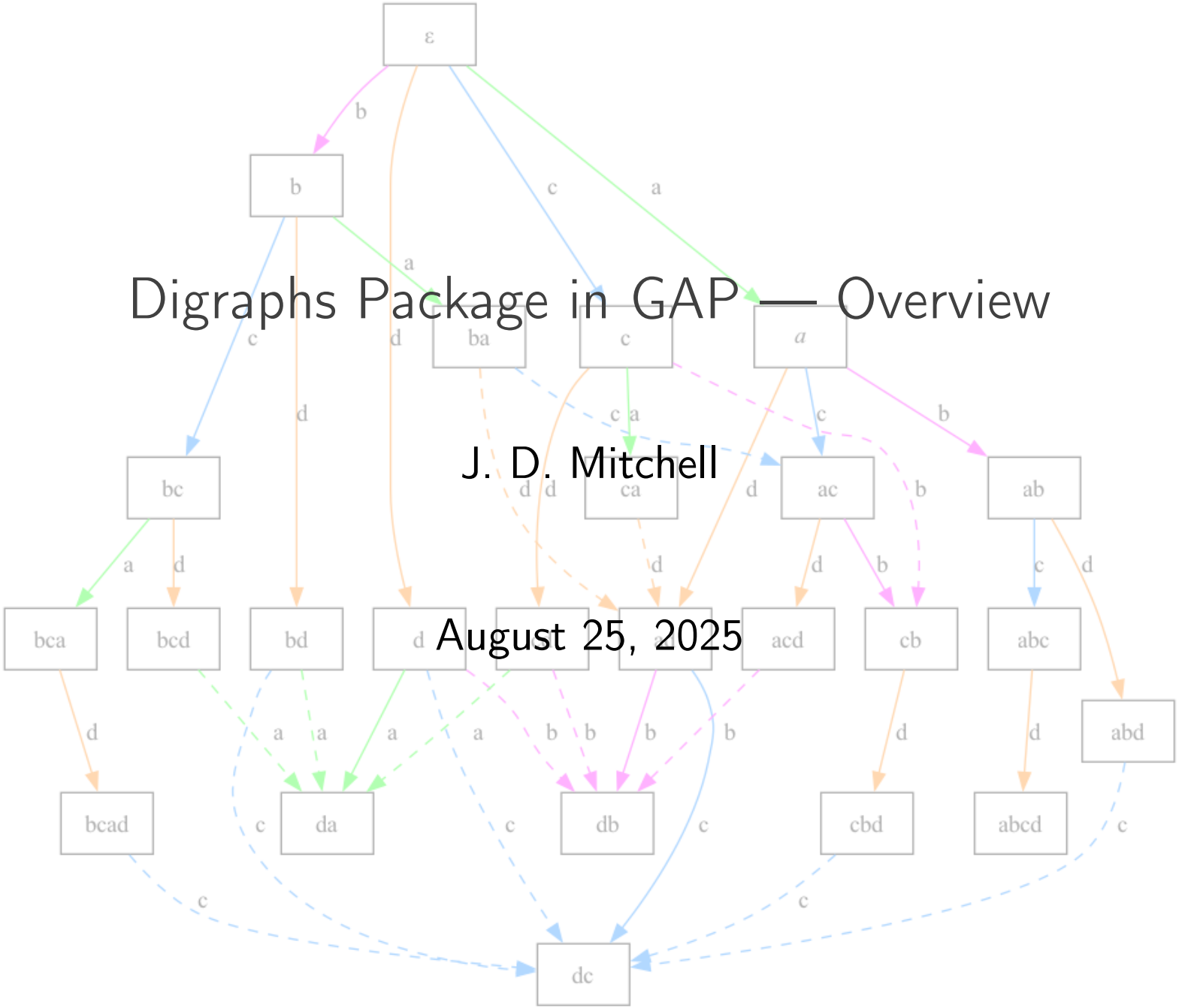


Digraphs Package in GAP — Overview

J. D. Mitchell

August 25, 2025

Contents

What is Digraphs? 1/3

Overview

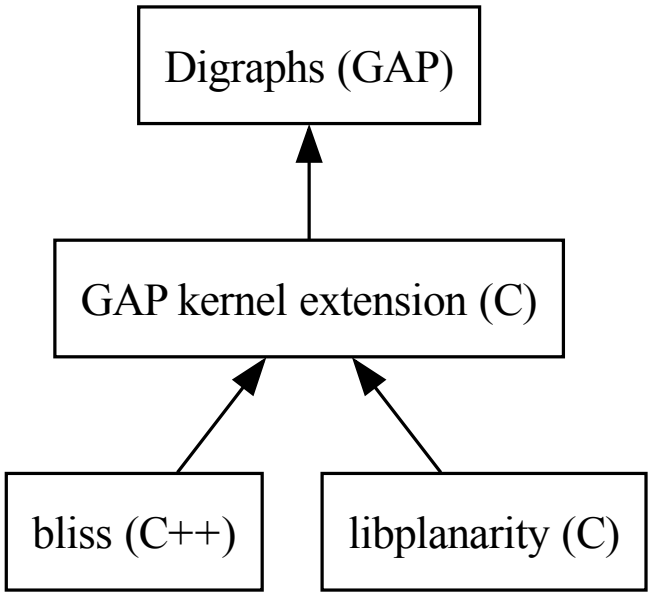
Digraphs is an add-on package for GAP that extends GAP's core capabilities so you can create, manipulate, and study directed graphs (digraphs); graphs; and multidigraphs.

In short:

- ★ GAP is mainly a computational algebra system, but it does not have any built-in graph theory tools.
- ★ Digraphs fills that gap (!) by adding a comprehensive set of data structures and algorithms for digraphs.
- ★ It is designed to work well with GAP's group-theoretic functionality, so you can combine graph theory and algebraic computations.

What is Digraphs? 2/3

Package structure




```
D := Digraph([[], [1], [2], [2]]);
SetDigraphVertexLabels(D,
 ["Digraphs (GAP)",
  "GAP kernel extension (C)",
  "bliss (C++)",
  "libplanarity (C)"]);
gv := GraphvizVertexLabelledDigraph(D);
GraphvizSetAttr(gv, "node [shape=box]");
GraphvizSetAttr(gv, "rankdir", "BT");
```

- ★ Digraphs: GAP code of the Digraphs package
- ★ Kernel extension: C code for
  - ☆ performance critical functions
  - ☆ interface with 3rd party libraries

What is Digraphs? 3/3

Third party libraries


Tommi Junttila and Petteri Kaski.
Engineering an efficient canonical labeling tool for large and sparse graphs.
DOI: 10.1137/1.9781611972870.13.


Tommi Junttila and Petteri Kaski.
Conflict propagation and component recursion for canonical labeling.
DOI: 10.1007/978-3-642-19754-3_16.

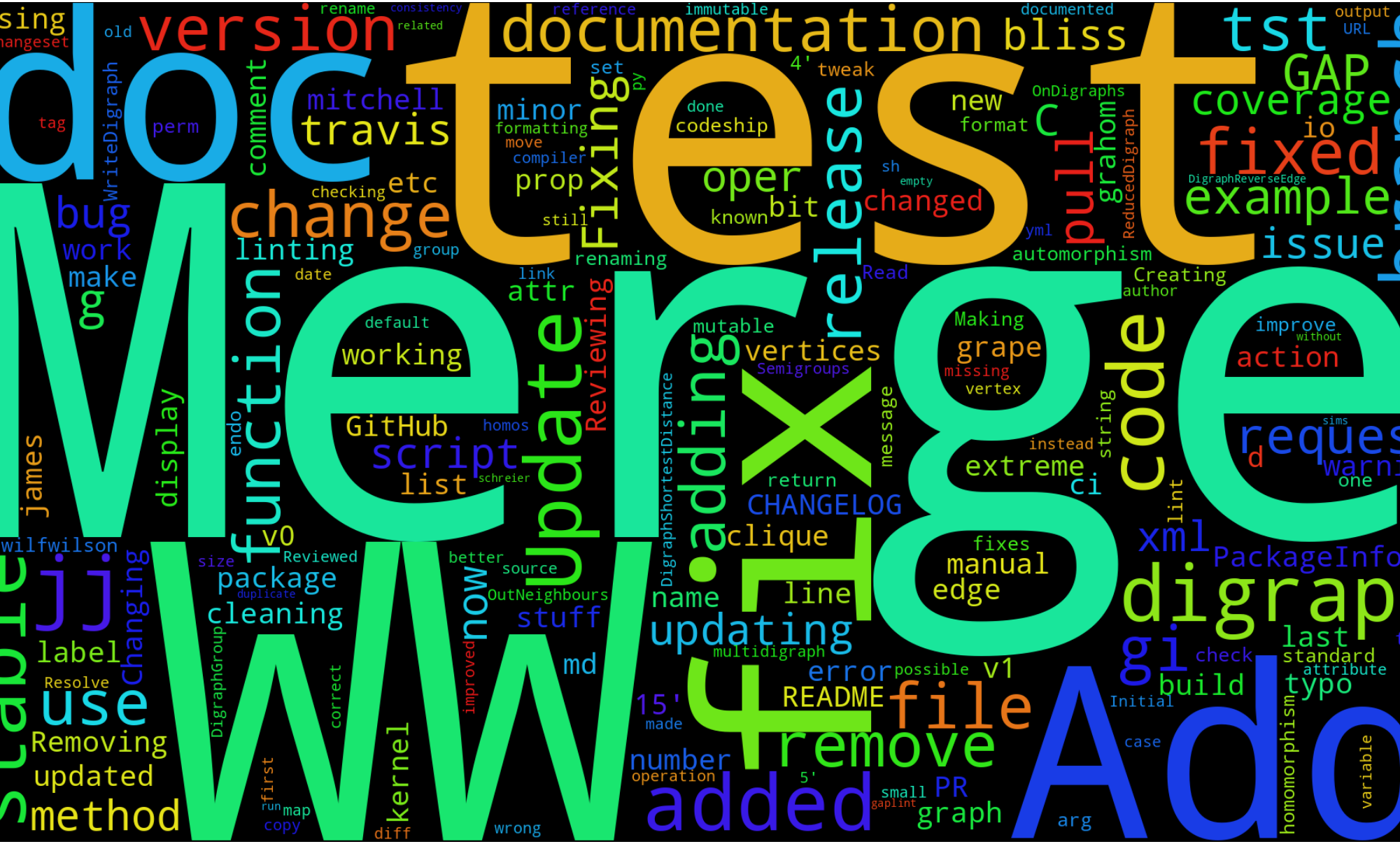

John Boyer.
Edge-Addition Planarity Suite.
Available at `https://github.com/graph-algorithms/edge-addition-planarity-suite`.

What is Digraphs? 4/3

History

- ★ First commit: 24th September 2014, 14:34:12
- ★ "Quick exercise to practice writing C code in GAP" JDM (2014)
- ★ Today: ∼106,447 lines of code, ∼2,824 total commits
- ★ Original authors:
  - ☆ Jan De Beule
  - ☆ Julius Jonusas
  - ☆ James Mitchell
  - ☆ Michael Torpey
  - ☆ Wilfred Wilson
- ★ Today: 38 additional contributors (many from St Andrews, some from Aachen, York, Kaiserslautern-Landau, and elsewhere).

Commit messages



More commit messages

Bad spelling award: Wilf Wilson

- ★ “Making **borbits** as slow as possible. WW”
- ★ “Remove last **vestigates** of US "Color" spelling variant”
- ★ “examples.gi: fix the method **strongs** (oops)”
- ★ “Fixing tests since I’ve solved the **troulbe**. WW”
- ★ “Fixing Dot functions to display **siolated** vertices. WW”

Unhelpful commit message award: James Mitchell

- ★ “added forgotted oper.xml”
- ★ “oops”
- ★ “dunno”
- ★ “merg”’
- ★ “stuf”’

Why Digraphs?

- ★ "Quick exercise to practise writing C code in GAP" JDM (2014) for Michael Young and Wilf Wilson who had just started their PhDs
- ★ Required a more structured approach to digraphs in Semigroups GAP package (Cayley graphs, partial orders, counting ideals, and so on)
- ★ Being an impatient type, JDM found the learning curve required to use Grape too steep (sorry Leonard!)

Design principles

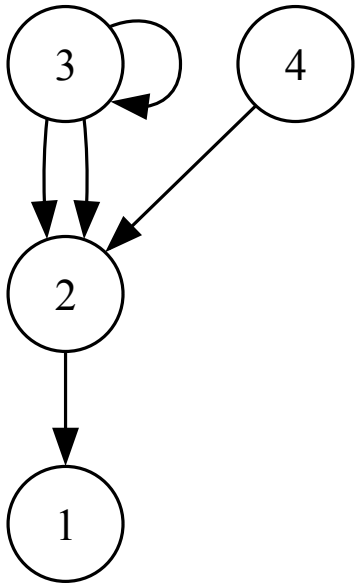
We wanted Digraphs to be:

- ★ easy to use ?
- ★ hard to misuse ?
- ★ fast ✓
- ★ feature rich ✓
- ★ high quality error messages ?
- ★ well-documented ✓
- ★ highly collaborative ✓
- ★ interoperable with Grape ✓

Definition of a digraph

Digraphs in DIGRAPHS can have loops, and multiple-edges.

```
D := Digraph([[], [1], [2,2,3], [2]]);
```



How digraphs are represented

The only representation of digraphs in DIGRAPHS is by out-neighbours!

The vertices are always `[1 .. n]` for some $n \geq 0$ and underneath the digraph is stored as a list of lists of vertices.

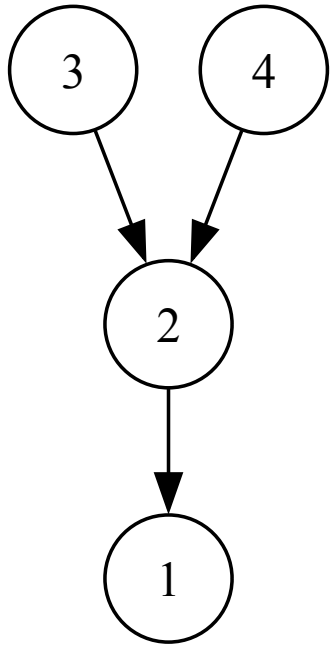
```
D := Digraph([[], [1], [2,2,3], [2]]);
```

Every digraph belongs to the representation `IsDigraphByOutNeighboursRep` but other representations are possible.

Constructing digraphs

Range and source

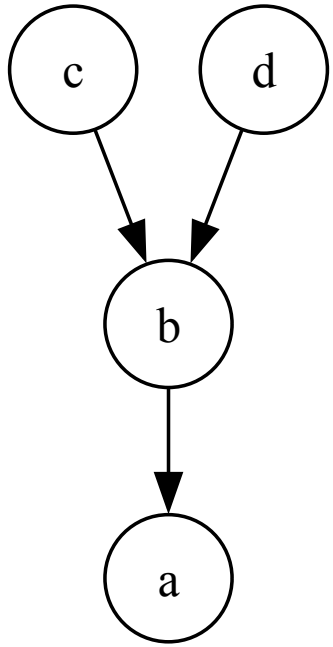
```
gap> D := Digraph([1, 2, 3, 4], [2, 3, 4], [1, 2, 2]);
<immutable digraph with 4 vertices, 3 edges>
gap> OutNeighbours(D);
[ [  ], [ 1 ], [ 2 ], [ 2 ] ]
```



Constructing digraphs

Range and source

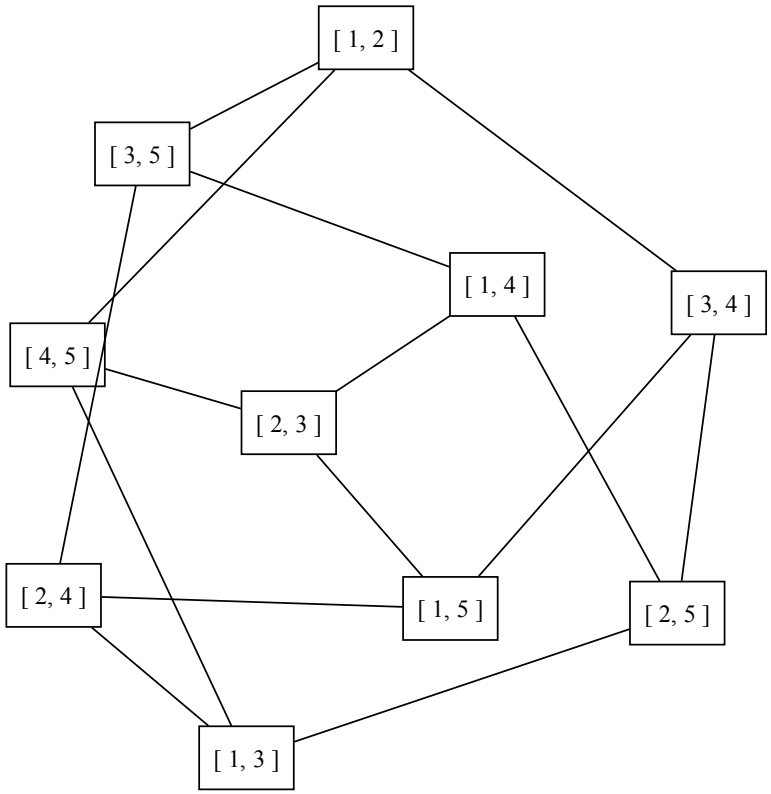
```
gap> D := Digraph(["a", "b", "c", "d"],
>                 ["b", "c", "d"],
>                 ["a", "b", "b"]);
<immutable digraph with 4 vertices, 3 edges>
gap> DigraphVertexLabels(D);
[ "a", "b", "c", "d" ]
```



Constructing digraphs

Adjacency function

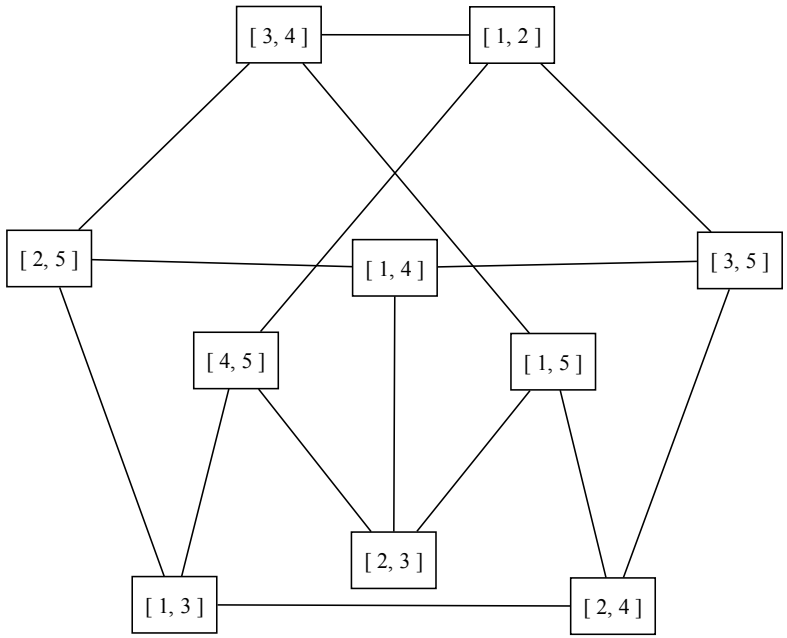
```
gap> D := Digraph(Combinations([1 .. 5], 2),
> {x, y} -> Intersection(x, y) = []);
<immutable digraph with 10 vertices, 30 edges>
```



Constructing digraphs

Grape

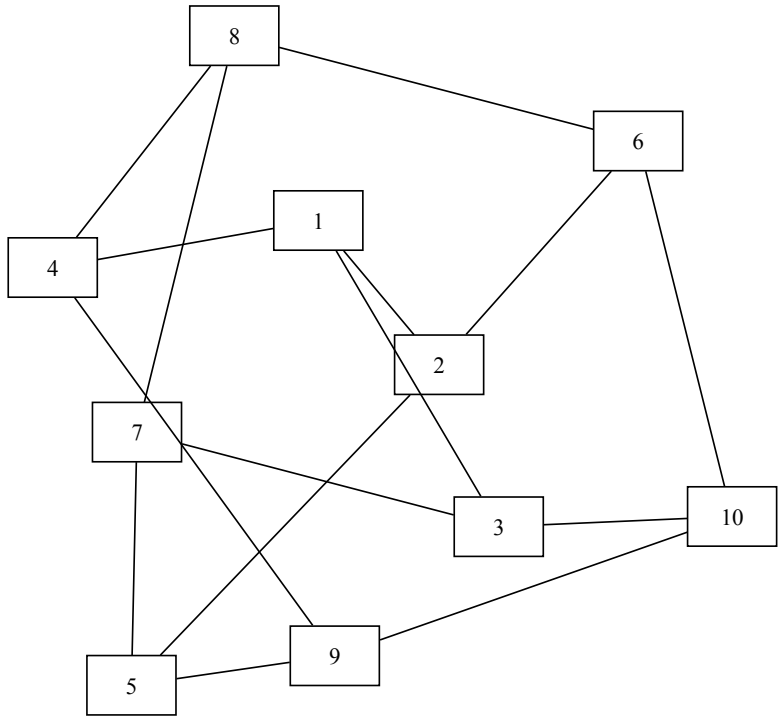
```
gap> G := Graph(SymmetricGroup(5), [[1, 2]], OnSets,
> {x, y} -> Intersection(x, y) = []);
..
gap> D := Digraph(G);
<immutable digraph with 10 vertices, 30 edges>
```



Constructing digraphs

By name

```
gap> ListNamedDigraphs("petersen");
[ "petersen", .. ]
gap> D := Digraph("petersen"); # or PetersenGraph();
<immutable digraph with 10 vertices, 30 edges>
```



Constructing digraphs

Other ways

- ★ `DigraphByAdjacencyMatrix([[0, 0, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0]])`
- ★ `DigraphByEdges([[2, 1], [3, 2], [4, 2]])`
- ★ `EdgeOrbitsDigraph`
- ★ `DigraphByInNeighbours`
- ★ `CayleyDigraph`
- ★ `ReadDigraph` (more later)
- ★ `RandomDigraph`

Mutable versus immutable

It seemed to happen often that we create a digraph and immediately want to change it:

```
gap> D := NullDigraph(10 ^ 8);
<immutable empty digraph with 100000000 vertices>
gap> time;
12813
gap> DigraphAddEdge(D, 1, 2);;
gap> time;
9041
gap> D := NullDigraph(IsMutable, 10 ^ 8);
<mutable empty digraph with 100000000 vertices>
gap> time;
11020
gap> DigraphAddEdge(D, 1, 2);
<mutable digraph with 100000000 vertices, 1 edge>
gap> time;
241
```

Mutable versus immutable

Almost every function in Digraphs that returns a digraph has the optional 1st argument IsMutable or IsImmutableDigraph, which specifies the mutability of the output.

Most functions that modify a mutable digraph will do so in-place (without copying it). Some care is required!

The manual entry for the function should clearly specify what happens in regard of mutability.

Why not always mutable?

Workflow:

- ★ create a mutable digraph
- ★ modify it until it has the desired properties
- ★ call MakeImmutable

New graphs from old

There are at least 48 different functions in Digraphs for forming new graphs from old ones:

- ★ copying, adding/removing edges, or vertices
- ★ subdigraphs, quotients, homomorphisms
- ★ spanning trees; and forests
- ★ reversing the direction of edges, duals
- ★ closures (symmetric, transitive, reflexive)
- ★ reductions (remove isolated vertices, and so on)
- ★ products (disjoint union, edge union, join, cartesian, direct, and so on)
- ★ line, double, bipartite double, distance digraph, the Mycielskian, and so on
- ★ matchings, maximal matchings,

Properties + Attributes

There are more than 100 properties and attributes implemented for digraphs and graphs in Digraphs:

- ★ vertices and edges, edge weights
- ★ neighbours and degree
- ★ reachability, connectivity, cycles, circuits
- ★ planarity
- ★ hashing
- ★ homomorphisms, chromatic number
- ★ clique and independent sets.

Serialisation

There are a fairly large number of different functions for serialisation into different formats including:

- ★ `graph6`, `sparse6`, `digraph6`, `disparse6` from `nauty`
- ★ plain text (you define the deserialisation function, lots of defaults implemented)
- ★ pickled files (GAP specific)
- ★ DIMACs
- ★ dreadnaut

The function `WriteDigraphs` attempts to write a list of digraphs to a file using its best guess at what format to use.

If you `Print` a digraph, then you'll get the shortest string it can:

```
gap> D := Digraph("moserspindle");;
gap> Print(D);
DigraphFromGraph6String("F`o~_")
```

Visualisation

In the current released version of DIGRAPHS, there are a number of functions for creating graphviz representations of digraphs:

- ★ `DotDigraph`
- ★ `DotVertexLabelledDigraph`
- ★ ...

This is not very flexible, so we made a new package called GRAPHVIZ, mostly written by Matthew Pancer, which isn't yet complete:

```
D := Digraph([[], [1], [2], [2]]);
SetDigraphVertexLabels(D,
 ["Digraphs (GAP)",
  "GAP kernel extension (C)",
  "bliss (C++)",
  "libplanarity (C)"]);
gv := GraphvizVertexLabelledDigraph(D);
GraphvizSetAttr(gv, "node [shape=box]");
GraphvizSetAttr(gv, "rankdir", "BT");
```

Live coding — part 1

What would you like to see how to do?

One task for tomorrow's first session is to think of something that you'd like to know how to do, and to tell me then.

Live coding — part 2

What would you like to see how to do?

The boolean semiring

Let $\mathbb{B}$ denote the boolean semiring:

$\oplus$	0	1
0	0	1
1	1	1

$\otimes$	0	1
0	0	0
1	0	1

and let $M_n(\mathbb{B})$ denote the monoid of all $n \times n$ matrices over $\mathbb{B}$.

Clearly $|M_n(\mathbb{B})| = 2^{n^2}$, grows rather quickly.

Minimum size generating sets?

If $X \subseteq M_n(\mathbb{B})$, then we write

$$\langle X \rangle = \{x_1 \cdots x_n \mid x_i \in X,\ n \in \mathbb{N}\} \leq M_n(\mathbb{B}).$$

Question: What is $d(M_n(\mathbb{B}))$ $\min |X|$ such that $\langle X \rangle = M_n(\mathbb{B})$?

n	d($M_n(\mathbb{B})$)
1	*2
2	*3
3	*5
4	*7
5	*13
6	68
7	2 142
8	459 153
9	?

Overview

Theorem (Devadze '68, Konieczny '11)

Every minimum size generating set for $M_n(\mathbb{B})$ contains 4 specific matrices, and one representative of every prime $\mathscr{J}$-class.

If P denotes any set of representatives of prime matrices, then we:

- ★ compute a set Q_n of matrices containing such a set P;
- ★ if $A, B \in Q_n$ and the row space $\mathrm{row}(A)$ of A embeds into $\mathrm{row}(B)$, then $A \notin P$.