JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

pubs.acs.org/jcim

Get Your Atoms in Order—An Open-Source Implementation of a Novel and Robust Molecular Canonicalization Algorithm

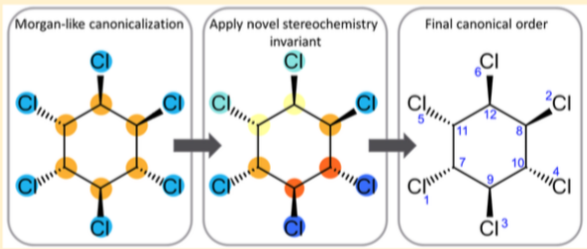
Nadine Schneider,*[,†] Roger A. Sayle,[‡] and Gregory A. Landrum[†]

[†]Novartis Institutes for BioMedical Research, Novartis Pharma AG, Novartis Campus, CH-4002 Basel, Switzerland

[‡]NextMove Software Ltd., Innovation Centre, Unit 23, Science Park, Milton Road, Cambridge CB4 0EY, U.K.

Supporting Information

ABSTRACT: Finding a canonical ordering of the atoms in a molecule is a prerequisite for generating a unique representation of the molecule. The canonicalization of a molecule is usually accomplished by applying some sort of graph relaxation algorithm, the most common of which is the Morgan algorithm. There are known issues with that algorithm that lead to noncanonical atom orderings as well as problems when it is applied to large molecules like proteins. Furthermore, each cheminformatics toolkit or software provides its own version of a canonical ordering, most based on unpublished algorithms, which also complicates the generation of a universal unique identifier for molecules. We present an alternative canonicalization approach that uses a standard stable-sorting algorithm instead of a Morgan-like index. Two new invariants that allow canonical ordering of molecules with dependent chirality as well as those with highly symmetrical cyclic graphs have been developed. The new approach proved to be robust and fast when tested on the 1.45 million compounds of the ChEMBL 20 data set in different scenarios like random renumbering of input atoms or SMILES round tripping. Our new algorithm is able to generate a canonical order of the atoms of protein molecules within a few milliseconds. The novel algorithm is implemented in the open-source cheminformatics toolkit RDKit. With this paper, we provide a reference Python implementation of the algorithm that could easily be integrated in any cheminformatics toolkit. This provides a first step toward a common standard for canonical atom ordering to generate a universal unique identifier for molecules other than InChI.



[...] All of the remaining 1 455 764 molecules except for two passed the [...] test. [...] The two molecules that did not result in a canonical representation are rather complex. [...]

SMILES

- The original version is proprietary, but OpenSMILES exist.
- Unclear/unfinished specification.
- Missing/unclear molecule model.
- No true support for conjugated bonds.
- Everyone seems to implement their own specification.
- Widespread belief that “the canonicalisation algorithm” works. (hint: it doesn’t)

InChI

- "IUPAC International Chemical Identifier"
- Identity crisis: is it a standard, tool, or algorithm?
- Missing/unclear molecule model.
- No separation between standard/specification and implementation.

InChI

- "IUPAC International Chemical Identifier"
- Identity crisis: is it a standard, tool, or algorithm?
- Missing/unclear molecule model.
- No separation between standard/specification and implementation.
- "the InChI source code [...] acts as the final arbiter of the correctness" — [InChI Tech. FAQ]
- "Mathematical details of the algorithms used will not be presented. They have been derived from methods reported in the literature [...]. They will be made available in the form of tested and documented source code along with the final version of the InChI" — [InChI Tech. Manual]

InChI

- "IUPAC International Chemical Identifier"
- Identity crisis: is it a standard, tool, or algorithm?
- Missing/unclear molecule model.
- No separation between standard/specification and implementation.
- "the InChI source code [...] acts as the final arbiter of the correctness" — [InChI Tech. FAQ]
- "Mathematical details of the algorithms used will not be presented. They have been derived from methods reported in the literature [...]. They will be made available in the form of tested and documented source code along with the final version of the InChI" — [InChI Tech. Manual]
-
```
if ( k < r ) {
    goto L9; /* cannot understand it... */
}
```

[InChI source code, the canonicalisation code]

Existing Tools

Canonicalization and automorphisms: nauty, Traces, Bliss
Automorphisms: Saucy
Isomorphism (and automorphisms): Conauto

- All based on individualization-refinement.
- (Almost) independent implementations.
- Different sets of heuristics and variations.
- Comparison has traditionally been done tool vs. tool.

Here:

- A generic framework for canonicalization algorithms.
- Algorithm variations implementable as individual plugins.
- Specified via generic programming, implemented in C++.
- Allows direct comparison of variations.
- Lower barrier of entry for implementing new ideas.
- Generality wrt. vertex/edge attributes.

[McKay, Congressus Numerantium, 1981] [McKay and Piperno, J. Symb. Comp., 2014] [Junttila and Kaski, ALENEX, 2007] [Darga et al., DAC, 2008] [López-Presa and Fernández Anta, SEA, 2009]

Abstract Algorithm

- Construct the search tree.
- For each leaf, construct the permuted graph using the discrete partition as a vertex permutation.
- For two such permuted graphs $G^{\overline{\pi_1}}$ and $G^{\overline{\pi_2}}$,
  - if $G^{\overline{\pi_1}} \overset{r}{<} G^{\overline{\pi_2}}$, discard π_2,
  - if $G^{\overline{\pi_1}} \overset{r}{=} G^{\overline{\pi_2}}$, yield $\overline{\pi_1}\pi_2$ as automorphism, and discard either π_1 or π_2.
- Return the permutation represented by the remaining leaf.

Pruning Techniques

- Use automorphisms to skip redundant subtrees.
- Use node invariants to remove subtrees (possibly changing which leaves are ever constructed).

Algorithm Variation

Categories

- Tree traversal
- Target cell selection

- Refinement
- Pruning with automorphisms
- Detection of implicit automorphisms
- Node invariants

DEMONSTRATION

Algorithm Variation

Tree Traversal:

- nauty, Bliss: depth-first (DFS)
- Traces: breadth-first with experimental paths (BFSExp)
- Here:
  - Arbitrary traversals are possible.
  - Garbage collected search tree via reference counting.
  - Extensions must keep owning references to tree nodes.
  - Implemented: DFS, BFSExp, and a new hybrid (BFSExpM).

Target Cell Selector:

- Many have been developed.
- Currently implemented:
  - first (F)
  - first largest (FL)
  - first largest with maximum number of non-uniformly joined neighbour cells (FLM)

Benchmarks

44 graph collections, with 4,715 graphs in total.
Time limit: 1000 s
Memory limit: 8 GB
Repetitions: 5

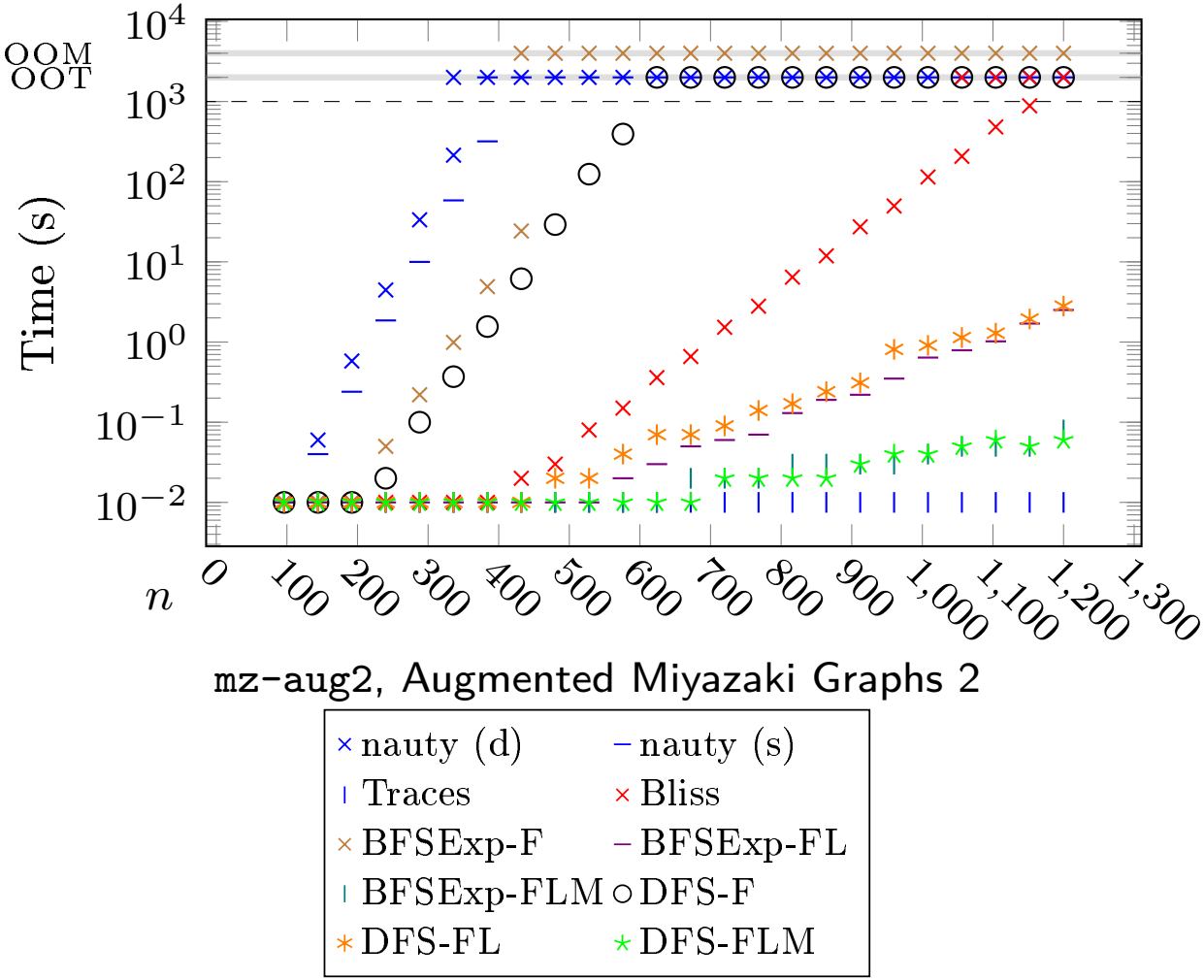
Algorithm configurations: $\{\mathrm{BFSExp}, \mathrm{DFS}\} \times \{\mathrm{F}, \mathrm{FL}, \mathrm{FLM}\}$

Compute nodes with two Intel E5-2680v3 CPUs (24 cores)
Compute node hours: approx. 12,000

BFSExp with FLM is often best.

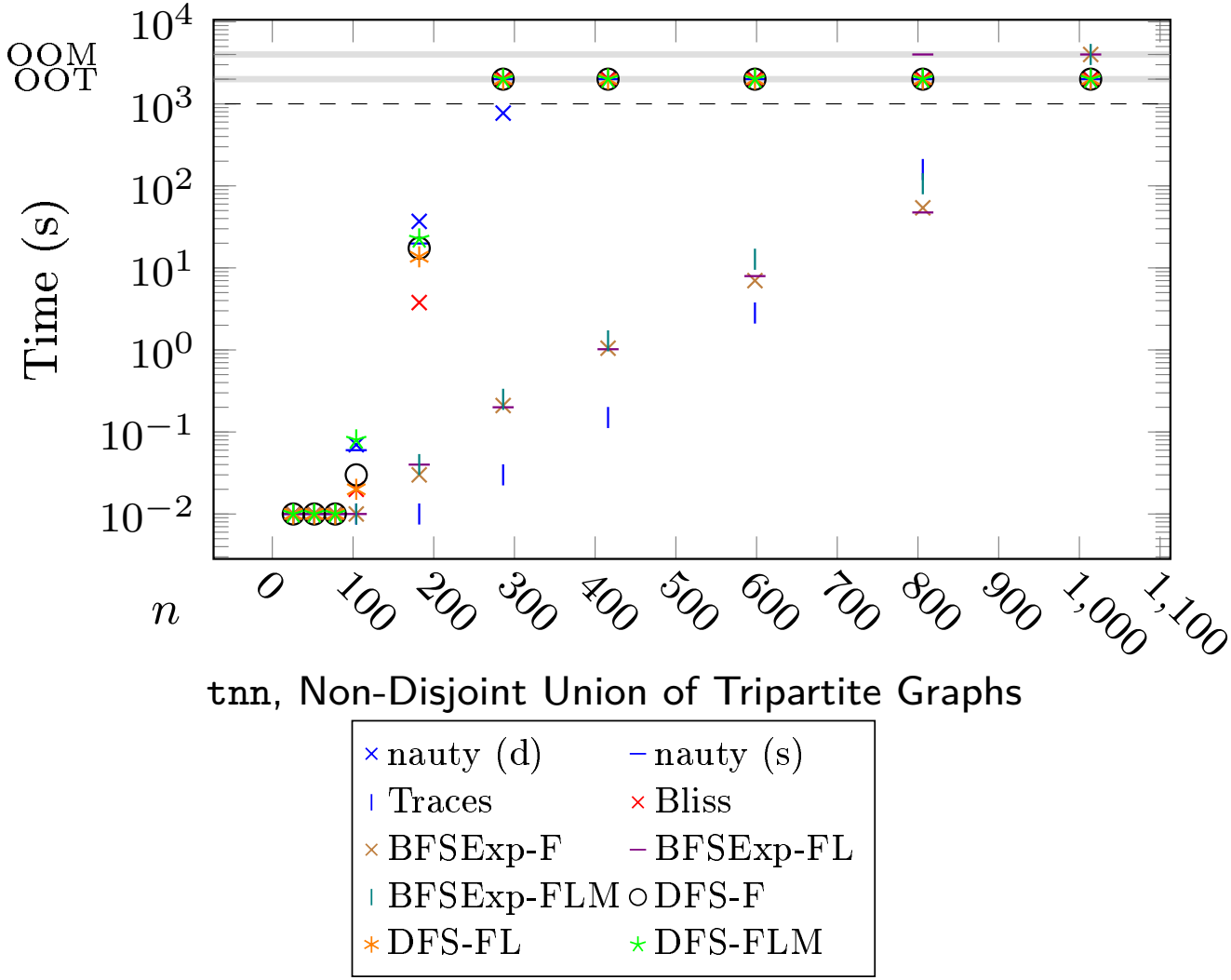
CFI-Rigid: [Neuen and Schweitzer, ESA, 2017]
nauty, Traces: [http://pallini.di.uniroma1.it/Graphs.html]
Bliss: [http://www.tcs.hut.fi/Software/bliss/benchmarks/index.shtml]
Conauto: [https://sites.google.com/site/giconauto/home/benchmarks]
Saucy: [http://vlsicad.eecs.umich.edu/BK/SAUCY/]

Tree Traversal and Target Cell Selector

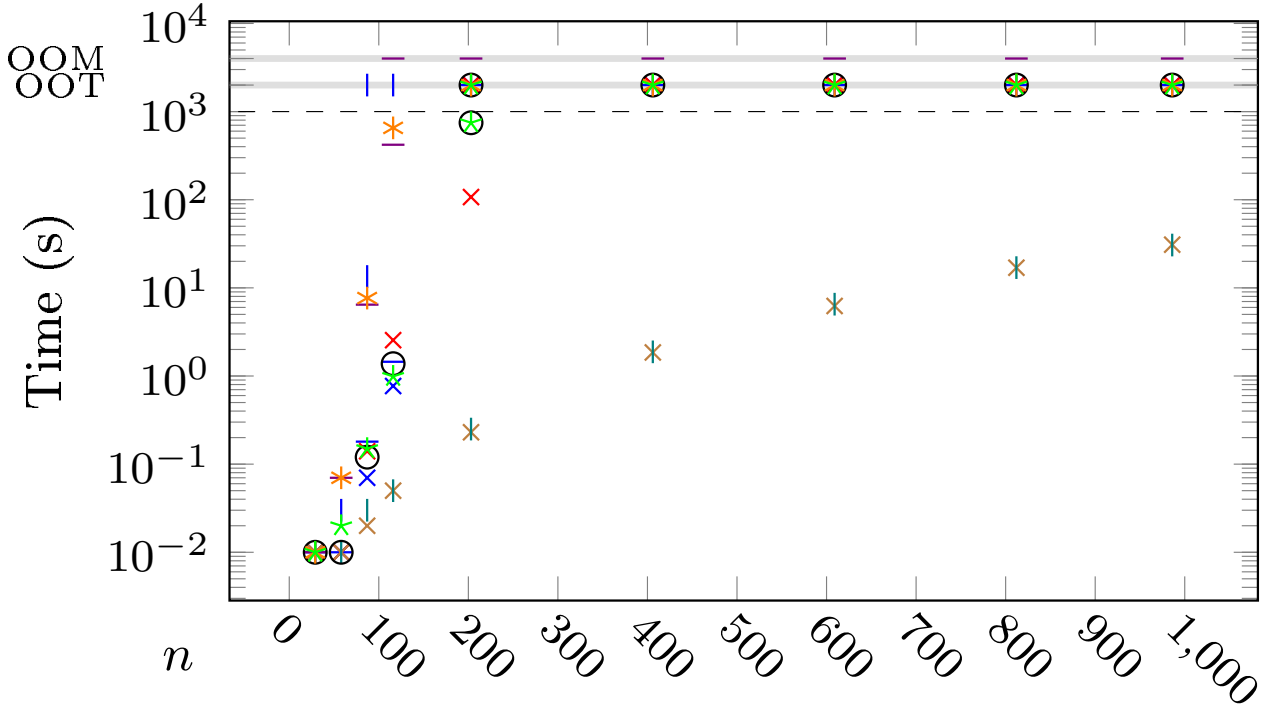


Similar characteristics observed for other Miyazaki graphs.

Tree Traversal and Target Cell Selector



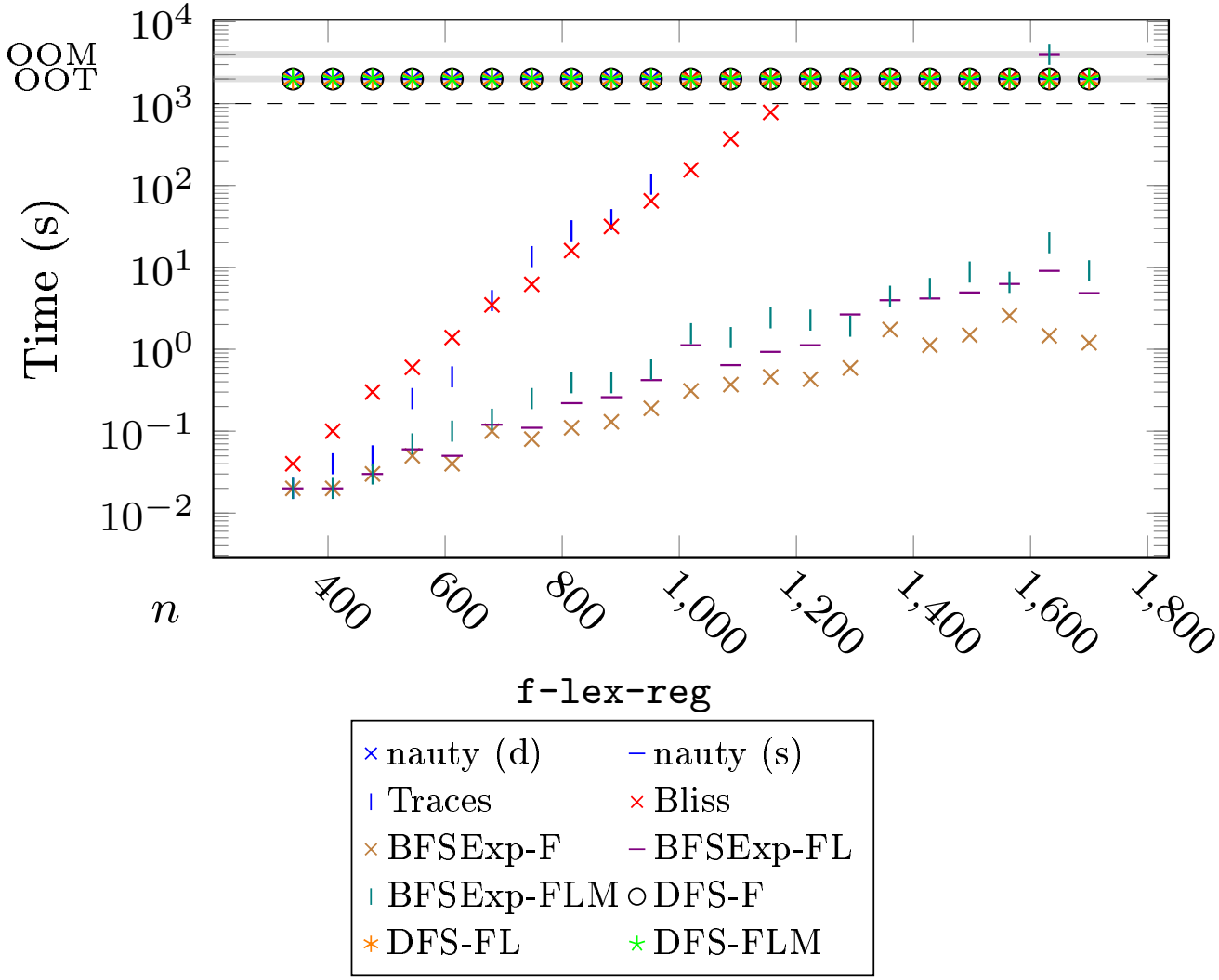
Tree Traversal and Target Cell Selector



usr, Union of Strongly Regular Graphs

× nauty (d)	– nauty (s)
| Traces	× Bliss
× BFSExp-F	– BFSExp-FL
| BFSExp-FLM	○ DFS-F
* DFS-FL	* DFS-FLM

Tree Traversal and Target Cell Selector



CFI-Rigid [Neuen and Schweitzer, ESA, 2017]

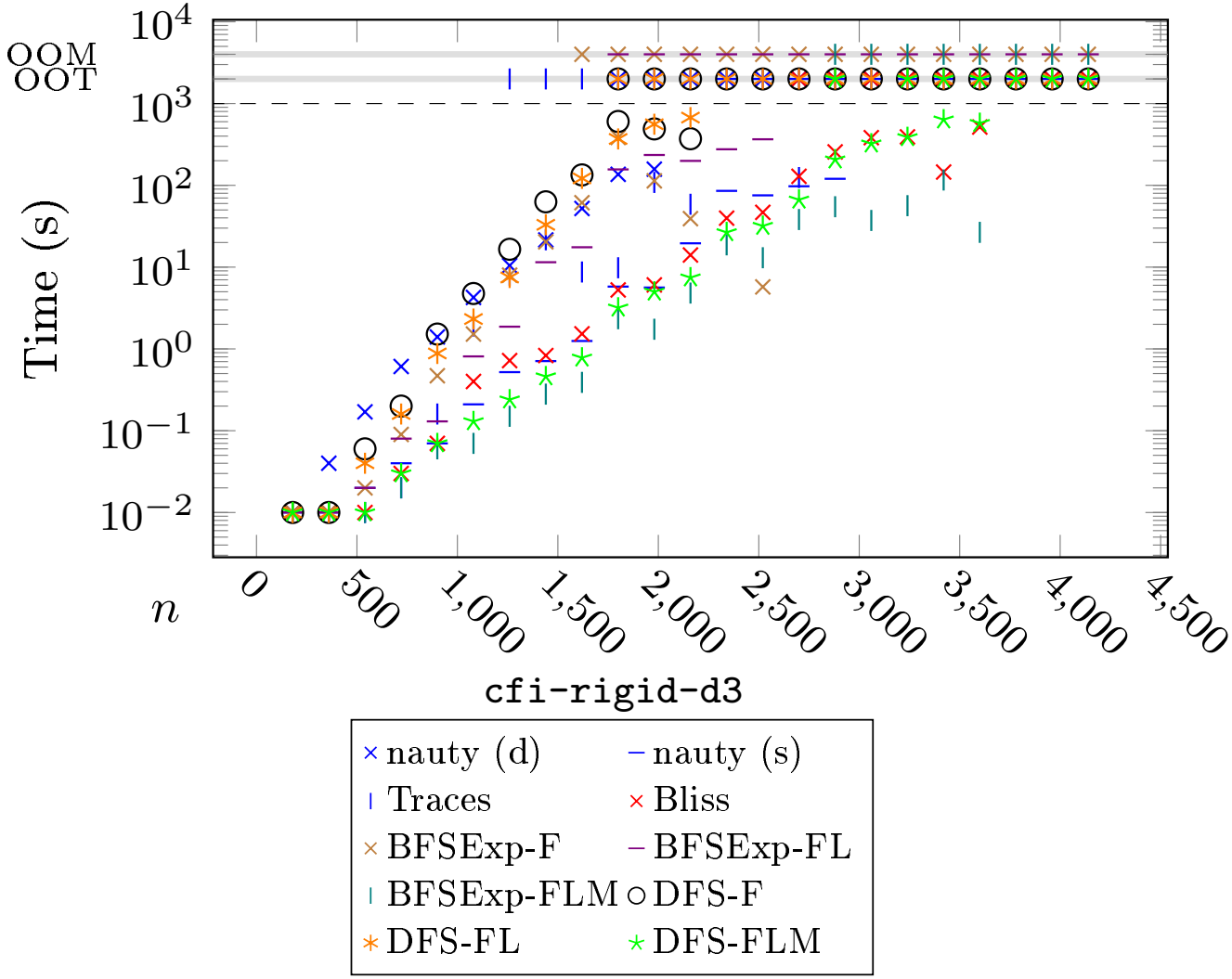
- 6 collections
- Designed to be the hard benchmarks.
- Expected to have very little symmetry.

Algorithm configurations:

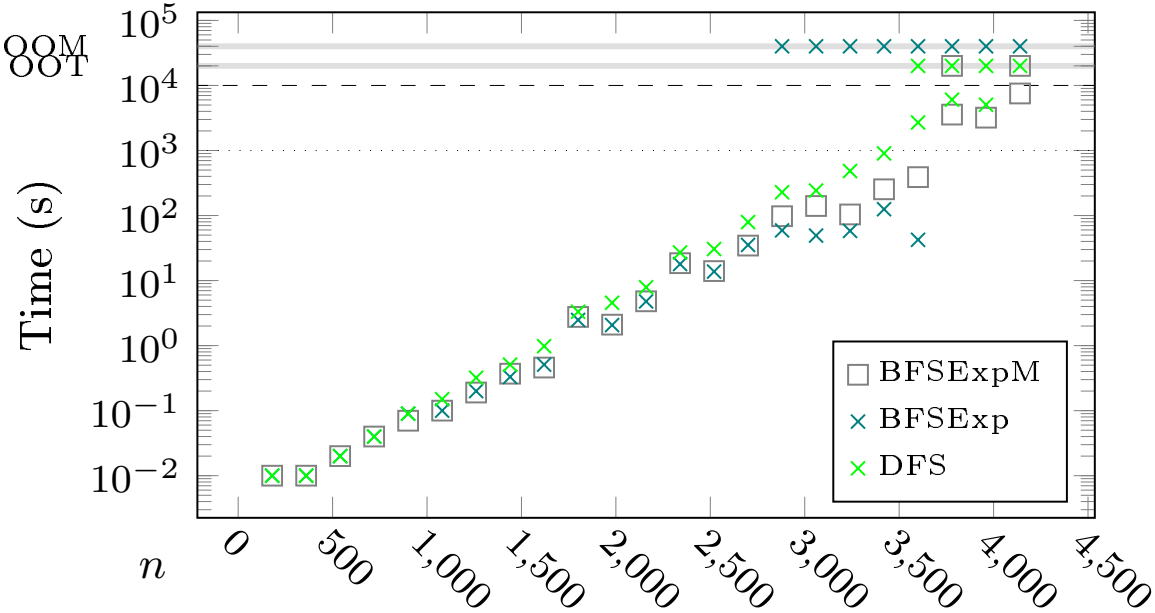
$$\{\mathrm{BFSExp}, \mathrm{DFS}\} \times \{\mathrm{F}, \mathrm{FL}, \mathrm{FLM}\} \times 2^{\{\mathrm{PL},\mathrm{Q},\mathrm{T}\}}$$

Col.	Group	Reduction	Best Algorithm	Invariants Matter	FLM Sep.	Max. Solved n
d3	D_3	—	BFSExp-FLM	yes (any)	yes	3,600
z3	$\mathbb{Z}_3$	—	BFSExp-FLM	yes (any)	yes	3,780
z2	$\mathbb{Z}_2$	—	Bliss, nauty (s)	yes (any)	no	2,992
r2	$\mathbb{Z}_2$	R^*	Bliss, nauty (s)	no	no	1,584
s2	$\mathbb{Z}_2$	B^*	FLM, Bliss, nauty (s)	yes (PL or Q)	no	2,496
t2	$\mathbb{Z}_2$	$R^* \circ B^*$	FLM, Bliss, nauty (s)	yes (PL or Q)	yes	1,056

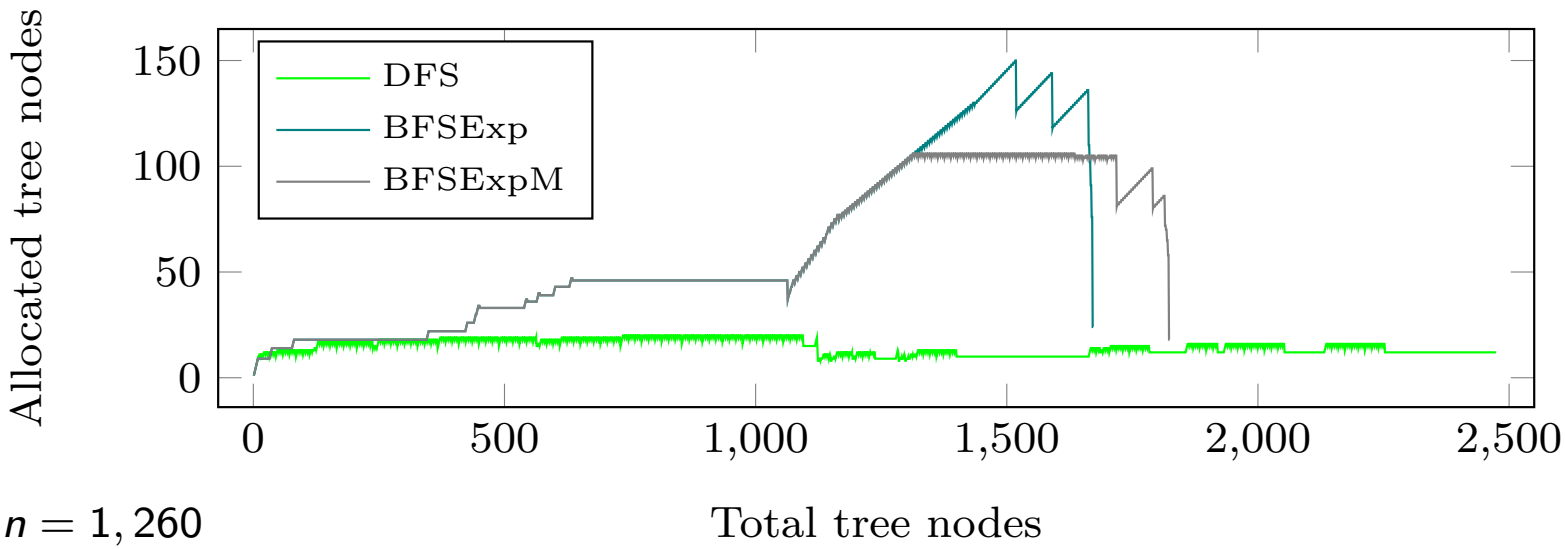
CFI-Rigid [Neuen and Schweitzer, ESA, 2017]



BFSExpM: Memory-Sensitive BFSExp



`cfi-rigid-d3`, Memory-limit: 2 MB



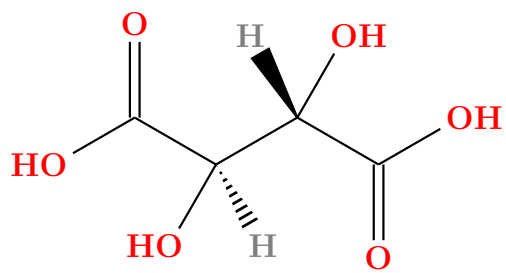
$n = 1,260$

Summary and Outlook

- Generic algorithm framework.
- (Relatively) easy to develop new variations.
- Allows direct comparison of algorithmic ideas.
- Competitive with established tools.
- Very easy to extract data for visualization.
- Details: see ALENEX 18 paper

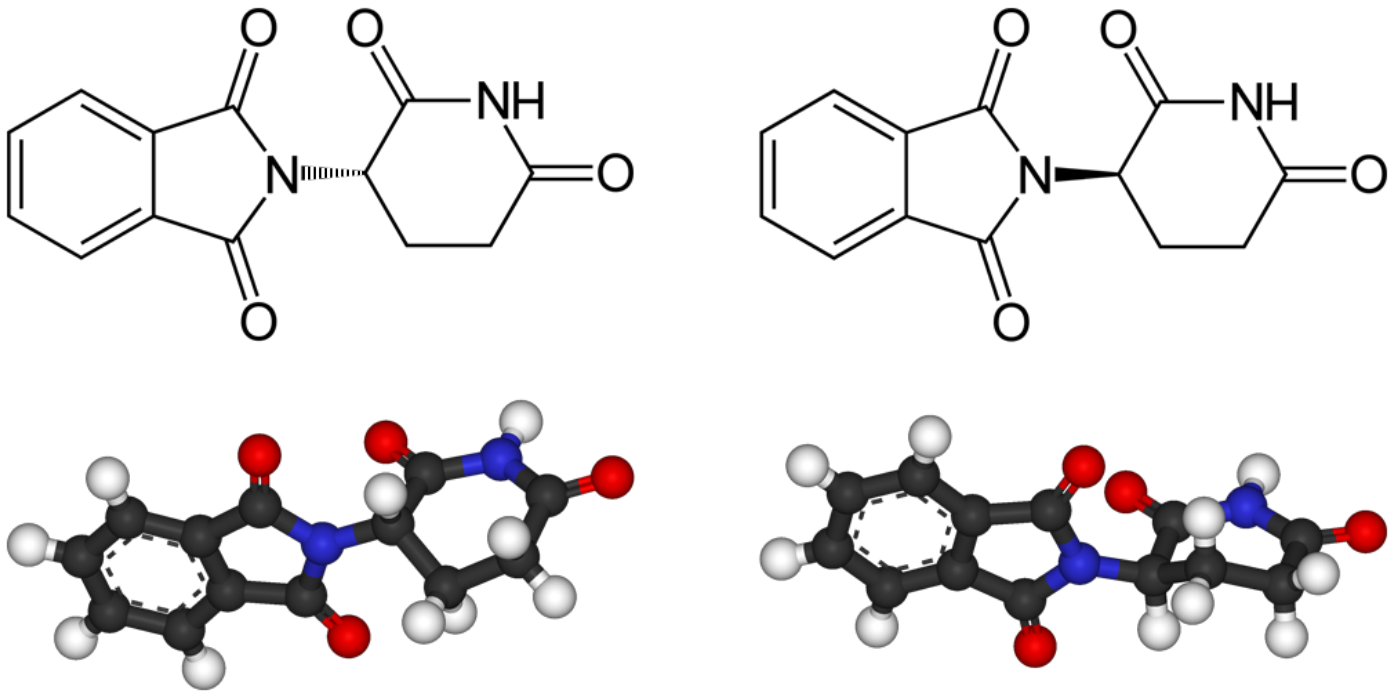
Current and future developments:

- Integration into our Cheminformatics framework (`mod.imada.sdu.dk`)
- Implement more variations (Schreier-Sims)
- Generalize handling of attributes.
  For example, stereo-information in molecule graphs.

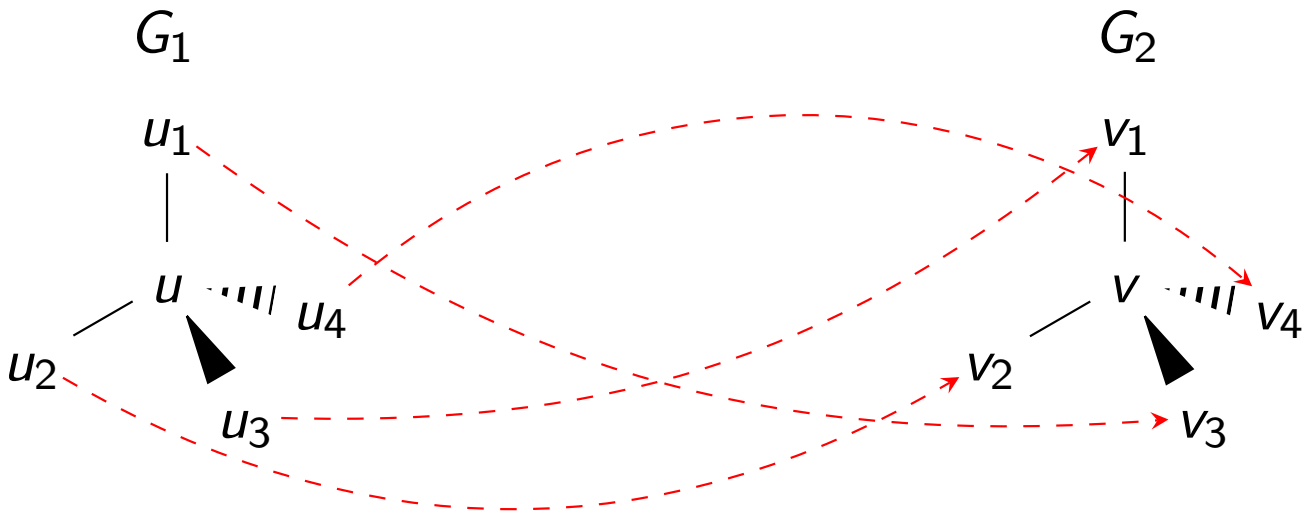


Sometimes Stereoisomerism is Important

- Thalidomide (Contergan): sedative vs. birth defects



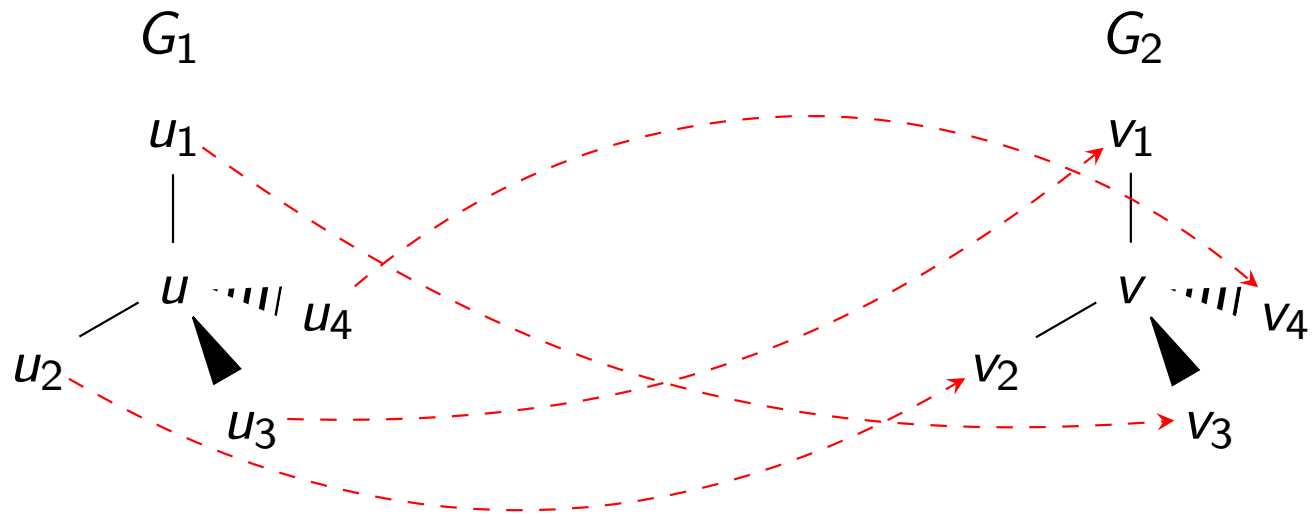
Morphism Example (Stereochemistry): TETRAHEDRAL



$N_u = [u_1, u_2, u_3, u_4]$

$N_v = [v_1, v_2, v_3, v_4]$

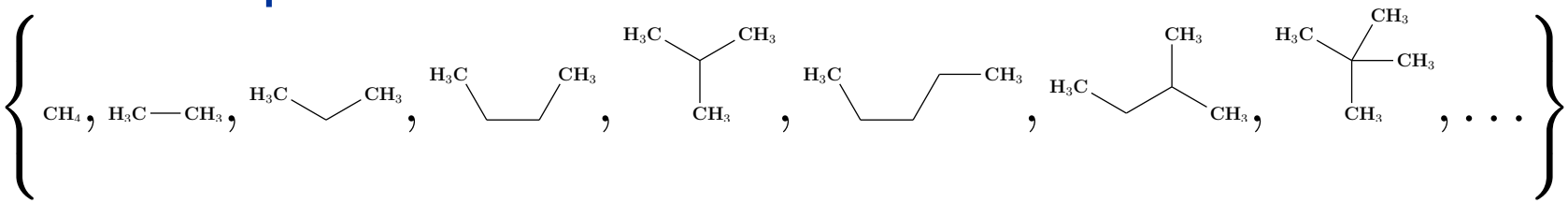
Morphism Example (Stereochemistry): TETRAHEDRAL



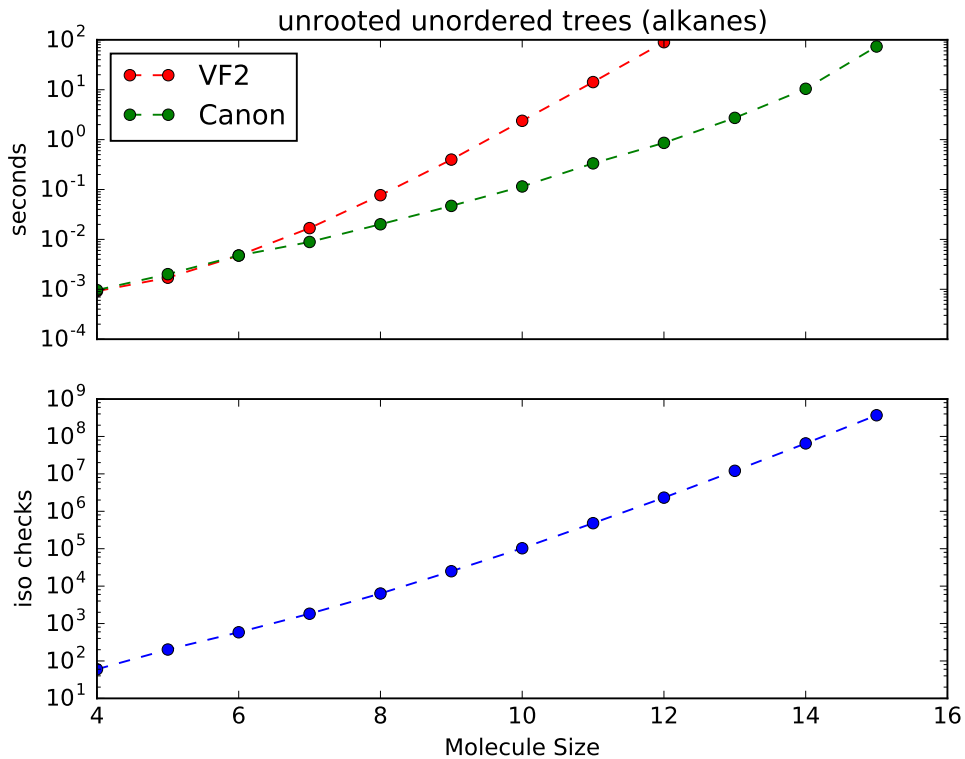
$N_u = [u_1, u_2, u_3, u_4]$ $\qquad$ $N_v = [v_1, v_2, v_3, v_4]$

Index map: $m_I = \{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 1, 4 \mapsto 4\}$
$= (1\ 3)(2)(4)$
$\notin \langle (1)(2\ 3\ 4), (1\ 2)(3\ 4) \rangle$

A rather stupid test case: Alkanes



- n-node unrooted quartic trees
- Number of trees: $A(n) \approx \frac{2.81546^n}{n^{5/2}}$
- Overall runtime / #isomorphism checks $\approx 160ns$



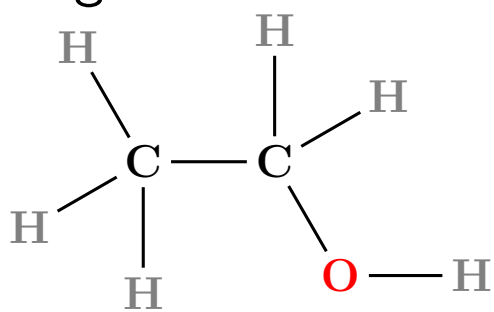
Algorithm Variation

Node Invariants:

- Totally ordered isomorphism-invariant information.
- Invariants can be implemented independently.
- A special visitor coordinates invariants.
- Implemented:
  - cell splitting positions (T), from Traces
  - quotient graph values (Q), from nauty, Traces, Bliss (but not hashed)
  - partial leaf (PL), from Bliss (but not hashed)
    Construct parts of the permuted graph earlier in the tree.

Refinement functions implemented:

- 1D Weisfeiler-Leman, generalized to exploit edge attributes.
- A function to handle degree-1 vertices.



Algorithm Variation

Detection of implicit automorphisms:

- Sometimes we can detect/guess automorphisms at internal tree nodes.
- nauty: several special cases of ordered partitions.
- Saucy: heuristics for guessing sparse automorphisms.
- Traces: reportedly a generalization of the Saucy heuristics.
- Implemented:
  - Partitions where all cells have size 1 or 2.
  - The degree-1 vertex refinement function.

Pruning with automorphisms:

Calculation of orbits in stabilizers of the found automorphisms.
Stabilizer calculation:

- nauty (early versions) and Bliss: conservative (implemented)
- Traces and nauty (recent versions): randomized Schreier-Sims

The implemented visitor for automorphism pruning is generic with respect to stabilizer implementation.