Canonicalization of Data Structures

Jakob Lykke Andersen

Research Group Bioinformatics and Computational Biology
Faculty of Computer Science
University of Vienna

Institute for Theoretical Chemistry
University of Vienna

Bled, February 2018

Introduction

Model

A mathematical object in some class M.

Example: a rational number, $\frac{3}{4}$

Introduction

Model

A mathematical object in some class M.

Example: a rational number, $\frac{3}{4}$

Representation

An object of an abstract data type R used to store the model.

Example: a pair of integers, $(3, 4)$

Introduction

Model

A mathematical object in some class M.

Example: a rational number, $\frac{3}{4}$

Representation

An object of an abstract data type R used to store the model.

Example: a pair of integers, $(3, 4)$

Implementation

An object of a concrete type used to store the model.

Example: `std::pair<int, int>(3, 4)`

Introduction

Model

A mathematical object in some class M.

Example: a rational number, $\frac{3}{4}$

Representation

An object of an abstract data type R used to store the model.

Example: a pair of integers, $(3, 4)$

Implementation

An object of a concrete type used to store the model.

Example: `std::pair<int, int>(3, 4)`

What if a model has multiple representations?

Example

$M = \mathbb{Q}$, the rational numbers

$R = \mathbb{Z} \times \mathbb{Z}$, pairs of integers

But $\frac{2}{5} = \frac{4}{10}$, so $(2, 5)$ should be considered “equal” to $(4, 10)$.

Notation: $(2, 5) \cong (4, 10)$ (“isomorphic to”)

$(2, 5) \stackrel{r}{=} (2, 5)$ (“representationally equal to”)

Canonicalization

Given a representation $G \in R$ find a new representation $C(G)$, such that:

- It represents the same model: $C(G) \cong G$
- All canonicalized isomorphic representations are the same:
  $\forall G' \in R, G' \cong G : \quad C(G') \stackrel{r}{=} C(G)$

How do we specify and implement canonicalization in practice?

Representations

Besides the $\stackrel{r}{=}$ operation we need:

- A class of operations, $\textsc{Op}$, that do not change the model.
- A total order $\stackrel{r}{<}$ among (isomorphic) representations.

Fraction Example:

$\textsc{Op}$:

- Multiplying with an integer: $(2,5)\cdot 2 = (2\cdot 2, 5\cdot 5) \cong (2,5)$
- Dividing with a common factor: $\frac{(4,10)}{2} = \left(\frac{4}{2}, \frac{10}{2}\right) \cong (4,10)$
- (and compositions of those operations)

$\stackrel{r}{<}$:

- Prefer both positive over both negative: $(2,5) \stackrel{r}{<} (-2,-5)$
- Prefer (neg., pos.) over (pos., neg.): $(-2,5) \stackrel{r}{<} (2,-5)$
- Prefer smaller (absolute) numbers (lexicographically): $(2,5) \stackrel{r}{<} (4,10), \quad (1,2) \stackrel{r}{<} (2,3)$

Canonicalization

Given $G \in R$:

- Find $op \in \textsc{Op}$ that minimizes $op(G)$, wrt. $\overset{r}{<}$
- Return $op(G)$ as the canonical form.

Fraction Example:

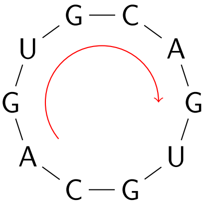
Given (a, b),

- Find $f = GCD(|a|, |b|)$
- If $b < 0$: let $op = \textsc{Div}(f) \circ \textsc{Mul}(\text{-}1)$
  else: let $op = \textsc{Div}(f)$
- Return $op((a, b))$

In Practice:

- Probably return op. The user can compute $op(G)$ if needed.
- $\overset{r}{<}$ may be implicitly defined by the canonicalization algorithm.

Example: Circular RNA (circRNA)



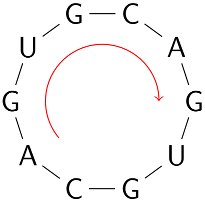
Representation: A sequence of symbols A, C, G, U.
  Example: AGUGCAGUGC

Operations: $\textsc{Rotate}(i)$, for $i \in \mathbb{Z}$
  Example: $\textsc{Rotate}(2, \text{AGUGCAGUGC}) = \text{UGCAGUGCAG}$

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: find the lexicographically smallest rotation
(can be done in linear time)

Example: Circular RNA (circRNA)



Representation: A sequence of symbols A, C, G, U.
  Example: AGUGCAGUGC

Operations: $\textsc{Rotate}(i)$, for $i \in \mathbb{Z}$
  Example: $\textsc{Rotate}(2, \text{AGUGCAGUGC}) = \text{UGCAGUGCAG}$

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: find the lexicographically smallest rotation
(can be done in linear time)

Symmetry Discovery: op is a symmetry if $op(G) \overset{r}{=} G$
  Example: $\textsc{Rotate}(5)$ is a symmetry of AGUGCAGUGC, because

$$\begin{aligned} \textsc{Rotate}(5, \text{AGUGCAGUGC}) &= \text{AGUGCAGUGC} \\ &\overset{r}{=} \text{AGUGCAGUGC} \end{aligned}$$

$\textsc{Rotate}(0)$ is a trivial symmetry

Example: Double Stranded RNA

Representation:

A pair of sequences of symbols A, C, G, U, of equal length.

Example: $\begin{smallmatrix}\mathsf{AGUGC}\\\mathsf{UCACG}\end{smallmatrix}$

Operations: $\textsc{Reverse} \circ \textsc{Swap}$

Example: $(\textsc{Reverse} \circ \textsc{Swap}) \binom{\mathsf{AGUGC}}{\mathsf{UCACG}} = \begin{smallmatrix}\mathsf{GCACU}\\\mathsf{CGUGA}\end{smallmatrix}$

$\stackrel{r}{=}$ and $\stackrel{r}{<}$: component-wise and lexicographic comparison

Example: $\begin{smallmatrix}\mathsf{AGUGC}\\\mathsf{UCACG}\end{smallmatrix} \stackrel{r}{<} \begin{smallmatrix}\mathsf{GCACU}\\\mathsf{CGUGA}\end{smallmatrix}$

Canonicalization: take the $\stackrel{r}{<}$-smallest of the two possibilities

Example: Double Stranded RNA, Only Binding Structure

Swapping all A with U and G with C preserves structure.

Representation:
A pair of sequences of symbols A, C, G, U, of equal length.

Operations: $\textsc{Reverse} \circ \textsc{Swap}$ and $\textsc{Invert}$ $(= \textsc{Swap})$
Example: $\textsc{Invert}\begin{pmatrix}\mathsf{AGUGC}\\ \mathsf{UCACG}\end{pmatrix} = \begin{pmatrix}\mathsf{UCACG}\\ \mathsf{AGUGC}\end{pmatrix}$

$\stackrel{r}{=}$ and $\stackrel{r}{<}$: component-wise and lexicographic comparison

Canonicalization: take the $\stackrel{r}{<}$-smallest of the four possibilities

Example: Anti-Parallel Strong Traces, Take 1

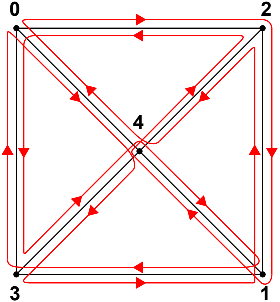
Model: A graph G (representing a polygon), with a closed walk visiting all edges twice and $\langle$more constraints$\rangle$.

Representation: A sequence of vertices $t = (v_{i_1}, v_{i_2}, \ldots, v_{i_{2m}})$.

Operations: $\textsc{Reverse}(t)$, $\textsc{Rotate}(i, t)$, and $\textsc{Permute}(\gamma, t)$ for any automorphism (i.e., symmetry) γ of G.

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: take the $\overset{r}{<}$-smallest (not trivial to do efficiently)



[Hellmuth et al., ALENEX, 2018]

[Bašić et al., MATCH, 2017]

Example: Anti-Parallel Strong Traces, Take 1

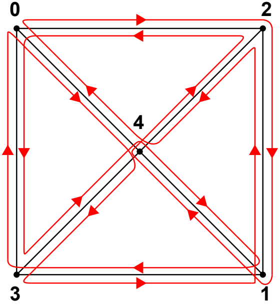
Model: A graph G (representing a polygon), with a closed walk visiting all edges twice and $\langle$more constraints$\rangle$.

Representation: A sequence of vertices $t = (v_{i_1}, v_{i_2}, \ldots, v_{i_{2m}})$.

Operations: $\textsc{Reverse}(t)$, $\textsc{Rotate}(i, t)$, and $\textsc{Permute}(\gamma, t)$ for any automorphism (i.e., symmetry) γ of G.

$\stackrel{r}{=}$ and $\stackrel{r}{<}$: component-wise and lexicographic comparison

Canonicalization: take the $\stackrel{r}{<}$-smallest (not trivial to do efficiently)



[Hellmuth et al., ALENEX, 2018]

But what about the graph?
What is a vertex?
What is the representation?

[Bašić et al., MATCH, 2017]

Example: Graphs, Part 1

Model: A graph $G = (V, E)$.

Representation: An adjacency list which implicitly assigns $1, 2, \ldots, n$ to V.

Operations: $\textsc{Permute}(\gamma)$ for any permutation of $1, 2, \ldots, n$.

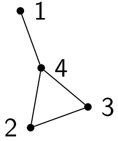
$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: $\langle$more on this later$\rangle$

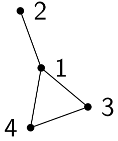
Graph Representation and Graph Permutation

$G = (V, E) \quad V = \{1, 2, \ldots, n\}$

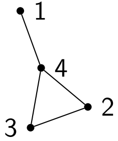
Isomorphic graphs, different representations:



G_1



G



G_2

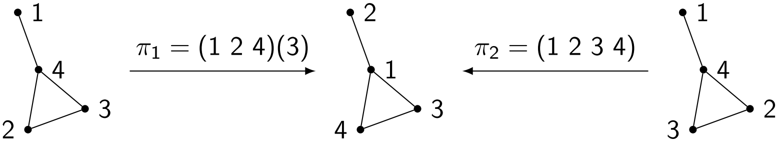
Adjacency list representation (with sorted neighbour lists):

1: 4	1: 2, 3, 4	1: 4
2: 3, 4	2: 1	2: 3, 4
3: 2, 4	3: 1, 4	3: 2, 4
4: 1, 2, 3	4: 1, 3	4: 1, 2, 3

Graph Representation and Graph Permutation

$G = (V, E) \quad V = \{1, 2, \ldots, n\}$

Isomorphic graphs, different representations:



G_1 $\qquad G_1^{\pi_1} \stackrel{r}{=} G \stackrel{r}{=} G_2^{\pi_2}$ $\qquad G_2$

Adjacency list representation (with sorted neighbour lists):

1: 4	1: 2, 3, 4	1: 4
2: 3, 4	2: 1	2: 3, 4
3: 2, 4	3: 1, 4	3: 2, 4
4: 1, 2, 3	4: 1, 3	4: 1, 2, 3

Example: Anti-Parallel Strong Traces, Take 2

Model: A graph G (representing a polygon), with a closed walk visiting all edges twice and $\langle$more constraints$\rangle$.

Representation: An adjacency list, and a sequence of integers $t = (v_{i_1}, v_{i_2}, \ldots, v_{i_{2m}})$.

Operations:

- $\textsc{Reverse}(t)$
- $\textsc{Rotate}(i, t)$
- $\textsc{Permute}(\gamma, t)$ for any automorphism γ of G.
- $\textsc{Permute}(\gamma, t, G)$ for any permutation γ of V.

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

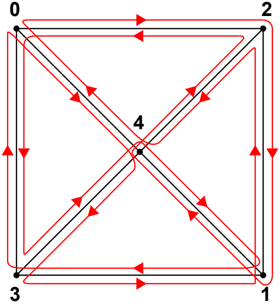
Canonicalization: take the $\overset{r}{<}$-smallest

[Bašić et al., MATCH, 2017]

Example: Anti-Parallel Strong Traces, Take 3

Model: A graph G (representing a polygon), with a closed walk visiting all edges twice and $\langle$more constraints$\rangle$.

Representation: An adjacency list, and a sequence of integers representing a *gap vector* $g = (a_1, a_2, \ldots a_{2m})$.



$$t = (0, 2, 1, 4, 2, 0, 3, 4, 1, 3, 0, 4, 3, 1, 2, 4)$$
$$g = (5, 3, 6, 4, 10, 5, 3, 4, 5, 3, 6, 4, 10, 5, 3, 4)$$

[Hellmuth et al., ALENEX, 2018]

Example: Anti-Parallel Strong Traces, Take 3

Model: A graph G (representing a polygon), with a closed walk visiting all edges twice and $\langle$more constraints$\rangle$.

Representation: An adjacency list, and a sequence of integers representing a *gap vector* $g = (a_1, a_2, \ldots a_{2m})$.

Operations:

- $\textsc{Rotate}(i, g)$
- $\textsc{MakeGap} \circ \textsc{Reverse} \circ \textsc{MakeTrace}$

$\overset{r}{=}$ and $\overset{r}{<}$: component-wise and lexicographic comparison

Canonicalization: find the $\overset{r}{<}$-smallest rotation of g and its reverse.

[Hellmuth et al., ALENEX, 2018]

Example: Graphs, Continued

Model: A graph $G = (V, E)$.

Representation: An adjacency list which implicitly assigns $1, 2, \dots, n$ to V.

Operations: $\textsc{Permute}(\gamma)$ for any permutation of $1, 2, \dots, n$.

$\stackrel{r}{=}$ and $\stackrel{r}{<}$: component-wise and lexicographic comparison

Computational Complexity: $\exp\left(O\left(\sqrt{n \log n}\right)\right)$

Brute-Force Algorithm:

1. Construct G^γ for all permutations $\gamma \in S_n$.
2. Select the "best" one (for example the $\stackrel{r}{<}$-smallest).

[Babai and Luks, STOC, 1983]

[Babai, Handbook of Combinatorics, 1996]

Existing Tools for Canonicalization in Practice

Published Tools: nauty, Traces, Bliss (and Saucy and Conauto)

- All based on the idea of individualization-refinement.
- Different sets of heuristics and variations.
- Many more algorithm variations are possible.
- Which is the best? for a specific class of graphs?
- What if the graph has vertex and edge labels?
- What if those labels are “complicated”? (e.g., stereo-info)

[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]

Existing Tools for Canonicalization in Practice

Published Tools: nauty, Traces, Bliss (and Saucy and Conauto)

- All based on the idea of individualization-refinement.
- Different sets of heuristics and variations.
- Many more algorithm variations are possible.
- Which is the best? for a specific class of graphs?
- What if the graph has vertex and edge labels?
- What if those labels are "complicated"? (e.g., stereo-info)

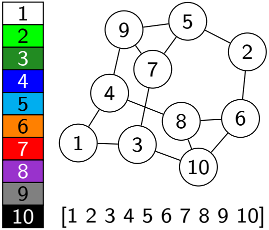
GraphCanon: [Andersen and Merkle, ALENEX, 2018]

- A generic C++ library for canonization algorithms.
- Algorithm variations implementable as individual plugins.
- Allows direct comparison of algorithm 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]

Individualization-Refinement Paradigm

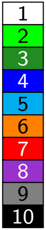
Initially: all vertices are unordered (same colour).

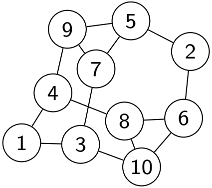


Individualization-Refinement Paradigm

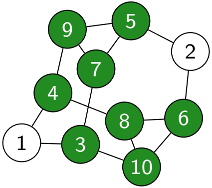
Refine the ordering by propagation of “cheap” local information.

Example: sort and partition by degree (1D Weisfeiler-Leman).





[1 2 3 4 5 6 7 8 9 10]

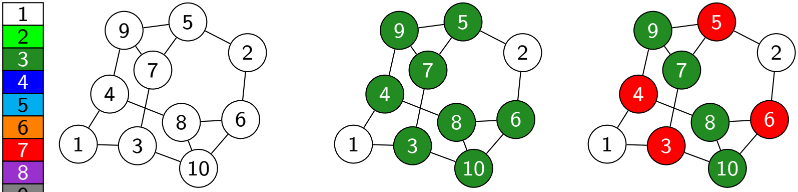


[1 2 | 3 4 5 6 7 8 9 10]

Individualization-Refinement Paradigm

Refine the ordering by propagation of “cheap” local information.

Example: sort and partition by degree (1D Weisfeiler-Leman).



[1 2 3 4 5 6 7 8 9 10]

[1 2 | 3 4 5 6 7 8 9 10]

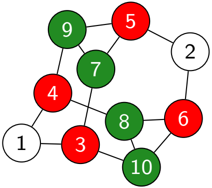
[1 2 | 7 8 9 10 | 3 4 5 6]

Individualization-Refinement Paradigm

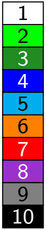
Let this be the root of a search tree, and select a colour.

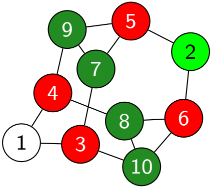
For each vertex of that colour;

create a child with this vertex given a unique new colour.

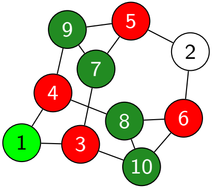


[1 2 | 7 8 9 10 | 3 4 5 6]



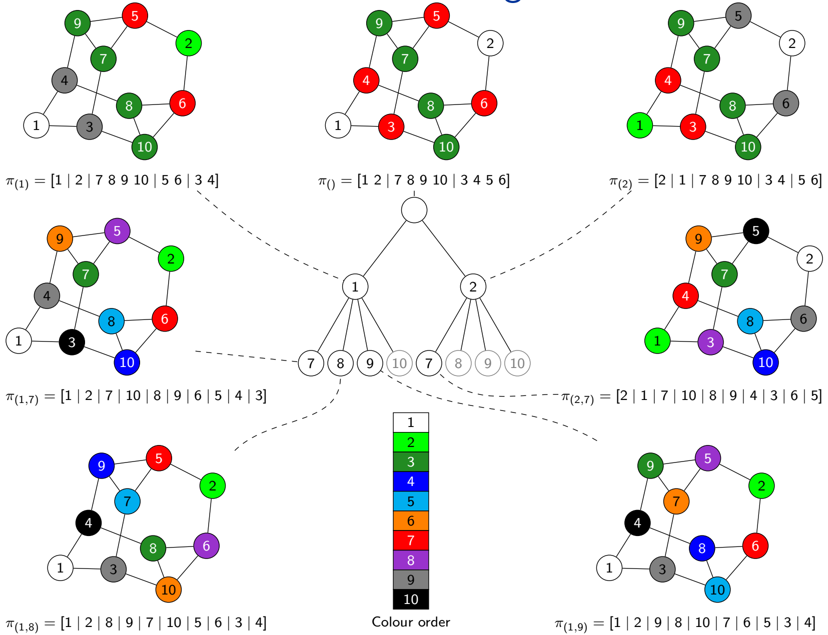


[1 | 2 | 7 8 9 10 | 3 4 5 6]



[2 | 1 | 7 8 9 10 | 3 4 5 6]

Individualization-Refinement Paradigm



Algorithm Variation

Categories

- Tree traversal
- Target cell selection
- Refinement
- Pruning with automorphisms
- Detection of implicit automorphisms
- Node invariants

GraphCanon: A common extension infrastructure.

Each variation implemented as a visitor:

- A set of callback methods for events of interest.
- Additional data structures instantiated
  - per search tree
  - per tree node

Benchmarks

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

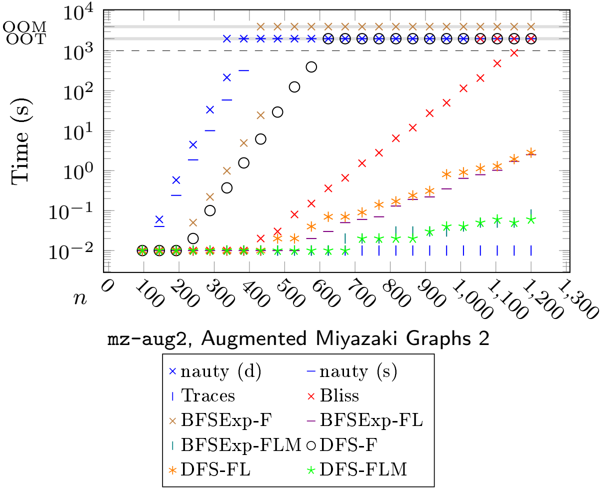
Algorithm configurations: $\{\mathrm{BFSExp}, \mathrm{DFS}\} \times \{\mathrm{F}, \mathrm{FL}, \mathrm{FLM}\}$

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/]

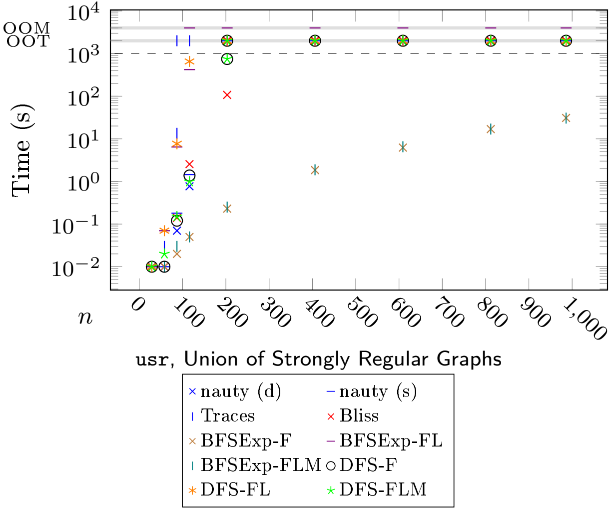
Tree Traversal and Target Cell Selector



`mz-aug2`, Augmented Miyazaki Graphs 2

Similar characteristics observed for other Miyazaki graphs.

Tree Traversal and Target Cell Selector



usr, Union of Strongly Regular Graphs

CFI-Rigid

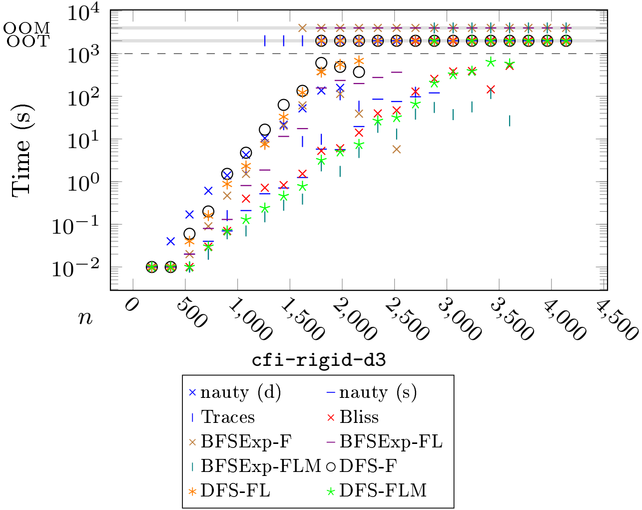
- 6 collections
- Designed to be the hard benchmarks.
- Expected to have very little symmetry.

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



10^4
OOM
OOT
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Time (s)
n
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
cfi-rigid-d3
nauty (d)
nauty (s)
Traces
Bliss
BFSExp-F
BFSExp-FL
BFSExp-FLM
DFS-F
DFS-FL
DFS-FLM

Summary

- Canonicalization is a general principle.
- The concepts can be applied to any data structure.
- Brute-force: make it a graph.

GraphCanon:

- Generic algorithm framework.
- (Relatively) easy to develop new variations.
- Allows direct comparison of algorithmic ideas.
- Competitive with established tools.
- `https://github.com/jakobandersen/graph_canon`
- Very easy to extract data for visualization:
  `https://jakobandersen.github.io/graph_canon_vis/`

MØD v0.7 (to be released soon™):

- Integrates GraphCanon .
- Finally, true canonical SMILES strings!
- The automorphism group of molecules is now available.
  (important for atom tracing)

Algorithm Variation

Tree Traversal:

- nauty, Bliss: depth-first (DFS)
- Traces: breadth-first with experimental paths (BFSExp)
- GraphCanon:
  - 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)

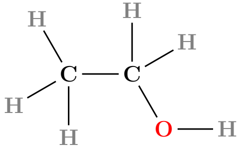
Algorithm Variation

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.