Classifying Discrete Objects with *Orbiter*

Anton Betten

Abstract

Orbiter is a software package to classify discrete objects such as designs, graphs, codes, and objects from finite geometry. It employs the method of *breaking the symmetry* to attack difficult problem instances by means of subobjects that serve as a stepping stone. The algorithms combine techniques from Group Theory and from Combinatorics. *Orbiter* is a library of C++ functions that provide functionality for Discrete Mathematics. In order to be applied to a specific problem, code has to be written taylored to the specific application.

The Experimental Approach

Research in Mathematics often benefits from examples. There are many areas where the existing theory does not suffice to explain all the examples that are known. Therefore, studying examples is often the first step in finding new constructions or new theory. This new theory helps to explain where the known examples come from, and often predicts more examples for larger instances of the parameter set. In some cases, infinite families of objects can be constructed. In order to help with this process, knowing examples and their automorphism groups is crucial. In Discrete Mathematics and Combinatorics, we are able to examine (at least for small cases) complete lists of objects up to isomorphism. This kind of data is very valuable to researchers. *Orbiter* is intended to assist with these classification problems. One could call this the *experimental approach* to mathematics. Designs, graphs and codes are related topics, and all are well-suited to computer investigation. The various links between these areas are stressed in [7].

Classifying all orbits of a permutation group means listing a set of representatives for the orbits, together with their respective stabilizer subgroups. Also, if an object is given, we can compute a group element that maps the given object to its representative in the classification. These kinds of problems are notoriously difficult, since they involve the isomorphism problem as a subproblem and isomorphism is usually NP-hard.

Breaking The Symmetry

While group theoretic algorithms to classify orbits are available, experience shows that many problems require a combination of methods to be solved efficiently. The method of *breaking the symmetry* allows to classify objects using subobjects that serve as a stepping stone. The subobjects are easier to classify, and it is possible to lift the classification of subobjects to the classification of the original objects. The method combines group theory with traditional "solvers." Here, we understand solvers

as any kind of computational primitive to solve the problem of *finding* all objects that arise from a given *starter object*. Once these objects have been found, an additional isomorph rejection step is performed to solve the classification problem. The theory behind this is explained in [2], where the method is applied to the classification of packings in $\mathrm{PG}(3,3)$. It is important to realize that the method is very general, and can be applied to broad classes of problems.

The use of subobjects is well-known. In [11], homomorphisms of group actions are discussed. In [5] and [6], the technique of "breaking the symmetry" is developed. In [16], an algebraic algorithm to compute the orbit decomposition is presented. In *Orbiter*, many of these algorithms are combined, and an isomorph classification module based on the theory described in [2] is present. *Orbiter* offers some algorithms to solve these systems of equations but also allows to interface with third party software. The communication between *Orbiter* and the outside solvers can happen through files. Once the data from the solver is received, the isomorph classification module starts its job and computes the final list of isomorphism types together with the stabilizer groups. Data from the classification is stored in files to allow identifying objects of the givem type at a later point in time. This means that given an object, a group element can be computed that maps the object to one of the representatives from the classification. An implementation of Knuth's dancing links (DLX) [10] is available. Wassermann's algorithm [17] or any other suitable piece of software can be used as an external solver.

The underlying idea behind *Orbiter* is to provide isomorph classification for a variety of types of objects. To be able to handle things uniformly, we rely on the use of C++ function pointers to realize permutation group algorithms for arbitrary objects. The only requirement is that the object can be represented as a set (or set of sets). The entries of the set are integers that represent the components of the object. For instance, when classifying sets of points in a finite projective space subject to certain conditions, the components are projective points, and they are represented numerically. When classifying combinatorial designs, the objects consist of sets of subsets of a set X. These subsets are known as blocks, and they form the components of the object. We can use the lexicographic ordering of subsets of X to identify blocks with integers. The process of converting components into integers and integers into components is called *ranking* and *unranking*. Sometimes, the terms indexing or enumerating are used also. Basically, the possible components are mapped bijectively to an interval of integers. We require that rank and unrank functions to encode the object under consideration are available. For may types of combinatorial objects, such functions exist or can be devised easily. Using this kind of methodology, *Orbiter* is able to realize permutation groups acting on objects. The permutation group algorithms and the functionality for the specific object are completely separate. The group does not know what objects it is acting on, and the objects does not know what group is acting on them. This methodology makes the code easily adaptable to different actions. The most basic group actions are that of the symmetric group acting on a set, and the projective linear group acting on projective space (as well as the affine group acting on a vector space). From these basic actions, one can define induced actions in various ways. For instance, the symmetric group induces an action on the k-subsets. The projective linear group induces an action on the Grassmannian of subspaces. Many other induced actions are available.

Orbiter's predecessor is DISCRETA [4], which is specialized to t-designs with prescribed groups of automorphisms, and comes with a graphical user interface. Since *Orbiter* applies to a much more

general class of problems, there was no longer the possibility for a graphical user interface. Instead, the user of *Orbiter* will have to write code to facilitate the algorithms that are provided. *Orbiter* is available from the website [15].

Applications

Recently, *Orbiter* has been used to classify packings [2], unitals [1] and BLT-sets [3]. Other applications exist. Some are described in the Orbiter Manual.

Other Work

A general reference for the problem of classifying designs and codes is [9]. This book emphasizes the use of canonical forms, facilitated for instance via the software package nauty [14], or the partition backtrack approach [12]. Both of these algorithms are available through the computer algebra system MAGMA [13]. Nauty is also available through *Orbiter*. A different computer algebra system with an emphasis on Group Theory is GAP [8].

References

[1] John Bamberg, Anton Betten, Cheryl Praeger, and Alfred Wassermann. Unitals in the Desarguesian Projective Plane of Order Sixteen. To appear in Journal of Statistical Planning and Inference.

[2] Anton Betten. The Packings of PG(3, 3). Submitted to Journal of Combinatorial Designs.

[3] Anton Betten. Rainbow Cliques and the Classification of Small BLT-Sets. Accepted for the Proceedings of ISSAC 2013.

[4] Anton Betten, Reinhard Laue, and Alfred Wassermann. DISCRETA, a tool for constructing t-designs. In: Computer Algebra Handbook, Edited by Johannes Grabmeier, Erich Kaltofen, Volker Weispfennig, Springer 2003, pp 372-375.

[5] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. In *Applied algebra, algebraic algorithms and error-correcting codes (Rome, 1988)*, volume 357 of *Lecture Notes in Comput. Sci.*, pages 99–110. Springer, Berlin, 1989.

[6] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. *Nordic J. Comput.*, 3(3):203–219, 1996.

[7] P. J. Cameron and J. H. van Lint. *Designs, graphs, codes and their links*, volume 22 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1991.

[8] GAP – Groups, Algorithms, and Programming, Version 4.4. The GAP Group, Aachen, Germany and St. Andrews, Scotland, 2004.

[9] P. Kaski and P. Östergård. *Classification algorithms for codes and designs*, volume 15 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2006.

[10] D. E. Knuth. Dancing links. *eprint arXiv:cs/0011047*, November 2000. in Davies, Jim; Roscoe, Bill; Woodcock, Jim, Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, Palgrave, pp. 187-214.

[11] R. Laue. Construction of combinatorial objects—a tutorial. *Bayreuth. Math. Schr.*, 43:53–96, 1993. Konstruktive Anwendungen von Algebra und Kombinatorik (Bayreuth, 1991).

[12] J.S. Leon. Partitions, refinements, and permutation group computation. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 123–158. Amer. Math. Soc., Providence, RI, 1997.

[13] Magma. The Computational Algebra Group within the School of Mathematics and Statistics of the University of Sydney, 2004.

[14] Nauty User's Guide (Version 2.4), Brendan McKay, Australian National University, Nov 4, 2009.

[15] Orbiter – A Program To Classify Discrete Objects, http://www.math.colostate.edu/∼betten/orbiter/orbiter.html, Anton Betten, 2013.

[16] B. Schmalz. t-Designs zu vorgegebener Automorphismengruppe. *Bayreuth. Math. Schr.*, 41:1–164, 1992. Dissertation, Universität Bayreuth, Bayreuth, 1992.

[17] Alfred Wassermann. Finding simple t-designs with enumeration techniques. *J. Combin. Des.*, 6(2):79–90, 1998.