GAP code

Function 1. `S5xS9Needed:= function(input)`

```
#The input is a list of sets (lists) of size three.  Each set of size three
#represents an S3 x S11 that is removed from the "standard" cover.  This
#function computes the list of 5-sets, each representing an S5 x S9, needed
#to cover the elements of type (3,5,6) that have been removed.

local l, templ, x, y, tempx, newl;

templ:= [];;
for x in input do
    tempx:= [];
    for y in [1..14] do
      if not y in x then
        Add(tempx, y);
      fi;
    od;
    Add(templ, tempx);
od;

newl:= List(templ, i -> Combinations(i,5));;

l:= [];

for x in newl do
  for y in x do
    if not y in l then
      Add(l, y);
    fi;
  od;
od;

return l;

end;
```

Function 2. `NumberS5xS9Needed:= function(input)`

```
#Input is a list of sets (lists) of size three.  Computes the length of the
#output of the function S5xS9Needed.

return Length(S5xS9Needed(input));

end;
```

Function 3. `S5xS9Overlaps:= function(input)`

```
#Input is a list of sets (lists) of size three.  Computes the number of
#intersections among the S5 x S9's needed to cover the elements of type
#(3,5,6).

local l, combs, concat, filt;

l:= S5xS9Needed(input);

combs:= Combinations(l,2);
concat:= List(combs, i -> Concatenation(i[1], i[2]));;
filt:= Filtered(concat, i -> IsDuplicateFree(i));;

return filt;

end;
```

Function 4. `NumberOverlaps:= function(input)`

```
#Input is a list of sets (lists) of size three.  Computes the length of
#the output of the function S5xS9Overlaps.

return Length(S5xS9Overlaps(input));

end;
```

Function 5. `455Shortage:= function(input)`

```
#Input is a list of sets (lists) of size three.  Computes how many
#elements are actually contained in the S5 x S9's twice, divided by
#the number of elements in a single S4 x S10. In terms of a cover,
#this can be interpreted as the minimum number of S3 x S11's that are
#not contained in the hypothetical smaller smaller cover if the groups
#corresponding to input are removed.

return (NumberOverlaps(input)*6*24*24/435456);

end;
```

Function 6. `S14SubgroupsNeeded:= function(input)`

```
#Input is a list of sets (lists) of size three.  This is a lower bound
#for the number of subgroups needed in any cover of the elements of S14
#if the groups represented by the lists in input are removed.

local L1, L2;

L1:= NumberS5xS9Needed(input);;

L2:= 455Shortage(input);;
```

```
return (L1 + (1001 - L1 + L2) +
(364 - (L1 + (1001 - L1 + L2))*84*2*24*120/2661120) + 1 + 1716 + 14);

end;
```

FUNCTION 7. `PossibleExtensions:= function(input)`

```
#Input is a list of sets (lists) of size three, representing the
#S3 x S11's removed. This computes the possible "extensions" of this
#list, i.e., all lists of size Length(input) + 1 that are feasible and
#contain input.  The function then computes which lists are the same
#with respect to conjugation by an element of S14 and returns a
#representative of each type of genuinely different extension.

local listn, test, temp, x, y, list, templist, reps, covered, counter, G;

listn:= [];;

for x in Combinations([1..14],3) do
  test:= true;
  for y in input do
      if Size(Intersection(x,y)) = 3 then
    test:= false;
      fi;
  od;
  if test then
    Add(listn, x);
  fi;
od;

list:= [];
templist:= [];
reps:= [];

for x in listn do
    temp:= [x];
    for y in input do
      Add(temp, y);
    od;
    Add(templist, temp);
od;

for x in templist do
    if S14SubgroupsNeeded(x) < 3096 then
        Add(list, x);
    fi;
od;

if Length(list) = 0 then
```

```
  return list;
fi;

reps:=[list[1]];

G:= SymmetricGroup(14);

covered:= Orbit(G, AsSet(list[1]), OnSetsSets);

counter:= 1;

for x in list do
  if Length(reps) > counter then
    covered:= Concatenation(covered,
          Orbit(G, AsSet(reps[Length(reps)]), OnSetsSets));
    counter:= counter + 1;
  fi;
  if not AsSet(x) in covered then
    Add(reps, x);
  fi;
od;

return reps;

end;
```

FUNCTION 8. `PossibleExtensions_2:= function(input)`

```
#Input is a list of sets (lists) of size three, representing the
#S3 x S11's removed. Function is the same as PossibleExtensions, except
#we now assume that anything added must have nonempty intersection with
#the sets in input.

local listn, test, temp, x, y, list, templist, reps, covered, counter, G;

listn:= [];;

for x in Combinations([1..14],3) do
  test:= true;
  for y in input do
      if Size(Intersection(x,y)) in [0,3] then
    test:= false;
      fi;
  od;
  if test then
    Add(listn, x);
  fi;
od;
```

```
list:= [];
templist:= [];
reps:= [];

for x in listn do
    temp:= [x];
    for y in input do
      Add(temp, y);
    od;
    Add(templist, temp);
od;

for x in templist do
    if S14SubgroupsNeeded(x) < 3096 then
    Add(list, x);
    fi;
od;

if Length(list) = 0 then
  return list;
fi;

reps:=[list[1]];

G:= SymmetricGroup(14);

covered:= Orbit(G, AsSet(list[1]), OnSetsSets);

counter:= 1;

for x in list do
  if Length(reps) > counter then
    covered:= Concatenation(covered,
          Orbit(G, AsSet(reps[Length(reps)]), OnSetsSets));
    counter:= counter + 1;
  fi;
  if not AsSet(x) in covered then
    Add(reps, x);
  fi;
od;

return reps;

end;
```