The DifSets Package

Version 2.3.0

14 July 2019

Dylan Peifer

Dylan Peifer
Email: djp282@cornell.edu
Homepage: http://www.math.cornell.edu/~djp282
Address: Department of Mathematics
105 Malott Hall
Cornell University
Ithaca, NY 14853-4201 USA

Abstract

The DifSets Package implements an algorithm for enumerating all difference sets up to equivalence in an arbitrary finite group. The algorithm functions by finding difference sums, which are potential images of difference sets in quotient groups of the original group, and searching their preimages. In this way, the search space can be dramatically decreased, and searches of groups of relatively large order (such as order 64 or order 96) can be completed.

Copyright

Contents

Chapter 1

Definitions

1.1 Difference Sets

A $\langle v,k,\lambda\rangle$-difference set is a nonempty proper subset D of a finite group G such that $|G| = v$, $|D| = k$, and each nonidentity element of G can be written as $d_i d_j^{-1}$ for $d_i, d_j \in D$ in exactly λ different ways. The standard example is the $\langle 7,3,1\rangle$-difference set $\{1,2,4\}$ of the group $\mathbb{Z}/7\mathbb{Z}$ under addition. Additionally, it can easily be shown that every one element subset of a group is a difference set, and the complement of any difference set is also a difference set.

We will often abuse notation and let D denote both the set D and the element

$$D = \sum_{d\in D} d$$

of the group ring $\mathbb{Z}[G]$. Then define

$$gD = \sum_{d\in D} gd,$$

$$D^{\phi} = \sum_{d\in D} \phi(d),$$

$$D^{(-1)} = \sum_{d\in D} d^{-1},$$

where $g \in G$ and ϕ is a homomorphism with domain G. Using this notation, a difference set in G is an element of the group ring $\mathbb{Z}[G]$ with coefficients from $\{0,1\}$ such that $DD^{(-1)} = (k-\lambda) + \lambda G$, where by convention the isolated coefficients $(k-\lambda)$ are assumed to be coefficients of the identity.

Two difference sets D_1, D_2 are equivalent if both are in the same group G and $D_1 = gD_2^{\phi}$ for some $g \in G$ and $\phi \in \mathrm{Aut}(G)$. In other words, D_1 is equivalent to D_2 if D_1 can be mapped to D_2 by translation and automorphism in the group G. We say D_1, D_2 are translationally equivalent if they are equivalent solely by translation, meaning $D_1 = gD_2$ for some $g \in G$.

In the package, difference sets are stored as lists of integers that represent the index of the elements in the difference set as found in the list of all elements in the group returned by the GAP function `Elements(G)`. For example, the difference set `[1, 3, 6, 9, 11, 13]` in `SmallGroup(16, 5)` really consists of the first, third, sixth, ninth, eleventh, and thirteenth elements of the list returned by `Elements(SmallGroup(16, 5))`. When given as arguments, difference sets in the package are never assumed to be sorted, but many functions will return difference sets in sorted order since sorting is used internally.

1.2 Difference Sums

A $\langle v,k,\lambda\rangle$-difference sum in a group G modulo its normal subgroup N is an element S of the group ring $\mathbb{Z}[G/N]$ such that $SS^{(-1)} = (k-\lambda)+\lambda|N|G/N$ and the coefficients of S have values in $\{0,1,\ldots,|N|\}$. Note that the original G and N are included in the definition, so it makes no sense to talk about a difference sum in some arbitrary group H. The size of a difference sum is the sum of its coefficients, and by defining the complement of S to be $|N|G/N-S$ we can see that, similar to difference sets, size one sums and complements of difference sums are always difference sums.

Two difference sums S_1,S_2 are equivalent if both are in the same group G mod its normal subgroup N and $S_1 = gS_2^{\phi}$ for some $g \in G/N$ and ϕ an automorphism of G/N induced by an automorphism of G. Note that not all automorphisms of G/N are induced by automorphisms of G, so our definition here is more restrictive than perhaps expected. As with difference sets, the sums S_1,S_2 are translationally equivalent if $S_1 = gS_2$ for some $g \in G/N$.

In the package, difference sums are stored as lists of integers that represent the values of the coefficients of the group ring elements, with position in the list given by the position of the coset in the list of elements returned by the GAP function `Elements(G/N)`. For example, the difference sum `[2, 4]` in `G := SmallGroup(16, 5)` mod its normal subgroup `Subgroup(G, [G.2, G.3, G.4])` has coefficient 2 on the identity coset, and coefficient 4 on the nonidentity coset.

Difference sums can be thought of as a generalization of difference sets. More importantly, however, difference sums can be thought of as images of difference sets in quotients of the original group. In particular, if $\theta : G \to G/N$ is the natural projection, then for any difference set D in $\mathbb{Z}[G]$ we have a difference sum D^{θ} in G modulo its normal subgroup N. Additionally, difference sums induce other difference sums in any further quotient. The fundamental idea of the algorithm in this package is that we can reverse this process. Starting with G mod G, where the only difference sum of size k is `[k]`, we can successively refine this difference sum up a series of quotients of G until reaching G itself. In each step we enumerate all preimages of the difference sums and remove preimages that are not difference sums themselves. In the final step we refine to difference sets. Furthermore, since equivalent difference sums will have equivalent collections of difference sets as preimages, in each step we remove all but one representative of each equivalence class from our collection. This method dramatically decreases the search space for an exhaustive enumeration of all difference sets up to equivalence in G.

Chapter 2

Package Contents

The DifSets Package consists of a collection of functions implementing the main algorithm, and some additional functions for experimentation and testing. Several functions not appearing in this documentation are used internally for certain subtasks. See the code itself for details.

2.1 The Main Functions

The purpose of this package is to provide a function that efficiently enumerates all difference sets up to equivalence in a given group. Similarly, we can also enumerate all difference sums up to equivalence. The following are these functions. Their components are described in further sections.

2.1.1 DifferenceSets

▷ `DifferenceSets(G)` (function)

Returns a list of all difference sets up to equivalence in the group `G`. Only the smaller of each complementary pair of difference sets is included, and one-element difference sets are ignored.

Example

```
gap> G := SmallGroup(16, 9);;
gap> DifferenceSets(G);
[ [ 1, 2, 3, 4, 7, 10 ], [ 1, 2, 3, 4, 8, 9 ] ]
```

2.1.2 DifferenceSums

▷ `DifferenceSums(G, N)` (function)

Returns a list of all difference sums up to equivalence in the group `G` mod its normal subgroup `N`. Only the smaller of each complementary pair of difference sums is included, and difference sums of size 1 are ignored.

Example

```
gap> G := SmallGroup(16, 8);;
gap> N := Subgroup(G, [G.3, G.4]);;
gap> DifferenceSums(G, N);
[ [ 3, 1, 1, 1 ], [ 2, 2, 2, 0 ] ]
```

2.2 Sizes

The first step of the algorithm is to determine what possible sizes of difference sets and sums the group can contain. Each size is then handled individually since different size sets or sums will never be equivalent.

2.2.1 PossibleDifferenceSetSizes

▷ `PossibleDifferenceSetSizes(G)` (function)

Returns a list of the possible sizes of difference sets in group `G`. Only the smaller of any pair of complementary sizes is returned, and the trivial size 1 is never included. Current implementation simply returns all values of k such that lambda = k(k-1)/(v-1) is an integer, where v is the order of `G`, and the resulting parameters v, k, lambda pass the Bruck-Ryser-Chowla test.

Example

```
gap> G := SmallGroup(31, 1);;
gap> PossibleDifferenceSetSizes(G);
[ 6, 10, 15 ]
```

2.2.2 DifferenceSetsOfSizeK

▷ `DifferenceSetsOfSizeK(G, k)` (function)

Returns a list of all difference sets up to equivalence in the group `G` that have size `k`.

Example

```
gap> G := SmallGroup(16, 9);;
gap> DifferenceSetsOfSizeK(G, 1);
[ [ 1 ] ]
```

2.2.3 DifferenceSumsOfSizeK

▷ `DifferenceSumsOfSizeK(G, N, k)` (function)

Returns a list of all difference sums up to equivalence in the group `G` mod its normal subgroup `N` that have size `k`.

Example

```
gap> G := SmallGroup(16, 8);;
gap> N := Subgroup(G, [G.3, G.4]);;
gap> DifferenceSumsOfSizeK(G, N, 1);
[ [ 1, 0, 0, 0 ] ]
```

2.3 Refining

Refining refers to the process of enumerating the preimages of a difference sum and filtering out preimages that are not themselves difference sets or sums. For each size k we know that the only difference sum of size k in G mod G is `[k]`. Starting with this difference sum, we successivly refine through a series of quotients of G to eventually reach the desired sums or sets. In the algorithm, we use `SomeRefinedDifferenceSets` (2.3.4) and `SomeRefinedDifferenceSums` (2.3.8) rather than

`AllRefinedDifferenceSets` (2.3.2) and `AllRefinedDifferenceSums` (2.3.6) since the former are faster and we only need at least one representative of each equivalence class since additional equivalent sums or sets will just be removed anyway.

2.3.1 RefiningSeries

▷ `RefiningSeries(G)` (function)

Returns a normal series for group `G`. Current implementation produces a chief series through a nontrivial normal subgroup of smallest possible size in `G`.

Example

```
gap> G := SmallGroup(8, 3);;
gap> List(RefiningSeries(G), N -> Size(N));
[ 8, 4, 2, 1 ]
```

2.3.2 AllRefinedDifferenceSets

▷ `AllRefinedDifferenceSets(G, N, difsums)` (function)

Returns a list of all difference sets that are preimages of difference sums contained in the list *difsums* of difference sums in group `G` mod its normal subgroup `N`. Difference sums in *difsums* are all assumed to be the same size.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N := Subgroup(G, [G.2, G.4]);;
gap> AllRefinedDifferenceSets(G, N, [[3,1,1,1], [2,2,2,0]]);
[ [ 1, 3, 2, 8, 4, 15 ], [ 1, 3, 2, 8, 9, 11 ], [ 1, 3, 2, 13, 4, 11 ],
  [ 1, 3, 2, 13, 9, 15 ], [ 1, 3, 6, 8, 4, 11 ], [ 1, 3, 6, 8, 9, 15 ],
  [ 1, 3, 6, 13, 4, 15 ], [ 1, 3, 6, 13, 9, 11 ], [ 1, 5, 2, 6, 4, 15 ],
  [ 1, 5, 2, 6, 9, 11 ], [ 1, 5, 2, 13, 4, 9 ], [ 1, 5, 2, 13, 11, 15 ],
  [ 1, 5, 6, 8, 4, 9 ], [ 1, 5, 6, 8, 11, 15 ], [ 1, 5, 8, 13, 4, 15 ],
  [ 1, 5, 8, 13, 9, 11 ], [ 1, 10, 2, 6, 4, 11 ], [ 1, 10, 2, 6, 9, 15 ],
  [ 1, 10, 2, 8, 4, 9 ], [ 1, 10, 2, 8, 11, 15 ], [ 1, 10, 6, 13, 4, 9 ],
  [ 1, 10, 6, 13, 11, 15 ], [ 1, 10, 8, 13, 4, 11 ], [ 1, 10, 8, 13, 9, 15 ],
  [ 3, 5, 2, 6, 4, 11 ], [ 3, 5, 2, 6, 9, 15 ], [ 3, 5, 2, 8, 4, 9 ],
  [ 3, 5, 2, 8, 11, 15 ], [ 3, 5, 6, 13, 4, 9 ], [ 3, 5, 6, 13, 11, 15 ],
  [ 3, 5, 8, 13, 4, 11 ], [ 3, 5, 8, 13, 9, 15 ], [ 3, 10, 2, 6, 4, 15 ],
  [ 3, 10, 2, 6, 9, 11 ], [ 3, 10, 2, 13, 4, 9 ], [ 3, 10, 2, 13, 11, 15 ],
  [ 3, 10, 6, 8, 4, 9 ], [ 3, 10, 6, 8, 11, 15 ], [ 3, 10, 8, 13, 4, 15 ],
  [ 3, 10, 8, 13, 9, 11 ], [ 5, 10, 2, 8, 4, 15 ], [ 5, 10, 2, 8, 9, 11 ],
  [ 5, 10, 2, 13, 4, 11 ], [ 5, 10, 2, 13, 9, 15 ], [ 5, 10, 6, 8, 4, 11 ],
  [ 5, 10, 6, 8, 9, 15 ], [ 5, 10, 6, 13, 4, 15 ], [ 5, 10, 6, 13, 9, 11 ] ]
```

2.3.3 NrAllRefinedSets

▷ `NrAllRefinedSets(G, N, difsums)` (function)

Returns the number of preimages that will need to be checked during a call to `AllRefinedDifferenceSets` (2.3.2) with the same arguments. This can give a rough estimate of how long the call will take to complete.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N := Subgroup(G, [G.2, G.4]);;
gap> NrAllRefinedSets(G, N, [[3,1,1,1], [2,2,2,0]]);
472
```

2.3.4 SomeRefinedDifferenceSets

▷ SomeRefinedDifferenceSets(*G, N, difsums*) (function)

Returns a list of some difference sets that are preimages of difference sums contained in the list *difsums* of difference sums in group *G* mod its normal subgroup *N*. At least one member of each equivalence class that would appear in the set of all preimages will be returned, but all preimage difference sets may not appear. Difference sums in *difsums* are all assumed to be the same size. Current implementation forces the choice of an identity element when possible.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N := Subgroup(G, [G.2, G.4]);;
gap> SomeRefinedDifferenceSets(G, N, [[3,1,1,1], [2,2,2,0]]);
[ [ 1, 3, 2, 8, 4, 15 ], [ 1, 3, 2, 8, 9, 11 ], [ 1, 3, 2, 13, 4, 11 ],
  [ 1, 3, 2, 13, 9, 15 ], [ 1, 3, 6, 8, 4, 11 ], [ 1, 3, 6, 8, 9, 15 ],
  [ 1, 3, 6, 13, 4, 15 ], [ 1, 3, 6, 13, 9, 11 ], [ 1, 5, 2, 6, 4, 15 ],
  [ 1, 5, 2, 6, 9, 11 ], [ 1, 5, 2, 13, 4, 9 ], [ 1, 5, 2, 13, 11, 15 ],
  [ 1, 5, 6, 8, 4, 9 ], [ 1, 5, 6, 8, 11, 15 ], [ 1, 5, 8, 13, 4, 15 ],
  [ 1, 5, 8, 13, 9, 11 ], [ 1, 10, 2, 6, 4, 11 ], [ 1, 10, 2, 6, 9, 15 ],
  [ 1, 10, 2, 8, 4, 9 ], [ 1, 10, 2, 8, 11, 15 ], [ 1, 10, 6, 13, 4, 9 ],
  [ 1, 10, 6, 13, 11, 15 ], [ 1, 10, 8, 13, 4, 11 ], [ 1, 10, 8, 13, 9, 15 ] ]
```

2.3.5 NrSomeRefinedSets

▷ NrSomeRefinedSets(*G, N, difsums*) (function)

Returns the number of preimages that will need to be checked during a call to SomeRefinedDifferenceSets (2.3.4) with the same arguments. This can give a rough estimate of how long the call will take to complete.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N := Subgroup(G, [G.2, G.4]);;
gap> NrSomeRefinedSets(G, N, [[3,1,1,1], [2,2,2,0]]);
300
```

2.3.6 AllRefinedDifferenceSums

▷ AllRefinedDifferenceSums(*G, N1, N2, difsums*) (function)

Returns a list of all difference sums in group *G* mod its normal subgroup *N2* that are preimages of difference sums contained in the list *difsums* of difference sums in group *G* mod its normal subgroup *N1*. The subgroup *N2* must be contained in *N1*. Difference sums in *difsums* are all assumed to be the same size.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N1 := Subgroup(G, [G.2, G.4]);;
gap> N2 := Subgroup(G, [G.2]);;
gap> AllRefinedDifferenceSums(G, N1, N2, [[3,1,1,1], [2,2,2,0]]);
[ [ 1, 1, 0, 1, 0, 1, 2, 0 ], [ 1, 1, 2, 1, 0, 1, 0, 0 ],
  [ 1, 0, 1, 1, 0, 2, 1, 0 ], [ 1, 2, 1, 1, 0, 0, 1, 0 ],
  [ 0, 1, 1, 2, 0, 1, 1, 0 ], [ 2, 1, 1, 0, 0, 1, 1, 0 ] ]
```

2.3.7 NrAllRefinedSums

▷ NrAllRefinedSums(*G, N1, N2, difsums*) (function)

Returns the number of preimages that will need to be checked during a call to AllRefinedDifferenceSums (2.3.6) with the same arguments. This can give a rough estimate of how long the call will take to complete.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N1 := Subgroup(G, [G.2, G.4]);;
gap> N2 := Subgroup(G, [G.2]);;
gap> NrAllRefinedSums(G, N1, N2, [[3,1,1,1], [2,2,2,0]]);
22
```

2.3.8 SomeRefinedDifferenceSums

▷ SomeRefinedDifferenceSums(*G, N1, N2, difsums*) (function)

Returns a list of some difference sums in group *G* mod its normal subgroup *N2* that are preimages of difference sums contained in the list *difsums* of difference sums in group *G* mod its normal subgroup *N1*. At least one member of each equivalence class that would appear in the set of all preimages will be returned, but all preimage difference sums may not appear. The subgroup *N2* must be contained in *N1* and difference sums in *difsums* are all assumed to be the same size. Current implementation forces a choice of nonzero identity coefficient when possible.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N1 := Subgroup(G, [G.2, G.4]);;
gap> N2 := Subgroup(G, [G.2]);;
gap> SomeRefinedDifferenceSums(G, N1, N2, [[3,1,1,1], [2,2,2,0]]);
[ [ 1, 1, 0, 1, 0, 1, 2, 0 ], [ 1, 1, 2, 1, 0, 1, 0, 0 ],
  [ 1, 0, 1, 1, 0, 2, 1, 0 ], [ 1, 2, 1, 1, 0, 0, 1, 0 ],
  [ 2, 1, 1, 0, 0, 1, 1, 0 ] ]
```

2.3.9 NrSomeRefinedSums

▷ NrSomeRefinedSums(*G, N1, N2, difsums*) (function)

Returns the number of preimages that will need to be checked during a call to SomeRefinedDifferenceSums (2.3.8) with the same arguments. This can give a rough estimate of how long the call will take to complete.

Example

```
gap> G := SmallGroup(16, 5);;
gap> N1 := Subgroup(G, [G.2, G.4]);;
gap> N2 := Subgroup(G, [G.2]);;
gap> NrSomeRefinedSums(G, N1, N2, [[3,1,1,1], [2,2,2,0]]);
21
```

2.4 Equivalence

Since we are searching for all difference sets or sums up to equivalence, at each stage we remove excess equivalent sums or sets from our collection. This can be done with `EquivalentFreeListOfDifferenceSets` (2.4.1) and `EquivalentFreeListOfDifferenceSums` (2.4.3). The additional functions `TranslateFreeListOfDifferenceSets` (2.4.2) and `TranslateFreeListOfDifferenceSums` (2.4.4) can be used to eliminate translate equivalent sums or sets, but they are not used in the main algorithm. Alternatively, `SmallestEquivalentDifferenceSet` (2.4.5) uses the `SmallestImageSet` function from the GRAPE package to produce the lexicographically minimal difference set equivalent to a given set. Eliminating equivalent sets can then be done by mapping each set to its minimal representative and then simply eliminating duplicates. This is done automatically by `SmallestEquivalentFreeListOfDifferenceSets` (2.4.6), which is used in the last stage of the main algorithm instead of `EquivalentFreeListOfDifferenceSets` (2.4.1). While the full algorithm with `SmallestEquivalentFreeListOfDifferenceSets` (2.4.6) is roughly 20% slower on average (and is almost 4x as slow on a few groups of order 64), this function is used since it is much faster on large automorphism groups (such as the automorphism group of `SmallGroup(64, 267)`, which is impossible with `EquivalentFreeListOfDifferenceSets` (2.4.1)) and provides a unique minimal result at the end of the algorithm.

2.4.1 EquivalentFreeListOfDifferenceSets

▷ `EquivalentFreeListOfDifferenceSets(`*`G, difsets`*`)` (function)

Returns a list of inequivalent difference sets in the group *`G`* that consists of one representative from each equivalence class found in the list *`difsets`* of arbitrary difference sets in *`G`*.

Example

```
gap> G := SmallGroup(16, 4);;
gap> sets := [[8,9,12,13,14,15], [7,8,9,13,15,16], [1,7,10,11,14,15]];;
gap> EquivalentFreeListOfDifferenceSets(G, sets);
[ [ 8, 9, 12, 13, 14, 15 ] ]
```

2.4.2 TranslateFreeListOfDifferenceSets

▷ `TranslateFreeListOfDifferenceSets(`*`G, difsets`*`)` (function)

Returns a list of translationally inequivalent difference sets in the group *`G`* that consists of one representative from each translational equivalence class found in the list *`difsets`* of arbitrary difference sets in *`G`*.

Example

```
gap> G := SmallGroup(16, 4);;
gap> sets := [[8,9,12,13,14,15], [7,8,9,13,15,16], [1,7,10,11,14,15]];;
gap> TranslateFreeListOfDifferenceSets(G, sets);
[ [ 8, 9, 12, 13, 14, 15 ], [ 7, 8, 9, 13, 15, 16 ] ]
```

2.4.3 EquivalentFreeListOfDifferenceSums

▷ EquivalentFreeListOfDifferenceSums(*G*, *N*, *difsums*) (function)

Returns a list of inequivalent difference sums in the group *G* mod its normal subgroup *N* that consists of one representative from each equivalence class found in the list *difsums* of arbitrary difference sums in *G* mod *N*.

Example

```
gap> G := SmallGroup(16, 4);;
gap> N := Subgroup(G, [G.1 * G.2 * G.3, G.3, G.4]);;
gap> EquivalentFreeListOfDifferenceSums(G, N, [[4,2], [2,4]]);
[ [ 4, 2 ] ]
```

2.4.4 TranslateFreeListOfDifferenceSums

▷ TranslateFreeListOfDifferenceSums(*G*, *N*, *difsums*) (function)

Returns a list of translationally inequivalent difference sums in the group *G* mod its normal subgroup *N* that consists of one representative from each translational equivalence class found in the list *difsums* of arbitrary difference sums in *G* mod *N*.

Example

```
gap> G := SmallGroup(16, 4);;
gap> N := Subgroup(G, [G.1 * G.2 * G.3, G.3, G.4]);;
gap> TranslateFreeListOfDifferenceSums(G, N, [[4,2], [2,4]]);
[ [ 4, 2 ] ]
```

2.4.5 SmallestEquivalentDifferenceSet

▷ SmallestEquivalentDifferenceSet(*G*, *D*) (function)

Returns the set that is lexicographically smallest among all sets that are equivalent to the difference set *D* in the group *G*.

Example

```
gap> G := SmallGroup(16, 4);;
gap> SmallestEquivalentDifferenceSet(G, [8,9,12,13,14,15]);
[ 1, 2, 3, 4, 8, 15 ]
```

2.4.6 SmallestEquivalentFreeListOfDifferenceSets

▷ SmallestEquivalentFreeListOfDifferenceSets(*G*, *difsets*) (function)

Returns a list containing the lexicographically smallest set for each set in the list of difference sets *difsets* in the group *G*. Duplicates are removed, so the returned list contains exactly one representative from each equivalence class found in *difsets*.

Example

```
gap> G := SmallGroup(16, 4);;
gap> sets := [[8,9,12,13,14,15], [7,8,9,13,15,16], [1,7,10,11,14,15]];;
gap> SmallestEquivalentFreeListOfDifferenceSets(G, sets);
[ [ 1, 2, 3, 4, 8, 15 ] ]
```

2.5 Testing

These additional functions are provided to check work and perform other experimentation. They are inefficient when used repeatedly. For example, when testing a large number of difference sets in a single group, it is better to precompute the needed group operations and store them in a table for lookup, but `IsDifferenceSet` (2.5.1) simply does the multiplication directly since it is only testing one set.

2.5.1 IsDifferenceSet

▷ `IsDifferenceSet(`*G, D*`)` (function)

Returns true if the set *D* is a difference set in the group *G*, and false otherwise.

Example

```
gap> G := SmallGroup(16, 4);;
gap> IsDifferenceSet(G, [1, 2, 3, 4, 5, 6]);
false
gap> IsDifferenceSet(G, [1, 2, 8, 10, 11, 15]);
true
```

2.5.2 IsDifferenceSum

▷ `IsDifferenceSum(`*G, N, S*`)` (function)

Returns true if the sum *S* is a difference sum in the group *G* mod its normal subgroup *N*, and false otherwise.

Example

```
gap> G := SmallGroup(16, 4);;
gap> N := Subgroup(G, [G.1 * G.2 * G.3, G.3, G.4]);;
gap> IsDifferenceSum(G, N, [2, 4]);
true
gap> IsDifferenceSum(G, N, [1, 1]);
false
```

2.5.3 IsEquivalentDifferenceSet

▷ `IsEquivalentDifferenceSet(`*G, D1, D2*`)` (function)

Returns true if sets *D1* and *D2* are equivalent in the group *G*, and false otherwise.

Example

```
gap> G := SmallGroup(16, 4);;
gap> IsEquivalentDifferenceSet(G, [1,5,8,9,10,14], [1,5,7,8,10,15]);
false
```

2.5.4 IsEquivalentDifferenceSum

▷ `IsEquivalentDifferenceSum(G, N, S1, S2)` (function)

Returns true if sums *S1* and *S2* are equivalent in the group *G* mod its normal subgroup *N*, and false otherwise.

Example

```
gap> G := SmallGroup(16, 4);;
gap> N := Subgroup(G, [G.1 * G.2 * G.3, G.3, G.4]);;
gap> IsEquivalentDifferenceSum(G, N, [2,4], [4,2]);
true
```

2.6 Loading Results

The `data` directory of the DifSets Package contains precomputed results for 1006 of the 1032 groups of order less than 100. The following two functions are the easiest way to access these precomputed lists of difference sets up to equivalence.

2.6.1 CanLoadDifferenceSets

▷ `CanLoadDifferenceSets(v, n)` (function)

Returns true if a precomputed list of all difference sets up to equivalence can be loaded from the package library for the group `SmallGroup(v, n)`, and false otherwise.

Example

```
gap> CanLoadDifferenceSets(36, 9);
true
gap> CanLoadDifferenceSets(79, 1);
false
```

2.6.2 LoadDifferenceSets

▷ `LoadDifferenceSets(v, n)` (function)

Returns the precomputed list of all difference sets up to equivalence for the group `SmallGroup(v, n)` stored in the package library. An error is thrown if no precomputed list is available. Note that the listed difference sets are specific to `SmallGroup(v, n)`, as GAP may label entries of other isomorphic versions of the same group differently.

Example

```
gap> LoadDifferenceSets(15, 1);
[ [ 1, 2, 3, 4, 8, 11, 12 ] ]
gap> G := SmallGroup(15, 1);; H := AbelianGroup([15]);;
gap> IdGroup(G) = IdGroup(H);
```

```
true
gap> IsDifferenceSet(G, [1, 2, 3, 4, 8, 11, 12]);
true
gap> IsDifferenceSet(H, [1, 2, 3, 4, 8, 11, 12]);
false
```

Chapter 3

Results

The DifSets Package was designed with the goal of finding all difference sets up to equivalence in groups of order 64 and 96, a goal which was accomplished. Overall, the algorithm has successfully computed results for 1006 of the 1032 groups of order less than 100. Full results, which include timings, number of sets, and the sets themselves can be found in the `data` subdirectory of the package, which is organized by group order and contains a single `.txt` file for each computed group. A list of all timings can also be found in the file `groups.csv` in the `data` directory, and the difference sets themselves can be loaded using the function `LoadDifferenceSets` (2.6.2). All computations were performed using GAP 4.9.1 on a 4.00GHz i7-6700K using 8GB of RAM. Here we give a basic overview of results and comments on timings. Throughout this chapter we will refer to the group returned by the GAP function `SmallGroup(v, n)` as `[v, n]`.

3.1 Order 16 and 36

Difference sets in groups of order 16 and 36 form the first nontrivial examples of the Hadamard parameters, and exhaustive enumerations are already well known. Still, computation of these sets gives a useful benchmark and check of accuracy.

Almost all groups in these orders take less than a second. The group `[36, 9]`, however, takes several orders of magnitude longer than other groups of order 36. This is because `[36, 9]` does not have small normal subgroups (in particular, its smallest nontrivial normal subgroup has order 9), and refining across a large gap in sizes, expecially near the end of the algorithm, requires checking significantly more preimages.

Group	Difference Sets	Time (seconds)
[16, 1]	0	0.030
[16, 2]	3	0.103
[16, 3]	4	0.100
[16, 4]	3	0.100
[16, 5]	2	0.061
[16, 6]	2	0.071
[16, 7]	0	0.072
[16, 8]	2	0.070
[16, 9]	2	0.082
[16, 10]	2	0.187
[16, 11]	2	0.121
[16, 12]	2	0.195
[16, 13]	2	0.117
[16, 14]	1	0.059

Group	Difference Sets	Time (seconds)
[36, 1]	0	0.335
[36, 2]	0	0.201
[36, 3]	0	0.407
[36, 4]	0	0.322
[36, 5]	0	0.218
[36, 6]	6	0.412
[36, 7]	1	0.795
[36, 8]	4	0.340
[36, 9]	5	340.989
[36, 10]	6	1.137
[36, 11]	3	0.699
[36, 12]	6	0.417
[36, 13]	1	0.801
[36, 14]	3	0.434

3.2 Order 64 and 96

Difference sets in groups of order 64 also satisfy the Hadamard parameters, while difference sets in groups of order 96 satisfy the McFarland parameters. Since there are many groups of both orders, here we just give some examples and summaries. In particular, the tables below list the fastest, slowest, and median five groups of each order, sorted by time.

Groups of order 64 are p-groups, and thus always have enough normal subgroups to form long refining series. This means the refining steps are relatively efficient for all groups in this order. The main difference between groups is the size of the automorphism group, and, in particular, four of the five groups taking the largest amount of time are four of the five groups with the largest automorphism groups in this order. The additional group in the top five, [64, 235], has a relatively large number of difference sets, but is otherwise unremarkable. In general, smaller numbers of difference sets correspond to faster times, and in fact the eight groups with no difference sets were computed the fastest, beating the next fastest groups by an order of magnitude. Overall, the mean computation time

for a group of order 64 was 3988.476 seconds, with a median time of 1493.175 seconds. This means that the total computer time to compute all difference sets in groups of order 64 was roughly 12 days.

In groups of order 96 we do not always have large numbers of normal subgroups, and, as with `[36, 9]`, this can substantially slow down computation. In fact, the five groups taking the longest computation time are five of the six groups with fewest normal subgroups in this order. We are helped, however, by the fact that the only valid choice of k is 20, which is relatively small and thus does not lead to large numbers of preimages even across large gaps in the refining series. Many groups in this order have no difference sets, but even for these groups computation can be slow. While the fastest groups contain no difference sets, many groups with no difference sets actually take much longer than other groups that do contain difference sets. Overall, the mean computation time for a group of order 96 was 24447.991 seconds, with a median time of 11278.765 seconds. This means that the total computer time to compute all difference sets in groups of order 96 was roughly 65 days.

Group	Difference Sets	Time (seconds)
`[64, 52]`	0	3.451
`[64, 54]`	0	3.463
`[64, 47]`	0	3.594
`[64, 186]`	0	3.940
`[64, 1]`	0	3.950
`[64, 166]`	2312	1424.692
`[64, 134]`	342	1439.484
`[64, 135]`	540	1493.175
`[64, 7]`	1320	1515.710
`[64, 160]`	3192	1518.693
`[64, 192]`	222	21131.394
`[64, 267]`	4	23662.500
`[64, 235]`	4317	24566.186
`[64, 260]`	30	144338.020
`[64, 262]`	148	229488.988

Group	Difference Sets	Time (seconds)
`[96, 2]`	0	8.731
`[96, 59]`	0	8.791
`[96, 189]`	0	29.378
`[96, 66]`	0	29.777
`[96, 46]`	0	44.478
`[96, 209]`	4	10809.673
`[96, 133]`	16	11198.052
`[96, 224]`	0	11278.765
`[96, 89]`	0	11349.466
`[96, 102]`	0	11415.688
`[96, 227]`	42	308246.830
`[96, 64]`	14	310447.407
`[96, 70]`	28	514559.313
`[96, 72]`	2	515196.547
`[96, 71]`	8	871439.024

3.3 Comments

Overall, the algorithm spends almost all of its time performing four operations: refining sums to sums in several stages using `SomeRefinedDifferenceSums` (2.3.8), refining sums to sets in the final stage using `SomeRefinedDifferenceSets` (2.3.4), removing equivalent difference sums in several stages using `EquivalentFreeListOfDifferenceSums` (2.4.3), and removing equivalent difference sets in the final stage using `SmallestEquivalentFreeListOfDifferenceSets` (2.4.6). On typical groups of order 16 and order 36 (i.e., not `[36, 9]`), each of these four operations takes roughly the same time. On groups of order 64, some testing indicates that one or two orders of magnitude more time are spent in the final stage, when the algorithm uses `SomeRefinedDifferenceSets` (2.3.4) and `SmallestEquivalentFreeListOfDifferenceSets` (2.4.6). This discrepency is likely to remain or increase for larger order groups, as the number of preimages to check increases exponentially with the number of cosets. For the tested groups of order 64, roughly 60% of the time in the final stage was spent refining, with the remaining 40% spent removing equivalent sets.

Large automorphism groups make removing equivalents time-consuming and large jumps in the size of the normal subgroups used, especially near the end of the algorithm, make refining difficult. So, in general, the algorithm seems to work well when the group has a small automorphism group and many (small) normal subgroups. In addition, the algorithm does better when the values of k that need to be checked are small, as this limits both the number of preimages to check as well as the amount of time required for checking sets and equivalences. It is also generally faster when the final result is a smaller number of difference sets.

There are twenty-six groups of order less than 100 in which the algorithm was not able to complete a search. Fourteen of these groups are prime order cyclic. As simple groups, these groups have no normal subgroups and thus no possibility for refining, which means the algorithm must search every possible subset of size k to find all difference sets of size k. Even for groups of relatively small order, such as order 31, this is infeasible, and with current implementation will overflow memory before even starting the search (one of these groups, `[37, 1]` is actually feasible to search without this implementation issue, but the others have too many sets to check). The remaining groups have either too few normal subgroups, large jumps in the refining series, large possible values of k, or a combination of these problems.

The next natural cases for exhaustive search are groups of order 100 and order 144, which give the next Hadamard parameters. Unfortunately, preliminary testing indicates that this algorithm is not likely to be able to compute all difference sets for these groups. For example, a typical difference sum in `[100, 9]` is `[5, 4, 3, 3, 0, 3, 2, 3, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 3]`, which has roughly 6×10^{16} preimage sets to check. In the search for difference sets in `[36, 9]` the single difference sum `[6, 3, 3, 3]`, with around 3×10^7 preimages, takes around 300 seconds to search. Thus even if we could check sets in `[100, 9]` as fast as in `[36, 9]`, the search would take roughly 20000 years. Some testing suggests that coding pieces of the algorithm in C could give one or two orders of magnitude of speedup, but even further speedup is required to make the search feasible, so some other improvements, either in theory or implementation, are needed as well.