Math.Comput.Sci. (2020) 14:559–575
https://doi.org/10.1007/s11786-020-00453-5

DiscreteZOO: A Fingerprint Database of Discrete Objects

Katja Berčič · Janoš Vidali

Received: 1 November 2018 / Revised: 29 November 2019 / Accepted: 15 December 2019 / Published online: 14 February 2020

Abstract In this paper, we present DiscreteZOO, a project which illustrates some of the possibilities for computer-supported management of collections of finite combinatorial (discrete) objects, in particular graphs with a high degree of symmetry. DiscreteZOO encompasses a data repository, a website and a SageMath Package.

Keywords Fingerprint database · Vertex-transitive graphs · Maniplexes · SageMath package · Website

Mathematics Subject Classification Primary 68R05; Secondary 68P20

1 Introduction

The fairly new discipline of Mathematical Knowledge Management [11] "studies the possibility of computer-supporting and even automating the representation, cataloguing, retrieval, refactoring, plausibilisation, change propagation and even application of mathematical knowledge" [16]. The discipline concerns itself with all forms of mathematical knowledge, including definitions, theorems, proofs and data. In mathematics, we commonly use the complementary strengths of humans and computers in creation of new knowledge, while computer support for managing existing mathematical knowledge is more rare.

This is also the case when it comes to the management of a specific kind of mathematical data: collections of **concrete** (as opposed to abstract) mathematical objects (Fig. 1). Concrete mathematical objects are those that can be represented as records or arrays, while abstract objects would be those represented by formulas or formal definitions. The DiscreteZOO project presented in this paper explores the possibilities for computer-supported management of collections of concrete mathematical objects in general and collections of examples of finite combinatorial (discrete)

K. Berčič (✉)
FAU Erlangen-Nürnberg, Erlangen, Germany
e-mail: katja.bercic@fau.de

J. Vidali
Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
e-mail: janos.vidali@fmf.uni-lj.si

J. Vidali
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

$G = (V, E)$, where $E = \{\{i, j\} : i, j \in V\}$

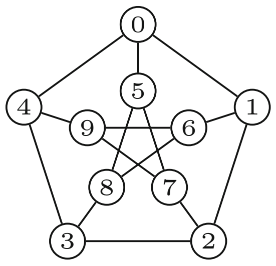


Fig. 1 An abstract object (top) and a concrete one (bottom)

objects in particular. We use collections of graphs with a high degree of symmetry as a case study towards databases of a broader range of mathematical objects.

Mathematicians have a history of constructing catalogues, tables and similarly organised collections of mathematical objects. In a thread started by Gordon Royle [19], the MathOverflow community collected several examples that predate computers. Such collections can be used to better understand the objects in focus (such as graphs with a high degree of symmetry) by studying small examples. In particular, they can be computed with, used to search for patterns (or counter examples), to uncover connections, and to experiment with. In combinatorics, enumeration of objects is sometimes done by actually constructing them and in some cases, it makes sense to store the objects as a catalogue. These collections are also where mathematics can fruitfully intersect with other disciplines.

Broadly speaking, a fingerprint is something that identifies something else (such as a person or data), often uniquely. They are widely used in science and technology, from forensics to genetics, biochemistry and computing [37]. In the context of mathematical knowledge, it is possible to organise fingerprints into a searchable database, i.e., a **fingerprint database**, to support querying for mathematical results [5]. Such databases make the data more accessible through querying and searching, and thus enable mathematicians to be more effective at their work. Last but not the least, it is easier for a collection of mathematical objects to get exposure when it is organised into a searchable database.

1.1 Overview

In Sect. 2, we sketch the state of art where it comes to collections of concrete mathematical objects. In Sect. 3, we present some use cases and combine them with the state of the art to identify certain requirements we consider necessary to bridge the experience gap between the tools that are currently offered and what the users (particularly mathematicians) need. In Sect. 4, we describe the DiscreteZOO project, paying particular attention to the structure of information and design decisions. We conclude in Sect. 5 with an outline of future work.

2 State of the Art

We use the expression **collection of mathematical objects** (CMO) to describe aggregations of concrete mathematical objects. In other contexts, these have been called atlases, catalogues, censuses, databases, directories, encyclopedias, and lists. Some of these are associated with some structure or features. Encyclopedias might be expected to have one interface page per object. A census is usually a collection that is in some way complete. For CMOs, we want to presuppose neither an interface nor any kind of underlying system. The only requirement we impose is a collection only contains concrete objects of one type. (such as collections of integer sequences or graphs). We do not require any degree of digitalisation for collections (and so, the *Atlas of Graphs* [29] contains a collection of graphs) nor do we want to bind collections of mathematical objects to any kind of infrastructure. We will use the word **database** whenever the collection has infrastructure that supports querying.

We know [3] of at least nine searchable databases of mathematical objects, which mostly address the needs of specific communities: *ATLAS of Finite Group Representations (Version 3)* [38], *The Database of Permutation Pattern Avoidance* [30], *NIST Digital Library of Mathematical Functions* [17], *Encyclopedia of Graphs* [26], *FindStat* [4], the *House of Graphs* [7], the *LMFDB* [32], the OEIS [25], the *QaoS* [31], the *Small Graph Database* [12] and the *SymbolicData Project* [34]. We found many more CMOs with fewer features, for example five collections of abstract polytopes, and, depending how one counts, several collections of lattices (at least three), graphs (at least ten) and other, mostly combinatorial, objects (at least ten). This is not at all surprising. On one hand, the accessibility and advancement of technology made it possible for more and more researchers to produce collections of mathematical objects. On the other hand, development of nontrivial infrastructure necessary for a searchable database goes beyond a typical mathematician's resources, skillset or interests.

There are some theoretical obstacles one encounters when making mathematical knowledge more accessible that are also true when working with CMOs.

Obstacle 1 Most mathematical results or objects do not have a natural order or structure that could be used as an intuitive basis for browsing.

The standard way to approach this is to support querying. Billey and Tenner proposed [5] **fingerprints**, or canonical representations of mathematical results, as a means to querying databases of theorems – an idea that also makes sense in the context of CMOs. Ideally, all fingerprints would be small, language-independent and would uniquely determine the fingerprintees. Finding suitable fingerprints is a domain-specific mathematical problem that does not always produce a result.

Obstacle 2 Many mathematical results or objects do not have a representation that is small, language-independent and canonical (uniquely determining).

Notable examples of objects that do not have a canonical form are graphs. They do, however, have a form that comes close. A **canonical form** or labelling [1] of a graph G is a labelled graph $\mathrm{Canon}(G) \simeq G$ such that that for every other graph H, $H \simeq G$ if and only if $\mathrm{Canon}(H) = \mathrm{Canon}(G)$, when both $\mathrm{Canon}(H)$ and $\mathrm{Canon}(G)$ can be computed. The labelling obtained depends on the algorithm that was used to compute it. Nauty [22] and Bliss [13] are two libraries commonly used to produce canonical forms of graphs.

The existence of a natural fingerprint likely played a part in the success of the *On-Line Encyclopedia of Integer Sequences* (OEIS) [25], perhaps the most famous example of a database of mathematical results. The OEIS is a searchable and collaborative database of integer sequences, which serve as fingerprints for their associated entries. The indexing is based on small, language-independent, and canonical data: the first elements of a sequence.

Even if most CMOs are only weakly supported by computers, they are already being used to manually find patterns (by writing programs) in the CMOs and look up properties of mathematical objects. A searchable database would make working with them more efficient and would open up additional options to researchers. Indeed, the initial motivation behind the DiscreteZOO project was to provide a home for the partially overlapping censuses of graphs with a high degree of symmetry [9,20,28], even if it quickly became clear that there was no reason to restrict it to being a database for symmetric graphs.

It is important to be able to refer to a mathematical object by a short unique identifier, for example in a paper, and some of the searchable databases we just mentioned already provide citable indexes. The sequences in the OEIS are citable via their unique identifier (such as `A000055`). Databases like *The Database of Permutation Pattern Avoidance* [30] and *FindStat* [4] use similar citable unique identifiers. A system of Math Object identifiers (MOI) has been proposed in [15], which tries to solve the CID problem for all kinds of mathematical knowledge.

While some of the more feature-rich databases of mathematical objects accept new contributions, there is currently no other simple way to organise a collection of mathematical objects into a searchable database. On the other hand, there are some helpful features that are not implemented even by the existing searchable databases. The LMFDB is a good example of a database with an interface that goes as far as it can, but does not have an API for programmatic access.

As mathematicians find themselves working with data more frequently and as the industry standards for interacting with data move forward, the current state of the art in becomes insufficient.

3 Requirements Analysis

There is a need in the mathematical community for a platform or framework that would provide infrastructural support for CMOs. We believe this is possible, because while every CMO necessarily has its own peculiarities, they have enough in common for a reusable infrastructure to make sense. This infrastructure could then provide common features to all databases while being flexible enough to accommodate any database-specific features. Such a platform would not need to implement many or complex features, as most CMOs would benefit even from just a very basic search functionality.

In the rest of this section, we will first describe some use cases we use as guidelines in the design of the DiscreteZOO system. In the second part of the section, we will describe some requirements for an ideal database of mathematical objects. Most of the features derived from these requirements could be provided by a general purpose platform or framework for CMOs.

Use case 1 (Classification of cubic vertex-transitive partial cubes) Tilen is interested in cubic partial cubes that are also vertex-transitive [1]. The census of cubic vertex-transitive graphs on up to 1280 vertices [28] contains (among others) all the graphs he is interested in on up to 1280 vertices. First, he needs to filter out those which are partial cubes. He would like to know whether there are any vertex-transitive cubic partial cubes in addition to the infinite family of even prisms ($K_2 \square C_{2n}$). Finally, he would like to compute isometric embeddings into hypercubes.

The census is published as Magma code [6], and the only way to do what he wants (apart from DiscreteZOO) is to program everything in a computer algebra system. If he does not have access to Magma, he needs to process the code so that it is readable by another computer algebra system. We will show how the task looks like with DiscreteZOO in Sects. 4.5 and 4.6.

Use case 2 (Classification of cubic vertex-transitive graphs with small girth) Primož wants to work on a classification of cubic vertex-transitive graphs with small girth [27]. To get an idea on how to proceed, he first wants to know how many graphs from the census [28] there are for each girth. Then he wants to check some additional properties for each graph.

Use case 3 (Abstract polytope lookup) Alice has recently come across an interesting abstract polytope. She knows it is self-dual and self-Petrial with the Schläfli symbol {12, 12}, but does not remember its name. She would like to look it up quickly in a database by filtering using the properties she remembers.

Use case 4 (Automorphism groups downloads) Bob wants to test a subgroup condition for automorphism groups of certain maps. He would like to use a database of maps (or equivalently, maniplexes of rank 3) to obtain a list of automorphism groups which he could then test with a GAP function he had already written.

Use cases 1 and 2 are already fully supported in DiscreteZOO. Use case 3 would require supporting properties which are numeric arrays (for the Schläfli symbol property). Use case 4 often appears in the area of symmetries of discrete objects, where researchers are often interested in properties of automorphism groups of objects which exhibit some specific property. It is beyond the scope of DiscreteZOO to provide such an in-depth analysis of automorphism groups, especially since computer algebra systems serve that purpose well. However, we believe that generating CAS code for lists of automorphism groups would be a valuable time-saving feature.

3.1 Requirements

Billey and Tenner [5] identified several properties important for fingerprint databases of theorems. Namely, such a resource should be a "searchable, collaborative database of citable mathematical results, indexed by small, language-independent, and canonical data". Given the rate of growth of mathematical knowledge, we believe these features to

[1] Tilen only used the original census for his research [18].

be (or become) critical for any mathematical knowledge resource, like databases of definitions, objects, theorems and proofs. We added some additional requirements we believe to be important in the following list. Note that another important consideration is sustainability. As it touches on broader problems, including social and organisational ones, it is beyond the scope of this paper.

R1: Searchable The database should have at least a basic search (filter) functionality for objects

R2: Collaborative The database should be collaborative. It needs to be easy for people to contribute and it must be possible for any data to find out who contributed it and when. In particular,

1. adding a new collection should ideally be a declarative task as opposed to a programming one – it should suffice to describe the structure of the collection,
2. changes such as adding new kinds of objects, new collections, new properties, or updates to existing values need to be tracked by the database.

R3: Citable The records in the database need to be citable.

R4: User-friendly The database should provide suitable interfaces. This means

1. simple, intuitive and easily accessible interfaces for casual users (it should avoid exposing the casual user to low level interfaces, such as Python, SageMath, SQL, …),
2. making usage easy or easier for power users.

R5: Self-explaining The database interfaces should make accessing definitions of concepts and further information easily available.

R6: Interoperable Other systems (like databases and computer algebra systems) should be able to interact with the database. This involves having well defined APIs.

R7: Coverage information The database interfaces should make explicit the assumptions made by the system and collections as well as providing information on the completeness of the search results. For example for graphs: “the search results are complete for your search parameters up to order 511”.

R8: Non-redundant The objects should be stored up to isomorphism.

R9: Decentralised The database should not rely on the existence of a central authority. An entry in a local copy of the database should be easily distributable to other copies without any third-party intervention.

Another reason for a decentralised database is the following scenario. A researcher may encounter an object that is not yet in the database; another researcher may then reproduce her work (or use his own methods) and easily verify that they have obtained the same object by comparing the identifier. Once the object is in the database, the identifier may then be used to quickly access the object and its properties.

4 DiscreteZOO, a First Step Towards a Generic CMO Framework

DiscreteZOO is an attempt to provide a generic platform with few features that are as general as possible, taking as a use case collections of symmetric graphs and related objects. We chose to focus on features that are simplest to implement and that have the greatest usability. We tried to keep the infrastructure as “object type agnostic”. That is, we avoided features specific to the object type (except for actual database encodings). An object types comes with a set of properties (mathematical invariants) that apply to objects of this type which are then used for searching and filtering. Here, we focused on property types that are easiest (and, in fact, easy) to implement: Boolean and numeric properties.

There are two main user interfaces: the website and the SageMath package. Ideas involved in the design process [14,24] were helpful in the preparation of the interfaces, especially so for the website. The interfaces implement a simple query language which we believe to be a flexible and general solution for accessing results in CMOs.

New data come into DiscreteZOO through the data repository, while the class specification repository describes the objects and their types. The need for these two repositories arose from the fact that this information is shared between different applications, with different needs, and written in multiple programming languages. Therefore, the data has to be represented in a programming language-independent manner. We chose the JSON format for its widespread support, and for being relatively human readable. Additionally, the usual line-based formatting means that such a format is well-suited for tracking changes with Git, which also addresses R2.

The layout of the data repository also closely follows the one used by Git, which stores its objects in files whose paths determine their hashes. Unlike Git, which essentially uses a content-addressable filesystem [8], the DiscreteZOO data repository is not entirely content-addressable. Instead, it could be described as *object-addressable* – for each object, the path to the file describing it (or a symbolic link to it) can be computed from the object itself. However, this file may still change (and keep the same path) if, say, new properties for this object are added.

4.1 Information Structure

DiscreteZOO aims to provide a platform for CMOs with a structure that is particularly simple: those which can be represented as tables, where each row represents information about one object and where each column represents a mathematical property (invariant) of objects. The actual database schema might be more complex for implementation purposes. Moreover, we require for now that these properties have either Boolean values (a graph is either Hamiltonian or not) or numeric (integer or real) values, small enough to fit standard database types.

In general, it would be possible to store any objects that have a reasonably small encoding as text or a binary blob. Each of the object types comes with a set of properties that are relevant for it, e.g., the properties that make sense for symmetric graphs are not the same as the ones that make sense for maniplexes.

A typical DiscreteZOO database record describes a single discrete object, several of its mathematical properties, indexes in collections, and any number of human-readable aliases and descriptions. Most of the properties are mathematical invariants. Every object also has a **GUID** (globally unique identifier) and a **CID** (citable identifier). For more detail on partial fingerprints, GUIDs and CIDs, see 4.1.1. All objects included so far also have a shareable encoding consisting of printable characters. Such an encoding makes it possible to transfer an object between software tools.

For example, the record representing the Petersen graph (Table 1) is marked as a graph, vertex-transitive graph and a cubic vertex-transitive graph and has all the precomputed properties relevant to these classes.

4.1.1 Fingerprints

Billey and Tenner [5] observed the natural conflict between keeping fingerprints small on one hand and their canonicity on the other. If we want to be able to use fingerprints for querying, they should also be readable or writable by humans (for example, for entry in a search input field on a website).

Mathematical invariants, such as valency or girth for graphs, behave as fingerprints in some ways and we believe it makes sense to use them as such. They are examples of data that are small, language independent, but do not necessarily uniquely identify an object: for example, there are infinitely many arc-transitive graphs of valency 3. We propose the term **partial fingerprints** to describe data that are small and language independent, but do not uniquely identify objects. In DiscreteZOO, we use the Boolean and numeric properties as partial fingerprints.

Ideally, the objects in DiscreteZOO have a GUID based on some canonical string representation. If such a representation exists, we use the 64-character hexadecimal representation of its SHA-256 hash for the GUIDs. Such GUIDs are fingerprints in the sense that they display a level of canonicity (two GUIDs obtained by the preferred algorithm are the same if and only if the objects are the same and it is possible to decode the complete object from them) and are reasonably small and language-independent, but are not human friendly. We plan to use randomly assigned GUIDs if we ever encounter an object type that has nothing approaching a canonical form. The use of a cryptographic hash function is a strong guarantee [23, §9] that no two objects are ever expected to be found to have the same identifier. The reason for choosing such a hash-based identifier instead of a sequential one (as with other databases) is decentralisation (R9).

Table 1 The Petersen graph

GUID	`c74c6028a25a65a6189db885bdaa11aa1c1f2a861f4c0d0e8efd6a4ec786e0a5`
CID	`Zc74c-6028-a25a+8`
data	`:IeIKqPD?hgAH?G̃`
aliases	Petersen graph

Class *graph*

is arc-transitive	true
is bipartite	false
is Cayley	false
is distance-regular	true
is distance-transitive	true
is edge-transitive	true
is Eulerian	false
is Hamiltonian	false
is overfull	false
is partial cube	false
is split	false
is strongly regular	true
clique number	2
connected components number	1
diameter	2
girth	5
odd girth	5
order	10
size	15
triangles count	0

Class *vertex-transitive graph*

Index in [20]	(10, 5)

Class *cubic vertex-transitive graph*

Index in [9]	(10, 1)
Index in [28]	(10, 3)
is Möbius ladder	false
is prism	false
is SPX	false

For graphs, the canonical labellings are as close as it is currently possible to get to a canonical form. We base the GUIDs for graph on the `graph6` and `sparse6` graph formats [21]. It is possible to use the graph canonical labellings to obtain canonical representations of other graph-like objects: for maniplexes, we can just add the edge labelling.

4.1.2 Citability

Ideally, a database of mathematical objects provides both an easy way to cite its contents as well as an efficient way to compare its contents with another collection of mathematical objects. These two objectives are, in some way, at cross purposes. The former requires the citable identifier to be as short as possible, while the other most likely means we need to use some kind of hashes (large, to avoid conflicts).

The 64-character strings of the GUIDs in DiscreteZOO are infeasibly long for reproduction in a text intended to be read by humans. We chose a standard abbreviation technique. In the Git versioning system [8], the objects are identified by 40-character hashes, but are usually referred to by simply taking the first 7 characters. In DiscreteZOO, the citable identifier is obtained by taking the first 12 hexadecimal digits of the hash and using further characters when necessary to avoid conflicts. A checksum hexadecimal digit, obtained by taking the XOR of the digits included in the citable identifier, is then added to the end. For readability, the characters are split into groups of 4 characters, the letter `Z` is prepended, and the checksum digit is separated by a plus sign:

`123456789abcdef...` $\rightarrow$ `Z1234-5678-9abc+c`.

The DiscreteZOO citable identifier (R3) is described in more detail in the project documentation [2, `Documentation` repository].

4.1.3 Query Language

The query language supports conditions on Boolean and numeric property types, as described below.

Definition 4.1 Let o be an object and let B be a Boolean property of this object. A **Boolean condition** is one of the following predicates.

- o has property B,
- o does not have property B.

Definition 4.2 Let o be an object and let N be a numeric property of this object and let n be a number and $R \in \{=, \neq, <, \leq, >, \geq\}$. A **numeric condition** is a predicate $N\ R\ n$.

Examples of Boolean conditions for graphs are "is bipartite", "is not Hamiltonian", etc. Examples of numeric conditions for graphs would be "girth equal to 3", "order at most 100", etc. Currently, expressing a condition "valency is prime" is not possible on the level of data (it is of course possible to check for it in the SageMath package).

It is only possible to query objects of the same type. Both the website and the SageMath package support simple queries (R1), consisting of conjunctions of the predicates described above.

Find all o of type T such that $P(o)$ holds for all $P \in \mathcal{C}$,
where $\mathcal{C}$ is a set of Boolean and numeric conditions.

The SageMath package allows arbitrary logical propositions (e.g., conjunctions, disjunctions, exclusive or, etc.) with the above predicates acting as atoms. Arithmetic can also be performed on numeric properties to derive new numeric conditions. The results can also be ordered according to some atomic or derived properties.

Let $\mathcal{P}$ be a logical proposition containing only Boolean and numeric conditions derived from the set of all properties of T. Let us additionally define $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_k$ as properties derived only from Boolean and numeric conditions from the set of all properties of T (through logical manipulation of Boolean properties and arithmetic manipulation of numeric properties). The SageMath supports the following queries.

Find all o of type T such that $\mathcal{P}(o)$ holds, in the lexicographic order of $(\mathcal{G}_1(o), \mathcal{G}_2(o), \dots, \mathcal{G}_k(o))$.

Additionally, besides iterating through the matches, it also allows counting queries (i.e., how many objects in the database satisfy the proposition). These can also be further enriched by grouping, so a single query can count the number of objects sharing the same value of a property (either atomic or derived) for all values appearing in the matches.

For each value $(\mathcal{G}_1(o), \mathcal{G}_2(o), \dots, \mathcal{G}_k(o))$ of an object o such that $\mathcal{P}(o)$ holds,
find the number of such objects.

Example 1 Find all graphs that are vertex-transitive, of valency 3 and girth at least 5.

Example 2 For each girth, count all cubic vertex-transitive graphs of diameter 5.

4.2 Contents

DiscreteZOO currently contains 212,269 graphs from the following three collections.

- The catalogue of vertex-transitive graphs on up to 31 vertices by McKay and Royle [20] (100,661 graphs),
- the census of cubic vertex-transitive graphs on up to 1280 vertices by Potočnik et al. [28] (111,360 graphs), and
- the census of cubic arc-transitive graphs on up to 2048 vertices by Marston Conder [9] (796 graphs).

A collection of 75,217 maniplexes is a work in progress by the first author.

Table 2 shows the properties which are currently computed for nearly all records and which do not have the same value for all of them. There are roughly 60 graph related properties in total, most of which are Boolean, integer or real-valued. We are also working on relations between objects (graph-valued graph properties) such as truncation.

4.3 Class Specifications and the Data Repository

The **class specifications** [2, `Specifications` repository] serve as a declarative (and to an extent, language-agnostic) way (R2.1) to describe CMOs. Together with the **data repository** [2, `Data` repository], they can be used to produce the databases used by the website and the SageMath interface. Each of them is available as a public repository on GitHub containing JSON files with the relevant information (Fig. 2).

The class specifications determine the possible classes of objects that can be stored in the DiscreteZOO data repository. The classes are organised in a hierarchical manner: `ZooEntity` represents the top-level class, and all other classes are descended from it. Directly descending from `ZooEntity` is the class `ZooObject`, which serves as a superclass for all mathematical objects in DiscreteZOO. Classes not descending from `ZooObject` are used to represent metadata – for instance, there is a `Change` class which is used to track changes to the database.

Each class specification is a JSON file storing an object which includes the following fields:

- `fields`: an object mapping property names to their types (one of `Integer`, `Rational`, `RealNumber`, `bool`, `str`, or a DiscreteZOO class);
- `primary_key`: the name of the property determining an instance (i.e., it acts as the primary key in the database);
- `condition`: an object specifying the required values of properties;
- `default`: an object specifying default values of properties (i.e., those which are determined by the conditions).

Note that the `fields` object only contains the names of the properties which are specific to the class – its instances will then additionally have all properties specified by the superclasses. The parent class is determined by the type of the property which is specified as `primary_key` (for `ZooEntity`, this is `Integer`, which is a builtin type) (Fig. 3).

For example, the `ZooObject` class has three properties: `alias`, `unique_id`, and `zooid`. The latter is also specified as the primary key, and its type `ZooEntity` means that it acts as a reference to the primary key of the `ZooEntity` class, i.e., its only property `zooid`. The first two properties have types deriving from the classes `ZooSet` and `ZooDict`, respectively. These two classes can be used to construct set and dictionary types, and do not have a corresponding JSON specification. The two properties thus provide every mathematical object stored in DiscreteZOO with a set

Table 2 Properties of graphs and maniplexes in DiscreteZOO

Graph property	
is arc-transitive	Boolean
is bipartite	Boolean
is Cayley	Boolean
is distance-regular	Boolean
is distance-transitive	Boolean
is edge-transitive	Boolean
is Eulerian	Boolean
is Hamiltonian	Boolean
is Möbius ladder	Boolean
is overfull	Boolean
is partial cube	Boolean
is prism	Boolean
is split	Boolean
is SPX	Boolean
is strongly regular	Boolean
chromatic index	numeric
clique number	numeric
connected components number	numeric
diameter	numeric
girth	numeric
odd girth	numeric
order	numeric
size	numeric
triangles count	numeric
Maniplex property	
is polytope	Boolean
is regular	Boolean
group order	numeric
number of orbits	numeric
rank	numeric

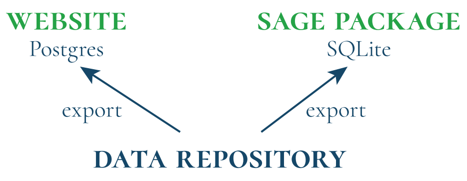


Fig. 2 Data repository and databases

of aliases and a dictionary of GUIDs – hashes of a canonical representations of an object as given by a chosen algorithm (Fig. 4).

The data repository stores information about the objects in DiscreteZOO. Each object is stored in a JSON file whose path is determined by its GUID: the first level folder gives the algorithm used, the second level folder is named for the first two nibbles in the hexadecimal representation of the GUID, and the filename corresponds to the

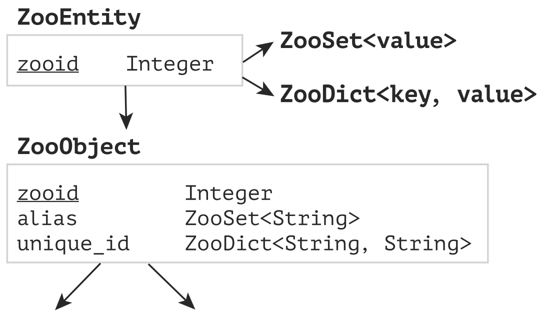


Fig. 3 Class hierarchy

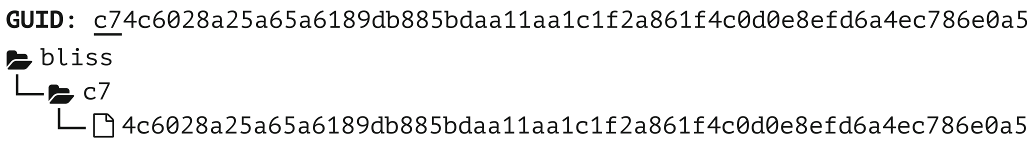

GUID: c74c6028a25a65a6189db885bdaa11aa1c1f2a861f4c0d0e8efd6a4ec786e0a5
bliss
c7
4c6028a25a65a6189db885bdaa11aa1c1f2a861f4c0d0e8efd6a4ec786e0a5

Fig. 4 Data repository: directory structure

remainder of the GUID. Since the GUID is only unique up to the algorithm used to compute it, an object may be represented by more than one file. In this case, only one of them is an ordinary file, while the rest are symbolic links pointing to the actual file.

Each data file contains a JSON object giving values of properties corresponding to the classes it belongs to. Note that any object in DiscreteZOO may belong to multiple classes – there is no restriction that an object only belong to superclasses of the most specific class. For instance, a graph may simultaneously be a cubic vertex-transitive graph and a split Praeger-Xu (SPX) graph, with neither of these classes being a subclass of the other. It is not required that every property be given – those which have not yet been computed will be simply left out.

The data repository also contains information about datasets. These are also stored as JSON files containing metadata, a list of classes the dataset represents, and a list of GUIDs of objects contained in the dataset. Datasets may also be nested – i.e., a parent dataset will contain all objects in the child dataset (but will not necessarily describe the same object types).

The information stored in the data repository can then be used to export the desired objects to a form suitable for querying and accessing the objects, e.g., an SQLite database. Note that, due to performance reasons, such a form may include additional identifiers which are not guaranteed to be globally unique, such as sequential row IDs. This information is therefore not stored in the data repository, which instead relies on GUIDs for the purposes of referencing other objects in the database.

4.3.1 Additional Features of the SageMath Package

For the SageMath package, it already suffices to describe a new collection by adding the appropriate data and dataset files to the data repository. Adding new object types, on the other hand, still require some programming.

The SQLite database in the SageMath package has a journaling system that keeps track of the tables, rows and columns changed, as well as of who introduced the changes. This allows users to prepare contributions to the database. To submit a contribution, an author with an account on GitHub can make a pull request to the data repository – i.e., a request to the repository maintainers to include the requested changes into the repository. If accepted, the contribution is merged with the database, the database downloads are updated and other users can choose to update their local databases (Fig. 5).

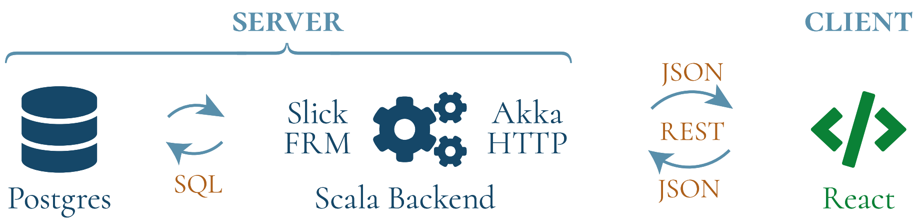


Fig. 5 Website stack

4.4 Implementation and User Interfaces

Each of the interfaces uses a simplified database optimised for its needs. In addition to the core and package databases, a user can download a subset of the core database for offline use with the SageMath package. This local database can store any properties she computes. From this, she can export the changes she makes to submit them to the core database.

Both the website and the SageMath package make it possible for the user to search for objects and filter object sets. The SageMath package supports adding new objects and properties into the local database. If a researcher wants to submit some of this to the core database, the package helps with preparing the changes file.

4.5 Website

The DiscreteZOO website is dedicated to simple searches, downloads and displaying encyclopaedic information.

We will describe the website functionality on Use case 1. The census of cubic vertex-transitive graphs on up to 1280 vertices [28] contains (among others) all graphs Tilen is interested in on up to 1280 vertices.

1. In the search box (see Fig. 6), he first chooses graphs as his objects ①
2. and then only selects the collection of cubic vertex transitive graphs from the list of graph collections ②.
3. He first adds the filter `is partial cube` from the available filters ④ to the selected filters ③, chooses `True` and confirms the selection by clicking on the checkmark.
4. He knows that even prisms ($K_2 \square C_{2n}$) are an infinite family of vertex-transitive cubic partial cubes and wants to remove them from the search results. He first adds the filter `is prism` from the available filters ④ to the selected filters ③, chooses `False` to filter for those that are not prisms and confirms selection by clicking on the checkmark (see Fig. 7).

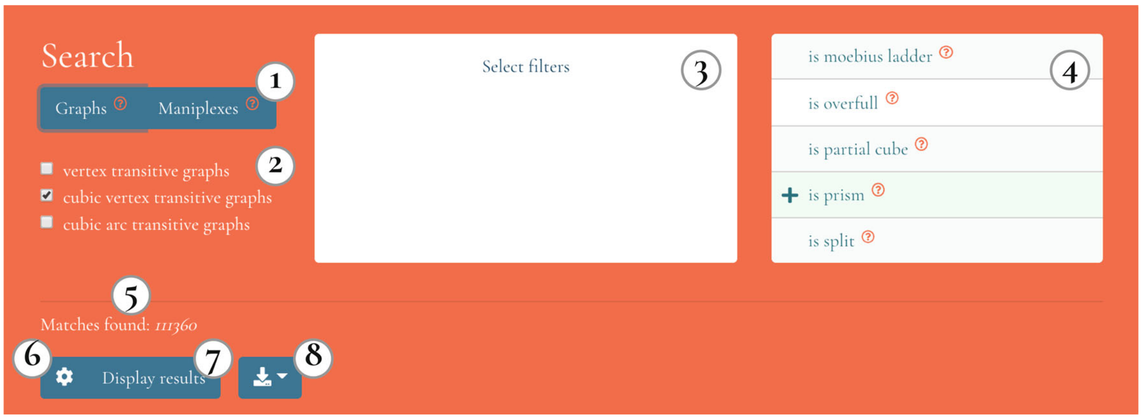
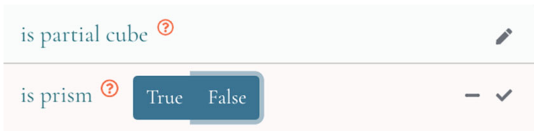


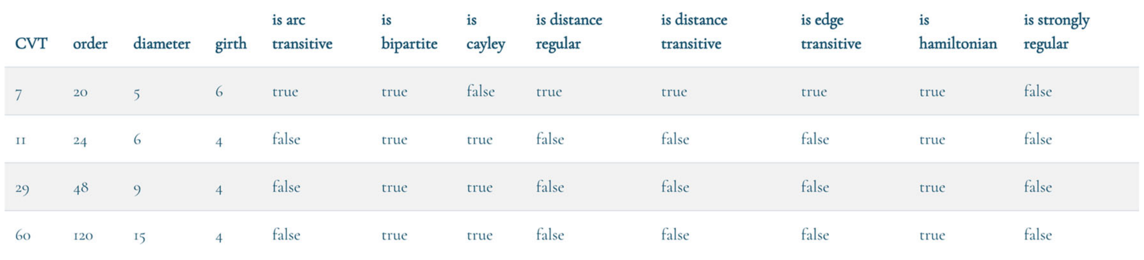
Fig. 6 Search box



is partial cube

is prism True False

Fig. 7 Selected filters



CVT	order	diameter	girth	is arc transitive	is bipartite	is cayley	is distance regular	is distance transitive	is edge transitive	is hamiltonian	is strongly regular
7	20	5	6	true	true	false	true	true	true	true	false
11	24	6	4	false	true	true	false	false	false	true	false
29	48	9	4	false	true	true	false	false	false	true	false
60	120	15	4	false	true	true	false	false	false	true	false

Fig. 8 Displayed results

5. The number of matches ⑤ immediately updates with the number of objects in the database that match the new search criteria, every time the chosen collections or filters get updated.
6. Finally, he downloads the list of graphs for SageMath from the download dropdown menu ⑧ and computes their isometric embeddings into hypercubes.

To optimise responsiveness, results are not displayed until the user presses the "Display results" button (Fig. 6 ⑥). Search result downloads do not require displaying the search results first.

The available filters area (Fig. 6 ④) contains all properties stored in the database for the chosen object type. Numeric properties can be filtered to satisfy simple equations (see Definition 4.2) (Fig. 8).

The following features are planned for implementation in the near future.

1. Download
   - graphs in the `sparse6` [21] format as well as formats readable by computer algebra systems,
   - automorphism groups of symmetric objects in the search results (in GAP and Magma code),
   - list of references in BibTeX (online resources, authors, papers) relevant for the search results.
2. Copy data to clipboard (for example, graph in the `sparse6`, automorphism group code, references).
3. Tooltip definitions of concepts (object types, filters).

4.6 SageMath Package

The objects in the database can also be accessed using the SageMath package [2, `DiscreteZOO-sage` repository]. SageMath [33] is an open source computer algebra system based on the Python programming language, and it already provides many structures for representing various mathematical objects. The DiscreteZOO SageMath package defines its own structures that inherit and override SageMath's structures, thus allowing one to utilise the full potential of SageMath while adding the functionality of accessing precomputed properties in the database, as well as storing newly computed properties back to the database for later reuse.

After installing the package and the database, one can either import the entire `discretezoo` package, or load submodules as one needs them. In the following examples, we will use the submodule for cubic vertex-transitive graphs, which includes the census of connected cubic vertex-transitive graphs by Potočnik et al. [28] (also known as the CVT census). We only need to import the class `CVTGraph` and the object `info` from the `discretezoo.entities.cvt` submodule, as well as the objects from the `discretezoo.entities.cvt.fields` submodule. The first submodule is intended for cubic vertex-transitive graphs, while the latter contains the objects representing the precomputed properties that one can use in search queries.

The CVTGraph class extends the ZooGraph class representing general graphs in the database, and the latter in turn extends both the general ZooObject class and SageMath's Graph class. One may construct a CVTGraph instance by specifying the order and index as given in the CVT census. For instance, one may obtain the Petersen graph and compare it to SageMath's builtin version using SageMath's is_isomorphic method.

```
sage: G = CVTGraph(10, 3)
sage: G.is_isomorphic(graphs.PetersenGraph())
True
```

A graph may also be constructed manually. In this case, its canonical form is computed and a GUID is derived from it. The GUID is then checked against the database, and if there is a matching entry, the precomputed properties will be loaded for the newly constructed graph.

```
sage: CVTGraph([[(u, i) for u in GF(7) for i in (−1, 1)],
....: lambda (u, i), (v, j): i != j and u∗i + v∗j in (1, 2, 4)])
Heawood graph: cubic vertex−transitive graph on 14 vertices, number 1
```

If the graph is not found in the database, it will be checked for the required properties (say, that it is cubic and vertex-transitive), and added to the local copy of the database. An entry indicating that a change to the database has occured will then also be added, thus allowing the user to track her changes and later make a contribution to the main database.

In Use case 1, Tilen is studying cubic vertex-transitive partial cubes. He uses the info object to make queries to the database. The info.all method returns a generator yielding the requested graphs. He restricts the queries by specifying conditions using the field objects.

```
sage: gen = info.all(is_partial_cube, orderby=order) # order by the number of vertices
sage: next(gen) # first matching graph
3−Cube: cubic vertex−transitive graph on 8 vertices, number 2
sage: next(gen) # second matching graph
6−Prism: cubic vertex−transitive graph on 12 vertices, number 3
```

He notices that the small examples he has obtained are even prisms. He now wants to obtain examples which are not prisms.

```
sage: partial_cubes = list(info.all(is_partial_cube, ~is_prism))
sage: partial_cubes
[Desargues graph: cubic vertex−transitive graph on 20 vertices, number 7,
 Truncated Octahedron: cubic vertex−transitive graph on 24 vertices, number 11,
 Truncated Cuboctahedron: cubic vertex−transitive graph on 48 vertices, number 29,
 Truncated Icosidodecahedron: cubic vertex−transitive graph on 120 vertices, number 60]
```

He then examines the first graph in the list. Asking whether the graph is a partial cube will give the precomputed value which has been used to find it.

```
sage: H = partial_cubes[0]
sage: H.is_partial_cube()
True
```

However, many methods in SageMath can also compute certificates which can be used to verify the correctness of the result. The certificate for a graph being a partial cube is an isometric embedding into a hypercube – i.e., a mapping from the vertices of the graph to the vertices of the hypercube ((0, 1)-strings of a fixed length). Since certificates are not stored in DiscreteZOO, this triggers the computation.

```
sage: _, cert = H.is_partial_cube(certificate=True)
sage: cert
{0: '00000', 1: '10111', 2: '11101', 3: '11011', 4: '11111', 5: '01011', 6: '11010', 7: '11100',
 8: '01101', 9: '10110', 10: '00111', 11: '11000', 12: '01001', 13: '10010', 14: '00011',
 15: '10100', 16: '00101', 17: '00010', 18: '00100', 19: '01000'}
```

Tilen thus sees that the Desargues graph can be isometrically embedded into the 5-dimensional hypercube.

On the other hand, in Use case 2, Primož wants to count the number of graphs of each girth in the CVT census. He can do this with the `info.count` method. Since the database contains some cubic vertex-transitive graphs which only appear in other censuses, he restricts his query to those graphs for which the index from the census exists.

```
sage: info.count(cvt_index) # number of graphs in the CVT census
111360
sage: info.count(cvt_index, groupby=girth) # break down by girth
{3: 160, 4: 5754, 5: 100, 6: 58674, 7: 192, 8: 13529, 9: 219, 10: 25806, 11: 80, 12: 5423, 13: 37,
 14: 1365, 15: 12, 16: 9}
```

He now wants to obtain a few small graphs of girth 7 and diameter 4. This can be achieved using the `info.one` method, which only returns a single graph.

```
sage: info.one(girth == 7, diameter == 4, orderby=order)
Generalised Petersen graph (13, 5): cubic vertex-transitive graph on 26 vertices, number 5
sage: info.one(girth == 7, diameter == 4, orderby=order, offset=1) # the second such graph
Coxeter graph: cubic vertex-transitive graph on 28 vertices, number 6
```

The two main parts of the SageMath package implementation are the **object classes** and the **database interfaces**.

The object class hierarchy closely follows the one given in the class specifications repository (see Sect. 4.3). Upon initialisation, each class is initialised with the information given in the corresponding JSON file. Additionally, it also implements methods needed for the functionality of the corresponding objects. Thus, most generic functionality resides in the methods of the `ZooEntity` and `ZooObject` classes. The subclasses of `ZooObject` also inherit their methods and properties from the appropriate classes provided by SageMath. These are however wrapped to deal with precomputed properties – i.e., when a method is called without extra parameters, a precomputed property will be returned if it is available, otherwise it will be written back to the database once computed. The methods such classes implement thus mostly deal with peculiarities related to object construction and calling certain methods inherited from SageMath.

To deal with queries, each object class is accompanied by an `info` object providing the user an interface for the queries, and a `fields` submodule containing objects representing the corresponding properties (including the ones inherited from its superclasses). These objects thus correspond to the properties of the query language (see Sect. 4.1.3), and they can be combined into more complex expressions using arithmetic and logical operators. Such expressions can then be given to the database interface, which transforms it into a query for the appropriate database. Currently, a generic SQL interface is available, with SQLite and PostgreSQL database interfaces as its special cases.

5 Conclusions and Future Work

We built on Billey and Tenner's observations on the need of the mathematical community for searchable databases and described some further requirements we consider important. We observed that their work applies equally well to databases of concrete mathematical objects, which are similar to fingerprint databases of theorems, but of slightly different flavour. We did not discuss the problem of ensuring sustainability, which is extremely important, but very broad. The ongoing work on http://data.mathhub.info at FAU by the first author and collaborators addresses it partially, as it offers hosting for datasets that would otherwise become orphaned. Another partial solution could involve starting a journal for data in mathematics. To get a sense of CMOs and databases we started an online catalogue, which we hope to extend into a survey paper in the future.

DiscreteZOO is an experimental project towards a general purpose framework for CMOs. The system is initially targeted at the community of researchers working with symmetric graphs (and some related objects). Above, we have already indicated some short term goals and features for the future. In particular, we would like to tackle coverage information (R7). An important practical consideration will be to further automate updates to the database, make them as declarative as possible and thus decrease the effort needed to contribute a new collection (R2.1). First steps towards a “programmable, mathematical API” for mathematical knowledge bases was investigated in [36] for the LMFDB. We are going to explore the possibilities offered by MMT [35], a framework for knowledge representation, to support describing the CMOs and objects they contain. In addition to the syntax, MMT also gives access to a large body of formalised knowledge, which would make implementing self-explanatory aspects (R5) much easier. We also plan to explore the Math-in-the-Middle (MitM) [10] approach for interoperability (R6) with computer algebra systems and other databases. MitM “allows translation between centrally formalized knowledge and the systems on the boundary via views and alignments”.

Acknowledgements Open Access funding provided by Projekt DEAL. The work by the first author reported here is supported in part by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541). The first author did the main portion of her work on the project while a postdoctoral scholar at the Center for Mathematical Sciences of the National Autonomous University of Mexico, Campus Morelia. The work by the second author reported here is supported by the Slovenian Research Agency (research program P1-0285 and project J1-8130). The first author would also like to thank Michael Kohlhase for his encouragement, his many insightful comments, and for pointing out the need to include a checksum in a CID. Finally, the authors would like to thank the second reviewer for important comments.

References

1. Babai, L., Luks, E.M.: Canonical Labeling of Graphs. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing. STOC ’83. New York, NY, USA: ACM, pp. 171–183 (1983). ISBN: 0-89791-099-0. https://doi.org/10.1145/800061.808746
2. Berčič, K., Vidali, J.: DiscreteZOO repositories. https://github.com/DiscreteZOO
3. Berčič, K.: Math Databases. https://github.com/MathHubInfo/Documentation/wiki/Math-Databases. Accessed 29 Nov 2018
4. Berg, C., Stump, C.: FindStat: The Combinatorial Statistic Finder (2014). http://www.FindStat.org
5. Billey, S.C., Tenner, B.E.: Fingerprint databases for theorems. Not. Am. Math. Soc. **60**(8), 1034–1039 (2013). https://doi.org/10.1090/noti1029. issn: 0002-9920
6. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. In: J. Symb. Comput. 24.3-4. Computational algebra and number theory (London, 1993), pp. 235–265, (1997). ISSN: 0747-7171. https://doi.org/10.1006/jsco.1996.0125
7. Brinkmann, G., et al.: House of graphs: a database of interesting graphs. Discrete Appl. Math. **161**(1), 311–314 (2013). https://doi.org/10.1016/j.dam.2012.07.018. issn: 0166-218X
8. Chacon, S., Straub, B.: Pro Git (2014). https://git-scm.com/
9. Conder, M.D.E., et al.: The Extended Foster Census. https://www.math.auckland.ac.nz/~conder/symmcubic2048list.txt
10. Dehaye, P.-O. et al.: Interoperability in the OpenDreamKit project: the math-in-the-middle approach. In: Kohlhase, M., et al. (eds.) Intelligent Computer Mathematics 2016. LNAI 9791. Springer, (2016). ISBN: 978-3-319-08434-3. https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf
11. Farmer, W.M.: Mathematical knowledge management. In: Schwartz, D., Te’eni, D. (eds.) Encyclopedia of Knowledge Management, 2nd edn, pp. 1082–1089. Idea Group Reference, Hershey (2011)
12. Grout, J.: Small Graph Database. https://jasongrout.org/graph_database. Accessed 3 Dec 2018
13. Junttila, T., Kaski, P.: Bliss: A Tool for Computing Automorphism Groups and Canonical Labelings of Graphs. http://www.tcs.hut.fi/Software/bliss/
14. Kholmatova, A.: Design Systems. Smashing Media AG, Breisgau (2017)
15. Kohlhase, M.: Math Object Identifiers—Towards Research Data in Mathematics. In: Kohlhase, A., Kübler, E. (ed.) Wissens- und Erfahrungsmanagement (Knowledge and Experience Management), FGWM. Workshop at LWDA 2017, pp. 214–252 (2017). http://ceur-ws.org/Vol-1917/paper33.pdf
16. Kohlhase, M.: Mathematical knowledge management: transcending the one-brain-barrier with theory graphs. In: EMS Newsletter, pp. 22–27 (2014). https://kwarc.info/people/mkohlhase/papers/ems13.pdf
17. Lozier, D.W.: NIST Digital Library of Mathematical Function. In: Annals of Mathematics and Artificial Intelligence—Special Issue on Mathematical Knowledge Management 38, pp. 105–119 (2003). http://dlmf.nist.gov/about/publications/MKM-Lozier.pdf

18. Marc, T.: Classification of vertex-transitive cubic partial cubes. J. Graph Theory **86**(4), 406–421 (2017). https://doi.org/10.1002/jgt.22134
19. MathOverflow contributors. What are some early examples of creation of lists / catalogues of (particularly) combinatorial objects? MathOverflow. https://mathoverflow.net/questions/47044/what-are-some-early-examples-of-creation-of-lists-catalogues-of-particularly. Accessed 20 Nov 2018
20. McKay, B., Royle, G.: Transitive Graphs. http://staffhome.ecm.uwa.edu.au/~00013890/remote/trans/index.html
21. McKay, B.: Description of graph6, sparse6 and digraph6 encodings. http://users.cecs.anu.edu.au/~bdm/data/formats.txt
22. McKay, B.D., Piperno, A.: Practical graph isomorphism, II. J. Symb. Comput. **60**, 94–112 (2014). https://doi.org/10.1016/j.jsc.2013.09.003. issn: 0747-7171
23. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography, p. xxviii+780. CRC Press, Boca Raton (1997)
24. Norman, D.: The Design of Everyday Things. Revised and Expanded. Basic Books, New York (2013)
25. OEIS Foundation Inc., ed. The On-Line Encyclopedia of Integer Sequences. http://oeis.org (visited on 05/28/2017)
26. Pisanski, T., et al.: The Encyclopedia of Graphs. http://atlas.gregas.eu. Accessed 3 Dec 2018
27. Potočnik, P., Vidali, J.: Girth-regular graphs. ARS Math. Contemp. **17**(2), 349–368 (2019). https://doi.org/10.26493/1855-3974.1684.b0d
28. Potočnik, P., Spiga, P., Verret, G.: Cubic vertex-transitive graphs on up to 1280 vertices. J. Symb. Comput. **50**, 465–477 (2013). https://doi.org/10.1016/j.jsc.2012.09.002. issn: 0747-7171
29. Read, R.C., Wilson, R.J.: An Atlas of Graphs. p. 464 (2005)
30. Tenner, B.E.: Database of permutation pattern avoidance. http://math.depaul.edu/bridget/patterns.html
31. The Kant Project. Querying Algebraic Objects System. http://page.math.tu-berlin.de/~kant/database.html. Accessed 3 Dec 2018
32. The LMFDB Collaboration. The L-functions and Modular Forms Database. http://www.lmfdb.org. Accessed 1 Feb 2016
33. The Sage Developers. SageMath, the Sage Mathematics Software System (Version 8.3) (2018). http://www.sagemath.org
34. The SymbolicData Project. SymbolicData Project. https://symbolicdata.github.io/. Accessed 3 Dec 2018
35. UniFormal/MMT—The MMT Language and System. https://github.com/UniFormal/MMT. Accessed 24 Oct 2017
36. Wiesing, T., Kohlhase, M., Rabe, F.: Virtual theories—a uniform interface to mathematical knowledge bases. In: Blömer, J., Kutsia, T., Simos, D. (eds.) MACIS 2017: Seventh International Conference on Mathematical Aspects of Computer and Information Sciences. LNCS 10693. Springer, pp. 243–257 (2017). https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/MACIS17-vt/crc.pdf
37. Wikipedia contributors. Fingerprint (disambiguation). Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Fingerprint_(disambiguation). Accessed 22 Nov 2018
38. Wilson, R., et al.: ATLAS of Finite Group Representations—Version 3. http://brauer.maths.qmul.ac.uk/Atlas/v3/. Accessed 3 Dec 2018