Finding orders with prescribed index in number fields

Alex J. Best
Supervised by Dr. Lassina Dembélé

February 17, 2014

Contents

Chapter 1

Introduction

This report details .

Chapter 2

Background material

In this section we fix several definitions and important results from algebraic number theory and commutative algebra. These results are well known and are used throughout the rest of the report. There is also some necessary background for the applications to elliptic curves, this is introduced as we need it in section 4.1 however in order to keep this section as brief as possible.

2.1 Commutative algebra

2.2 Algebraic number theory

Algebraic number theory can be thought of as beginning with the study of *algebraic numbers*. The subject as a whole now encompasses a huge amount of related mathematics, all involving the use of algebraic techniques to tackle number theoretic problems.

Definition 1 (Number field).

Definition 2 (Order).

Definition 3 (Ring of integers).

We are now ready to state in precise terms the project aimed to solve and to detail the methods used in its solution.

Chapter 3

The problem

3.1 Statement

The aim of the project was to find a solution to the following problem, and moreover to find an algorithmic solution that works efficiently.

Problem 1. Given an order R of an absolute number field K and an integer I find the set

$$\{\mathcal{O} \subseteq R \mid \mathcal{O} \text{ is a suborder, } [R : \mathcal{O}] = I\}.$$

To find a suborder we really mean compute a $\mathbb{Z}$ basis for the order, as such a basis defines an order completely.

One very natural extension of the above problem is to consider relative extensions of number fields. More precisely we wish to study the following problem.

Problem 2. Given an extension of number fields $L|K$, a $\mathbb{Z}_K$-order R of $\mathbb{Z}_L$ and an integer I find the set

$$\{\mathcal{O} \subseteq R \mid \mathcal{O} \text{ is a } \mathbb{Z}_K\text{-suborder, } [R : \mathcal{O}] = I\}.$$

We now detail the steps leading up to a solution of the problem in increasing generality. The solution is presented this way in order to motivate the ideas used in the more general cases

3.2 Quadratic number fields

It is well known [] that the ring of integers of a quadratic field $K = \mathbb{Q}(\sqrt{d})$ for d non-square always takes the form

$$\mathbb{Z}_K = \mathbb{Z} + \mathbb{Z}\alpha, \text{ with } \alpha = \begin{cases} \sqrt{d} & \text{if } d \equiv 2, 3 \pmod 4, \\ \frac{1+\sqrt{d}}{2} & \text{if } d \equiv 1 \pmod 4. \end{cases}$$

Indeed there is so little room for manoeuvre here that we obtain the following result on the structure of an order in this case.

Proposition 1. *Every $\mathcal{O}$ of a quadratic number field is given by*

$$\mathcal{O} = \mathbb{Z} + \mathbb{Z}m\alpha$$

for some $m \in \mathbb{Z}$, α as above.

3.3 Absolute number fields

We originally hoped that the correspondence between suborders and their conductors that exists in the quadratic case (Theorem **??**) could be generalised to higher degree number fields. However the direct generalisations of this result fail to hold even in degree 3 number fields. We now give examples of some results that would be good for our purposes if true and explicit counter examples for each of them.

3.4 Relative number fields

Chapter 4

Applications

Through the main problem itself is an interesting one which is worth studying in its own right we were also motivated to look at it by the potential applications to other questions within the same areas of mathematics. One of the most prominent areas in which a solution to the problem can be used is to answer questions about elliptic curves. How a solution to the problem considered above can be applied in this case is detailed below, along with results obtain from the application of our methods there. Also ???

4.1 Elliptic curves

Chapter 5

Conclusion

5.1 Further work

Chapter 6

Appendix: Code

Much of what was done has been implemented in both the Sage and Magma and so we provide annotated source code listings for the algorithms in both languages below.

6.1 Sage

```
def conductor(order):
    """
    Return the conductor of the order.
    """

    K = order.fraction_field()
    R = Integers()
    ZK = K.maximal_order()
    omega = ZK.basis()
    n = order.rank()
    M = matrix(R.fraction_field(), nrows = n*n, ncols = n)

    d = 1
    for i in range(n):
        for j in range(n):
            coords = order.coordinates(ZK.gen(i)*ZK.gen(j))
            for k in range(n):
                M[j*n + k, i] = coords[k]
                d = lcm(d, coords[k].denominator())

    H = (d * M).change_ring(R).hermite_form(include_zero_rows =
        False)

    return K.ideal(list(vector(omega) * d * H.inverse()))

def orders_of_index(O,I):
    """
    Returns a list of orders with the given index.
    """
```

```
    K = O.fraction_field()
    ZK = K.maximal_order()
    R = Integers() # This may need to be different when relative
         orders are looked for.

    # We find all ideals of norm dividing I^2, which are all
        possibly conductors of our order.
    possible_conductors = []
    for d in divisors(I):
        possible_conductors += ideals_of_norm(K, I*d)

    print possible_conductors

    orders = []
    for f in possible_conductors:
        #print f
        #print f.basis()
        cur_orders = orders_with_conductor_and_index(f, I)
        if cur_orders:
            orders += cur_orders
        print str(len(cur_orders)) + " order(s) found with right
             index."
    return orders

def ideals_of_norm(K,N):
    # We use the factorisation of prime ideals to find all
        ideals with given norm N
    primes = dict() # primes[p][i] will contain all prime ideals
         with norm p^i.
    for (p,e) in N.factor():
        #print str(p)+"^"+str(e)
        primes[p] = dict()
        for i in range(1,e+1):
            primes[p][i] = []

        for (P,E) in K.factor(p): # for P in K.prime_factors(p)
            ?
            v = valuation(P.norm(), p)
            # replacing the condition below with v <= e would
                probably be faster, if it turns out we cannot
                rule out any k <= e as exponents.
            if v in primes[p]: # Otherwise the ideal is too
                small to be of use to us.
                primes[p][v].append(P)

    #print primes
    ideals = [K.ideal(1)]
    for (p,e) in N.factor():
        possible_factorisations = []
        for partition in Partitions(e, parts_in = primes[p].keys
            ()):
            #print partition
            current_factorisation = [K.ideal(1)]
            for i in partition:
```

```
                new_factorisation = []
                for P in primes[p][i]:
                    for f in current_factorisation:
                        new_factorisation.append(f*P)
                current_factorisation = new_factorisation
            possible_factorisations += new_factorisation

        current_ideals = []
        for P in possible_factorisations:
            for f in ideals:
                current_ideals.append(f*P)
        ideals = current_ideals
    return ideals

def orders_with_conductor_and_index(f,I):
    K = f.number_field()
    Zk = K.maximal_order()
    naive_O = K.order(f.basis()) # Use ring_generators here?!
    if conductor(naive_O) == f:
        if naive_O.index_in(Zk) == I:
            return [naive_O]
        else: # Still not clever enough!
            orders = []
            quo = Zk.free_module().quotient(naive_O.free_module
                ())
            r = quo.cardinality() / I
            for a in quo:
                if a.additive_order() == r:
                    O = K.order(naive_O.gens() + [Zk(a.lift())])
                    if O.index_in(Zk) == I:
                        if not O in orders:
                            orders.append(O)
            return orders
    else:
        pass
        #raise Exception("AAAH")
    return []

def cocyclic_orders_of_index(O,I):
    """
    Returns a list of orders with the given index.
    """
    K = O.fraction_field()
    ZK = K.maximal_order()
    R = Integers() # This may need to be different when relative
         orders are looked for.

    possible_conductors = ideals_of_norm(K, I*I)

    print possible_conductors

    orders = []
    for f in possible_conductors:
        q = ZK.free_module().quotient(f.free_module())
```

```
        if q.ngens() == 2:
            if q.gens()[0].order() == I: # ????
                orders.append(cocyclic_order_with_conductor(f))
        #print f
        #print f.basis()
    return orders

def cocyclic_order_with_conductor(f):
    K = f.number_field()
    Zk = K.maximal_order()
    O = K.order(f.basis()) # Use ring_generators here?!
    if conductor(O) == f:
        return O
    raise Exception("Order was not cocyclic.")
    return []
```

6.2 Magma

```
function overOrder(I,K) // Returns the smallest order containing
     I
    return Order(Basis(I,K));
    //return Order(TwoElement(I));
end function;

function idealOfO(O,I)
    return ideal<O|[O!b : b in Basis(I,FieldOfFractions(O))]>;
end function;

function orderOfIndex(K, d)
    "Finding orders:";
    primeIdeals := AssociativeArray();
    Zk := Integers(K);

    for pe in Factorisation(d^2) do
        p := pe[1];
        e := pe[2];
        "Prime:", p;
        primeIdeals[p] := AssociativeArray();

        for i in [1..e] do
            primeIdeals[p][i] := [];
        end for;

        for PE in Factorisation(ideal<Zk|p>) do
            P := PE[1];
            E := PE[2];
            v := Valuation(Norm(P),p);
            if v in Keys(primeIdeals[p]) then // Otherwise the
                exponent is too high anyway
                Append(~primeIdeals[p][Valuation(Norm(P),p)],P);
                     // Add P to the set of prime ideals of norm
                    p^e
```

```
        end if;
    end for;
    for x in Keys(primeIdeals[p]) do x, primeIdeals[p][x];
        end for;
end for;

"Partitions:";

possConds := [ ideal<Zk|1> ];
for pe in Factorisation(d^2) do
    p := pe[1];
    e := pe[2];
    p,"^",e;
    possFacts := [ ];

    for part in RestrictedPartitions(e, Keys(primeIdeals[p])
        ) do
        part;
        curFact := [ideal<Zk|1>];
        for i in part do
            newFact := [ ];
            for P in primeIdeals[p][i] do
                for f in curFact do
                    Append(~newFact, f*P);
                end for;
            end for;
            curFact := newFact;
        end for;
        possFacts cat:= curFact;
    end for;
    curConds := [];
    for P in possFacts do
        for f in possConds do
            Append(~curConds, f*P);
        end for;
    end for;
    possConds := curConds;
end for;

out := [];

"Trying conductors:";
for f in possConds do
    f,Norm(f);
    Of := overOrder(f,K);
    if Conductor(Of) eq f then
        if Index(Zk,Of) eq d then // Sanity check
            Append(~out, Of);
        else;
            //error Error();
        end if;
    end if;
end for;
```

```
    return out;
end function;
```

Bibliography

[Bra09] Johannes Brakenhoff. *Counting problems for number rings.* PhD thesis, Universiteit Leiden, 2009.

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory.* Graduate Texts in Mathematics. Springer, 1993.

[Coh00] H. Cohen. *Advanced Topics in Computational Number Theory.* Graduate Texts in Mathematics. Springer New York, 2000.

[Lan94] S. Lang. *Algebraic Number Theory.* Applied Mathematical Sciences. Springer, 1994.

[NS10] J. Neukirch and N. Schappacher. *Algebraic Number Theory.* Grundlehren der mathematischen Wissenschaften. Springer, 2010.

[PZ89] M. Pohst and H. Zassenhaus. *Algorithmic Algebraic Number Theory.* Cambridge University Press, 1989.