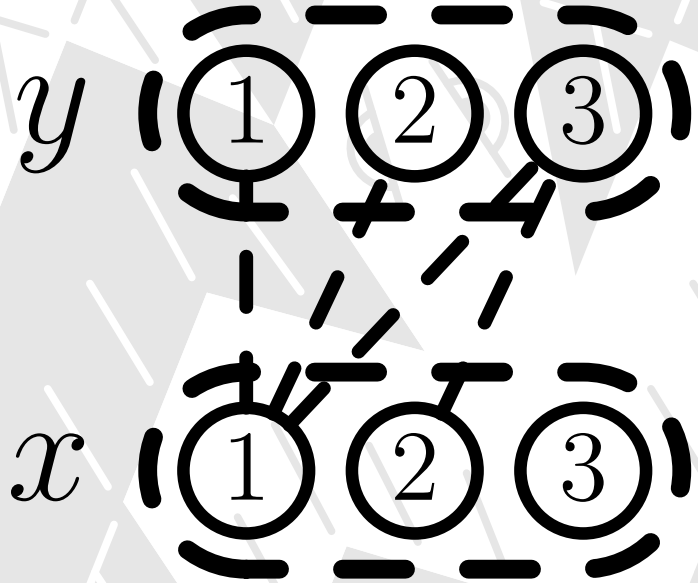
September 18, 2003

*This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075.

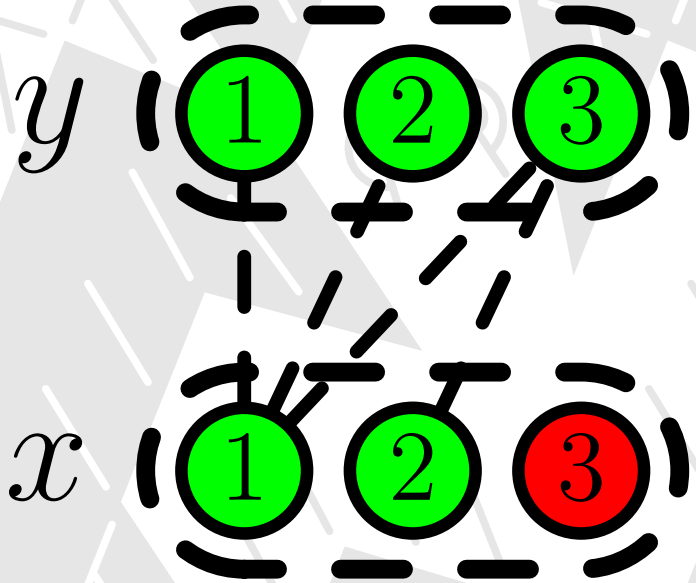
Outline

- Arc-Consistency;
- Maintain Arc-Consistency;
- Experimental Results;
- Conclusions & Future Work.

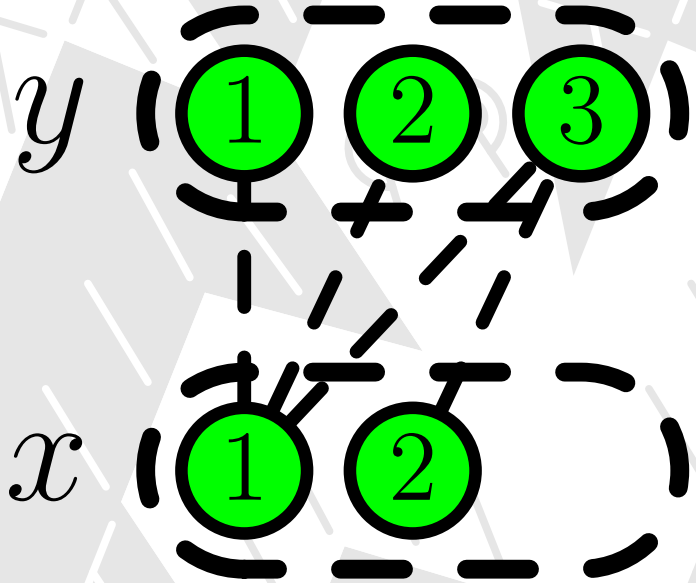
Constraint Propagation: Arc-Consistency



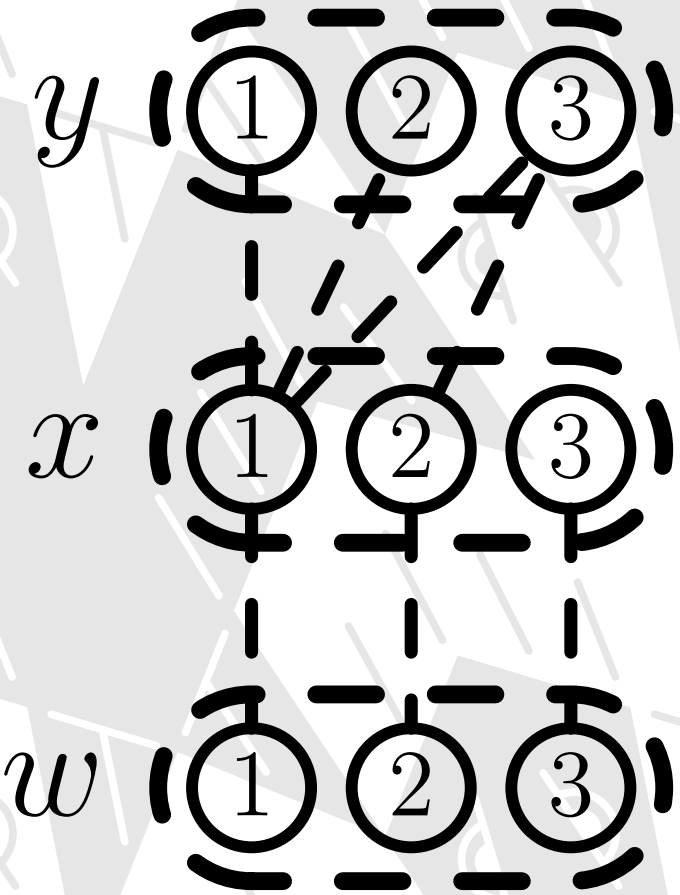
Constraint Propagation: Arc-Consistency



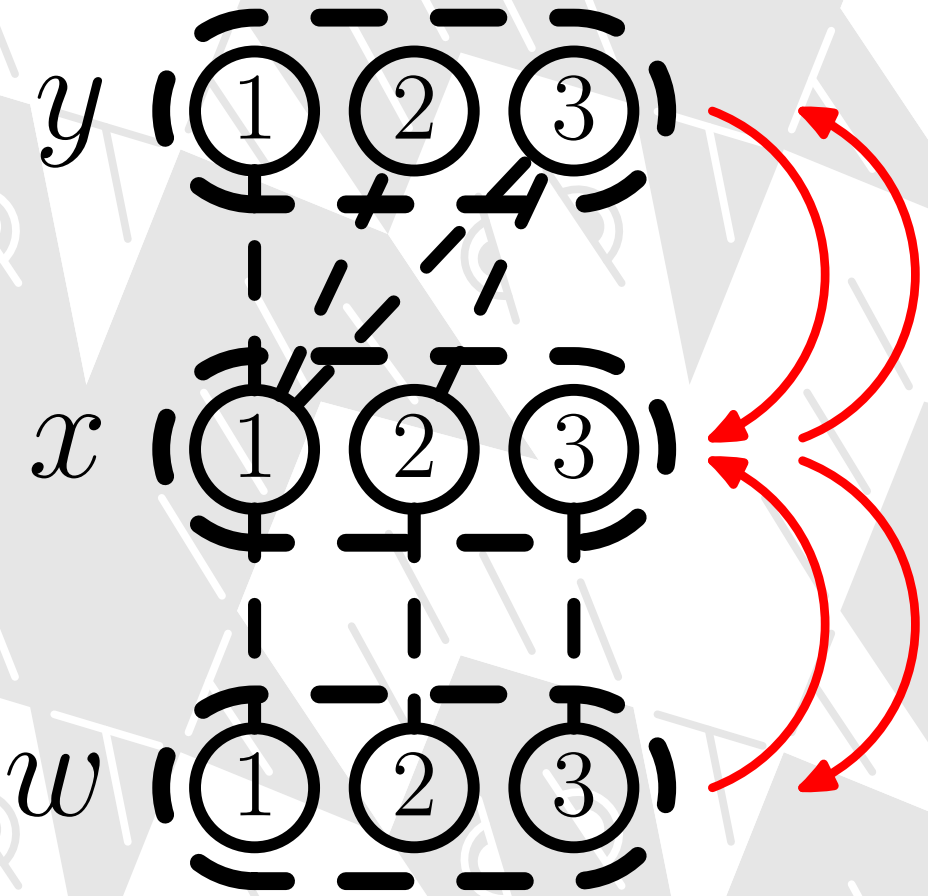
Constraint Propagation: Arc-Consistency



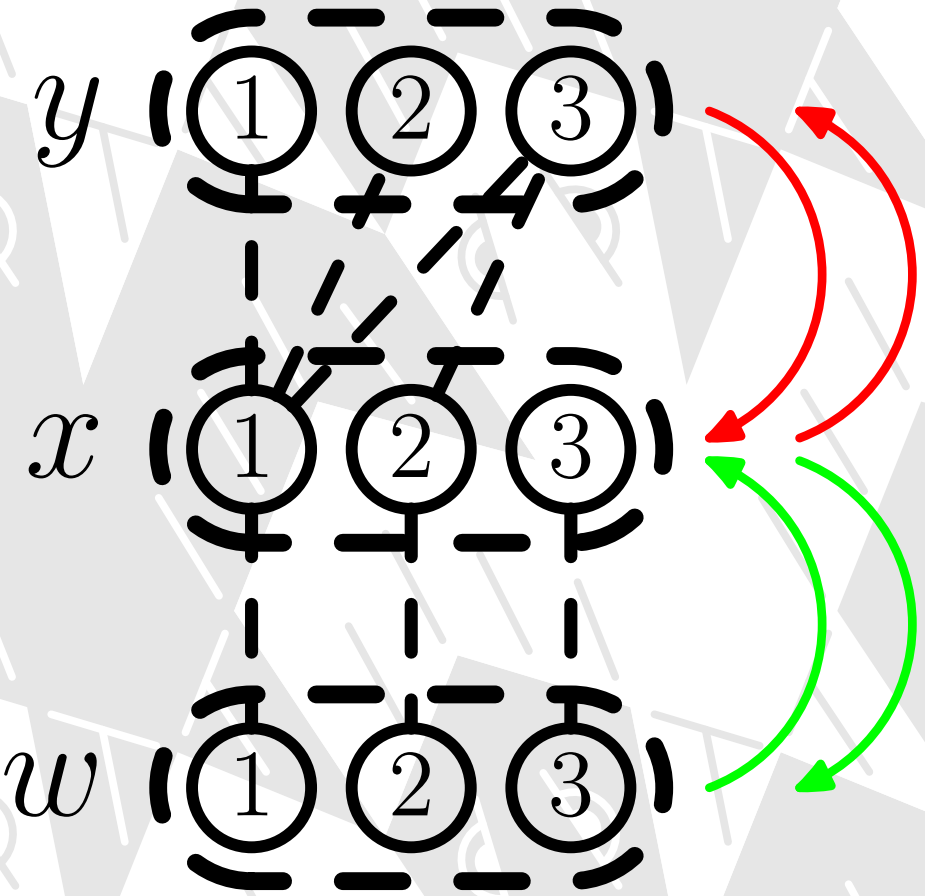
Arc-Based Arc-Consistency Algorithms



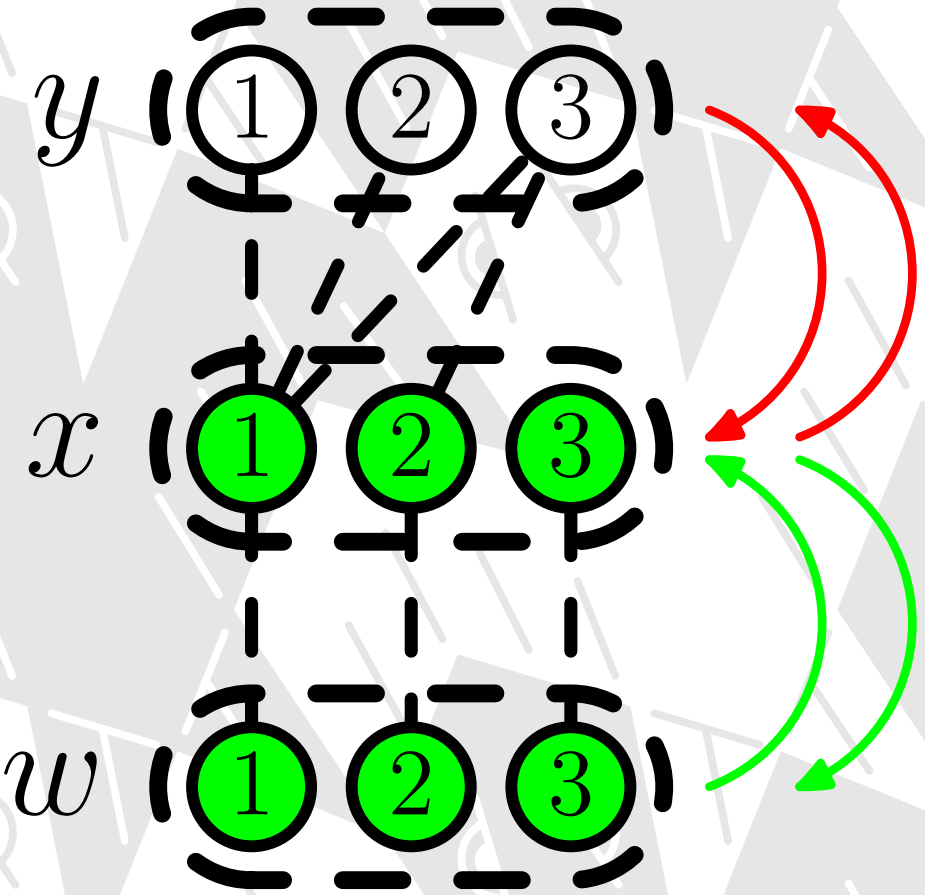
Arc-Based Arc-Consistency Algorithms



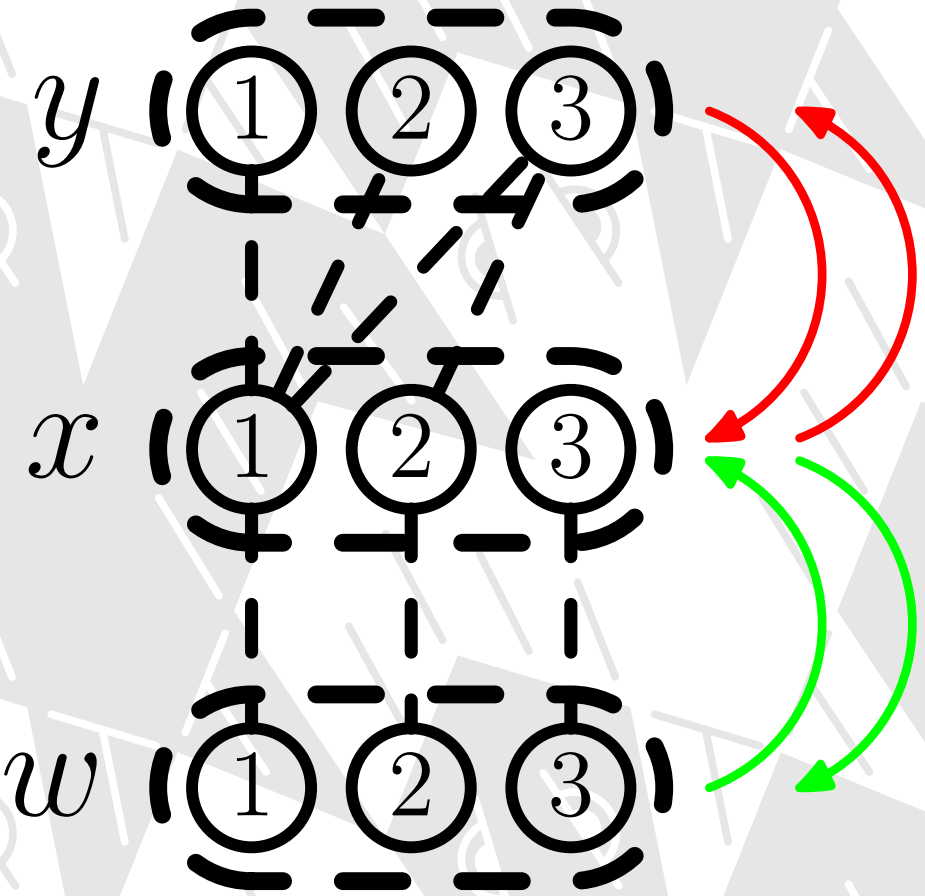
Arc-Based Arc-Consistency Algorithms



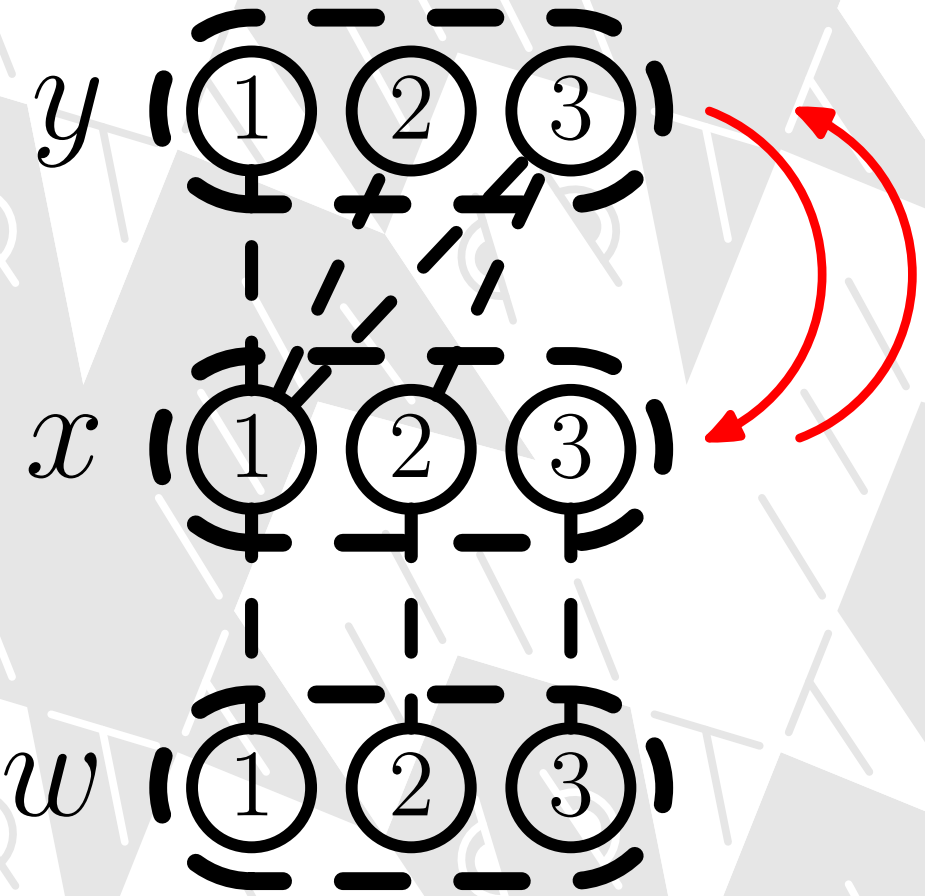
Arc-Based Arc-Consistency Algorithms



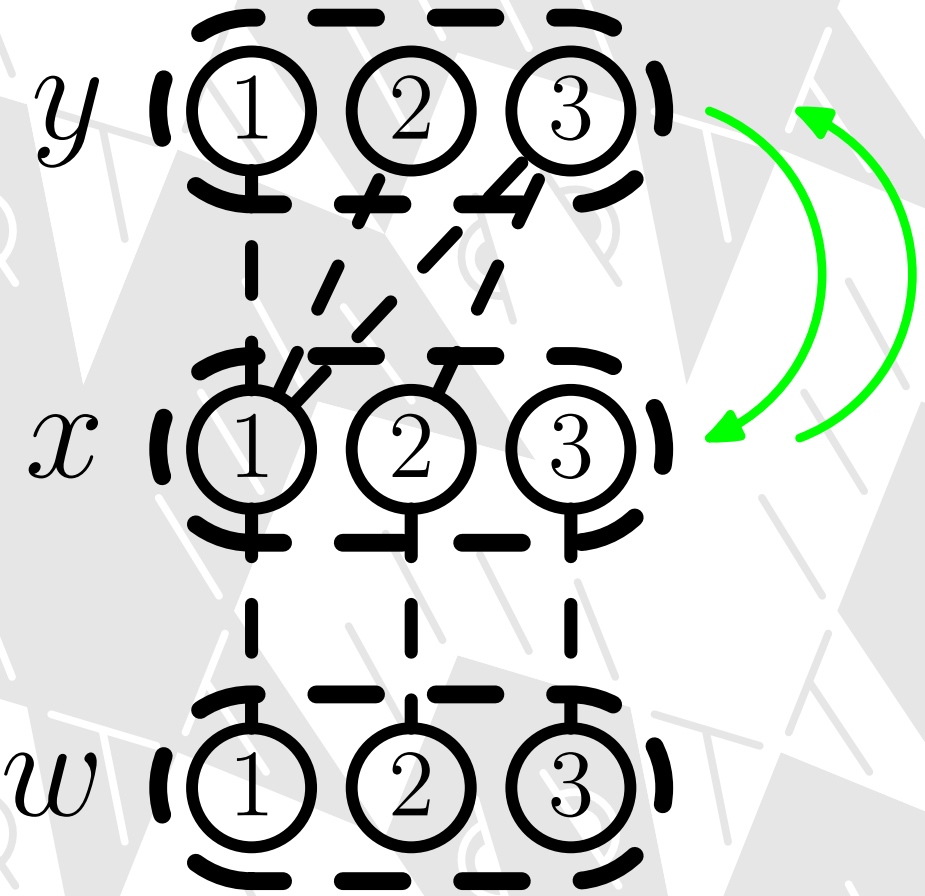
Arc-Based Arc-Consistency Algorithms



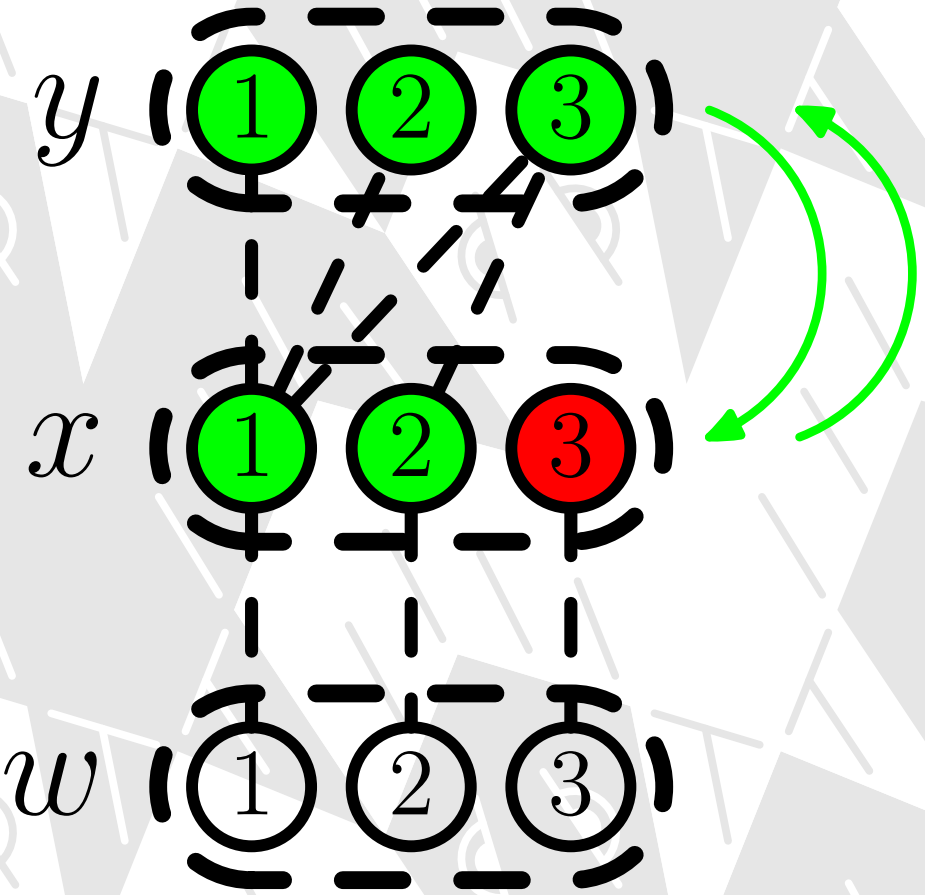
Arc-Based Arc-Consistency Algorithms



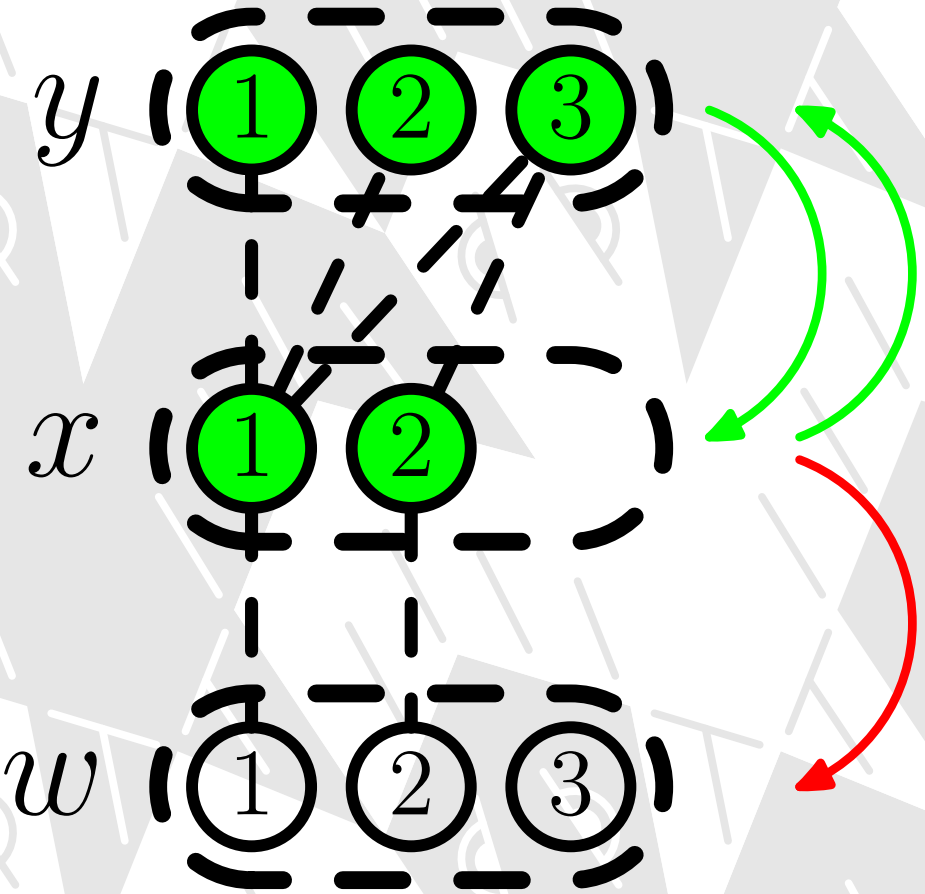
Arc-Based Arc-Consistency Algorithms



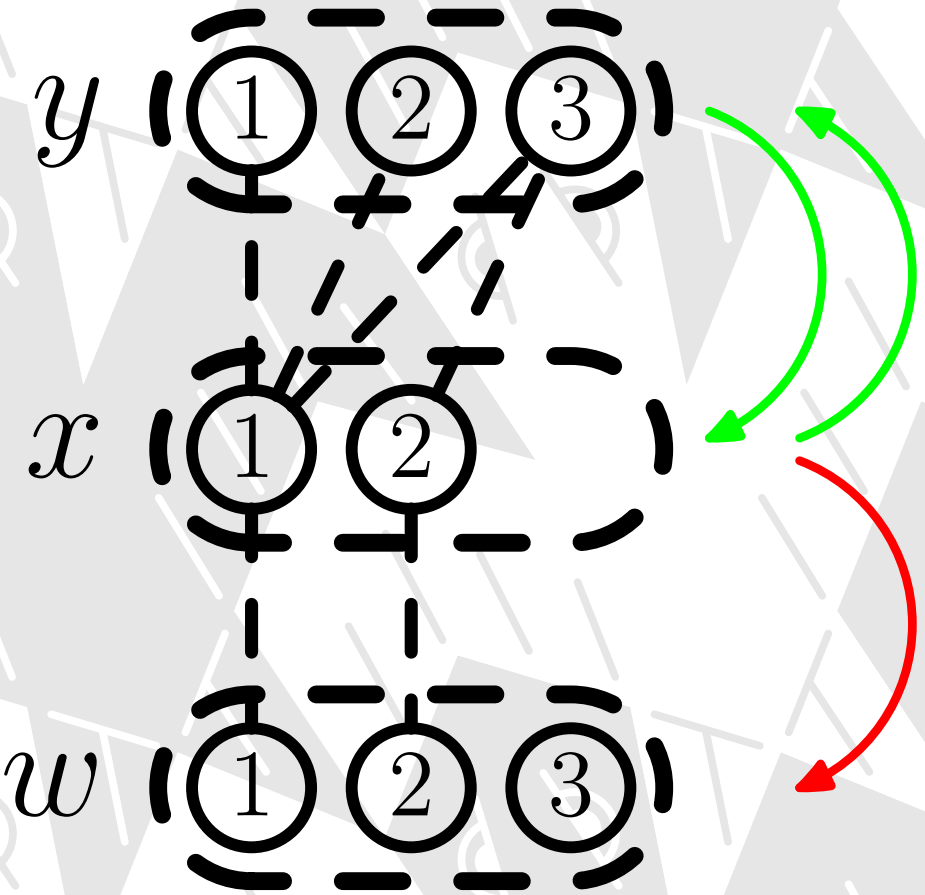
Arc-Based Arc-Consistency Algorithms



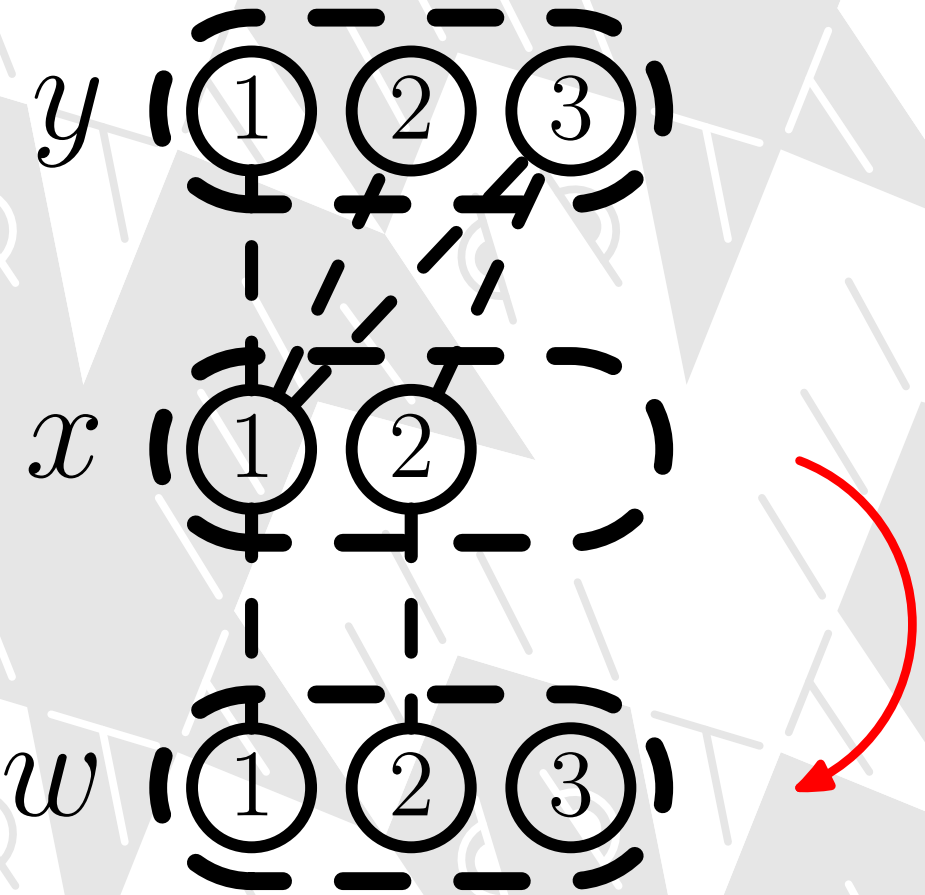
Arc-Based Arc-Consistency Algorithms



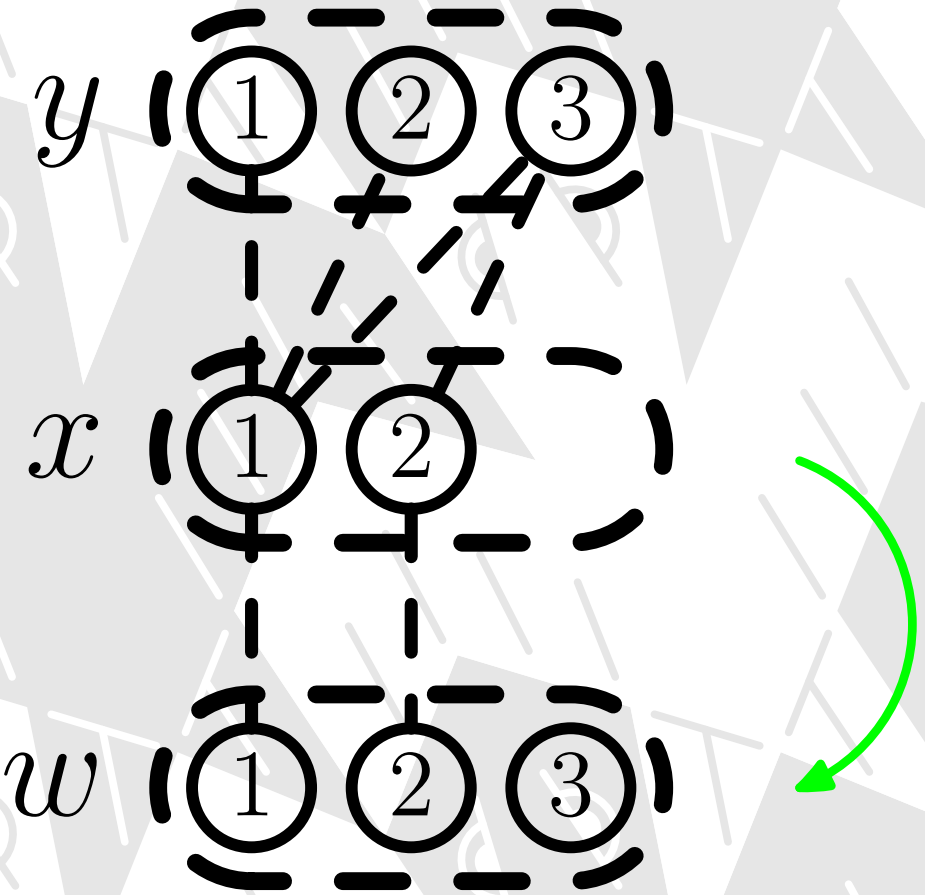
Arc-Based Arc-Consistency Algorithms



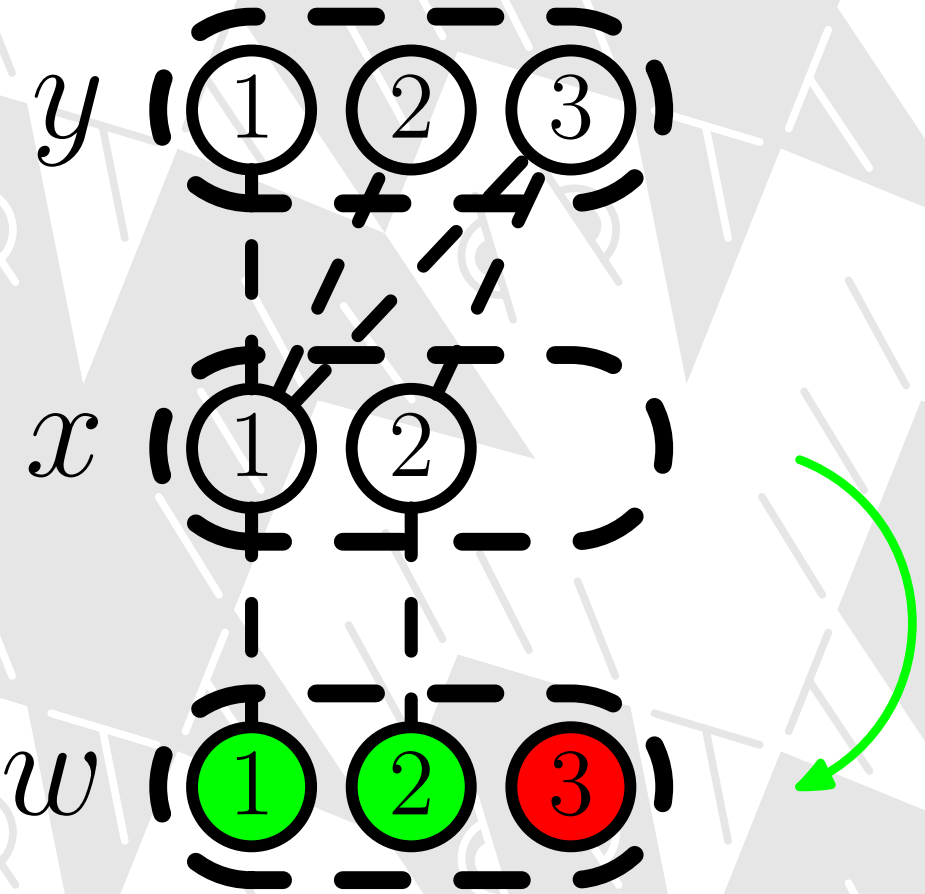
Arc-Based Arc-Consistency Algorithms



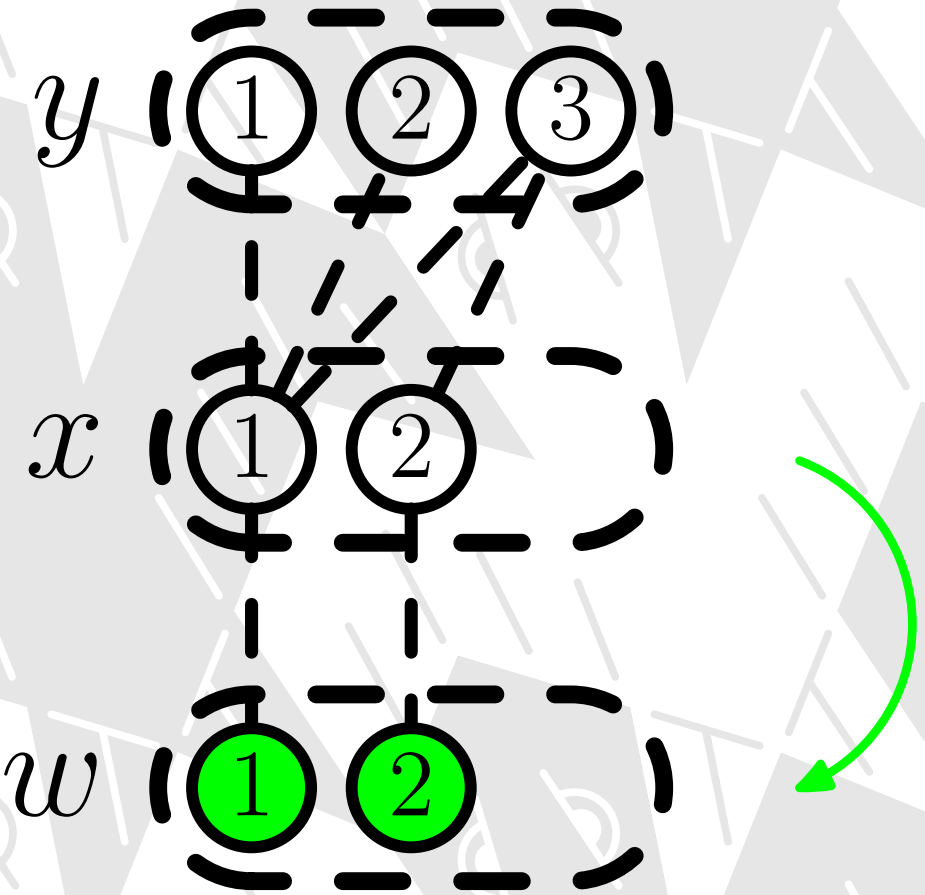
Arc-Based Arc-Consistency Algorithms



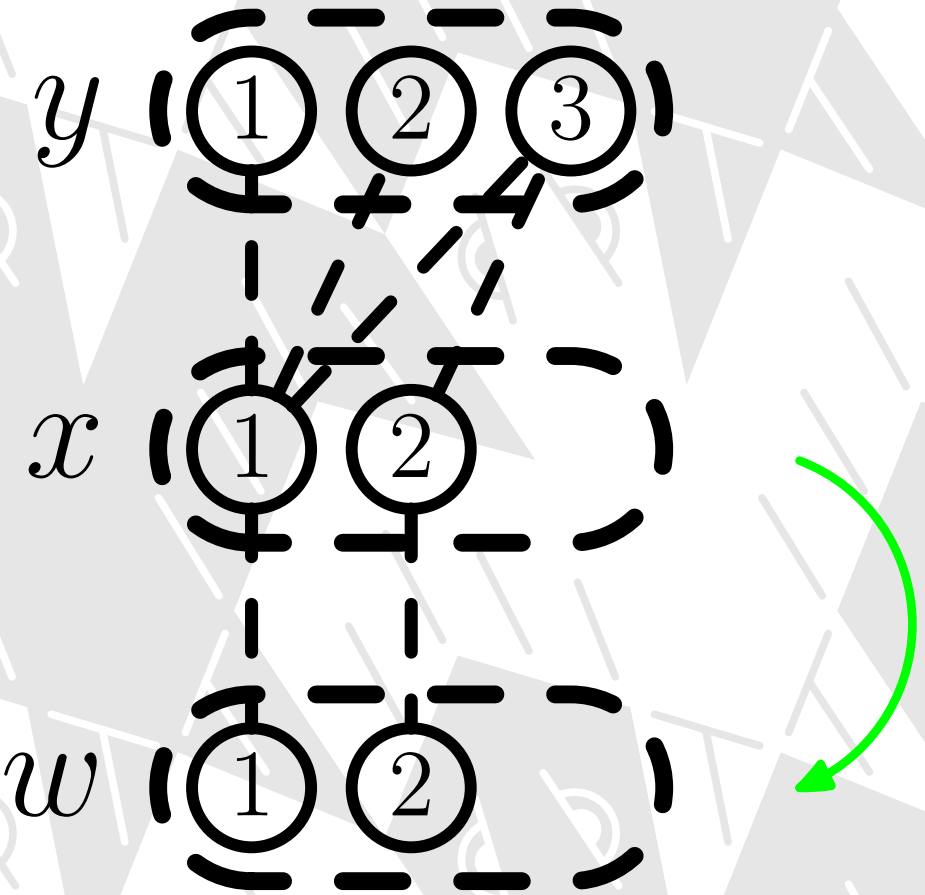
Arc-Based Arc-Consistency Algorithms



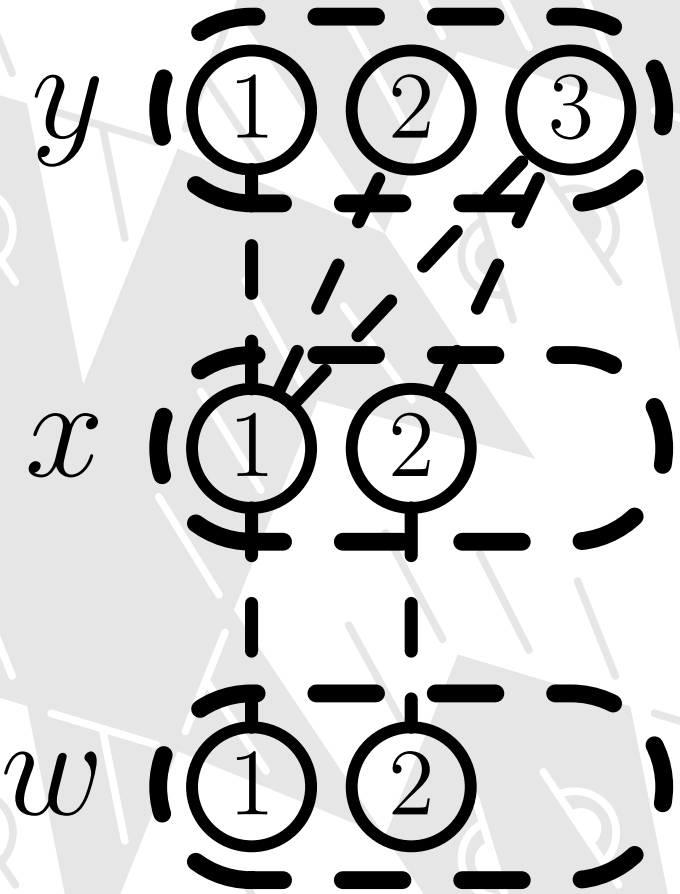
Arc-Based Arc-Consistency Algorithms



Arc-Based Arc-Consistency Algorithms



Arc-Based Arc-Consistency Algorithms



Consistency Before Search

To overcome backtracking's forgetfulness, people started using *constraint propagation* to make CSPs more *consistent* at a local level *before* search.

This significantly reduced the number of incompatible candidate partial assignments.

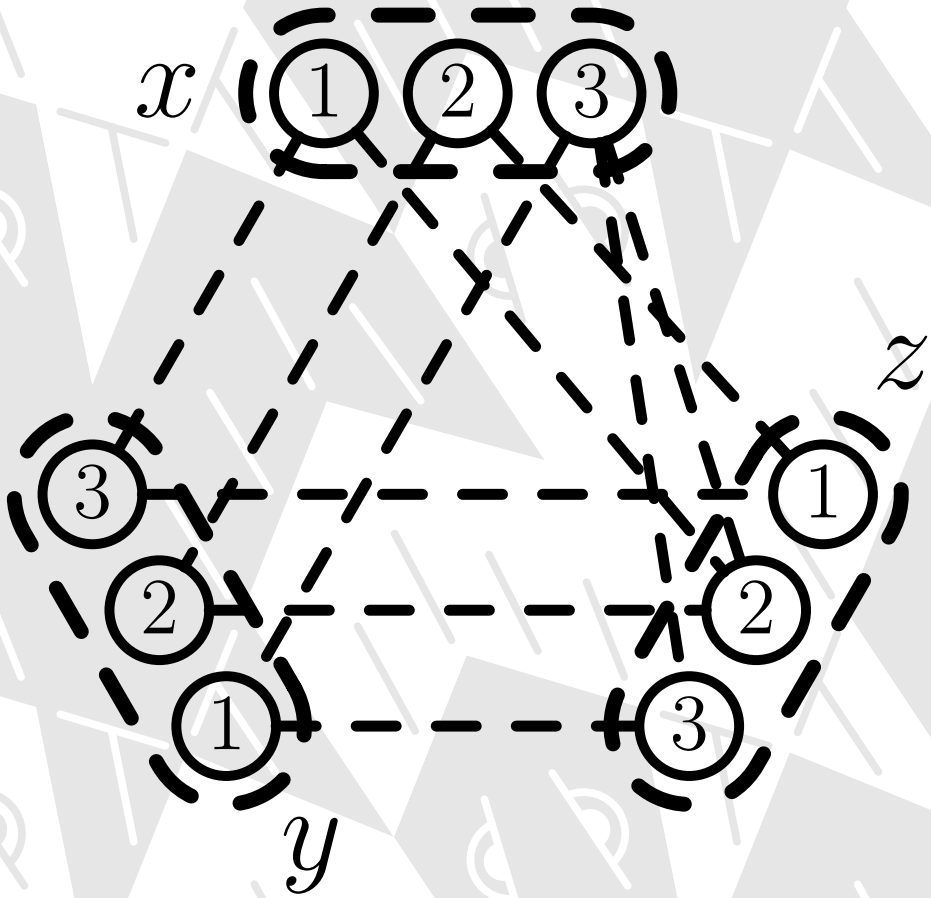
Unfortunately, it did not overcome backtracking's total amnesia.

Consistency During Search

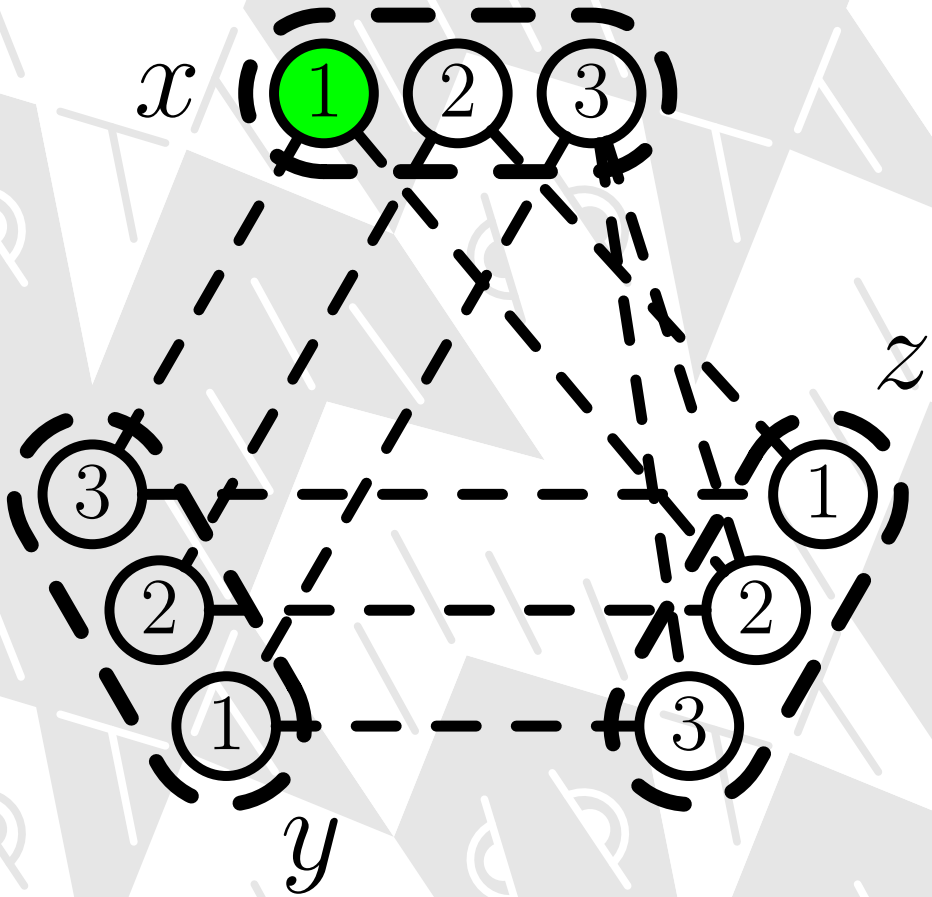
To further improve backtracking people started to use algorithms that maintained certain levels of consistency *during* search.

MAC (Maintain Arc-Consistency) is one such algorithm.

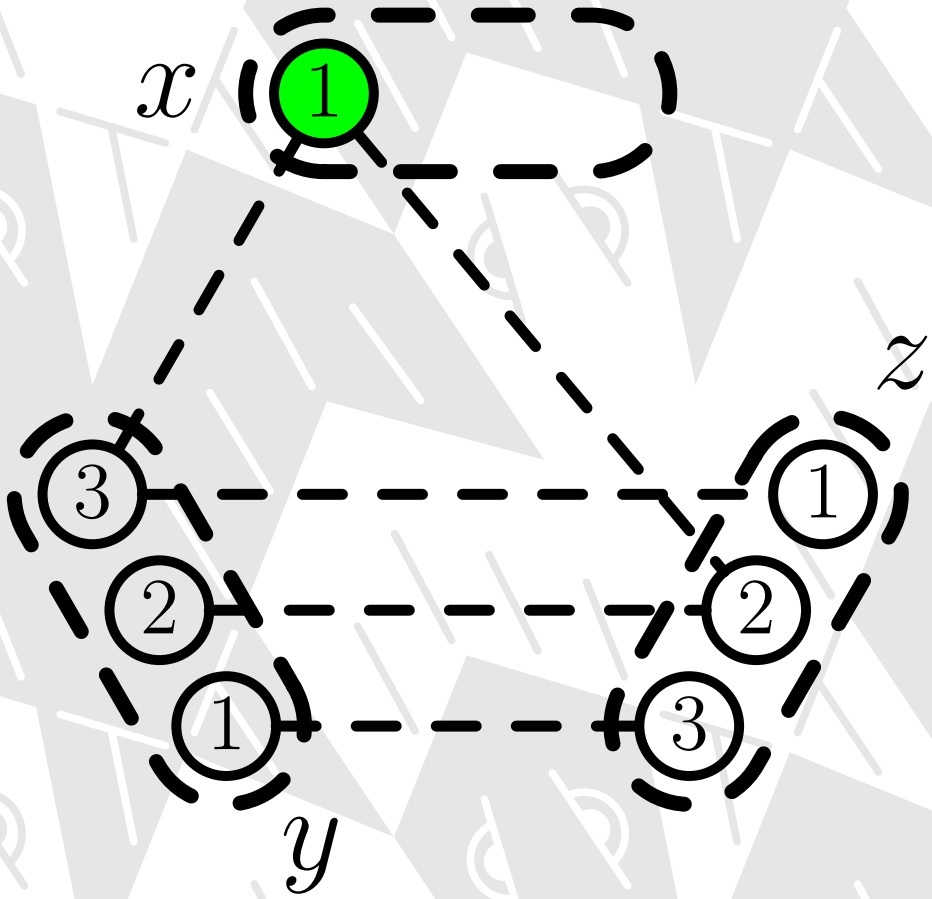
MAC



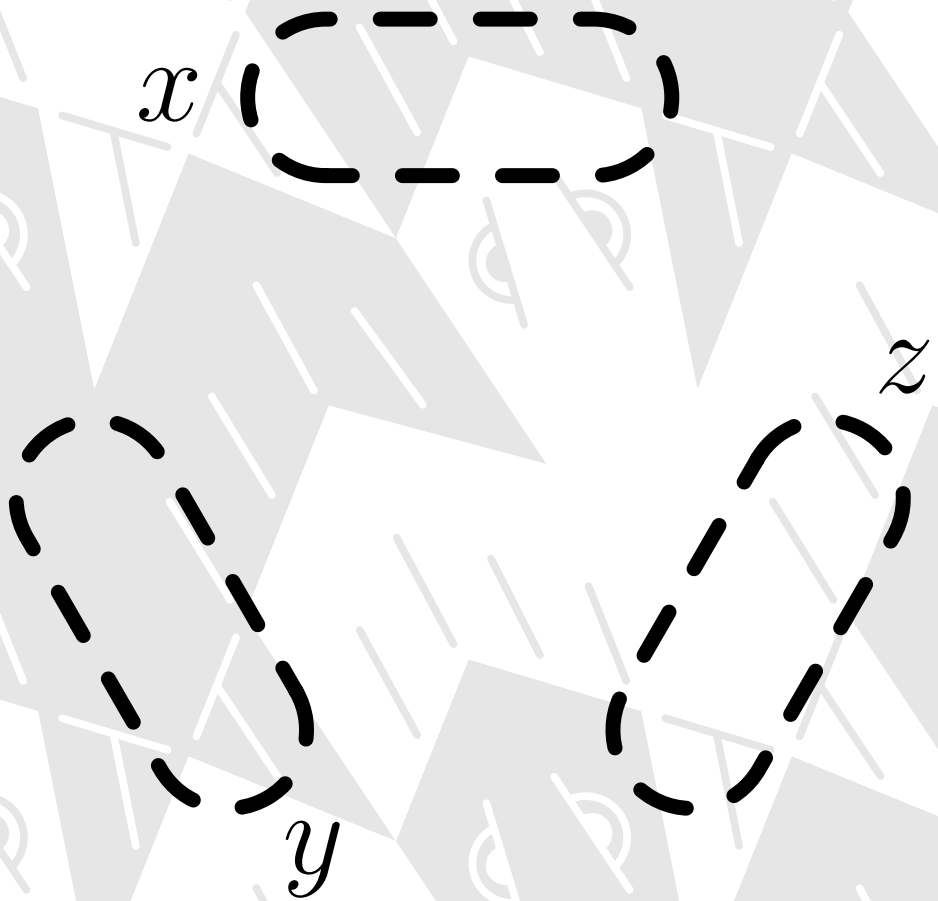
MAC



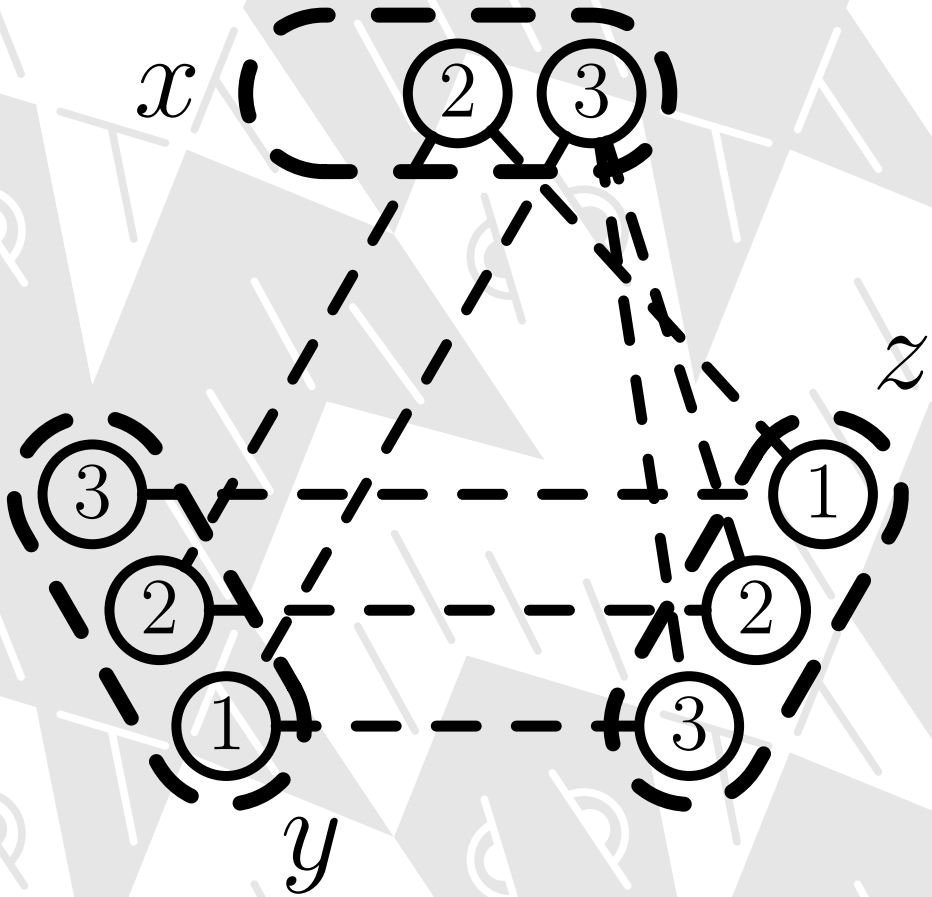
MAC



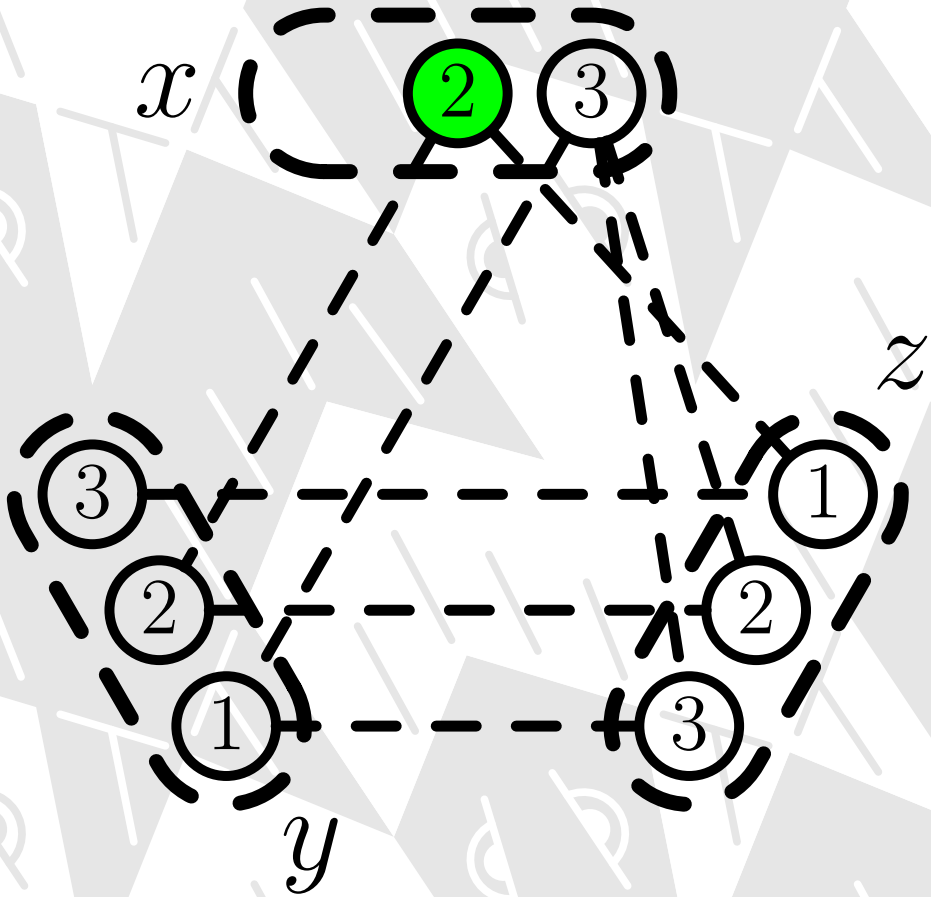
MAC



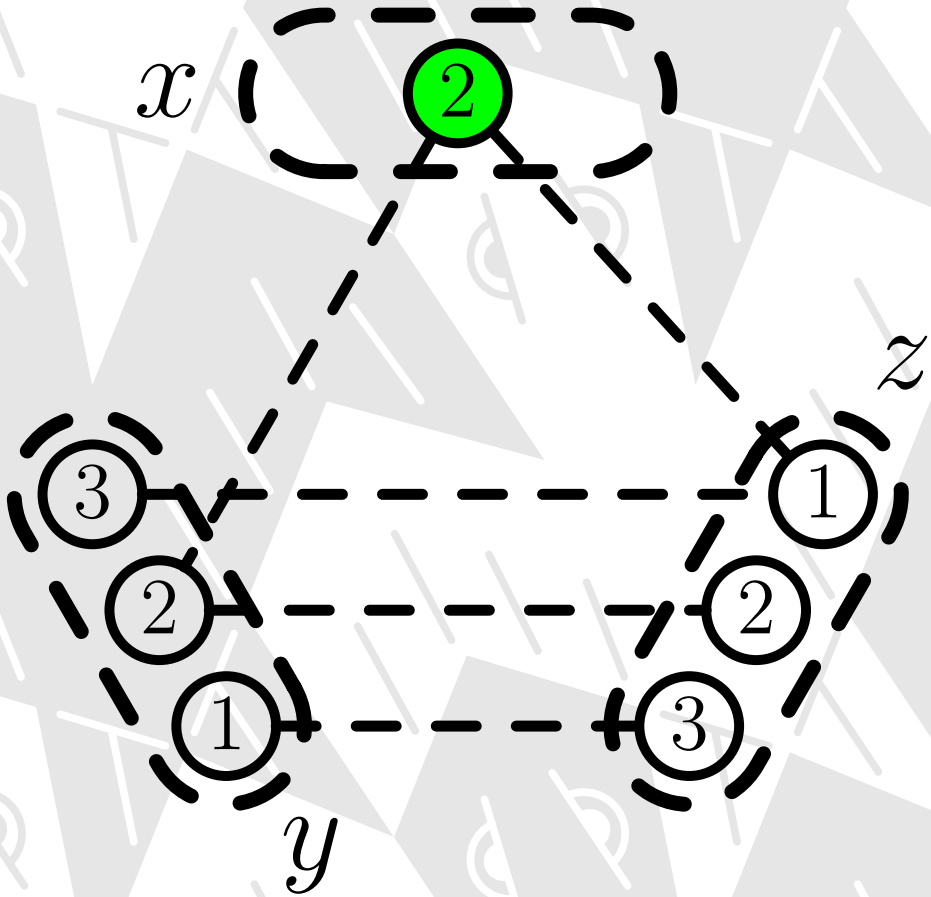
MAC



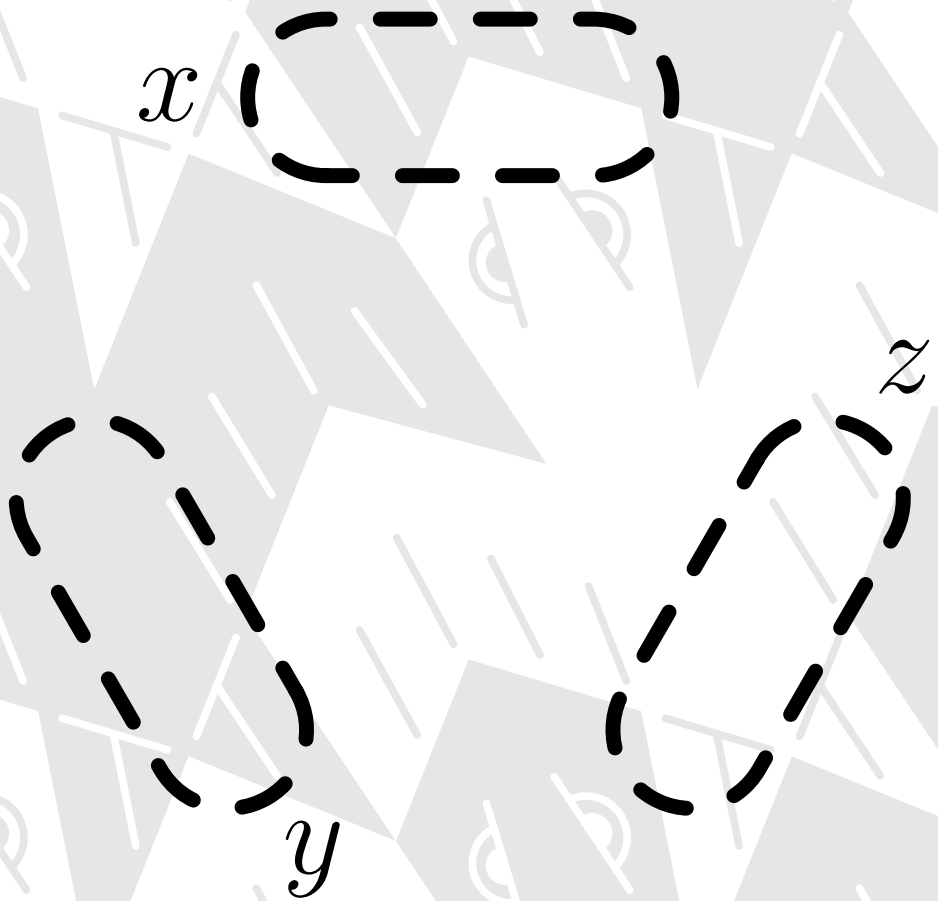
MAC



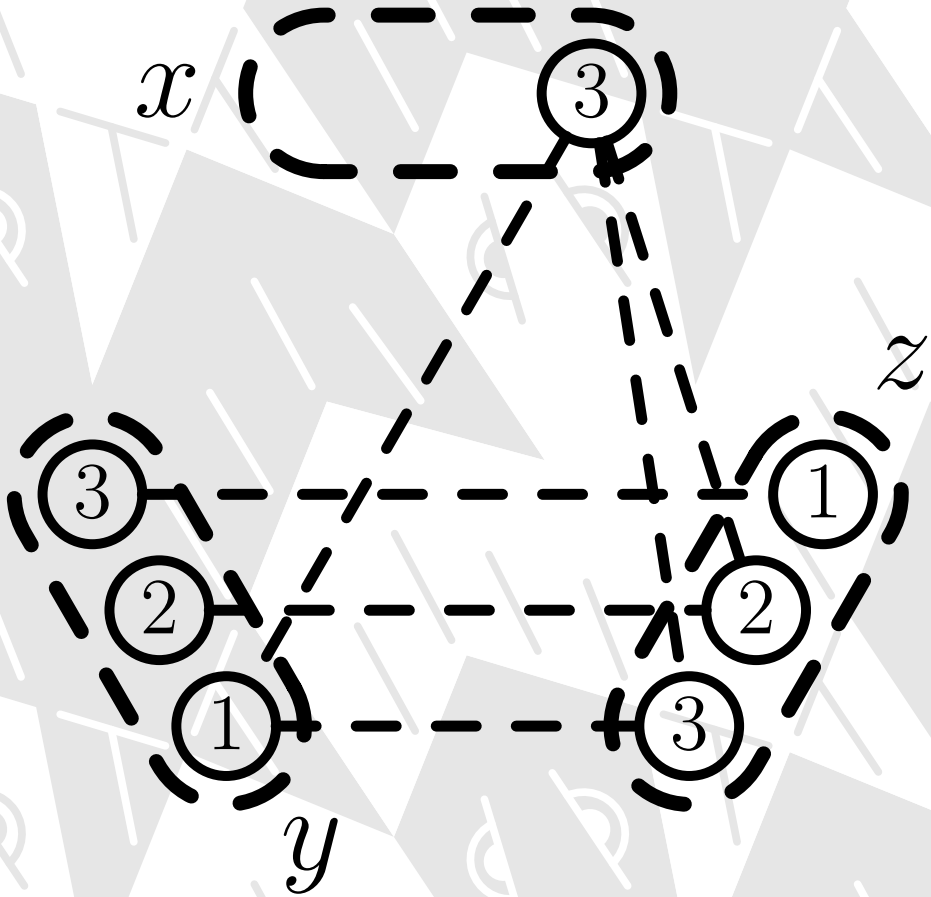
MAC



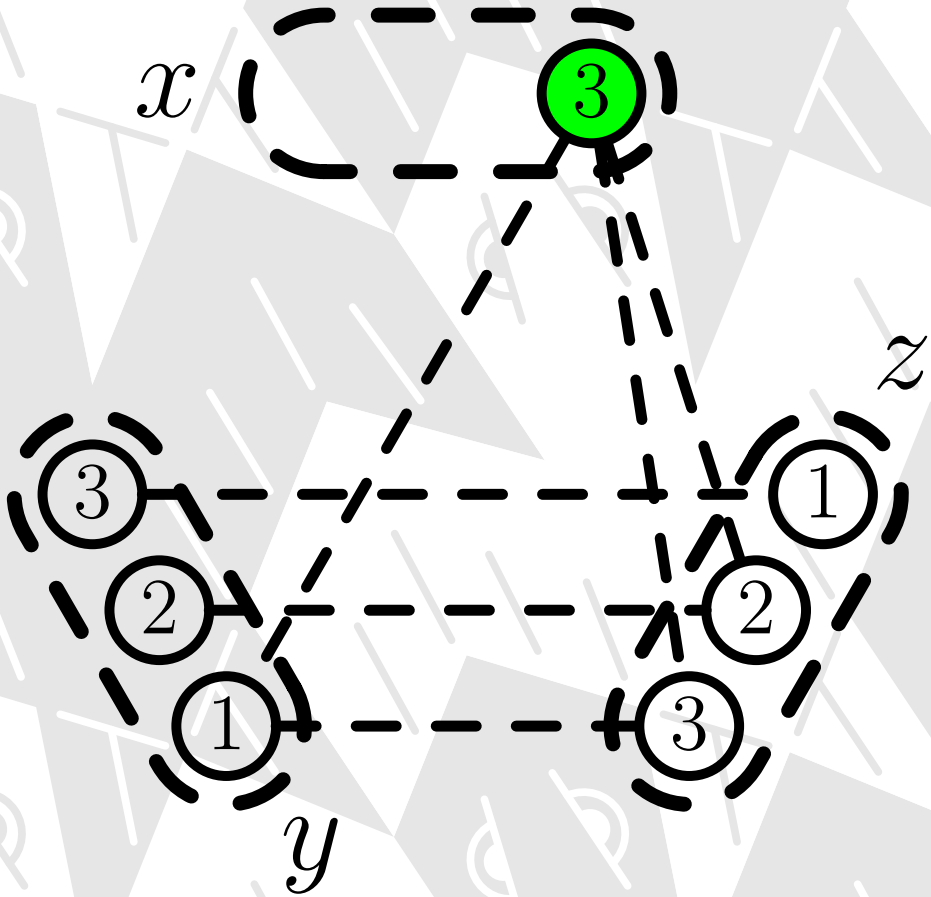
MAC



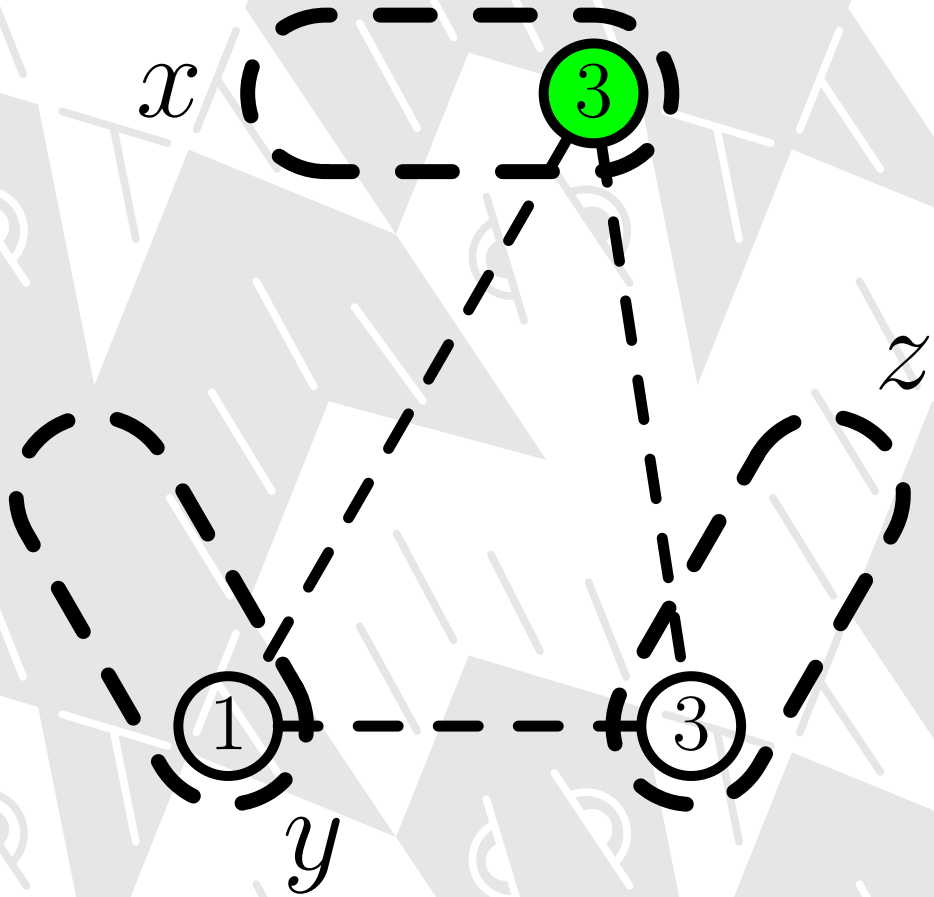
MAC



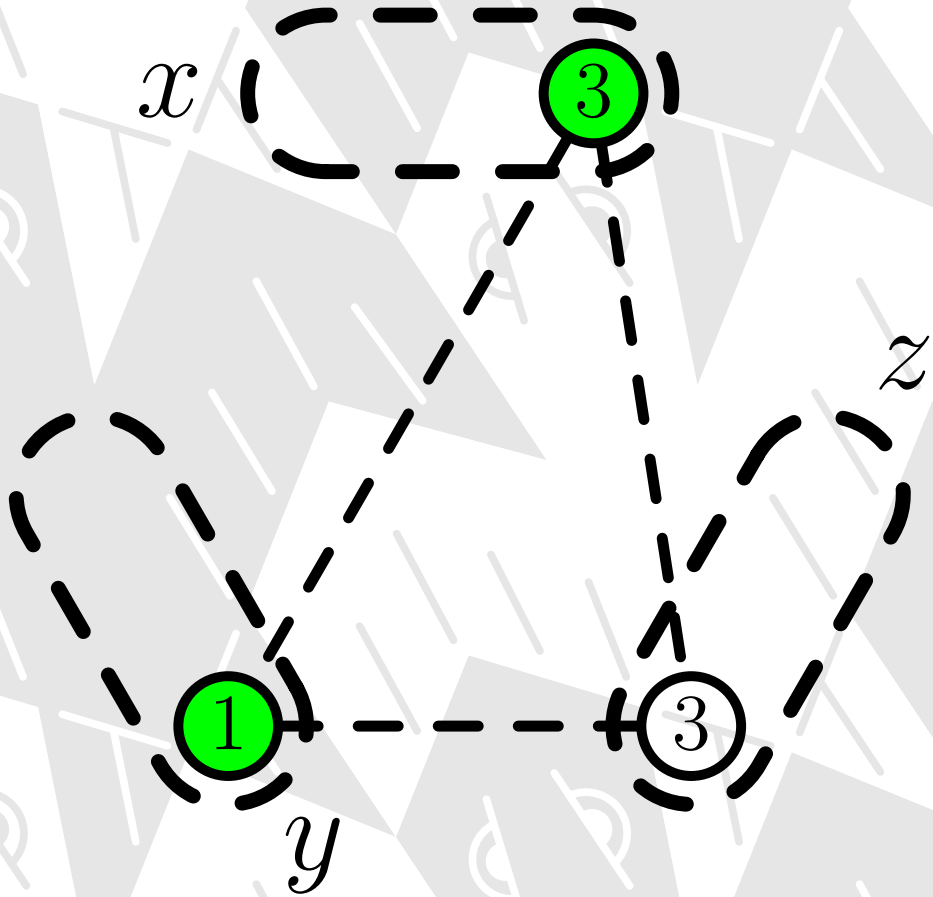
MAC



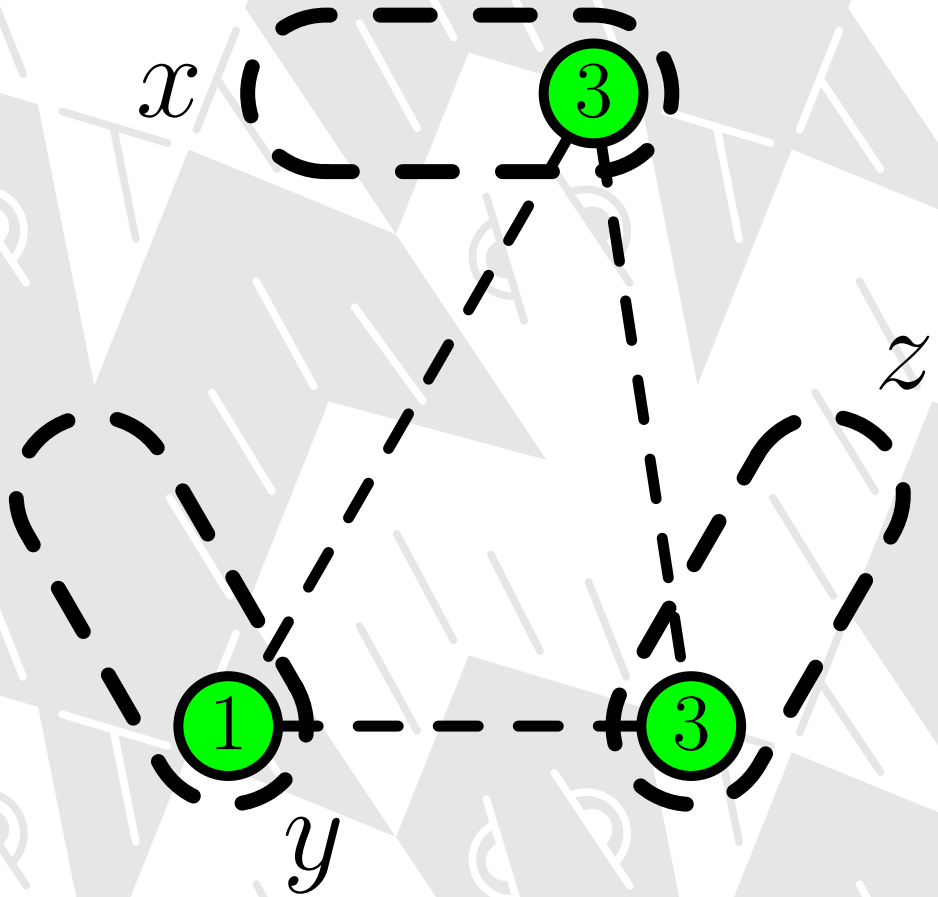
MAC



MAC



MAC



MAC-2001

Based on AC-2001.

Does not repeat checks (its worst case time-complexity is *optimal*).

Uses a lexicographical domain-heuristic.

For each constraint C and each value in the domain of the variables that are constrained by C it remembers the *last* supporting value.

To remember its checks AC-2001 requires a large $\mathcal{O}(ed)$ data structure that is maintained during search by MAC-2001.

Is reported to behave well on average.

"AC-2001, at the price of a slight extra data structure (just an integer for each value-constraint pair), reaches an optimal worst-case time complexity." [Bessière and Régin, 2001]

"AC-2001, at the price of a slight extra data structure (just an integer for each value-constraint pair), reaches an optimal worst-case time complexity." [Bessière and Régin, 2001]

Note that MAC-2001 has to save/restore its $\mathcal{O}(ed)$ counters.

But then MAC-2001 must have a $\mathcal{O}(ed\min(d,n))$ space-complexity because saving state means one of the following:

1. Save relevant counters before arc-consistency. This requires a $\mathcal{O}(ned)$ space-complexity because we may have to do this n times.

2. Save each counter before it is incremented. This comes at the price of a space-complexity of $\mathcal{O}(ed^2)$ because we may have to save each counter d times.

MAC-3$_d$

Revises one or 2 domains at a time.

When it revises 2 domains it uses a double-support domain-heuristic.

Does not remember its checks during search and its worst case time-complexity can therefore not be optimal.

Does not require additional data structures during search.

Has a $\mathcal{O}(e + nd)$ space-complexity.

This space-complexity is strictly better than MAC-2001's $\mathcal{O}(ed \min(d, n))$ and AC-2001's $\mathcal{O}(ed)$.

Experimental Results: The algorithms

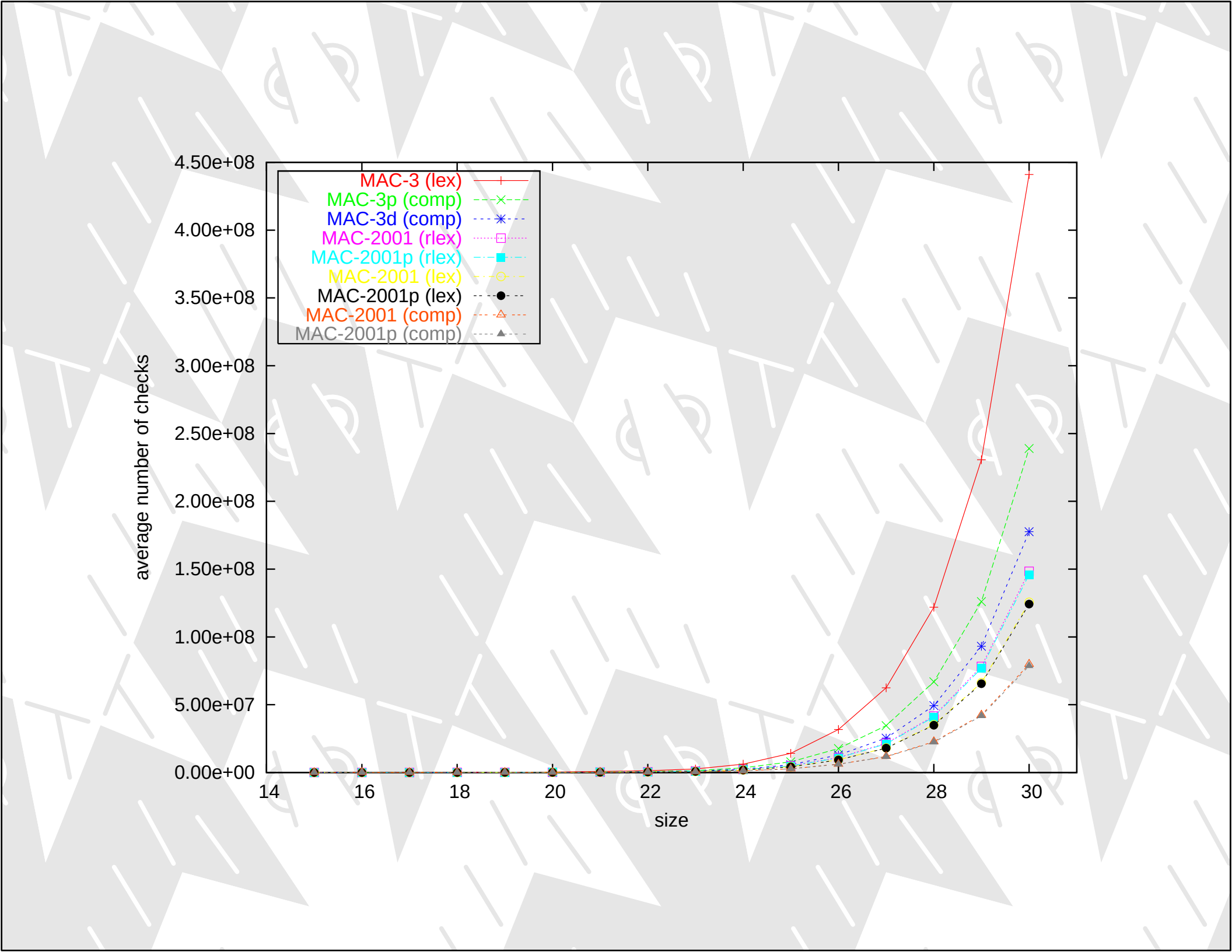
We compared five algorithms: MAC-3, MAC-3$_d$, MAC-3$_p$, MAC-2001, and MAC-2001$_p$.

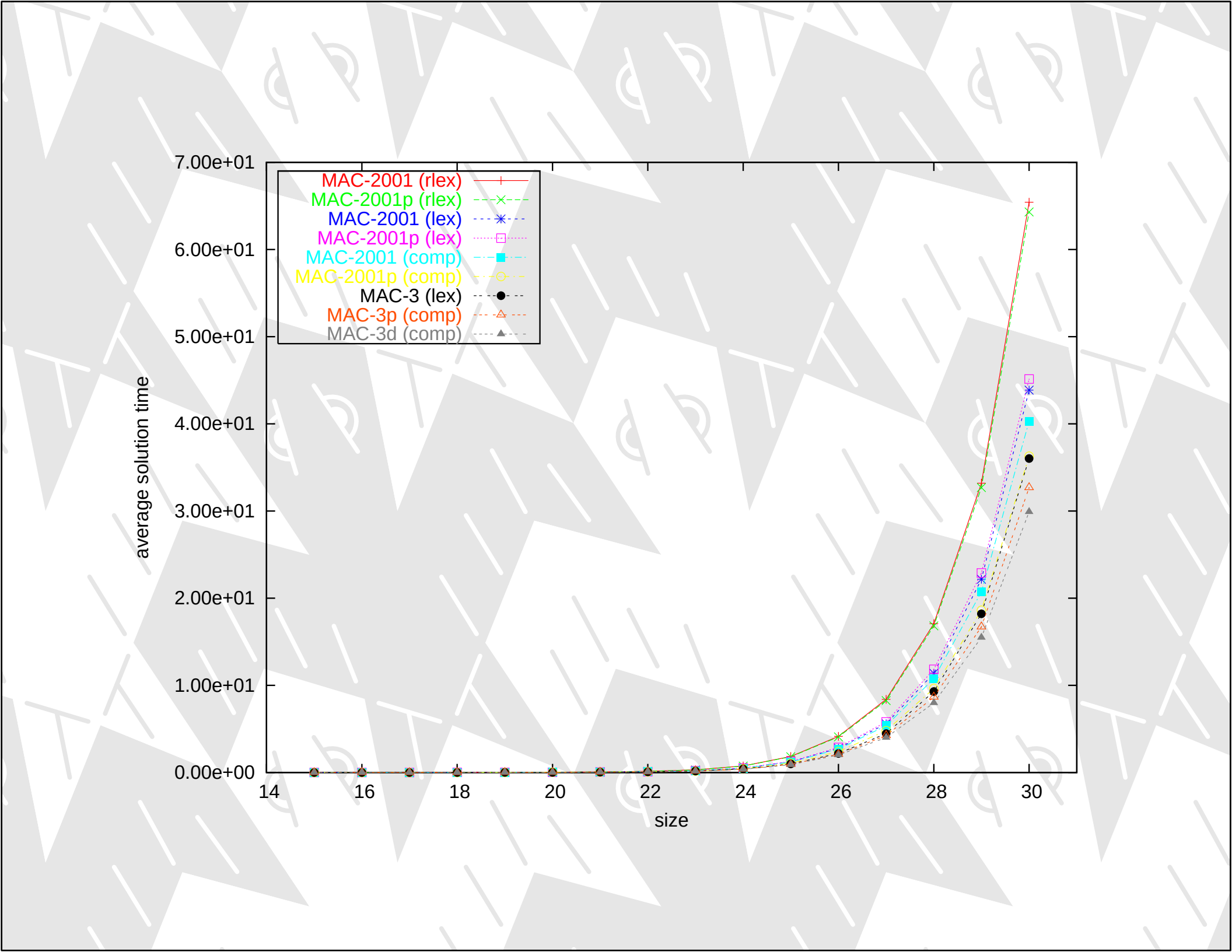
We used three different arc-heuristics: *lex*, *rlex*, and *comp*.

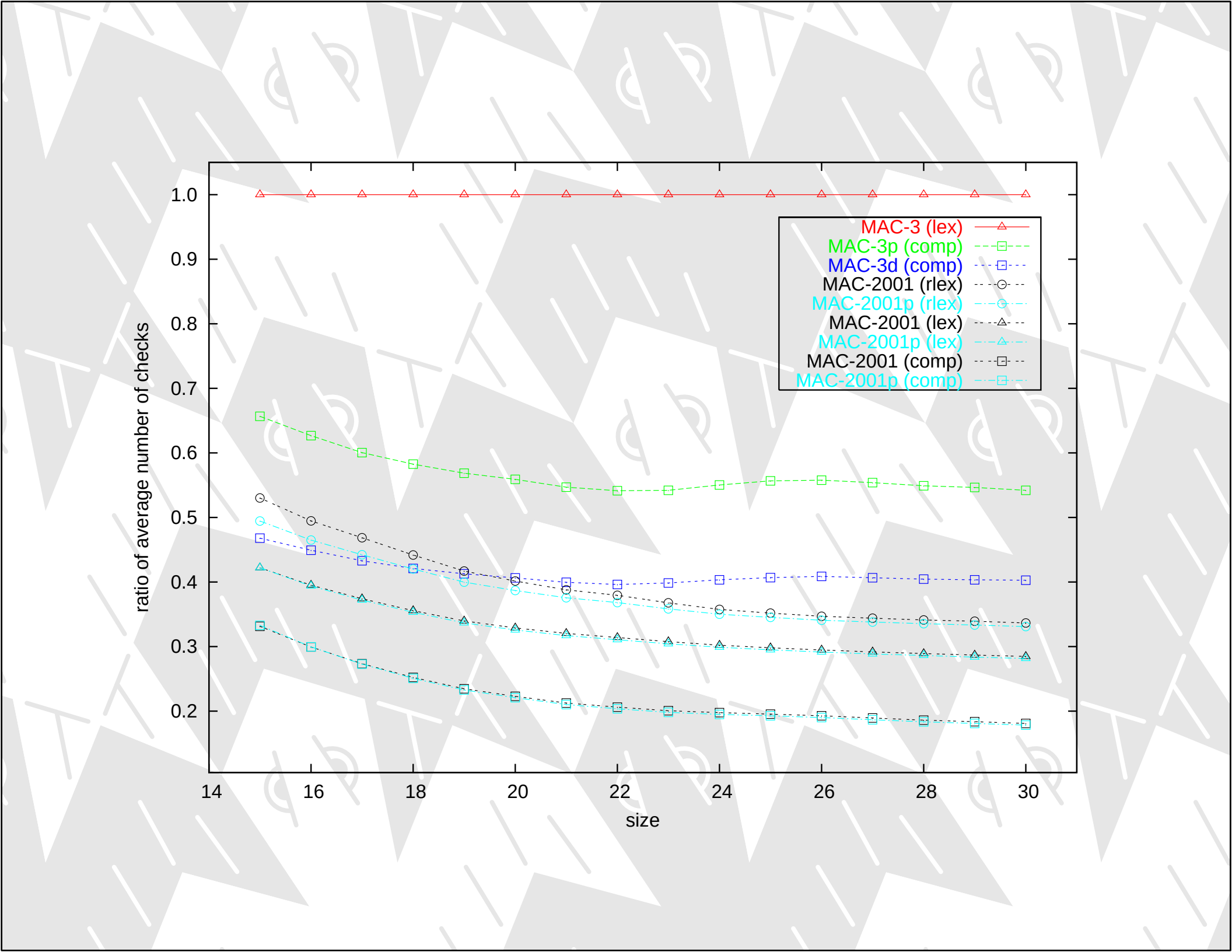
Experimental Results: The Random Problems

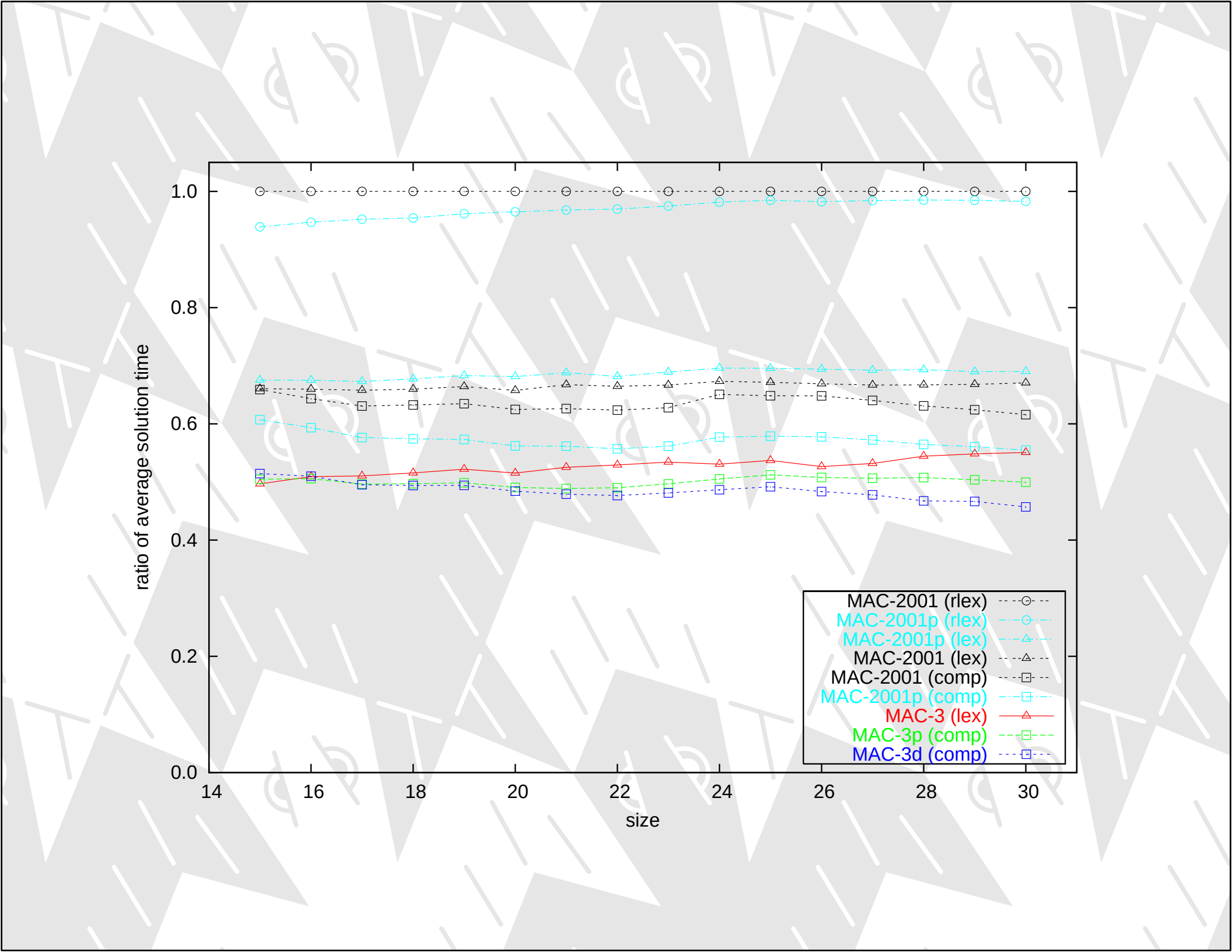
For each combination of density and uniform tightness in $\{(i/20, j/20) : 1 \leq i, j \leq 19\}$ we generated 50 random CSPs with s variables and s values, for $s \in \{15, \ldots, 30\}$.

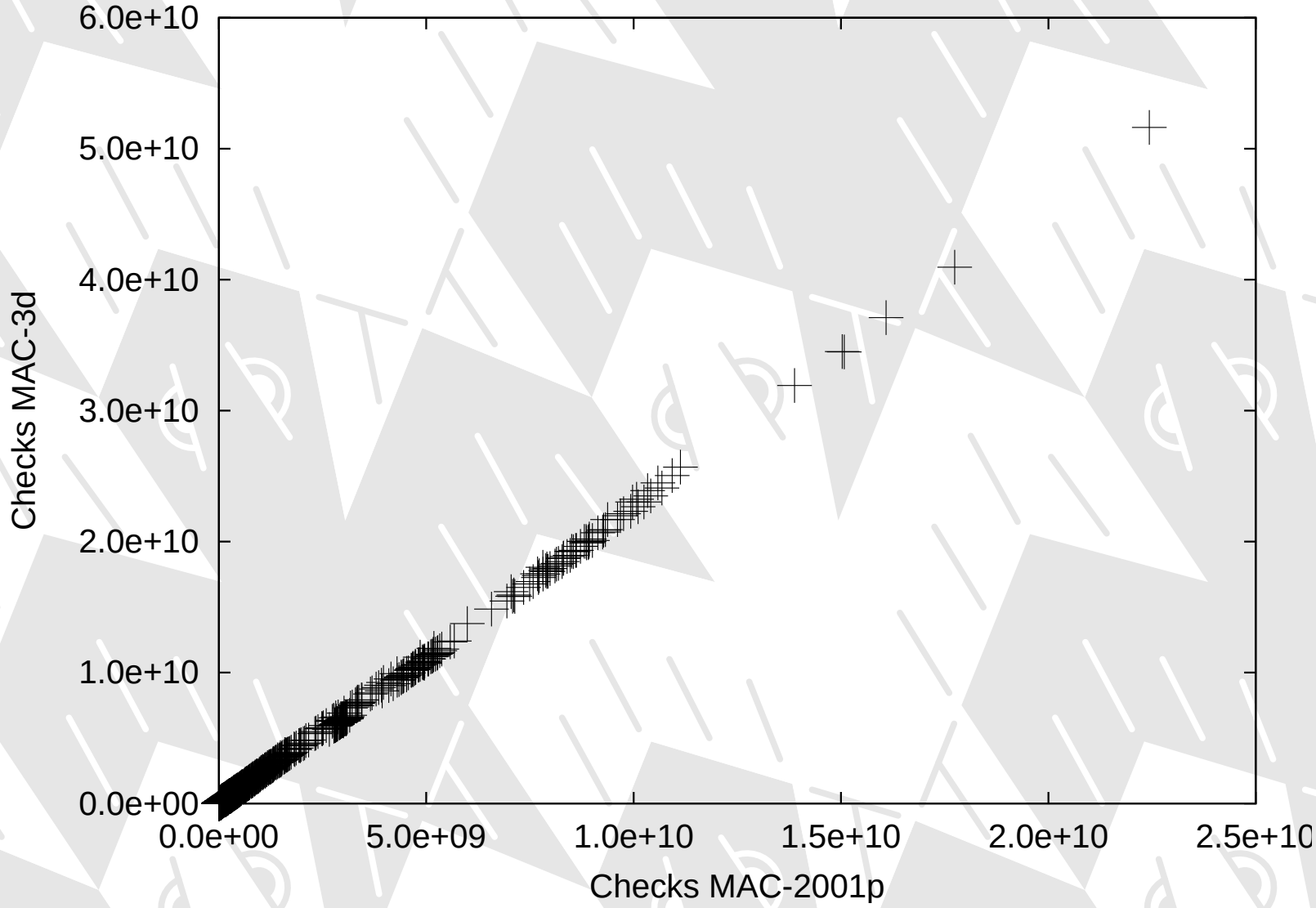
Next, for each combination of density and tightness, we computed the average number of checks and the average time that were required to solve the 50 CSPs for that combination.

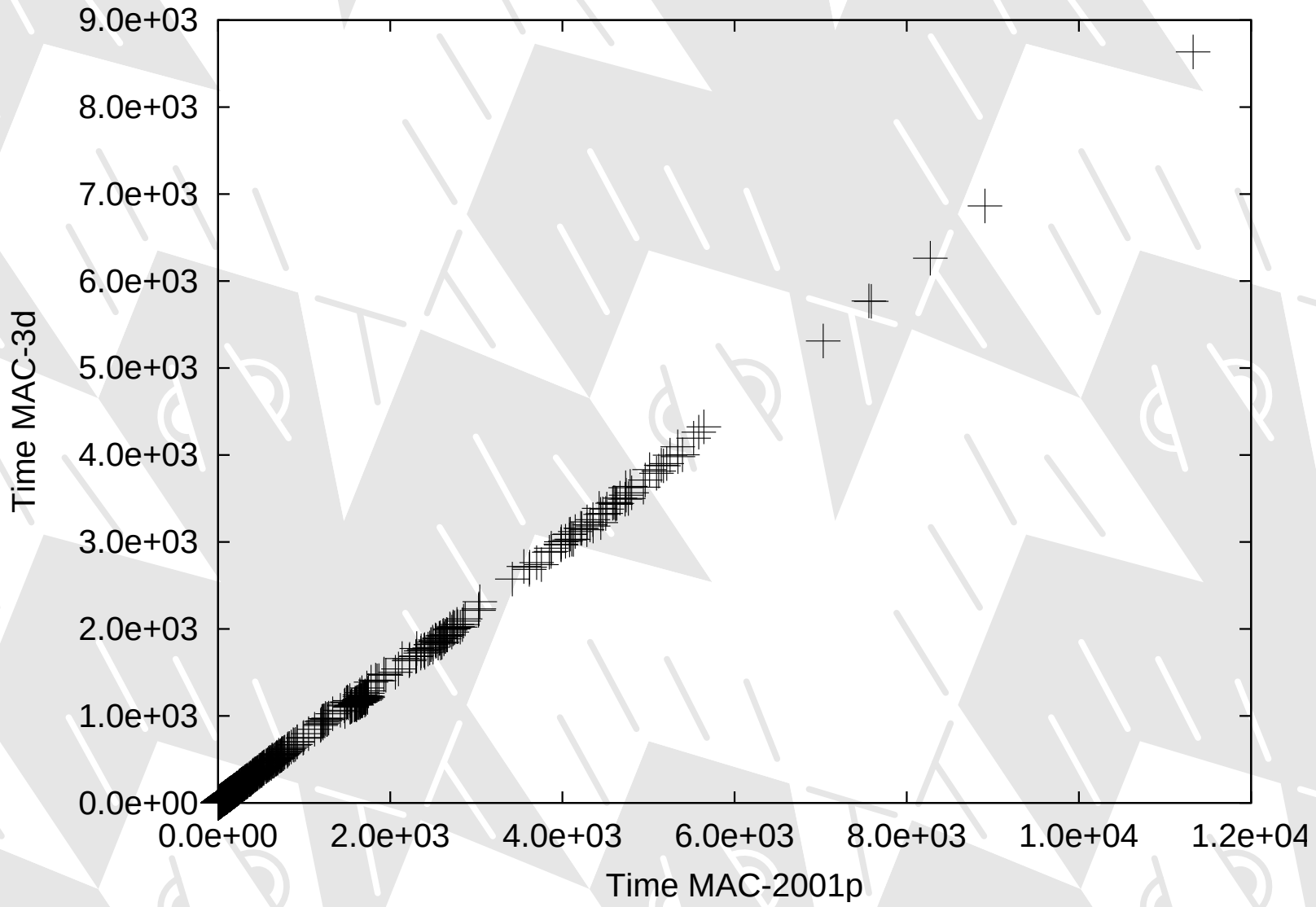

MAC-3 (lex)
MAC-3p (comp)
MAC-3d (comp)
MAC-2001 (rlex)
MAC-2001p (rlex)
MAC-2001 (lex)
MAC-2001p (lex)
MAC-2001 (comp)
MAC-2001p (comp)
average number of checks
4.50e+08
4.00e+08
3.50e+08
3.00e+08
2.50e+08
2.00e+08
1.50e+08
1.00e+08
5.00e+07
0.00e+00
14
16
18
20
22
24
26
28
30
size


MAC-2001 (rlex)
MAC-2001p (rlex)
MAC-2001 (lex)
MAC-2001p (lex)
MAC-2001 (comp)
MAC-2001p (comp)
MAC-3 (lex)
MAC-3p (comp)
MAC-3d (comp)
average solution time
7.00e+01
6.00e+01
5.00e+01
4.00e+01
3.00e+01
2.00e+01
1.00e+01
0.00e+00
14
16
18
20
22
24
26
28
30
size


ratio of average number of checks
size
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
14
16
18
20
22
24
26
28
30
MAC-3 (lex)
MAC-3p (comp)
MAC-3d (comp)
MAC-2001 (rlex)
MAC-2001p (rlex)
MAC-2001 (lex)
MAC-2001p (lex)
MAC-2001 (comp)
MAC-2001p (comp)


ratio of average solution time
size
1.0
0.8
0.6
0.4
0.2
0.0
14
16
18
20
22
24
26
28
30
MAC-2001 (rlex)
MAC-2001p (rlex)
MAC-2001p (lex)
MAC-2001 (lex)
MAC-2001 (comp)
MAC-2001p (comp)
MAC-3 (lex)
MAC-3p (comp)
MAC-3d (comp)


6.0e+10
5.0e+10
4.0e+10
3.0e+10
2.0e+10
1.0e+10
0.0e+00
Checks MAC-3d
0.0e+00
5.0e+09
1.0e+10
1.5e+10
2.0e+10
2.5e+10
Checks MAC-2001p


9.0e+03
8.0e+03
7.0e+03
6.0e+03
5.0e+03
4.0e+03
3.0e+03
2.0e+03
1.0e+03
0.0e+00
Time MAC-3d
0.0e+00
2.0e+03
4.0e+03
6.0e+03
8.0e+03
1.0e+04
1.2e+04
Time MAC-2001p

Experimental Results: The Real-World Problems

- RLFAP and GRAPH problems from the CELAR suite;
- Large very *sparse* optimisation problems;
- We only considered the satisfiability problem.

Algorithm	Problem		Checks			Time		
			lex	*rlex*	*comp*	*lex*	*rlex*	*comp*
MAC-3	RLFAP	1	4.24e+06	4.02e+06	4.17e+06	0.50	0.58	0.59
MAC-3_p	RLFAP	1	3.89e+06	3.97e+06	3.64e+06	0.47	0.52	0.51
MAC-3_d	RLFAP	1	2.60e+06	2.67e+06	1.92e+06	0.38	0.43	0.38
MAC-2001$_p$	RLFAP	1	1.85e+06	1.85e+06	1.78e+06	0.38	0.43	0.44
MAC-3	RLFAP	11	2.90e+08	1.57e+08	5.66e+07	34.12	20.63	7.86
MAC-3_p	RLFAP	11	2.13e+08	1.42e+08	4.37e+07	25.78	18.60	6.20
MAC-3_d	RLFAP	11	1.72e+08	1.15e+08	3.09e+07	23.10	16.91	5.35
MAC-2001$_p$	RLFAP	11	3.48e+07	2.91e+07	1.04e+07	11.74	10.25	3.87
MAC-3	GRAPH	9	4.43e+06	4.51e+06	3.90e+06	0.54	0.65	0.58
MAC-3_p	GRAPH	9	4.33e+06	4.48e+06	3.59e+06	0.54	0.60	0.52
MAC-3_d	GRAPH	9	3.31e+06	3.43e+06	2.18e+06	0.47	0.52	0.42
MAC-2001$_p$	GRAPH	9	1.86e+06	1.87e+06	1.79e+06	0.42	0.46	0.46
MAC-3	GRAPH	10	8.25e+06	8.30e+06	5.68e+06	1.00	1.13	0.85
MAC-3_p	GRAPH	10	8.08e+06	8.57e+06	5.50e+06	0.98	1.13	0.80
MAC-3_d	GRAPH	10	7.02e+06	7.49e+06	4.29e+06	0.90	1.05	0.71
MAC-2001$_p$	GRAPH	10	2.67e+06	2.74e+06	2.33e+06	0.60	0.72	0.61
MAC-3	GRAPH	14	3.89e+06	3.95e+06	3.40e+06	0.48	0.56	0.50
MAC-3_p	GRAPH	14	3.83e+06	3.92e+06	3.09e+06	0.48	0.51	0.45
MAC-3_d	GRAPH	14	2.87e+06	2.96e+06	1.73e+06	0.41	0.45	0.34
MAC-2001$_p$	GRAPH	14	1.65e+06	1.65e+06	1.59e+06	0.36	0.39	0.39

Conclusions & Future Work

- MAC-2001's being good at saving checks is expensive in time.
- ⋆ MAC-3_d does not rely on an optimal arc-consistency component;
  ⋆ MAC-3_d does not need additional data structures during search;
  ⋆ MAC-3_d has a better space-complexity; and
  ⋆ MAC-3_d *seems* to have a better average time-complexity.
- Can we explain these results theoretically?
- What *exactly* is the role of arc-heuristics?
- Can we learn from this lesson for other consistency algorithms?