

Ident512

**An identification routine for the groups
of order 512**

0.8

3 December 2025

Henrik Schanze

Henrik Schanze

Email: henrik.schanze@tu-braunschweig.de

Homepage: <https://www.iaa.tu-bs.de/henschan>

Address: Universitaetsplatz 2

Contents

1	Introduction	3
2	Functionality	4
2.1	Functions for building a search tree	4
2.2	Identifying groups via a serach tree	8
2.3	Functions to determine siblings and twins	9
	Index	13

Chapter 1

Introduction

The main purpose of the `Ident512` package is to implement an identification routine for the groups of order 512. This is done by calculating a number of invariants and comparing the results with the values of all possible candidates via a precalculated search tree. This Procedure determines the vast majority of groups. In the unfavorable case that the used invariants shrink the number of candidates down to more than one group the function `IsPqIsomorphicPGroup` from the `Anupq` package is used to determine the correct candidate.

All functions used to build the precalculated search tree are also content of this package. Thus one can build ones own search tree with a different set of invariants.

While determining suitable invariants for this task, the definition of subgroup- and factor-equivlance was developed together with the definition of twins (see). The latter part of the packag implements function to determine these sets of groups, load precalculated lists of such groups and calculate certain invariants for these sets.

Chapter 2

Functionality

This chapter lists all functions contained in the `Ident512` package.

2.1 Functions for building a search tree

This section describes all functions and variables connected to building a search tree to identify groups via implemented invariants.

2.1.1 Pack

▷ `Pack(list)` (function)

Returns: An integer encoding the content of *list*.

Encodes the content of the provided *list* into an integer.

Example

```
gap> Pack([1,2,3]);
123
gap> Pack([606, 109, 1405]);
63095
gap> Pack([[1,2,3], [606, 1405]]);
19765
```

2.1.2 BuildCoc

▷ `BuildCoc(G)` (function)

Returns: A list of lists containing the conjugacy classes of *G*.

Computes the conjugacy classes of the group *G* and clusters them such that each clusters contains all classes of a certain length and certain representative order (meaning the order of any representative).

2.1.3 ClusterList

▷ `ClusterList(tree[, ...])` (function)

Returns: A list of all clusters of groups in a given search tree (the leafs of the tree).

When initializing the only required argument is *tree*. The information stored for each cluster includes the size of the cluster, the path through the tree, a list of descriptions for performed `EvalFpCoc`-tests and the ids of the groups.

Example

```
gap> tree := rec( fp := [ 63049, 75721, 4310 ], next := [ rec( fp := [ 337 ], next := [ [ 3, 8 ]
> [ 4 ], rec( fp := [ 32173 ], next := [ [ 5, 6, 7 ] ] ) ) ] );
gap> ClusterList(tree);
[ rec( descl := [ ], ids := [ 3, 8 ], path := [ 63049, 337 ], size := 2 ),
  rec( descl := [ ], ids := [ 4 ], path := [ 75721 ], size := 1 ),
  rec( descl := [ ], ids := [ 5, 6, 7 ], path := [ 4310, 32173 ], size := 3 ) ]
```

2.1.4 PerformTestOnGroup

▷ `PerformTestOnGroup(G , $test$ [, opt])`

(function)

Returns: The value returned by $test$, if desired encoded as an integer.

Performs $test$ on the group G . The following tests are implemented:

- *"Parent"*: Identifies the parent of G via `IdGroup`.
- *"Center"*: Calculates the rank and abelian invariants of the center of G .
- *"AbelInv"*: Computes the order of each term of the derived series and the abelian invariants of the derived factors of G .
- *"Weights"*: Calculates the `LGWeights` of a special pcgs of G .
- *"PCSID"*: Identifies the proper subgroups of the p-central series of G via `IdGroup`.
- *"ElmOrds"*: Collects the orders of all elements of G .
- *"CocList"*: Collects the number of conjugacy classes of a G of same length and same representative order (the order of a representative).
- *"SpSubId"*: Computes the Frattini subgroup, the upper and lower central series of G and identifies all terms if possible via `IdGroup`.
- *"MaxSubId"*: Computes all maximal subgroups of G and identifies them via `IdGroup`.
- *"SubId"*: Collects all sizes and isomorphism types of conjugacyclasses of proper subgroups via `IdGroup`.
- *"MaxFactId"*: Calculates all minimal non-trivial subgroups of G and identifies their quotient with G via `IdGroup`.
- *"CentQuot"*: Calculates the central quotients of G and identifies them via `IdGroup`.
- *"FactId"*: Identifies all proper factors of G via `IdGroup`.
- *"OutId"*: Tries to identify the groups of outer automorphisms of G , either via `IdGroup` or via invariants.
- *"LattEqv"*: Calculates all conjugacy classes of proper subgroups of G , stores their size, identifies the isomorphism types of each class and determines the isomorphism types and conjugacy classes in G of its maximal subgroups and minimal supergroups.

This function accepts a record *opt* of options with the single content *opt.pack*, a boolean that determines if Pack should be applied to the result of a test. The default value of pack is true.

Example

```
gap> G := SmallGroup(512, 1405);;
gap> PerformTestOnGroup(G, "ElmOrds");
85416
gap> PerformTestOnGroup(G, "ElmOrds", rec(pack := false));
[ [ 1, 1 ], [ 2, 7 ], [ 4, 24 ], [ 8, 288 ], [ 16, 64 ], [ 32, 128 ] ]
```

2.1.5 PerformTestOnClusterList

▷ PerformTestOnClusterList(*CL*, *test*[], *opt*) (function)

Returns: Either a list of pairs each containing the path of the respective cluster and the subbranch resulting from *test* or just a list of subbranches.

Performs a test on the groups of clusters in the list *CL*. The tests are performed by PerformTestOnGroup. *CL* could either be a cluster list produced by ClusterList or be a list of lists containing ids. In the former case the output is a list of pairs containing the path of a cluster and the subbranch resulting from the test. In the latter case only a list of subbranches is returned. This function accepts a record *opt* of options with the following content:

- *pack*: A boolean that is handed to PerformTestOnGroup.
- *ign1cl*: A boolean that determines if clusters of size 1 should be ignored.
- *order*: An integer giving the order of the groups with ids in *CL*.
- *verbose*: A boolean that determines if a progress should be printed.

The default value is *pack:=true*, *ign1cl:=true*, *order:=512*, *verbose:=true*.

Example

```
gap> CL:=[[1..5],[6..10]];;
gap> PerformTestOnClusterList(CL,"ElmOrds",rec(verbose:=false));
[ rec( fp := [ 88570, 4310, 63049, 75721 ], next := [ [ 1 ], [ 2, 5 ], [ 3 ], [ 4 ] ] ),
  rec( fp := [ 4310, 63049 ], next := [ [ 6, 7, 9, 10 ], [ 8 ] ] ) ]
```

2.1.6 AddClusterTestResults

▷ AddClusterTestResults(*tree*, *TR*) (function)

Returns: A search tree (implemented as a record).

Extends *tree* by the test results stored in *TR*, a list of pairs each containing the path in *tree* and a branch (a tree of depth one), extending the corresponding cluster.

Example

```
gap> tree:=BuildSearchTree([3,4,5,6,7,8], 512, ["ElmOrds"]);;
gap> CL := ClusterList(tree);
[ rec( desc1 := [ ], ids := [ 3, 8 ], path := [ 63049 ], size := 2 ),
  rec( desc1 := [ ], ids := [ 4 ], path := [ 75721 ], size := 1 ),
  rec( desc1 := [ ], ids := [ 5, 6, 7 ], path := [ 4310 ], size := 3 ) ]
gap> TR:=PerformTestOnClusterList(CL,"MaxSubId",rec(verbose:=false));
[ [ [ 63049 ], rec( fp := [ 62603 ], next := [ [ 3, 8 ] ] ) ],
  [ [ 4310 ], rec( fp := [ 11744, 86646 ], next := [ [ 5 ], [ 6, 7 ] ] ) ] ]
gap> AddClusterTestResults(tree,TR);;
```

```
gap> ClusterList(tree);
[ rec( desc1 := [ ], ids := [ 3, 8 ], path := [ 63049, 62603 ], size := 2 ),
  rec( desc1 := [ ], ids := [ 4 ], path := [ 75721 ], size := 1 ),
  rec( desc1 := [ ], ids := [ 5 ], path := [ 4310, 11744 ], size := 1 ),
  rec( desc1 := [ ], ids := [ 6, 7 ], path := [ 4310, 86646 ], size := 2 ) ]
```

2.1.7 BuildSearchTree

▷ BuildSearchTree(*N*, *order*, *tests*) (function)

Returns: A search tree (implemented as a record).

Builds a search tree for the groups with ids in *N* or, if *N* is an integer, the first *N* groups of size *order*, using the tests from the list *tests* and PerformTestOnClusterList.

Example

```
gap> BuildSearchTree(10, 512, ["MaxSubId", "CentQuot"]);
rec( fp := [ 25611, 76646, 62603, 89031, 11744, 86646, 63803, 78015 ],
      next := [ [ 1 ], [ 2 ], rec( fp := [ 63542, 75796 ], next := [ [ 3 ], [ 8 ] ] ), [ 4 ], [ 5 ] ],
      rec( fp := [ 30861, 61235 ], next := [ [ 6 ], [ 7 ] ] ), [ 9 ], [ 10 ] )
gap> BuildSearchTree([1,2,3,5,7,11], 512, ["ElmOrds", "CocList"]);
rec( fp := [ 88570, 4310, 63049, 94635 ],
      next := [ [ 1 ], rec( fp := [ 96561, 32173 ], next := [ [ 2 ], [ 5, 7 ] ] ), [ 3 ], [ 11 ] ] )
```

2.1.8 FindParametersForCocTest

▷ FindParametersForCocTest(*coc*) (function)

Returns: A pair made of a list of return values from EvalFpCoc and a list describing the test performed by EvalFpCoc.

Does a parameter search such that evaluating EvalFpCoc on the clusters of conjugacy classes in *coc* results into at least 2 different values.

Example

```
gap> coc:=List([3, 8], id->BuildCoc(SmallGroup(512, id)));;
gap> FindParametersForCocTest(coc);
[ [ 1363, 46760 ], [ 3, 2, 4 ] ]
```

2.1.9 DeEncodeDesc

▷ DeEncodeDesc(*desc*) (function)

Returns: An integer or a list of integers

De- or encodes the description for EvalFpCoc, depending if an integer (decodes to a list) or a list of integers (encodes to an integer) is provided.

Example

```
gap> DeEncodeDesc([3,2,4]);
302004
gap> DeEncodeDesc(302004);
[3, 2, 4]
```

2.1.10 AlterCocAccordingToFp

▷ AlterCocAccordingToFp(*coc*, *desc*, *fp*) (function)

Returns: A list of lists containing the conjugacy classes

Alters *coc* according the result *fp* from EvalFPCoc evaluated with *desc*.

2.1.11 CocSplitRoutine

▷ CocSplitRoutine(*Ids*, *order*, *desclist*) (function)

Returns: A serach tree (implemented as a record) of depth one.

Builds a search tree of depth one to distinguish the groups of *order* with ids in *Ids* based on the returned values of EvalFpCoc. Past tests with EvalFpCoc which altered the clusters of conjugacy classes returned by BuildCoc can be recreated via the list *desclist*.

Example

```
gap> CocSplitRoutine([5,6,7], 512, []);
rec( desc := [104003], fp := [ 1214, 16 ],
      next := [ [ 5 ], rec( desc := [40204005], fp := [ 20792, 46881 ], next := [ [ 6 ], [ 7 ] ] ) )
```

2.1.12 PerformCocTestOnClusterList

▷ PerformCocTestOnClusterList(*CL*[, *opt*]) (function)

Returns: A list of pairs each containg the path of the respective cluster and the subbrunch resulting from CocSplitRoutine.

Tries to split the clusters in the list *CL* via CocSplitRoutine. *CL* needs to be a cluster list produced by ClusterList. This function accepts a record *opt* of options with the following content:

- *pack*: A boolean that determines if Pack should be applied to the result.
- *order*: An integer giving the order of the groups with ids in *CL*.
- *verbose*: An boolean that determines if a progress should be printed.

The default value is *pack:=true*, *order:=512*, *verbose:=true*.

Example

```
gap> tree:=PerformTestOnClusterList([3,8,5,6,7],"CocList",rec(verbose:=false));
gap> CL:=ClusterList(tree);
[ rec( descl := [ ], ids := [ 3, 8 ], path := [ 337 ], size := 2 ),
  rec( descl := [ ], ids := [ 5, 6, 7 ], path := [ 32173 ], size := 3 ) ]
gap> TR:=PerformCocTestOnClusterList(CL,rec(verbose:=false));
gap> AddClusterTestResults(tree, TR);
gap> ClusterList(tree);
[ rec( descl := [ [ 302004 ] ], ids := [ 3 ], path := [ 337, 1363 ], size := 1 ),
  rec( descl := [ [ 302004 ] ], ids := [ 8 ], path := [ 337, 46760 ], size := 1 ),
  rec( descl := [ [ 104003 ] ], ids := [ 5 ], path := [ 32173, 1214 ], size := 1 ),
  rec( descl := [ [ 104003 ], [ 40204005 ] ], ids := [ 6 ], path := [ 32173, 16, 20792 ], size := 1 ),
  rec( descl := [ [ 104003 ], [ 40204005 ] ], ids := [ 7 ], path := [ 32173, 16, 46881 ], size := 1 ) ]
```

2.2 Identifying groups via a serach tree

This section briefly describes the function and variable in this package used to identify a arbitrary group of order 512.

2.2.1 ID_GROUP_512_TREE

▷ ID_GROUP_512_TREE

(global variable)

The search tree of groups of order 512 that can determine the vast majority of groups via the packed invariants "MaxSubId", "CentQuot", "ElmOrds", "CocList" from PerformTestOnGroup and the results from CocSplitRoutine.

2.2.2 IdGroup512

▷ IdGroup512(G [, opt])

(function)

Returns: The id of the group of order 512 in SmallGroup isomorphic to G .

Identifies the group G of order 512. This function accepts a record opt of options with the following content:

- *pack*: A boolean that is handed to PerformTestOnGroup.
- *tests*: A list of strings, that determines which tests from PerformTestOnGroup are used to distinguish groups with *tree*.
- *tree*: A searchtree that was build using the invariants resulting from *tests*.
- *anupq*: A boolean that determines if the *IsPqIsomorphicPGroup* from the Anupq-Package should be used to distinguish remaining candidates for the

identity of G if necessary. The default value is *tests*:=["MaxSubId", "CentQuot", "ElmOrds", "CocList"], *pack*:= true, *anupq*:=true. *tree* is unbound and ID_GROUP_512_TREE is used by default.

Example

```
gap> G := Image(IsomorphismPermGroup(SmallGroup(512, 1984)));;
gap> IdGroup512(G);
1984
gap> H := Image(IsomorphismPermGroup(SmallGroup(512, 449)));;
gap> IdGroup512(H, rec(anupq := false));
[449, 450]
gap> IdGroup512(H, rec(anupq := true));
449
```

2.3 Functions to determine siblings and twins

This section describes all functions to calculate

- subgroup-equivalent groups, i.e. groups with isomorphic proper subgroups,
- factor-equivalent groups, i.e. groups with isomorphic proper factors,
- siblings, i.e. groups that are subgroup- and factor equivalent and where the bijection of subgroups preserves conjugacy and maps the Frattini subgroups, lower and upper central series onto the respective counterparts,
- twins, siblings that are also Brauer Pairs.

It also provides functions to load precalculated lists of such groups and functions to calculate invariants of these sets of groups and a way to export latex-tables-rows with these informations

2.3.1 SplitBinsByTest

▷ `SplitBinsByTest(Bins, order, test)` (function)

Returns: A list of sublists of the provided lists in *Bins*.

Takes a list of bins with ids of groups of size *order* and performs *test* via `PerformTestOnGroup` on them. Depending on the outcome a bin is split into smaller bins according to the test result.

Example

```
gap> SplitBinsByTest([[1..10]], 512, "MaxSubId");
[ [ 1 ], [ 2 ], [ 3, 8 ], [ 4 ], [ 5 ], [ 6, 7 ], [ 9 ], [ 10 ] ]
```

2.3.2 CalculateSiblings

▷ `CalculateSiblings(order)` (function)

Returns: A list of lists of ids of groups of size *order* that are siblings.

Performs all necessary tests on all groups of size *order* to determine all sets of Siblings.

2.3.3 SiblingsIds

▷ `SiblingsIds(order[, indices])` (function)

Returns: A list of lists of ids of groups of size *order* that are siblings.

Loads and returns a list of ids of siblings of size *order* if available. If a list *indices* is given only the respective entries are returned.

Example

```
gap> SiblingsIds(128);
[ [ 1317, 1322 ], [ 1327, 1329 ], [ 1597, 1598 ] ]
gap> SiblingsIds(256, [1..3]);
[ [ 227, 228 ], [ 258, 260, 261 ], [ 845, 846 ] ]
```

2.3.4 Siblings

▷ `Siblings(order[, indices])` (function)

Returns: A list of groups of size *order* that are siblings.

Loads and returns a list of ids of siblings of size *order* if available. If *indices* are given only the respective entries are returned.

2.3.5 DetermineBrauerPairs

▷ `DetermineBrauerPairs(Bins, order)` (function)

Returns: A list of sublists of the provided lists in *Bins*.

Determines the Brauer pairs among the bins of ids of groups of size *order* listet in *Bins*.

2.3.6 TwinsIds

▷ `TwinsIds(order[, indices])` (function)

Returns: A list of lists of ids of groups of size *order* that are twins.

Loads and returns a list of ids of twins of size *order* if available. If a list *indices* is given only the respective entries are returned.

Example

```
gap> TwinsIds(256);
[ [ 1734, 1735 ], [ 1736, 1737 ], [ 1739, 1740 ], [ 1741, 1742 ], [ 3678, 3679 ], [ 4154, 4157 ],
gap> TwinsIds(256, [1..3]);
[ [ 1734, 1735 ], [ 1736, 1737 ], [ 1739, 1740 ] ]
```

2.3.7 Twins

▷ `Twins(order [, indices])` (function)

Returns: A list of groups of size *order* that are twins.

Loads and returns a list of twins of size *order* if available. If a list *indices* is given only the respective entries are returned.

2.3.8 SplitExactOutEqv

▷ `SplitExactOutEqv(Bin, order)` (function)

Returns: Sublists of the provided *Bin* containing the outer-automorphism equivalent groups among them.

Splits list of ids of groups of size *order* in *Bin*, if their outer automorphism groups are isomorphic, via `IsPqIsomorphicPGroup` from the `Anupq` package.

2.3.9 BinsOfOutEqGroups

▷ `BinsOfOutEqGroups(Bins, order, exact)` (function)

Returns: A list of sublists of the provided lists in *Bins*.

Takes a list of bins with ids of groups of size *order* and evaluates which are outer- automorphism equivalent. If *exact* = true, sets of groups which could not be ruled out to be equivalent are tested via `SplitExactOutEqv`. Otherwise these sets are simply included.

2.3.10 LoadSuperTwins

▷ `LoadSuperTwins(order)` (function)

Returns: A list of lists of ids of twins of size *order* that are outer- automorphism equivalent.

Loads and returns a list of ids of twins of size *order* if available.

Example

```
gap> LoadSuperTwins(512);
[ [ 28340, 28341 ], [ 1364, 1368 ], [ 1365, 1369 ], [ 1366, 1370 ], [ 1367, 1371 ], [ 1682, 1683 ],
[ 1680, 1681 ], [ 1746, 1747 ], [ 1748, 1749 ], [ 1750, 1751 ], [ 1752, 1753 ], [ 53283, 53285 ],
[ 43261, 43262 ], [ 28426, 28427 ], [ 29033, 29034 ], [ 29035, 29036 ], [ 30917, 30923 ],
[ 30981, 30983 ], [ 32666, 32667 ], [ 53282, 53284 ], [ 139350, 139355 ], [ 53396, 53398 ],
[ 53382, 53384 ], [ 42518, 42519 ], [ 42482, 42483 ], [ 42520, 42521 ], [ 43208, 43209 ],
[ 45383, 45384 ], [ 53306, 53307 ], [ 53383, 53385 ], [ 53397, 53399 ], [ 53484, 53485 ],
[ 253429, 253432 ], [ 253447, 253450 ] ]
```

2.3.11 TableOfGroupInvariants

▷ `TableOfGroupInvariants(bins, order)` (function)

Returns: A list of records

Calculates a number of invariants and characteristics for the groups of order *order* with ids in lists of *bins*. If the invariants are equal for all ids of a bin, a record listing everything is added to the returned list. The invariants/characteristics are:

- the rank of the group
- the p-class of the group
- the information if the groups are isoclinic
- the size of the group of automorphisms
- the information if the groups are outer-automorphism-equivalent
- the information if the groups form a modular isomorphic pair (MIP)

The entries *isoclinic* and *MIP* are not determined in this function but could be added to the entries of a table via `DetermineMIP` and `DetermineIsoclinic`.

2.3.12 DetermineMIP

▷ `DetermineMIP(tab)` (function)

Returns: A list of records

Determines if the sets of groups represented by the records in *tab* are modular isomorphic pairs using the `ModIsom` package. This information is added to each record and the updated list is returned.

2.3.13 DetermineIsoclinic

▷ `DetermineIsoclinic(tab)` (function)

Returns: A list of records

Determines if the sets of groups represented by the records in *tab* are isoclinic using the `XMod` package. This information is added to each record and the updated list is returned.

2.3.14 MergeTablesOfInvariants

▷ `MergeTablesOfInvariants(tab1, tab2)` (function)

Returns: A list of records

Merges the tables *tab1* and *tab2* into one table. If a record entry of only one is empty the non-empty entry is used. This enables the parallel computation of isoclinism and MIPs.

2.3.15 PrintTableToFile

▷ `PrintTableToFile(tab, filename)` (function)

Exports the content of the table *tab* into *filename* table.txt as rows of a latex table.

Index

AddClusterTestResults, [6](#)
AlterCocAccordingToFp, [7](#)

BinsOfOutEqGroups, [11](#)
BuildCoc, [4](#)
BuildSearchTree, [7](#)

CalculateSiblings, [10](#)
ClusterList, [4](#)
CocSplitRoutine, [8](#)

DeEncodeDesc, [7](#)
DetermineBrauerPairs, [10](#)
DetermineIsoclinic, [12](#)
DetermineMIP, [12](#)

FindParametersForCocTest, [7](#)

IdGroup512, [9](#)
ID_GROUP_512_TREE, [9](#)

LoadSuperTwins, [11](#)

MergeTablesOfInvariants, [12](#)

Pack, [4](#)
PerformCocTestOnClusterList, [8](#)
PerformTestOnClusterList, [6](#)
PerformTestOnGroup, [5](#)
PrintTableToFile, [12](#)

Siblings, [10](#)
SiblingsIds, [10](#)
SplitBinsByTest, [10](#)
SplitExactOutEqv, [11](#)

TableOfGroupInvariants, [11](#)
Twins, [11](#)
TwinsIds, [10](#)