A Database for Number Fields

Mario Daberkow[1] and Andreas Weber*[2]

[1] Technische Universität Berlin
Fachbereich Mathematik MA 8-1
10623 Berlin, Germany
E-mail: daberkow@math.tu-berlin.de
[2] Department of Computer Science
Cornell University
Ithaca, NY 14853, U. S. A.
E-mail: aweber@cs.cornell.edu

Abstract. We describe a database for number fields that has been integrated into the algebraic number theory system Kant. The database gives efficient access to the tables of number fields that have been computed during the last years and is easily extended.
A set of functions that are specific for a number field database has been integrated into the user interface Kash of Kant. The user has thus the possibility to create queries which involve special functions on number fields provided by Kant.

1 Introduction

In the last years several extensive lists of number fields were calculated, containing data for hundreds of thousands of number fields (see e.g. [22, 2, 11, 3], or [5, App. B] for other references). These lists have been generated with the help of programs for algebraic number theory such as Kant [7, 15, 6] and Pari [1] and most of this data is available in formats that can be read by those programs. However, no mechanisms had been implemented that allow an efficient access to the data.

In this paper we describe a database for number fields that has been integrated into Kant and that gives efficient access to the computed number fields. The database has been designed to use one of the available relational database management systems that support SQL [8, 13] as a query language. It can thus use the technologies that have been developed for relational databases to handle large amounts of data. A set of functions that are specific to a number field database have been integrated into the user interface Kash [15] of Kant. Thus the user can create queries that involve special functions on number fields provided by Kant.

For the connection to the database management system that stores the data we use several layers of functionality and we utilize standards as often as possible. It is thus possible to change the underlying database management system without too much effort, should such an undertaking be necessary.

* Supported by the *Deutsche Forschungsgemeinschaft*.

The organization of the paper is as follows. The basic design considerations for the database are described in Sec. 2. The layers of functionality we have used are given in Sec. 3 A discussion of some special topics that arise in connection with a number field database is contained in Sec. 4.

In Sec. 5 we give some examples of the use of the database. Queries that can be handled by the database system alone are described in Sec. 5.1. However, the examples given in Sec. 5.2 involve the combination of the database management system and the number theory system in an essential way. This section exemplifies that the combination of the involved systems is more powerful than the "sum of its parts".

2 Design of the Database

Several partially contradicting goals had to be balanced when designing the database.

- The database has to allow efficient access to hundreds of thousands or even millions of number fields.
- The database should support access for all platforms supported by Kash.
- The format of the data in the database should be independent of the platform used and possible changes to the binary representation of number fields in the Kant kernel.
- Since Kash is available for academic institutions free of charge, the database system that is used to store number field data should be available under the same conditions.
- The database has to be extensible.

For these reasons we have chosen to use a database management system (DBMS) that supports the SQL standard [8, 13]. By utilizing a standard it is possible to support several databases — ones that are free of charge for academic institutions and also highly optimized ones if needed by users.

2.1 Currently Used Tables

Currently the database contains tables in a form roughly corresponding to the file output format for orders in Kash [15, 6].[3]

Within these tables we use the generating polynomial as a key. This is possible because two non-isomorphic number fields cannot have the same generating polynomial. However, two isomorphic number fields may have different generating polynomials, see the discussion given in Sec 4.

[3] In order to speed up common queries we store the full signature of the number fields and not only its real part. We also store the regulator both with the full given precision but also as a floating-point number of the database, because this enables fast queries which use a bound on the regulator as a criterion. These additions cause only a negligible storage overhead.

If we want to ensure that we do not multiply store information of isomorphic number fields, we have to use some functionality that is provided by the surrounding Kash system, cf. Sec. 4.

3 Layering the Database Functions

3.1 System Independent Connection Functions

Although the query language is standardized by SQL, the C-APIs[4] used to access the database are not. Some databases support standards also in this respect like Microsoft's "Open Database Connectivity" (ODBC) [20], but most of the databases available under UNIX do not. However, the functionality that is needed is similar for all databases. It is necessary that there are APIs for

- opening a connection to the DBMS;
- sending SQL queries to the DBMS;
- retrieving the result of a query from the DBMS.

Using several layers of functionality it is thus possible to introduce a layer of connection functions that is independent of the used system and through which all of the access of the higher layers to the database management system can go through.

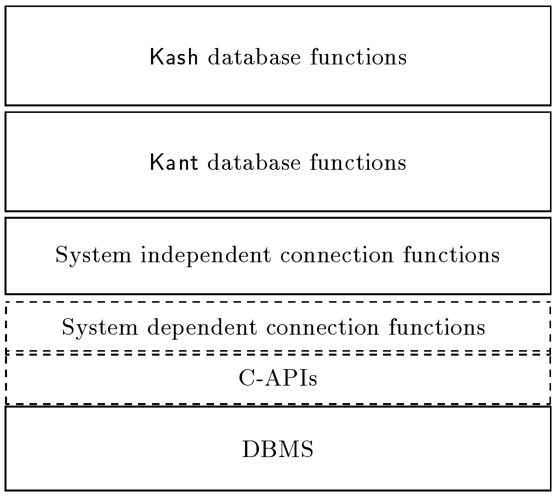

Fig. 1. Layers of the number field database

[4] We use the usual acronym API for *application programming interface*. Almost all database management systems provide APIs for the C programming language [16].

The use of the layer "system dependent connection functions" are similar in spirit to the "Z classes" of D. Fuqua [12][5] or of DBI [4].

In the current implementation we have provided system dependent connection functions for the shareware database mSQL [14], which is available free of charge for academic institutions. Extensions of this layer to support other database systems are relatively easy.[6] Implementations of the necessary system dependent connection functions for databases supporting ODBC [20] and Postgres95 [24] are currently under development.

3.2 Kant Database Functions and Kash Database Functions

Only a basic set of functions that is necessary to deal with the number field database had to be implemented as a C-library, a library extending the Kant library functions. Basically, following functionality is given by this C-library.

- Opening and closing connections to a number field database.
- Inserting new number fields into the database.
- Deleting number fields from the database.
- Sending a query to the database and successively retrieving the matching number fields. The retrieved number fields are parsed into the internally used binary format.
- Counting the number of matches of a query.

With the help of these functions it is easily possible to write a set of more sophisticated functions in Kash that facilitate the use of the database. For instance a function that checks whether there is a isomorphic number field in the database is written within a few lines of code in Kash.

4 Some Special Topics of a Number Field Database

One of the major problems for a database of number fields is the representation of the stored data, since there is no canonical form of a general number field known. Since a number field k is a finite extension of $\mathbb{Q}$, we can assume that k is given in the form $\mathbb{Q}(\rho)$, where ρ is a root of a monic, irreducible polynomial $f(t) \in \mathbb{Z}[t]$, which is called a generating polynomial of k. Hence, k is a finite

[5] The "Z classes" are implemented in C++ [23] to provide consistent C++ APIs to commonly used database. Our system dependent layer are C functions, whose implementation has been influenced by the implementation of the "Z classes".

[6] In Fig. 1 the layers that vary with the different database managements systems are given as dashed boxes. The layers that are invariant to such changes are given as framed boxes. Since the query language is the standardized SQL for the database management systems, the "user visible part" is independent of the used system and thus the layer "DBMS" is given as a framed box. Only the C-APIs are varying and have to be matched by a corresponding set of "system dependent connection functions". All other layers are not affected by changes of the underlying database management system.

vector space over $\mathbb{Q}$ and every element α in k can be uniquely represented in the form

$$\alpha = a_1 + a_2\rho + \ldots + a_n\rho^{n-1},$$

with $a_i \in \mathbb{Q}$ $(1 \leq i \leq n)$ and $n := \deg(f)$. Moreover, using this representation we can perform all kinds of arithmetic in k and therefore the description of a generating polynomial is an important part of the system we describe. But unfortunately, the polynomial f is not uniquely determined, as the next example shows:

Example. Consider the polynomials $f_1(t) = t^3 - 21t + 35$ and $f_2(t) = t^3 + 3t^2 - 18t + 15$ and the number fields k_i generated by root ρ_i of f_i $(i = 1, 2)$. These two number fields are isomorphic and an embedding of k_1 into k_2 is given by

$$\tau : k_1 \to k_2 \ : \rho_1 \mapsto 10 - 5\rho_2 - \rho_2^2.$$

Note, however, that this embedding is not unique, since

$$\tilde{\tau} : k_1 \to k_2 \ : \rho_1 \mapsto 1 + \rho_2$$

is also a proper embedding. We would like to remark that by ordering the roots of each polynomial, we could define a unique embedding from k_1 into k_2.

4.1 Test for Isomorphisms and Constructing Embeddings

As a consequence of the above representation of number fields using a primitive element, a test whether or not a certain field is already contained in the database is not trivial and in addition it is not a straight–forward procedure to extend the stored information of a number field, if the additional data is given in a different representation. Before we consider the situation that we want to extend the database, we put our focus on the situation that only two number fields k_1 and k_2 are given. Here we have to solve the following two problems:

1. decide, whether or not these two fields are isomorphic,
2. if k_1 and k_2 are isomorphic, find an embedding $\tau : k_1 \to k_2$.

We will see that these two problems are closely connected and that all algorithms we state here will prove that two fields are isomorphic by constructing an embedding from one field into the other.

Before we construct an embedding $\tau \ : k_1 \to k_2$ we perform some easy test to make sure that k_1 and k_2 are not definitely non isomorphic fields. Checking the invariants of a number field are a good way to do so, since two isomorphic number fields have the same invariants [21, 17]. Note, that the reverse conclusion is not true, and hence the invariants are only a test, but not a proof.

For our purposes the most important invariants of a number field k are the signature and the discriminant k. There are certainly more invariants, like the regulator R_k, the class group Cl_k, the Galois group $\mathrm{Gal}(k)$ or the ζ–function of k, but these are much harder to compute than the two mentioned above and long computations have shown that these two invariants are very good indicators for non isomorphic fields. For example there are only 2253 fields of degree 5 and signature $(3, 1)$ with a non unique discriminant (for 2139 of these fields the number of non isomorphic fields is exactly 2) in the set of all fields of this signature and a discriminant bounded by 5000000. The total number of fields with this property is 79394 [22]. We would like to mention, that some additional fast tests, like the verification of the decomposition behavior of small rational primes in a number field, are possible if the pseudo tests need to be improved. We will now sketch two algorithms to construct an embedding of k_1 into k_2 if the two fields passed the pseudo tests. In the following let k_i be generated by a root ρ_i of a monic, irreducible polynomial $f_i(t) \in \mathbb{Z}[t]$ $(i = 1, 2)$ of degree $n = \deg(f_1) = \deg(f_2)$.

Embedding by factorization. The first algorithm uses the factorization of polynomials over number fields [18, 9]. The number field k_1 is isomorphic to k_2 if and only if the polynomial f_1 has a linear factor in $k_2[t]$, e.g. there is an $\alpha \in k_2$ satisfying $(t - \alpha) \mid f_1(t)$. Using α we define an embedding of k_1 into k_2 by

$$\tau : k_1 \to k_2 \ : \rho_1 \mapsto \alpha.$$

Embedding by enumeration. This second method uses the LLL–algorithm [19] combined with the Fincke–Pohst algorithm for determining all lattice points in an ellipsoid. The basic idea is the same as in the first algorithm. We try to find a root α of the polynomial f_1 in the number field k_2. The roots of f_1 are algebraic integers, and since we already know the discriminant of the number field k_2, we can assume that we know an integral basis $\omega_1, \dots, \omega_n$ of k_2 [21, 5], i.e. a $\mathbb{Q}$–basis of k_2 such that an element $\alpha = \sum_{i=1}^{n} a_i \omega_i \in k_2$ is an algebraic integer if and only if $a_i \in \mathbb{Z}$ $(1 \le i \le n)$ and we denote the set of the algebraic integers of k_2 by $\mathcal{O}_{k_2}$.

To sketch the method, let $\sigma_1, \dots, \sigma_n$ be the set of all $\mathbb{Q}$–isomorphisms from k_2 into subfields of $\mathbb{C}$. By

$$\langle \cdot, \cdot \rangle \ : k_2 \times k_2 \to \mathbb{R} \ : (x, y) \mapsto \sum_{i=1}^{n} \sigma_i(x) \overline{\sigma_i(y)}$$

we define a scalar product on k_2, such that $(\mathcal{O}_{k_2}, \langle \cdot, \cdot \rangle)$ is a lattice. The Fincke–Pohst algorithm [10] computes all points in this lattice with a bounded or given value of the norm function T_2 associated to the scalar product. The enumeration procedure can be improved dramatically if an LLL reduced basis of L is used. Since we can compute the value of T_2 for all roots of f_1 by evaluating $\sum_{i=1}^{n} \rho_1^{(i)} \overline{\rho_1^{(i)}}$, where $\rho_1^{(i)}$ $(1 \le i \le n)$ are the roots of f_1, we are able to prove whether or not the polynomial f_1 has a root in k_2 by enumerating the finite number of points in $\mathcal{O}_{k_2}$ with the correct T_2 value. For each of the evaluated points we check if it is a root of f_1. Once such an element is found, we can define an embedding of k_1 into k_2 analog to the last algorithm. If no such element is found, k_1 is not isomorphic to k_2.

There has been a tremendous improvement on algorithms for the factorization of polynomials over number fields in the recent past, so that this method seems to be the best choice in general at the moment. Nonetheless the enumeration method has proven to be better for fields of degrees up to 5 with moderate discriminants. Moreover the second method has advantages over the factorization method, if we have to do several isomorphism tests for a single number field, since we can precompute and store a lot of the needed data.

4.2 Extending the Stored Information

We are now coming back to the problem of extending an existing database. Here we have to check whether for a given number field k an isomorphic one is already stored and if we can extend the data of the stored field. We proceed quite similar to the above case. First, we extract all fields from the database having the same signature and discriminant (if more informations are without any additional computations available, we extend the search criterion) as k and then we test if one of the extracted fields is isomorphic to k. In case this is true, the isomorphism test has computed an embedding from k into the field stored in the database as well, so if informations are known for k that are not already stored, we can transfer these data easily.

5 Use of the Database

5.1 Queries Involving the DBMS Only

Several queries can be handled by the underlying database management system alone. The Kash functions are only needed to parse the number fields from their representation in the relational database to their representation in Kash.

The queries that can be handled by the database management system alone are the ones that consist of logical connections of atomic queries with the

- generating polynomial,
- degree,
- discriminant,
- signature,
- regulator of the unit system,
- Galois group,
- class number,
- number of cyclic factors of the class group,
- the transformation matrix for an integral basis,

to give the most important ones.

In Table 1 we summarize the results of some queries. The timings that are given heavily depend on the used database system. The given timings show that for a subset of the currently computed number field data the shareware database mSQL is sufficiently efficient. Notice that the time for *counting* all matching number fields is generally smaller than the one for returning even one matching number field, because in the latter case the DBMS has to collect data that has to be returned.

Table 1. Summary of some queries

The timings where on a HP 735 using mSQL. The sample database contained 83806 number fields.
Query 1: `"degree=5 and [signature real]=3 and [class number]>1"`
Query 2: `"degree=5 and [signature real]=5 and regulator>99.5"`
Query 3: `"[signature real]=5 and [KANT name Galois group] LIKE '%D5%'"`

	Query 1	Query 2	Query 3
Time for returning first match	0.51 sec	4.95 sec	0.00 sec
Avg. time for returning a consecutive match	0.00 sec	0.00 sec	0.00 sec
Time for counting all matches	0.05 sec	0.76 sec	0.00 sec
Number of matches	1284	17939	26

5.2 Queries Involving Kash Functions

We will give some examples of queries that involve the combination of a database query with functions of Kash.

Example 1. We want to extract all fields k of degree 5 and class number 2 from the database, such that there are exactly two prime ideals above 3 and 5 in k. There have been 343 fields with this property stored in the sample database. The total number of fields of degree 5 and class number 2 in the database were 1244. Here is the Kash sample session:

```
kash> DbOpen ("donald:kantnf");;
kash> DbQueryFLDTable ("degree = 5 and [class number] = 2");
true
kash> L := [];
[ ]
kash>
kash> repeat
>   o := DbNextOrderFromQuery ();
>   if o <> false then
```

```
>     if (Length (Factor (3*o)) = 2) and (Length (Factor (5*o)) = 2)
>        then Add (L, o);
>     fi;
>   fi;
> until o = false;
kash> Length (L);
343
kash> Factor (3*L[100]);
[ [
        [3 1 2 2 0]
        [0 1 0 0 0]
        [0 0 1 0 0]
        [0 0 0 1 0]
        [0 0 0 0 1]
        , 1 ], [
        [3 2 2 0 1]
        [0 1 0 0 0]
        [0 0 1 0 0]
        [0 0 0 3 0]
        [0 0 0 0 1]
        , 2 ] ]
kash> Factor (5*L[100]);
[ [ <5, [2, 1, 0, 0, 0]>, 1 ], [ <5, [15, 10, 9, 15, 3]>, 1 ] ]
kash> DbCountMatchesQueryFLDTable
> ("degree = 5 and [class number] = 2");
1244
```

Example 2. In our second example we are interested in all totally real fields k of degree 3, such that there exists a totally real quadratic extension generated by a unit of k, which is unramified at all places. We know that for such an extension the class number of k has to be divisible by 2. The procedure given below found 146 fields.

The database request alone extracted 612 fields and has thus limited the search space for the Kash procedure tremendously.

```
kash> DbOpen ("donald:kantnf");;
kash> DbQueryFLDTable
> ("degree=3 and [signature real]=3 and [class number]>1");
true
kash> L := [];
[  ]
kash> repeat
>   o := DbNextOrderFromQuery ();
>   if o <> false then
>     if (OrderClassGroup (o) [1] mod 2 = 0) then
>       U := OrderUnitsFund (o);
>       for u in [U[1],U[2],U[1]*U[2],-U[1],-U[2],-U[1]*U[2]] do
>          tmp := Flat (MatToColList (EltCon (u)));
>          if ForAll (tmp, n -> Re (n) > 0) then
```

```
>           O := Order (o,2,u);
>           if (Norm (OrderKextDisc (O)) = 1) then
>             Add (L, [o,u]);
>           fi;
>         fi;
>      od;
>    fi;
>  fi;
> until o = false;
kash> Length (L);
146
kash> DbCountMatchesQueryFLDTable
> ("degree=3 and [signature real]=3 and [class number]>1");
612
```

Acknowledgments. We are grateful to A. Myka for suggesting to consider mSQL. The second author is grateful to R. Zippel for awaking his interest in databases for algebraic objects.

References

1. C. Batut, D. Bernardi, H. Cohen, and M. Olivier. *User's Guide to PARI-GP — Version 1.39.03*. Université Bordeaux I, 33405 Talence Cedex, France, 1995. Available at `ftp://megrez.math.u-bordeaux.fr/`.
2. J. Buchmann, D. Ford, and M. Pohst. Enumeration of quartic fields of small discriminant. *Mathematics of Computation*, 61:873–879, 1993.
3. J. Buchmann and D. J. Ford. On the computation of totally real quartic fields of small discriminant. *Mathematics of Computation*, 52:161–174, 1989.
4. Tim Bunce. Perl 5 database interface (DBI) API specification — version 0.6, 1994. Available at `ftp://ftp.mcqueen.com/pub/dbperl/DBI/`.
5. Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.
6. Mario Daberkow, Max Jüntgen, Andreas Jurk, Michael E. Pohst, and Johannes von Schmettow. KANT. In Fachgruppe Computeralgebra der GI, DMV, GAMM, editor, *Computeralgebra in Deutschland — Bestandsaufnahme, Möglichkeiten, Perspektiven*, pages 212–218. Passau, 1993.
7. Mario Daberkow, M. Pohst, et al. KANT V4. Submitted to *Journal of Symbolic Computation*.
8. C. J. Date and Hugh Darwen. *A Guide to The SQL Standard*. Addison-Wesley, 3rd edition, 1993.
9. M. J. Encarnación. *Fast algorithms for reconstructing rationals, computing polynomial GCDs and factoring polynomials*. PhD thesis, RISC Linz, 1995.
10. U. Fincke and M. Pohst. A procedure for determining algebraic integers of given norm. In *Proc. Eurosam 83*, volume 162 of *Lecture Notes in Computer Science*, pages 194–202. Springer-Verlag, 1983.
11. D. Ford. Enumeration of totally complex quartic fields of small discriminant. In A. Pethö, M. E. Pohst, H. C. Williams, and H. G. Zimmer, editors, *Proceedings*

of the Colloquium on Computational Number Theory, pages 129–138, Debrecen, Hungary, 1989. de Gruyter.
12. Dean Fuqua. Z classes for database access, 1995. Available at `http://alfred.nih.gov/Dean/ZDB.html`.
13. Martin Gruber. *SQL Instant Reference*. Sybex, 1993.
14. David J. Hughes. *Mini SQL — A Lightweigth Database Engine*, 1995. Available at `ftp://Bond.edu.au/pub/Minerva/msql/`.
15. KANT Group. *KASH — A User's Guide*. Straße des 17. Juni 136, 10623 Berlin, Germany, 1995. Available at `ftp://ftp.math.tu-berlin.de/pub/algebra/Kant/`.
16. Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ 07632, second edition, 1988.
17. Serge Lang. *Algebraic Number Theory*, volume 110 of *Graduate Texts in Mathematics*. Springer-Verlag, 1986.
18. A. K. Lenstra. Factoring polynomials over algebraic number fields. In *Proceedings of the 1983 European Computer Algebra Conference*, volume 144 of *Lecture Notes in Computer Science*, pages 32–39. Springer-Verlag, 1982.
19. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.
20. Microsoft Corporation. Accessing the world of information: Open Database Connectivity (ODBC), 1995. Available at `http://www.microsoft.com/DEVONLY/STATEGY/ODBC/ODBCBG.HTM`.
21. Michael E. Pohst and Hans Zassenhaus. *Algorithmic Algebraic Number Theory*. Cambridge University Press, Cambridge, 1989.
22. A. Schwarz, M. Pohst, and F. Diaz y Diaz. A table of quintic fields. *Mathematics of Computation*, 63:361–376, 1994.
23. Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, second edition, 1991.
24. Andrew Yu and Jolly Chen. *The Postgres95 User Manual*. Computer Science Division, University of California at Berkeley. Available at `http://epoch.cs.berkeley.edu:8000/postgres95/`.

This article was processed using the LaTeX macro package with LLNCS style