

```

1  load "SubidealLattices.m";
2
3  function RestrictAutomorphismTypes(l,n)
4  // Input: l in N; n in N
5
6  // Output: Restricts the possible automorphism types for extremal
   modular lattices as much as possible
7
8      min := ExtremalMinimum(l,n);
9
10     Types := AutomorphismTypes(l, Integers() ! (n/2), n, min);
11
12     RestrictedTypes := [];
13
14     for type in Types do
15         type;
16         p := type[1];
17         n1 := type[2];
18         np := type[3];
19         s := type[4];
20
21         if p ne 2 and IsPrime(l) then
22
23             k1 := type[6];
24             kp := type[7];
25
26             K<z> := CyclotomicField(p);
27             Kpos := sub<K | z + z^(-1)>;
28
29             f := InertiaDegree(Factorization(ideal<Integers(Kpos) | l>)
[1][1]);
30             deltap := (-1)^(Integers() ! (kp/f + (p-1)/2 * (Binomial
(Integers() ! (np / (p-1) + 1), 2) + Binomial(s, 2))));
31             delta1 := deltap * (-1)^(Integers() ! (s*(p-1)/2));
32
33             if l eq 2 then
34                 if IsDivisibleBy(np + s*(p-1), 8) then
35                     epsilonp := deltap;
36                 else
37                     epsilonp := -deltap;
38                 end if;
39
40                 if IsDivisibleBy(n, 8) then
41                     epsilon := 1;
42                 else
43                     epsilon := -1;
44                 end if;
45             else
46                 epsilonp := (-1)^(Integers() ! (kp / f + (l-1)/2*Binomial
(kp,2)));
47
48                 if IsDivisibleBy(n*(l+1), 16) then
49                     epsilon := 1;
50                 else
51                     epsilon := -1;
52                 end if;

```

```

53     end if;
54
55     epsilon1 := epsilonp*epsilon;
56
57     Sym1 := [* 2, n1 *];
58     Symp := [* 2, np *];
59     if l eq 2 then
60         Append(~Sym1, <2, k1, epsilon1, 2, 0>);
61         Append(~Sym1, <p, s, delta1>);
62         Append(~Symp, <2, kp, epsilonp, 2, 0>);
63         Append(~Symp, <p, s, deltap>);
64     else
65         if l lt p then
66             Append(~Sym1, <l, k1, epsilon1>);
67             Append(~Sym1, <p, s, delta1>);
68             Append(~Symp, <l, kp, epsilonp>);
69             Append(~Symp, <p, s, deltap>);
70         else
71             Append(~Sym1, <p, s, delta1>);
72             Append(~Sym1, <l, k1, epsilon1>);
73             Append(~Symp, <l, kp, epsilonp>);
74             Append(~Symp, <p, s, deltap>);
75         end if;
76     end if;
77
78     if n1 le 12 and n1 gt 0 then
79         List := [L : L in EnumerateGenusSymbol(Sym1) | Minimum(L)
ge min];
80         if #List eq 0 then
81             continue type;
82         end if;
83     end if;
84
85     if np le 12 and np gt 0 then
86         List := [L : L in EnumerateGenusSymbol(Symp) | Minimum(L)
ge min];
87         if #List eq 0 then
88             continue type;
89         end if;
90     end if;
91
92     else
93
94         if n1 le 12 and n1 gt 0 then
95             det1 := p^s;
96             for i := 5 to #type by 3 do
97                 det1 *:= type[i]^type[i+1];
98             end for;
99
100             List := [L : L in EnumerateGenusDeterminant(det1, n1,
true) | Minimum(L) ge min];
101             if #List eq 0 then
102                 continue type;
103             end if;
104         end if;
105

```

```

106     if np le 12 and np gt 0 then
107         detp := p^s;
108         for i := 5 to #type by 3 do
109             detp *:= type[i]^type[i+2];
110         end for;
111
112         List := [L : L in EnumerateGenusDeterminant(detp, np,
113 true) | Minimum(L) ge min];
114         if #List eq 0 then
115             continue type;
116         end if;
117     end if;
118
119     Append(~RestrictedTypes, type);
120 end for;
121
122 return RestrictedTypes;
123
124 end function;
125
126
127 function PossibleCharPos(l, n)
128 // Input: l in N; n in N
129
130 // Output: List of all characteristic polynomials of lattices
131 // possibly not found by the subideal-lattice algorithm. Format:
132 // [[<d_1,c_1>,...,<d_k,c_k>], ...] for the exponents c_i > 0 of the
133 // Phi_(d_l) for the divisors d_l
134
135 Types := RestrictAutomorphismTypes(l,n);
136
137 Results := [];
138
139 for phim in [Integers() ! (n/2)+1..n] do
140     for m in EulerPhiInverse(phim) do
141         Div := Sort(Divisors(m));
142         Phi := [EulerPhi(d) : d in Div];
143         k := #Div;
144
145         pList := [p : p in PrimeDivisors(m) | Gcd(p,l) eq 1];
146         FixDimLists := [];
147         for p in pList do
148             FixDims := [];
149             for type in Types do
150                 if type[1] eq p then
151                     FixDim := type[2];
152                     if not FixDim in FixDims then
153                         Append(~FixDims, FixDim);
154                     end if;
155                 end if;
156             end for;
157         end for;
158         if #FixDims eq 0 then
159             continue m;
160         end if;

```

```

158     Append(~FixDimLists, FixDims);
159 end for;
160
161 t := #pList;
162
163 M := ZeroMatrix(Integers(), k, t+1);
164 for i in [1..k] do
165     for j in [1..t] do
166         if IsDivisibleBy(Integers() ! (m/pList[j]), Div[i]) then
167             M[i,j] := Phi[i];
168         end if;
169     end for;
170     M[i, t+1] := Phi[i];
171 end for;
172
173 if t gt 0 then
174     TypeChoice := CartesianProduct([1..#List]: List in
FixDimLists]);
175
176     for IndexList in TypeChoice do
177         N := ZeroMatrix(Integers(), 1, t+1);
178         MaxDim := [];
179         for i in [1..t] do
180             N[1][i] := FixDimLists[i][IndexList[i]];
181         end for;
182         N[1][t+1] := n;
183
184         MaxDim := [Floor(n/EulerPhi(d)) : d in Div];
185         for i in [1..k] do
186             for j in [1..t] do
187                 if IsDivisibleBy(Integers() ! (m / pList[j]), Div
[i]) then
188                     MaxDim[i] := Minimum(MaxDim[i], Floor(N[1][j] /
Phi[i]));
189                 else
190                     MaxDim[i] := Minimum(MaxDim[i], Floor((n-N[1]
[j]) / Phi[i]));
191                 end if;
192             end for;
193         end for;
194
195         C := CartesianProduct([0..MaxDim[i]] : i in [1..k]);
196
197         for c in C do
198             v := Matrix(Integers(), 1, k, [x : x in c]);
199             if v*M eq N then
200                 if &or[c[i] gt 0 : i in [1..k-1]] and Lcm([Div[i] :
i in [1..k-1] | c[i] gt 0]) eq m then
201                     ExpList := [<Div[i], c[i]> : i in [1..k] | c[i] gt
0];
202                     Append(~Results, ExpList);
203                 end if;
204             end if;
205         end for;

```

```

206         end for;
207     else
208         C := CartesianProduct([[0..Floor(n/EulerPhi(d))] : d in
Div]);
209         N := Matrix(Integers(), 1, 1, [n]);
210         for c in C do
211             v := Matrix(Integers(), 1, k, [x : x in c]);
212             if v*M eq N then
213                 if Lcm([Div[i] : i in [1..k] | c[i] gt 0]) eq m then
214                     ExpList := [<Div[i], c[i]> : i in [1..k] | c[i] gt
0];
215                     Append(~Results, ExpList);
216                 end if;
217             end if;
218         end for;
219     end if;
220 end for;
221 end for;
222
223 return Results;
224
225 end function;

```