```
load "SubidealLattices.m";

function RestrictAutomorphismTypes(l,n)
// Input: l in N; n in N

// Output: Restricts the possible automorphism types for extremal modular lattices as much as possible

  min := ExtremalMinimum(l,n);

  Types := AutomorphismTypes(l, Integers() ! (n/2), n, min);

  RestrictedTypes := [];

  for type in Types do
    type;
    p := type[1];
    p := type[1];
    n1 := type[2];
    np := type[3];
    s := type[4];

    if p ne 2 and IsPrime(l) then

      k1 := type[6];
      kp := type[7];

      K<z> := CyclotomicField(p);
      Kpos := sub<K | z + z^(-1)>;

      f := InertiaDegree(Factorization(ideal<Integers(Kpos) | l>)[1][1]);
      deltap := (-1)^(Integers() ! (kp/f + (p-1)/2 * (Binomial(Integers() ! (np / (p-1) + 1), 2) + Binomial(s, 2))));
      delta1 := deltap * (-1)^(Integers() ! (s*(p-1)/2));

      if l eq 2 then
        if IsDivisibleBy(np + s*(p-1), 8) then
          epsilonp := deltap;
        else
          epsilonp := -deltap;
        end if;

        if IsDivisibleBy(n, 8) then
          epsilon := 1;
        else
          epsilon := -1;
        end if;
      else
        epsilonp := (-1)^(Integers() ! (kp / f + (l-1)/2*Binomial(kp,2)));

        if IsDivisibleBy(n*(l+1), 16) then
          epsilon := 1;
        else
          epsilon := -1;
```

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

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

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

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