Summary of New Features in Magma V2.10

April 2003

1 Introduction

This document provides a terse summary of the new features installed in Magma for release version V2.10 (April, 2003). Previous releases of Magma were: V2.9 (May 2002), V2.8 (July 2001), V2.7 (June 2000), V2.6 (November 1999), V2.5 (July 1999), V2.4 (December 1998), V2.3 (January 1998), V2.2 (April 1997), V2.1 (October 1996), V2.01 (June 1996) and V1.3 (March 1996).

2 Summary

Groups

- *Permutation Groups:* The performance of the Schreier–Todd-Coxeter–Sims algorithm for computing the order of a permutation group has been greatly improved in a number of important contexts. A new algorithm for chief series (developed by Bill Unger) provides greatly improved performance. The database of primitive permutation groups has been extended from degree 50 to degree 999.

- *Finite Groups:* A new algorithm developed by Holt and Cannon that determines the maximal subgroups of any finite group using a knowledge of the maximal subgroups of its simple composition factors is included for the first time.

- *Finite Groups:* The maximal subgroup machinery is now able to access the maximal subgroups and automorphism groups of a greatly expanded set of simple groups. These include HS and $L(6,2)$, $Alt(d)$ ($d < 1000$), $L(2,p)$, $L(2,p^2)$, $L(2,p^3)$, $L(3,p)$, $L(3,8)$, $L(3,9)$, $S(4,p)$, and $U(3,p)$ (p prime). This greatly extends the range of those algorithms used for determining subgroups and automorphism groups. Fast algorithms for computing all subgroups having index less than some specified (moderate) bound in permutation and pc-groups have been implemented.

- *Finite Matrix Groups:* It is now possible to compute automorphism groups and determine isomorphism of any finite matrix group for which it is possible to to compute a BSGS. A revised version of the package for performing Aschbacher analysis has been provided by Eamonn O'Brien.

- *Group Cohomology:* A new module for computing the cohomology of a permutation group or pc-group developed by Derek Holt is included. This module also provides machinery for constructing extensions of G-modules and general abelian groups by general finite groups. It will be extended to include finite matrix groups shortly.

- *Finitely Presented Groups:* A highly tuned version of the Holt algorithm for computing equivalence classes of homomorphisms from a finitely presented group to a permutation group has been developed. This has been successful in constructing epimorphisms onto groups of order up to 10^9.

- *Braid Groups:* Asymptotically fast algorithms have been implemented for group operations, lattice operations, normal form computations and for computing the super summit set and the set of positive conjugates of an element. This work has as its goal the implementation of a new fast algorithm due to Volker Gebhardt for solving the conjugacy problem.

- *Groups and Monoids Defined by Rewrite Systems:* A new version of the KBMAG package of Derek Holt has been installed. The new installation provides code for the construction of confluent presentations for finitely presented groups and monoids as well as short-lex automatic groups. This machinery is an enormous improvement over the previous KBMAG installation in Magma, particularly in the case of determining the automatic structure of a group.

Commutative Algebra

- *Polynomial Rings and Affine Algebras:* A new algorithm by Allan Steel (to be published) solves a suite of fundamental problems associated with arbitrary algebraic function fields. These fields may be constructed as chains of algebraic extensions (using polynomial quotient rings, affine algebras, or standard algebraic function fields) and transcendental extensions (using rational function fields). The fields may have any characteristic and the algebraic extensions may be inseparable (which can happen in small characteristic). The problems now solved include decomposition of multivariate ideals and factorization of polynomials over such fields.

Extensions of Rings

- *Algebraic Function Fields*: Conversion of arbitrary finite extensions of univariate function fields to simple extensions is possible, thus any field can be converted into the representation suited best for any given application.

- *Algebraic Function Fields*: The machinery for relative extensions of function fields has been extended so that essentially all intrinsics now apply to both absolute extensions and relative extensions. In particular, divisor theory is supported in the relative case.

- *Algebraic Function Fields:* Magma 2.10 includes an experimental release of a module for class field theory of global functions fields. In particular, it is possible to determine defining equations for arbitrary Abelian extensions.

- *Function Fields:* The computation of GCDs of polynomials over function fields has also been greatly improved through use of a new modular algorithm. Factorization of polynomials over arbitrary algebraic function fields of any characteristic is now fully supported.

- *p-adic Rings and Fields:* The module for p-adic rings (and fields) and their extensions has been completely rewritten in Magma 2.10, resulting in greatly improved performance that is highly competitive for cryptographic applications. Furthermore, the new "Round 4" algorithm of Pauli for factoring polynomials over local fields is included.

Representation Theory

- *Modular Representations:* A package is provided which attempts to construct all (absolutely) irreducible representations of a finite group over a given finite field. For soluble groups the Brückner adaptation of the Glasby–Howlett method is used. For non–soluble groups the irreducibles are obtained by chopping up representations using the Meataxe.

Lie Theory

- *Lie Theory:* The entire module has been heavily revised and greatly expanded. Only a few highlights are noted here.

- *Coxeter Groups:* A new category of Coxeter groups has been implemented. It represents elements as words in the standard Coxeter group presentation. The normal form of an element is found using an efficient new algorithm developed by Robert Howlett.

- *Reflection Groups:* Some basic functions are now provided for creating and identifying reflection groups over an arbitrary field. The functionality for real reflection groups has been expanded.

- *Group of Lie type:* A much faster algorithm is now used for multiplying elements in groups of Lie type. The standard automorphisms of these groups are available (i.e. inner, diagonal, diagram, and field automorphisms).

- *Lie Group Representations:* The highest weight representations of a group of Lie type can be computed—this gives all rational representations over the base field.

Algebraic Geometry

- *Schemes:* Maps between schemes can now be represented with multiple sets of polynomials, which must agree wherever both sets define a valid map. This allows the representation of true morphisms. Additionally, a routine is available that uses Groebner basis techniques to derive extra sets of defining equations, to extend the domain of definition of a map. For maps between smooth curves, this always yields a morphism. A routine is provided that properly tests a map for birationality and finds a birational inverse, if it exists.

- *Curve Maps:* Maps between curves can now be used to pull back functions, differentials and divisors and to push forward functions and divisors.

- *Projective Varieties (Local solubility):* Given a projective variety over Q or over a number field, one can now test if there are any points over the completion of the base field at a finite prime. For plane curves, one can choose to check solubility of the desingularised curve. The system will perform the necessary blowups as necessary. For hyperelliptic curves, local solubility at a finite prime with a sufficiently large residue field of odd characteristic is tested using an algorithm that is independent of the size of the residue field.

- *Plane Curves:* Plane curves and their function fields are now tightly integrated. It is now possible go backwards and forwards between the coordinate rings of a curve and its function field and to construct projective embeddings from functions.

- *Elliptic Curves:* 2-Selmer groups and 2-Isogeny Selmer groups of elliptic curves over number fields can now be computed explicitly and are returned as abstract abelian groups. Maps are provided to represent elements of the Selmer group as elements of etale algebras and to get the corresponding principal homogeneous space, represented as a cover of the elliptic curve.

- *Elliptic Curves:* The Magma V2.9 Cremona database of elliptic curves (up to conductor 10 000) has been replaced in V2.10 by a version that includes all curves having conductor up to 20 000.

- *Modular Forms:* William Stein has supplied a new revision of his packages for modular symbols and modular forms.

- *Subgroups of $PSL(2, R)$:* A new revision of Helena's Verrill's package for subgroups of $PSL(2, R)$ has been installed.

- *Numerical Graded Rings:* The existing K3 database in Magma contains the 391 examples of K3 surfaces in the known lists. The same ideas have now been used to construct much bigger lists of K3 surfaces, that include all possible configurations of input data, (Gavin Brown).

Incidence Structures

- *Networks:* Flow networks have been introduced as a new type. These may have multiple edges and loops. Two flow algorithms have been implemented, the Dinic algorithm and the push-relabel method.

- *Graphs:* Magma V2.10 contains an implementation of the linear-time algorithm of Hopcroft and Tarjan for finding the 3-connected components of a graph. In addition, it is now possible to determine the vertex connectivity and the edge connectivity of a graph. Finally, a maximum matching algorithm has been implemented for bipartite graphs.

- *Incidence Structures and Designs:* A much faster but more space expensive mechanism is provided to test whether a structure is t-balanced.

- *Finite Projective Planes:* A much improved method is used to test whether an incidence structure is a finite projective plane. The impact of the new method is particularly dramatic when attempting to construct planes of large order.

Coding Theory

- *Linear Codes over Finite Fields:* Magma V2.10 contains a database of constructions of best known codes up to length 100 over $GF(4)$. The codes of length up to 18 are optimal. The database is over 98 per cent complete with only 73 of the 5150 codes missing, the first such missing code occurs at length 92. Many of the codes constructed in this database have bounds that are vast improvements on the previously bounds for best codes over $GF(4)$. (Marcus Grassl and Greg White).

- *Linear Codes over Finite Rings:* Magma V2.10 supports error- correcting codes over integer residue rings and Galois rings, with specialised functionality for codes over $\mathbf{Z}_4$, the ring of integers modulo 4. The machinery for codes over general finite rings includes basic constructions such as cyclic code and permutation code and the determination of the complete weight enumerator and Hamming weight distribution.

- *Linear Codes over $\mathbf{Z}_4$:* Features include the Gray map and a host of specific constructions including Kerdock, Preparata, Reed–Muller, Goethals codes and more. Optimised code for calculating the Lee and Euclidean weight distributions is included.

3 Removals and Changes

This section lists the most important changes in Version 2.10. Other minor changes are listed in the relevant sections.

– The user interface for the Number Field Sieve (NFS) integer factorization package has been completely rewritten. All of the previous NFS intrinsics have been removed and replaced with new equivalents.

4 Documentation

New chapters in the Handbook for V2.10 are:

- Cohomology
- Introduction to Lie Theory
- Root Systems
- Coxeter Systems
- Coxeter Groups (the previous Coxeter Group chapter has been renamed to "Coxeter Groups as Permutation Groups")
- Numerical Graded Rings
- Networks

There has been some rearrangement of chapters within the Handbook.

The HTML version of the Handbook now has an improved index (with more detailed subindices).

5 Aggregates

5.1 Mappings [HB 13]

New features:

- The intrinsic `Components` returns the maps which were composed to form the input.

6 Semigroups and Monoids

6.1 Monoids Defined by Rewrite Systems [HB 15]

The 2001 version of the `kbmag` package of Derek Holt has been installed. The first Magma version of this package was released in V2.5 (July 1999) but it suffered from numerous problems. Since then Derek Holt revised his package and the revised edition was installed from scratch by Graham Matthews. The new installation works brilliantly – the various problems that crippled the V2.5 installation have all been overcome. The functionality is unchanged.

New features:

- A new installation of the entire program provides much greater reliability and efficiency.

7 Groups

7.1 Permutation Groups [HB 17]

New Features:

- The database of primitive groups has been extended to degree 999 by Colva Roney-Dougal and Bill Unger. We believe the list to be complete. Numbering from Sims' list of degree up to 50 is changed. There are also added access functions, but the basic functions `PrimitiveGroup(d,n)` and `NumberOfPrimitiveGroups(d)` will work as before. A new function, which has been added, is `PrimitiveGroupIdentification`.
- A new `FPGroupStrong` algorithm has been implemented, a combination of STCS, BCS verification and results of Volker Gebhardt.
- The range of groups that `MaximalSubgroups` and `AutomorphismGroup` may be applied to is increased. The almost simple groups database now includes HS and $L(6,2)$. Derek Holt and Colva Roney-Dougal have also supplied routines so that the maximal subgroups and automorphism groups of the groups $Alt(d)$ $(d < 1000)$, $L(2,p)$, $L(2,p^2)$, $L(2,p^3)$, $L(3,p)$, $L(3,8)$, $L(3,9)$, $S(4,p)$, and $U(3,p)$ (p prime) may be found as if they were in the database. Functions `MaximalSubgroups` and `AutomorphismGroup` may be applied to any permutation group with all non-abelian composition factors in the database or listed above.
- A function `DoubleCosetRepresentatives`, computing a set of representatives of the double cosets of subgroups H and K of a permutation group G has been provided.

Bug fixes:

- A bug in computing an EARNS for a primitive group has been fixed.
- Several memory leaks, particularly associated with computing socle quotients and socle factors, have been stopped.
- Fixed user reported bug in ReduceGenerators.

7.2 Matrix Groups over Finite Fields (Aschbacher Analysis) [HB 18]

A revised version of this package that performs Aschbacher analysis has been provided by Eamonn O'Brien.

Changes:

- The intrinsics `BlockSystem` and `TensorInducedFactors` have been removed.

New features:

- The intrinsic `NormalSubgroupRandomElement` returns a random element of the normal closure of a subgroup given in terms of normal generators.
- The intrinsic `ExtraSpecialNormaliser` returns the action of the generators of a matrix group on an normal extraspecial or symplectic subgroup.

– The intrinsic `ExtraSpecialAction` gives the action of an element on an extraspecial or symplectic group.

– The intrinsic `ExtraSpecialBasis` returns a basis of an extraspecial or symplectic subgroup normalised by the group.

– The intrinsic `TensorInducedAction` returns the tensor induced action of an element.

– The intrinsic `Blocks` returns the blocks of imprimitivity of a matrix group.

– The intrinsic `ImprimitiveAction` gives the action of an element on a set of blocks of imprimitivity.

7.3 Finite Soluble Groups [HB 19]

Bug Fixes:

– A bug in the implementation of the collection algorithm has been fixed.

7.4 Databases of Groups [HB 24]

Changes:

– The database of primitive groups has been extended to degree 999 by Colva Roney-Dougal and Bill Unger. We believe the list to be complete. Numbering from Sims' list of degree up to 50 is changed. There are also added access functions, but the basic functions `PrimitiveGroup(d,n)` and `NumberOfPrimitiveGroups(d)` will work as before. A new function, which has been added, is `PrimitiveGroupIdentification`.

– The almost simple groups database now includes HS and $L(6,2)$. Derek Holt and Colva Roney-Dougal have also supplied routines so that the maximal subgroups and automorphism groups of the groups $Alt(d)$ ($d < 1000$), $L(2,p)$, $L(2,p^2)$, $L(2,p^3)$, $L(3,p)$, $L(3,8)$, $L(3,9)$, $S(4,p)$, and $U(3,p)$ (p a prime) may be found as if they were in the database.

Bug Fixes:

– A bug in the identification of groups in the small groups database has been fixed.

7.5 Finitely Presented Groups [HB 26]

Changes:

– A new method has been implemented for computing preimages of submodules of a $K[G]$-module M, defined by an elementary abelian section of a finitely presented group G. The new approach yields significantly reduced running times and presents an important improvement to the tools for constructing normal subgroups of finitely presented groups.

– The possibility of imposing time limits has been provided for the functions `LowIndexSubgroups` and `LowIndexSubgroupsProcess`. This gives the user increased control when employing these functions in combination with other strategies in the search for subgroups satisfying certain properties.

New features:

– A new function, `Homomorphisms`, computing equivalence classes of homomorphisms from a finitely presented group F to a permutation group G modulo a user specified subgroup of the automorphism group of G has been provided. The efficient implementation of the underlying backtrack search allows to compute quotients of order 10^8 or 10^9 for finitely presented groups with few generators in very reasonable time. An interactive version of this algorithm exists as well.

– Building on the above machinery, functions have been implemented to enumerate all epimorphisms of a finitely presented group onto any simple group having order less than 10^9. The main function is `SimpleQuotients` and, in addition, a process version is provided.

– The new function `IsPerfect`, testing whether a finitely presented group is perfect, has been provided.

– The function `AbelianQuotientInvariants` has been improved by the use of new sparse matrix techniques.

Bug fixes:

– A bug in simplifying presentations using Tietze transformations has been fixed. This problem affected various other functions making indirect use of Tietze transformations.

– A problem with the `sub<`$G|f$`>` constructor, creating a subgroup of G as point stabiliser of the transitive permutation representation f was fixed. It is now checked that f is transitive and invalid data is reported as runtime error.

7.6 Braid Groups [HB 29]

The implementation of braid groups in Magma has been revised completely for version V2.10. Working with both Artin's original presentation and the band generator presentation introduced by Birman, Ko and Lee is now fully supported and elements can be represented as words or in terms of simple elements as desired. Both presentations and different representations of elements can be used simultaneously.

New features:

– The following functions for controlling for a given braid group B the default presentation used for B, the element representation used for arithmetic operations with elements of B, and the print format for elements of B have been provided: `GetPresentation`, `SetPresentation`, `GetForceCFP`, `SetForceCFP`, `GetElementPrintFormat` and `SetElementPrintFormat`.

– For accessing representations of elements, the functions `WordToSequence`, `Infimum`, `Supremum`, `CanonicalLength` and `CanonicalFactorRepresentation` have been provided.

– Various normal forms of elements can be computed with the new functions `LeftNormalForm`, `RightNormalForm`, `LeftMixedCanonicalForm` and `RightMixedCanonicalForm`.

– The new functions `Cycle` and `Decycle` implement the cycling and decycling operations introduced by Garside.

– The new boolean predicates `IsSimple`, `IsSuperSummitRepresentative` and `IsConjugate` for elements have been introduced. Moreover, the partial orderings on the elements of a braid group can be accessed with the new operators `ge` and `le` and with the predicates `IsGE` and `IsLE`. The latter versions allow specification of a presentation.

– The functions `LeftGCD`, `RightGCD`, `LeftLCM` and `RightLCM`, providing the lattice operations for elements of a braid group, have been introduced.

– The set of positive conjugates of an element and the super summit set of an element can be computed with the new functions `PositiveConjugates` and `SuperSummitSet`, respectively. Alternatively, these sets can be computed using the interactive versions of these tools.

– The functions `SuperSummitInfimum`, `SuperSummitSupremum` and `SuperSummitCanonicalLength`, computing invariants of the conjugacy class of an element, have been provided.

– The symmetric representation, the integral Burau representation and the modular Burau representations of a braid group can be created using the new functions `SymmetricRepresentation` and `BurauRepresentation`.

Changes:

– Pseudo random elements of a braid group can now be created using each of the possible representations of elements. The relevant functions are `Random`, `RandomWord` and `RandomCFP`.

– Arithmetic operations now are by default performed using representation of the operands in terms of simple elements. In general, this yields much better performance.

Removals:

– The function `NormalFormWord` has been removed, as it was made obsolete by the more general representation used for elements and does not correspond to the new design of the category.

7.7 Groups Defined by Rewrite Systems [HB 30]

The 2001 version of the `kbmag` package of Derek Holt has been installed. The first Magma version of this package was released in V2.5 (July 1999) but it suffered from numerous problems. Since then Derek Holt revised his package and the revised edition was installed from scratch by Graham Matthews. The new installation works brilliantly – the various problems that crippled the V2.5 installation have all been overcome. The functionality is unchanged.

New features:

– A new installation of the entire program provides much greater reliability and efficiency.

7.8 Automatic Groups [HB 31]

The 2001 version of the `kbmag` package of Derek Holt has been installed. The first Magma version of this package was released in V2.5 (July 1999) but it suffered from numerous problems. Since then Derek Holt revised his package and the revised edition was installed from scratch by Graham Matthews. The new installation works brilliantly – the various problems that crippled the V2.5 installation have all been overcome. The functionality is unchanged.

New features:

– A new installation of the entire program provides much greater reliability and efficiency.

7.9 Subgroups of $PSL(2, R)$ [HB 33]

Bug fixes:

- Fixed printing of congruence subgroups given in terms of intersection of the basic congruence subgroups.
- Fixed testing of equivalence of matrices with respect to the action of some congruence group.
- Fixed testing of equivalence of points in the upper half plane under action of some congruence subgroup.
- Corrected type of labels attribute of the `SymFry` type.

New features:

- New function `FindWord` which expresses an element of a congruence subgroup in terms of the given list of generators for the group.
- Function `CongruenceSubgroup` now works properly with dirichlet characters allowed to be used in defining a congruence subgroup.
- For convenience, the signature product `SpcHypElt * RngElt` is now supported.

8 Basic Rings

8.1 Real and Complex Fields

New features:

- The extended reals (type `ExtRe`) have been introduced to provide a satisfactory common universe for `Infinity` and integers/rationals/reals. This allows sequences of valuations to be created even when some of the valuations are infinite.

Bug fixes:

- The calculation $\infty^{-\infty}$ no longer crashes.

8.2 Polynomial Rings [HB 38]

New features:

- Factorization of polynomials over arbitrary algebraic function fields of small characteristic is now supported for the first time (see Commutative algebra below).
- New fast modular algorithm for GCD of polynomials over algebraic function fields and quotient rings which are fields (both univariate and multivariate).
- The polynomial quotient rings $K[x]/\langle f \rangle$, where K is a field and f may be reducible, are now fully supported as general Euclidean rings.

9 Linear Algebra and Module Theory

9.1 Matrices [HB 42]

New features:

- Linear algebra over the polynomial quotient rings $K[x]/\langle f \rangle$, where K is a field and f may be reducible, is now fully supported.
- New functions and procedures `RemoveRow`, `RemoveColumn` and `RemoveRowColumn` to remove a row and/or a column from a matrix.

10 Commutative Algebra

10.1 Ideal Theory and Gröbner Bases and Affine Algebras [HB 47]

A new algorithm by Allan Steel (to be published) solves a suite of fundamental problems associated with arbitrary algebraic function fields. These fields may be constructed as chains of algebraic extensions (using polynomial quotient rings, affine algebras, or standard algebraic function fields) and transcendental extensions (using rational function fields). The fields may have any characteristic and the algebraic extensions may be inseparable (which can happen in small characteristic).

The problems now solved include decomposition of multivariate ideals and factorization of polynomials over such fields.

New features:

- Primary decomposition and radical computation is now fully supported for ideals over arbitrary algebraic function fields of any characteristic (including non-perfect fields).
- Primary decomposition and radical computation is now supported for ideals over finite fields of arbitrary dimension.
- Factorization of polynomials is now fully supported over arbitrary algebraic function fields of any characteristic.
- New fast modular algorithm for GCD of polynomials over arbitrary algebraic function fields.

11 Extensions of Rings

11.1 Algebraic Number Fields [HB 50]

Changes:

- The `ideal<>` constructor now checks that the input defines an ideal.

– Factorization of ideals has been improved by storing a product representation on appropriate ideals and using a coprime factorization algorithm on this partial factorization.

– Changed `SplittingField` for polynomials over **Q** to allow it to return a tower of fields rather than a simple optimized representation.

– Improved performance of `OptimizedRepresentation` and `LLL` by using a different LLL-version. Furthermore, if this function fails to find a better representation, the old representation is returned.

– A rewrite of the code computing completions of number fields and prime ideals using the new local rings and allowing for precision change.

New Features:

– Homomorphisms from orders of algebraic number fields can be created by giving the images of the basis elements (not just the primitive element).

– The greatest common divisor of two ideals can be obtained.

Bug Fixes:

– Bug fix in `IsSubfield`, in the pre-image code for homomorphisms between orders and number fields, in the pre-image code for the unit-map.

11.2 Quadratic Fields [HB 52]

New Features:

– `Conductor` now returns the ramified real places of the field if asked for.

Bug Fixes:

– A bug involving preimages of ideals under the class group map has been fixed.

– Any number of problems with the conversion of quadratic forms to and from ideals have been fixed.

11.3 Cyclotomic Fields [HB 53]

New Features:

– `Conductor` now returns the ramified real places of the field if asked for.

– Some generic functions to check an element for being a root of unity, returning the torsion subgroup of some rings.

11.4 Abelian Extensions [HB 54]

New Features:

– Better support for the computation of automorphism groups of class fields.

– Computation of the second cohomology group.

11.5 Algebraic Function Fields [HB 57]

The functionality of extensions of algebraic function fields has been extended to almost match that of ordinary algebraic function fields. The functionality of the orders of algebraic function fields has been extended to almost match the functionality of the orders of algebraic number fields.

Removals and Changes:

- The tuples returned by `DecompositionType` are now of the form $\langle f, e \rangle$ where f is the inertia degree and e is the ramification degree of the corresponding ideal or place in the decomposition. It is no longer always necessary to compute the decomposition to obtain this information.
- Factorization of ideals has been improved by storing a product representation on appropriate ideals and using a coprime factorization algorithm on this partial factorization.
- `Eltseq` of an element of an order is now over the field of fractions of the coefficient ring.
- Printing of divisors is no longer only done as a linear combination of places. In some cases this was too expensive.

New Features:

- Experimental introduction of class field theory for function fields. The new facilities include functions for `RayClassGroup` computation, defining equations of class fields and computation of norm groups.
- Places and Divisors of extensions of algebraic function fields can be created. They have the full functionality of the extensions of rational function fields except for the functions mentioned below.
- Differentials of extensions of algebraic function fields can be created. They have the full functionality of the extensions of rational function fields except for the functions mentioned below.
- All functions for algebraic function fields can be called on extensions of function fields with the exception of those involving series rings, galois groups and subfields. These functions are `Reduce`, `Expand`, `Residue`, `GaloisGroup`, `Subfields`, `Automorphisms`, `IsSubfield` and `IsIsomorphic`.
- The functions `Modexp` and `Modinv` can now be called on elements of orders of all algebraic function fields.
- `RationalFunction` can now be returned over a coefficient ring or field given as a second argument.
- The following intrinsics have been preexisting for orders of number fields and have recently been added for orders of algebraic function fields :
  - `!!` for ideals,
  - `+` for orders,
  - `AbsoluteOrder`, `AbsoluteDiscriminant`, `Basis` of an order of an algebraic function field over a ring given as a second argument and `BasisMatrix` of an order,
  - `Different` for orders, ideals and elements,
  - `Index`, `EquationOrder`, `IsAbsoluteOrder` for orders,
  - `IsInert`, `IsRamified`, `IsSplit`, `IsTamelyRamified`, `IsTotallyRamified`, `IsTotallySplit`, `IsUnramified`, `IsWildlyRamified` for ideals and orders,
  - creating an order from a basis,

  - `PrimitiveElement`, `Simplify`, `SubOrder`, `pMaximalOrder`, `pRadical` for orders,
  - `ColonIdeal`, `meet` of an ideal with a ring, `IsPower` and `Root` functions for ideals.

- The function `RationalExtensionRepresentation` now allows even relative extensions to be expressed as a direct extension of the rational function field. Expressing an algebraic function field as an extension of one of its coefficient fields can be accomplished using `UnderlyingField`.
- `Completion`s of non relative function fields and their orders at ideals of degree one over the constant field (places of degree 1) can now be taken.
- `ConstantFieldExtension` extends a function field by extending the constant field.
- The following intrinsics were preexisting for extensions of rational function fields and are now available for extensions of algebraic function fields :
  - `Differential`, `Differentiation`, `SeparatingElement`, `DifferentiationSequence`,
  - `DifferentialSpace`, `DifferentialBasis`,
  - `SpaceOfDifferentialsFirstKind` (`SpaceOfHolomorphicDifferentials`),
  - `BasisOfDifferentialsFirstKind` (`BasisOfHolomorphicDifferentials`),
  - `CartierRepresentation`, `HasseWittInvariant`, `ClassGroupPRank`,
  - `Identity` of a differential space, `IsCanonical`,
  - `Divisor`, `PrincipalDivisor`,
  - `RamificationDivisor` , `WronskianOrders`, `GapNumbers`, `WeierstrassPlaces`,
  - `DifferentDivisor`, `CanonicalDivisor`, `ComplementaryDivisor`,
  - `Identity` of a divisor group,
  - `Degree`, `Minimum`, `IsConstant`, `IsSeparating`, `Zeros`, `Poles`,
  - `SerreBound`, `IharaBound`,
  - `NumberOfPlacesOfDegreeOneBound`, `NumberOfPlacesOfDegreeOne`,
  - `NumberOfPlaces`,
  - `HasPlace`, `Places`, `RandomPlace`, `DivisorOfDegreeOne`,
  - `ClassGroupGenerationBound`, `ClassNumberApproximation`,
  - `LPolynomial`, `Genus`, `ExactConstantField`,
  - `DegreeOfExactConstantField` (`DimensionOfExactConstantField`) .

  Some of these functions compute the rational extension representation of the relative field and perform the computations on this. Others can do the computations on the relative field directly.
- By means of a `Type` parameter to `FunctionField` rational function fields can be created as algebraic. Extensions of such fields will not be considered relative but still as rational extensions.
- The `Trace` and `Norm` of an element can be computed over a coefficient ring or field given as a second argument.
- The ideal constructor has been expanded to allow an ideal to be created from a basis and a denominator. The result is checked to see whether the input did indeed define a true ideal.
- Homomorphisms from orders of function fields can be created by giving the images of the basis elements.

– An extra argument has been added to `Zeros`, `Poles` and `CommonZeros`. The first argument can now be the field the places to be returned should be of.

Bug Fixes:

– A bug in `Expand` has been fixed.

11.6 Newton Polygons [HB 60]

New Features:

– It is now possible to create a newton polygon given a polynomial over the integers or rationals, a number field or an order of such or an algebraic function field or an order of such and either a prime number or a prime ideal, respectively.

– For any newton polygon the function `Slopes` will return the slopes of all the faces of the newton polygon.

11.7 p-adic Rings and Fields [HB 61]

The p-adic rings and fields have been completely rewritten. The new implementation is substantially faster, to the extent that Magma implementations of point-counting algorithms such as the AGM are now within an order of magnitude of optimized C implementations. The new p-adics are built around a fixed precision model, where all elements are of the same precision; while it is still possible to construct free precision rings (where each element can have a different precision), these are internally mapped into the fixed precision structures. Thus, the user can sacrifice automated precision management for greater speed.

New Features:

– There are now four ring/field types: `RngPadRes` and `RngPadResExt`, representing fixed-precision rings, and `RngPad` and `FldPad`, representing free precision rings and fields, respectively.

– The number of constructors for the p-adics has been cut substantially. In particular, the allowed parameters of the `ext` constructor have been reduced and all `LocalRing` and `LocalField` constructors have been removed. Similarly, the allowed parameters of the `elt` constructor have also changed.

– In free precision rings, exact elements are no longer supported (that is, the rationals are no longer directly embedded into the p-adics). For instance, in the previous implementation, the element 1 could be represented exactly in a p-adic ring; now, it can only be represented up to some finite precision.

– Extensions may now be constructed in chains of arbitrary height. This means that intrinsics such as `Degree`, `RamificationDegree`, `InertiaDegree`, `Eltseq`, `Trace`, `Norm`, and `MinimalPolynomial` are now overloaded to take a second argument, which is the base ring or field with respect to which the calculated is performed.

– The intrinsics `EisensteinPolynomial`, `InertialPolynomial`, and `InertiaRing` have been removed.

– The attribute `SeriesPrinting` has been removed.

– The generators of a local ring or field L have now changed. In previous versions of magma, $L.1$ referred to the uniformizing element, and $L.2$ referred to the inertial element. Now, a local ring or field L has only one generator, $L.1$, which refers to the element whose powers generate a basis of L as a vector space over its base ring or field. The uniformizing element can still be obtained using `UniformizingElement`.

– Equality of elements in a free precision ring is now banned, due to the fact that there are several possible definitions of equality in an inexact ring.

– The intrinsics `LocseqInert`, `InertseqpAdic`, and `Locseq` have been removed, as their functionality is now available in `Eltseq`.

– In a free precision p-adic ring R, division by / now returns an element in its field of fractions, whereas `div` returns an element in R (and hence may fail). An intrinsic `IsExactlyDivisible` has been added which allows the user to check whether `div` will succeed.

– The Hensel lifting of polynomial factorizations has been improved, and now the intrinsic `HenselLift` can take a sequence of factors, instead of just two factors.

– Intrinsics `InverseSqrt`, `InverseSquareRoot` and `InverseRoot` have been added, which perform efficient computation of $x^{-1/n}$ for some local ring or field unit x.

– The intrinsic `HasPRoot` has been removed.

12 Representation Theory

12.1 $K[G]$-Modules and Group Representations [HB 73]

A package is provided which attempts to construct all (absolutely) irreducible representations of a finite group over a given finite field. For soluble groups the Brueckner adaptation of the Glasby–Howlett method is used. For non–soluble groups the irreducibles are obtained by chopping up representations using the Meataxe.

13 Lie Theory

13.1 Root Systems and Root Data (New) [HB 79–80]

We now distinguish between root systems, defined over real vector spaces, and root data, defined over integer lattices. The former are designed for the study of reflection groups and Coxeter groups; the later are designed for Lie algebras and groups of Lie type. A small number of commands that worked previously will no longer work: for example `RootDatum("H3")` must be changed to `RootSystem("H3")` since the root system of type H_3 cannot be defined in an integer lattice.

13.2 Coxeter Systems (New) [HB 82]

Cartan matrices have been expanded to include infinite Coxeter systems. Cartan names have been extended to include affine Coxeter systems. It is now possible to compute with

Coxeter matrices, Coxeter graphs, and Dynkin digraphs. We can identify and construct affine and hyperbolic Coxeter systems.

13.3 Coxeter Groups (New) [HB 83]

A new category of Coxeter groups has been implemented. It is called `GrpFPCox` and it represents elements as words in the standard Coxeter group presentation. Elements are automatically put in normal form using an efficient new algorithm designed and implemented by Robert Howlett.

13.4 Coxeter Groups as Permutation Groups [HB 84]

The old category `GrpCox` of finite Coxeter groups as permutations on the roots has been renamed `GrpPermCox`. Such a group can be created from either a root system or a root datum.

13.5 Reflection Groups [HB 85]

We now have some basic functions for creating and identifying reflection groups over an arbitrary field. The functionality for real reflection groups has been expanded.

13.6 Groups of Lie Type [HB 86]

Changes:

- A much faster algorithm has been implemented for multiplication in groups of Lie type. This requires some preprocessing, so it may take longer to create such a group. The previous method is still available as an optional parameter. We can now compute the multiplicative Jordan decomposition of an element.

New features:

- The standard automorphisms of these groups are available (i.e. inner, diagonal, diagram, and field automorphisms). In many cases these give the full automorphism group.
- The highest weight representations of a group of Lie type can be computed—this gives all rational representations over the base field. The computation of the inverse of a representation of a group of Lie type over its base field can be achieved using `GeneralisedRowReduction`. We have a new function to compute the adjoint representation and a faster method for computing the standard representation.

14 Algebraic Geometry

14.1 Schemes [HB 87]

Removals and Changes:

- Improved checking for inverses when the domain is not an ambient.
- `AlternateDefiningPolynomials` has been renamed to `AllDefiningPolynomials` which includes the original definition of the map.

New Features:

- More checks for well–definedness when multiple sets of defining equations are given.
- Multiple definitions for the inverse of a map can be given. Such definitions can be retrieved using `AllInverseDefiningPolynomials`.
- Inverses of projective closure maps are set on creation.
- The function `Difference` will remove from a scheme the intersection of that scheme with a second argument (taking into account multiplicities) and return the closure of this.
- The functions `IntegralSplit`, `Numerator` and `Denominator` have been provided for working with function field elements and associated schemes, as well as `AlgebraicFunction` and `Restriction`.
- Projective maps can be created using function field elements by the function `ProjectiveMap`.
- A test for the birationality of scheme maps is provided by `IsInvertible`. A birational inverse is returned if it exists.
- `GenericPoint` returns a point in the pointset of the function field of the scheme.
- Compositions of scheme maps can be multiplied out fully (including multiple definitions) using `Expand`. A map between schemes can be `Extend`ed to be defined on the whole domain. For a map with multiple definitions `Prune` will remove any unnecessary definitions.
- It is now possible to test whether a pointset over a local field is empty using `IsEmpty`. Further one can use `IsLocallySolvable` to test whether a scheme over a number field has a local point and `LiftPoint` to increase the precision of a point.
- One can check whether a function field can be computed for a scheme using `HasFunctionField`.

Bug Fixes:

- Greater strictness on base rings for which Gröbner basis algorithms are not available. One can no longer retrieve the ideal of a scheme over a ring without these algorithms or do anything which requires such information.

14.2 Algebraic Curves [HB 88]

New Features:

- The functions `FormalPoint` and `EvaluateByPowerSeries` use series rings to work with points.
- `Degree` and `RamificationDivisor` of maps between schemes. Also the `Pullback` and `Pushforward` function field elements, places and divisors and curve places and divisors along a map between schemes.

14.2.1 Function Fields and Divisors on Curves

New Features:

- Some functions for curves relating to function fields have been added. These are `NumberOfPlaces`, `CartierRepresentation`, `FieldOfGeometricIrreducibility`, `CommonZeros`, `Reduction`, `GCD` and `LCM` of divisors, `ResidueClassField` and `Lift` at a place, `IsAbsolutelyIrreducible`, `ClassNumber` and `GlobalUnitGroup`.

14.3 Elliptic Curves [HB 91]

Removals and Changes:

- Elliptic curves can now be created from general schemes instead of only hyperelliptic curves. Similarly, elliptic curves can now be created from a general scheme and a point rather than a curve and a point. Only one map is returned in each case but it is birational.
- Redesigned routines to put a curve of genus 1 into Weierstrass form. Many commonly occurring cases are recognised and handled more efficiently. The system will automatically choose the most appropriate method.
- Elliptic curves may now be created by specifying the polynomials, similar to the hyperelliptic case. To resolve a naming conflict the former intrinsic `EllipticCurve(j)` has been changed to `EllipticCurveFromjInvariant(j)`.

New features:

- Elliptic curves can be created from a curve of genus 1 and a place of degree 1.
- `HyperellipticPolynomials` of an elliptic curve can be retrieved as for hyperelliptic curves.
- Elliptic curves can be created from the corresponding hyperelliptic polynomials.
- The `DualIsogeny` of an isogeny of an elliptic curve can be obtained as well as the `TwoIsogeny` of a point of an elliptic curve.

14.3.1 Elliptic Curves over the Rational Field

New features:

- The improved bound (due to Samir Siksek) on the difference between the naive and canonical heights of a point has been implemented. This bound is now used internally for the group closure operation and is accessible via `SiksekBound`.
- The Cremona database of elliptic curves now includes all curves of conductor up to 20 000.

Bug fixes:

- A long-standing bug in the 2-descent for certain curves has been fixed. Up to conductor 10 000 there were 14 curves affected, the smallest being 903B3.

14.3.2 Elliptic Curves over an Algebraic Number Field

New features:

- `IsIntegralModel` and `IntegralModel` for elliptic curves over number fields.
- `IsIntegralModel` for elliptic curves at a given prime ideal of a number field.
- Several functions have been provided which work with elliptic curves over number fields. They are the `Reduction` of a curve at an ideal, and the `TorsionBound`, `pPowerTorsion` and `TorsionSubgroup` of a curve.
- For isogenies between elliptic curves defined over number fields, `SelmerGroup` has been implemented. Accompanying this are the functions `IsogenyMu` and `RankBound`.
- The functions `AbsoluteAlgebra`, `pSelmerGroup` and `LocalTwoSelmerMap` have been provided for working with etale algebras.

14.4 Hyperelliptic Curves [HB 92]

Removals and Changes:

- The input arguments of the `HyperellipticCurve` intrinsic that creates an hyperelliptic curve have been swapped for general consistency. If $f(x)$ and $h(x)$ are two univariate polynomials then `HyperellipticCurve(f, h)` returns the curve defined by the equation $y^2 + h(x)y = f(x)$. (Formerly the syntax was `HyperellipticCurve(h, f)`.)
- Again for consistency, the input polynomials have been swapped in the intrinsics `IgusaInvariants`, `JInvariants`, `ScaledIgusaInvariants` and `IgusaClebschInvariants`. A typical call to compute the Igusa J-invariants of the curve $y^2 + h(x)y = f(x)$ would be `IgusaInvariants(f, h)`.

14.5 Numerical Graded Rings (New) [HB 99]

The K3 database in Magma contains the 391 examples of K3 surfaces in the known lists (excluding various standard degenerations). These were created by assembling a lot of data that should occur on these surfaces, feeding it into the Riemann–Roch formula to get a Hilbert series, and then attempting to describe a plausible K3 surface embedded in weighted projective space that had that Hilbert series. The ideas are described in the paper

S. Altınok, G. Brown, M. Reid, Fano 3-folds, K3 surfaces and graded rings, Contemp. Math. **314**, 2002, pp.25–61.

The same ideas have now been used to make much bigger lists of K3 surfaces, that include all possible configurations of input data. The price is that the new lists (one for each integer ≥ -1, and each of size between 4000 and 6500) contain many very complicated surfaces that need to be embedded in very large weighted projective spaces, that is, in very high codimension. However, the construction of the lists does include some tricks for making these candidate descriptions reasonable. The main one is to recognise that the surfaces are related by projections. In the easiest cases, so–called Type I projections, this is simply the elimination of a variable from the space. Thus if one has information

about the projective space after projection, it is easy to inherit that before projection by reintroducing the variable. This is described in the forthcoming paper

G. Brown, Datagraphs in algebraic geometry, to appear in Proceedings of SNSC01, F. Winkler (ed), RISC-Linz, 2001.

Notice that these routines do not describe explicit graded rings: they do not explain how to write the equations, but only say which weighted variables should be used in the equations. (The Hilbert series does include more information, but still much less than would dictate explicit equations, in general.) The process of recovering equations through projections is called 'unprojection', and that is still some way off being implemented.

The new packages still generate only lists of K3 surfaces. But prototype versions have been used experimentally in PhD theses over the past two years to generate Fano 3-folds and Calabi–Yau 3-folds. A graduate project also made lists of subcanonical curves using similar methods. These will be incorporated into Magma in due course.

15 Incidence Structures

15.1 Graphs [HB 102]

Changes:

- The semantics for graph equality have been made consistent with the fact that graphs may have different labellings and/or different supports. If G and H are two graphs, `G eq H` if and only if G and H are structurally equal, if they have the same support and if they have the same vertex and edge labelling.
- If G is a graph with support S and vertex and/or edge labels L, the function `StandardGraph` returns a graph H with same vertex and edge set as G, with vertex and/or edge labels L, but with standard support $1, \ldots,$ `Order(G)`.
- The interface allowing for vertex and edge addition or deletion has been modified so to keep in line with the Magma philosophy. In some instances it was possible to specify a vertex or an edge by means of integers in the appropriate range. Now vertices and edges will have to be specified by a standard Magma object of type `GrphVert` and `GrphEdge` respectively.

  The old interface has been retained for backward compatibility but is no longer documented.
- Similarly, the signature for the function `AssignLabels` has been modified so that vertices and edges are now only specified by objects of type `GrphVert` and `GrphEdge` respectively. Again, previous signatures that referred to vertices and edges by means of integers have been retained for backward compatibility.
- Our definition of a eulerian graph or digraph has been modified in order to conform with the currently accepted definition. An undirected graph is eulerian if and only if all vertices have even degree. A directed graph is eulerian if and only if each vertex has same in- and out-degree. That is, a graph is eulerian if and only if it has a eulerian circuit. Note that we do not require the graph to be connected.

New Features:

– It is now possible to create a `NullGraph` in Magma, that is, a graph with no vertices. It is also possible to test whether a graph is null, using `IsNullGraph`.

– The function `Conversion` is now available for digraphs. If G is a digraph, `Conversion(G)` returns a digraph with the same vertex set as G and whose edges are the edges of G with the direction reversed.

– If G is a graph with support S and vertex and/or edge labels L, the function `UnlabelledGraph` returns a graph H with same vertex and edge set as G, with support S but without vertex and/or edge labelling.

– The functions `IsVertexLabelled` and `IsEdgeLabelled` applied to a graph G determine if the vertices or edges of G are labelled.

– The classical linear-time 3-connectivity algorithm from Hopcroft and Tarjan has been implemented, with corrections of our own and from Gutwenger and Mutzel.

  The function `IsTriconnected` determines if a graph is 3-connected. The function `Splitcomponents` splits a graph G into components that can be easily reconstructed as 3-connected graphs. Finally, the function `SeparationVertices` finds the cut vertices and/or the separator pairs of a graph.

– A general connectivity machinery has been put in place for graphs and digraphs. It allows to test for vertex and edge connectivity (`IsKVertexConnected` and `IsKEdgeConnected`) and to compute the vertex and edge connectivity of a graph (`VertexConnectivity` and `EdgeConnectivity`).

  The underlying algorithms are flow-based algorithms; we have implemented two of them: the Dinic algorithm and a push-relabel method. For more details on these algorithms see

  Subsection 15.2 below. The push-relabel method is usually (much) faster than Dinic, expect possibly in very sparse graphs with a small connectivity number. This is why the former algorithm is the default algorithm used in all connectivity computations; users may choose Dinic by setting an optional parameter.

– A natural extension of the flow-based machinery is the algorithm that computes a maximum matching for a bipartite graph G: the function `MaximumMatching` returns a maximum matching of G as a list of edges of G.

Bug Fixes:

– Several problems related to the deletion of a graph or of its vertex or edge set have been solved.

15.2 Networks (New) [HB 103]

Networks are a new Magma category with type `GrphNet`. Networks are defined as digraphs whose edges are associated with a capacity; there may be parallel edges and loops. The fundamental network flow problem is the minimum cost flow problem, that is, determining a maximum flow at minimum cost from a specified source to a specified sink. So far we have concentrated on the maximum flow problem: Finding the maximum flow that can be pushed from a source to a sink subject to the capacity constraints of the edges in the network.

Two flow-based algorithms have been implemented: The Dinic algorithm with added heuristics from B. McKay, and the push-relabel method with heuristics mainly due to Cherkassky and Goldberg *et al* . The latter usually out-performs Dinic, however Dinics

might be best for very sparse networks with a very small flow and whose edge capacities are small.

The Dinic algorithm consists of two phases. The first phase constructs a layered network which consists of the "useful" edges of the network: Those edges through which a flow can be pushed. The second phase finds a maximal flow by constructing paths from the source to the sink. These two phases are repeated until no new layered network can be constructed due to the fact that there is no path of "useful" edges from the source to the sink. The flow is then maximum.

The generic push-relabel algorithm constructs a flow by pushing the maximum possible flow out of the source into its neighbours, and then pushing the excess flow at those vertices into their own neighbours. This is repeated until all vertices of the network except the source and the sink have a excess flow of zero. Of course this means that some flow might have been pushed back into the source. Specific heuristics for zero-one networks (i.e. whose edge capacities are either zero or one) and for general networks result in a very efficient implementation (when compared to Dinic). Thus the push-relabel method is the default algorithm when computing a maximum flow, although users may choose Dinic if they so desire.

Since networks allow parallel edges, they will be represented as an adjacency list: This is the sparse graph representation which was first introduced in the previous Magma release V2.9. The possible existence of parallel edges actually introduces the most significant distinction between a simple graph and a network (from the users point of view): Edges in a network will require a unique identification, an identification which is obviously not a necessity in a graph where multiple edges are disallowed.

The scheme chosen to identify edges in a network is to associate it with its index in the graph's adjacency list. This is possible since our implementation guarantees that edge indices remain constant during the graph's lifetime, even though vertices and edges may have been added or removed.

In short, most basic access functions and predicates that apply to simple graphs are also available for networks. They are not listed here, a full description can be found in the Magma manual. The following list highlights the most significant features of networks in Magma. Please be aware that for the time being it is not possible to label vertices and edges in a network. This feature should be available in the next release.

New Features:

- The function `EdgeIndices(u, v)` returns the indices in the adjacency list of the graph of all the directed edges from u to v. The function `EdgeMultiplicity(u, v)` returns the multiplicity of the edge from u to v.
- An edge e of type `GrphEdge` in a network N is uniquely identified by `EndVertices(e)` and `Index(e)` in the adjacency list of N.
- If E is the edge set of a network N, then $E.i$ designates the edge in N with index i in N's adjacency list. Note that those semantics are different from the ones applying in the simple graph case.

For if F is the edge set of a simple graph G, then $F.i$ designates the ith edge in the list of edges of G as given by `Edges(G)`. It is likely that from the next Magma release onwards the latter semantics will be changed so that $F.i$ always designates the edge with index i in the adjacency list of the graph, be it simple or not.

- If e of type `GrphEdge` is an edge of a network N from vertex u to vertex v, then `Capacity(e)` returns the capacity associated with e while `Capacity(u, v)` returns the *total* capacity from vertex u to vertex v.
- Incremental construction of networks by addition or deletion of vertices and edges works in a similar fashion as in the simple digraph case, the only difference being the possibility of creating parallel edges and the necessity to associate each edge with a capacity.
- The semantics for network equality work as follows: Let N and M be two networks. Then `N eq M` if and only if `Order(N) eq Order(M)`, if N and M have same support and the *total* capacity from vertex u to vertex v in N is equal to the *total* capacity from vertex u to vertex v in M for all ordered pairs of vertices. That is, it is possible that N equals M without it being necessary that they have the same number of edges from u to v for any given two vertices u and v.
- The semantics for `IsSubgraph(N, M)` are similar. M is a subgraph of N if and only if the *total* capacity from vertex u to vertex v in M is no greater than the total capacity from vertex u to vertex v in N.
- Given two vertices s and t in a network N, `MaximumFlow(s, t)` returns the value of the maximum flow in N from s to t. Given two sequences S and T of vertices in a network N, `MaximumFlow(S, T)` returns the value of the maximum flow in N from the vertices in S to the vertices in T.
- Let e of type `GrphEdge` be an edge in a network N from vertex u to vertex v. After a maximum flow computation in N, `Flow(e)` returns the flow carried by the edge e, while `Flow(u, v)` returns the *total* flow vertex u to vertex v. If no maximum flow computation has been performed on N then both functions return zero.
- Given two vertices s and t in a network N, `MinimumCut(s, t)` returns the minimum cut in N corresponding to the maximum flow from s to t in N. Similarly, given two sequences S and T of vertices in a network N, `MinimumCut(S, T)` returns the minimum cut corresponding to the maximum flow from the vertices in S to the vertices in T.

15.3 Incidence Structures and Designs [HB 104]

New Features:

- Now a `pRank` function is available for incidence structures (it was formerly restricted to finite planes only).

Changes:

- The existing brute force t-balance test is quite expensive, especially for large t. We have implemented a new test, also using brute force but that performs much better. However it has a drawback as it might run out of memory space. For this reason the choice is left to the user as to which implementation to use for the t-balance test.

  The functions where this choice is made possible are `IsDesign`, `IsSteiner`, `IsBalanced`, `Design` as well as the `Design<>` constructor. The choice is made using the optional parameter `Al` which can be set to the values

- `"NoOrbits"` This is the existing implementation of the brute force test and is the default setting.
- `"Orbits"` This is the existing implementation of a test that uses the orbits of t-sets under the automorphism group of the incidence structure under consideration. This is much faster than `"NoOrbits"` for some cases, but slower for others.
- `"FastBalanceTest"` This is the new implementation of the brute force test. Its usage is recommended as it significantly outperforms the two previous tests. These later tests should only be used in the case where using `"FastBalanceTest"` would not be possible due to shortage of memory space. Typically this would happen for large t and large incidence structures.

– Testing whether an incidence structure is a near linear space has been significantly improved by the implementation of a specialised and very fast 2-balance test.

Bug Fixes:

– A bug in `IsHadamardEquivalent` that caused the function to give a wrong answer has been fixed.

– Bugs in `AutomorphismGroup` and `IsIsomorphic` for designs that caused these functions to hang have been fixed.

15.4 Finite Planes [HB 105]

Changes:

– Testing whether an incidence structure is a projective or affine plane has been rewritten using a different set of axioms and implementing a specialised and very fast 2-balance test. This results in a significant performance improvement and affects all functions creating planes or testing if an incidence structure is a plane.

Bug Fixes:

– Bugs in `AutomorphismGroup` and `IsIsomorphic` for planes that caused these functions to hang have been fixed.

16 Coding Theory

16.1 Linear Codes over Finite Fields [HB 107]

Changes:

– Verbose output has for the best known linear codes, with flag `BestCode`, has been re-written to be more user friendly. It now outputs construction steps in a point-wise fashion, rather than using indenting. The new output is especially helpful for complex code constructions.

New Features:

– A new database for codes over $GF(4)$ contains constructions of best codes up to length 100. The database is over 98% complete, the first missing code appearing at length 92. Many of the codes constructed in this database are vast improvements on the previously known bounds for best codes over $GF(4)$.

16.2 Linear Codes over Finite Rings [HB 108]

Changes:

– Codes over integer residue rings are now classified by the total number of codewords they contain. Previously they were classified by the number of generators, which was not invariant for equivalent codes.

New Features:

– Magma now supports error correcting codes over Galois rings. Constructions include `CyclicCode`, as well as facilities to factorize $x^n - 1$. Important structural calculations are available such as the `CompleteWeightEnumerator` and the Hamming `WeightDistribution`.

– Magma now has special functionality for codes over $Z4$, the ring of integers modulo 4. A host of specific constructions include `KerdockCode`, `PreparataCode`, `ReedMullerCode`, `GoethalsCode`, and more. The optimized calculations of `LeeWeightDistribution` and `EuclideanWeightDistribution` are also included. Derived binary codes can be obtained, including those from the `GrayMap`.