ferret

Backtrack Search in Permutation Groups

1.0.9

18 October 2022

Christopher Jefferson

Christopher Jefferson
Email: caj21@st-andrews.ac.uk
Homepage: https://caj.host.cs.st-andrews.ac.uk/
Address: St Andrews
Scotland
UK

Copyright

Contents

Chapter 1

The Ferret Package

This chapter describes the GAP package Ferret. Ferret implements highly efficient implementations of a range of search algorithms on permutation groups. If you are interested in if Ferret can be applied to another problem, please contact the authors, who will be happy to look into if your problem can be solved with Ferret.

1.1 Replacing Built-in functionality

Ferret automatically installs methods which replace GAP's a number of GAP's built-in functionality:

- *Intersection* for a list of permutation groups.
- *Stabilizer(G,S,Action)* for a permutation group G, and the actions:
  - `OnSets`
  - `OnOnSets`
  - `OnSetsDisjointSets`
  - `OnSetsSets`
  - `OnTuples`
  - `OnPairs`
  - `OnDirectedGraph`
- *Stabilizer(G, S)* for a permutation group G and a:
  - permutation
  - transformation
  - partial permutation

If you would like to disable this functionality, you can use 1.1.1.

1.1.1 EnableFerretOverloads

▷ `EnableFerretOverloads(`*`[active]`*`)` (function)

if *`active`* (a bool) is true, then enable Ferret specialisations of Intersection and Stabilizer. Call with *`active`* false to disable.

1.1.2 FerretOverloadsEnabled

▷ `FerretOverloadsEnabled()` (function)

Return if Ferret specialisations of Intersection and Stabilizer are currently enabled.

1.2 Using 'Solve' to solve problems directly

The main method of using Ferret's functionality is the `Solve` (2.1.3) method. This method intersects a list of permutation groups. The unusual feature is that these permutation groups can be represented in a variety of ways. They can be usual GAP permutation groups given as a list of generators, or they can be the group which is the stabilizer of combinatorial object under some action. Larger problems are then composed from these pieces. For example, the stabilizer of a set S under a group G can be expressed as the intersection of the group which stabilizes the set S and the group G. For this problem, there would be no point using `Solve` (2.1.3), as GAP's built in 'Stabilizer' function provides the same functionality. However, with `Solve` (2.1.3) we can intersect any number of groups at the same time, for example finding the intersection of two groups G and H, the stabilizer of a set S and the stabilizer of a set of sets T, with the following code:

Example

```
gap> Solve([ConInGroup(G), ConInGroup(H),
> ConStabilize(S, OnSets), ConStabilize(T, OnSetSets)])
```

The currently allowed arguments to `Solve` (2.1.3) are:

- `ConInGroup` (2.1.2), which represents a Permutation Group in GAP
- `ConStabilize` (2.1.1), which takes an object and an action.

Chapter 2

The Solve Method

The central functionality of the Ferret package is based around the Solve method. This function performs a backtrack search, using the permutation backtracking algorithm, over a set of groups or cosets. Often users will want to use a higher level function which wraps this functionality, such as `Stabilizer` or `Intersection`. The solve function accepts a list of groups, and finds their intersection. For efficiency reasons, these groups can be specified in a variety of different ways. As an example, we will consider how to implement `Stabilizer(G, S, OnSets)`, the stabilizer of a set S in a permutation group G using Solve (this is not necessary, as when Ferret is loaded this method is replaced with a Ferret-based implementation). Another way of viewing `Stabilizer(G, S, OnSets)` is as the intersection of G with `Stabilizer(Sym(n), S, OnSets)`, where $Sym(n)$ is the symmetric group on n points, and n is at least as large as the largest moved point in G. Solve takes a list of objects which represent groups. Two of these are `ConInGroup(G)`, which represents the group G, and `ConStabilize(S, OnSets)`, which represents the group which stabilizes S. We find the intersection of these two groups by `Solve([ConInGroup(G), ConStabilize(S, OnSets)])`.

2.1 Methods of representing groups in Ferret

Groups and cosets must be represented in a way which Ferret can understand. The following list gives all the types of groups which Ferret accepts, and how to construct them.

2.1.1 ConStabilize (for an object and an action)

▷ `ConStabilize(`*`object, action`*`)` (function)
▷ `ConStabilize(`*`object, n`*`)` (function)

This function creates a Constraint which can be given to `Solve` (2.1.3). It does not perform any useful actions by itself

In the first form this represents the group which stabilises *`object`* under *`action`*. The currently allowed actions are `OnSets`, `OnSetsSets`, `OnSetsDisjointSets`, `OnSetsTuples`, `OnTuples`, `OnPairs` and `OnDirectedGraph`.

In the second form it represents the stabilizer of a partial perm or transformation in the symmetric group on *`n`* points.

2.1.2 ConInGroup

▷ `ConInGroup(`*`G`*`)` (function)

This function creates a Constraint which can be given to `Solve` (2.1.3). It does not perform any useful actions by itself

Represents the set of permutations in a permutation group *`G`*, as an argument for `Solve` (2.1.3).

These methods are both used with Solve:

2.1.3 Solve

▷ `Solve(`*`constraints[, rec]`*`)` (function)

Finds the intersection of the list *`Constraints`*. Each member of *`constraints`* should be a group or coset generated by one of `ConInGroup` (2.1.2) or `ConStabilize` (2.1.1). The optional second argument allows configuration options to be passed in. These follow options are supported:

`rbaseCellHeuristic` **(default "smallest")**
: The cell to be branched on. This is the option which will most effect the time taken to search. the default is usually best. Other options are: "First" (first cell), "Largest" (largest cell), "smallest2" (the 2nd smallest cell), "random" (a random cell) and "randomsmallest" (one of the smallest cells, chosen randomly)

`rbaseValueHeuristic` **(default "smallest")**
: Choose which cell to branch on within a cell. While this will generally make a big difference to search, it is hard to predict the best value, and small changes to the problem will change the best heuristic. Options are the same as `rbaseCellHeuristic`.

`searchValueHeuristic` **(default** `RBase`**)**
: The order to branch during search. In general the best order is very hard to predict. Options are "RBase", "InvRBase", "Random", "Sorted" or "Nosort" (which uses the order the values naturally end up in by the algorithm).

`searchFirstBranchValueHeuristic` **(default** `RBase`**)**
: Choose the search order used just on the left-most branches of search. Allows the same options as `searchValueHeuristic`

`stats` **(default** `false`**)**
: Change the return value to provide a range of information about how search performed (implies `recreturn`). This information will change between releases.

`nodeLimit` **(default** `false`**)**
: Either FALSE, or an integer which places a limit on the amount of search which should be performed. WARNING: When this option is set to an integer, Ferret will return the current best answer when the limit is reached, which may be a subgroup of the actual result. To know if this limit was reached, set `stats` to TRUE, and check the nodes.

`recreturn` **(default** `false`**)**
: Return a record containing private information, rather than the group.

`only_find_generators` **(default** `true`**)**
: By default only find the generators of the group. If false, then find all members of the group. This option is only useful for testing. If 'true', then sets 'recreturn' to true.

Chapter 3

Installing and Loading the Ferret Package

3.1 Unpacking the Ferret Package

If the Ferret package was obtained as a part of the GAP distribution from the "Download" section of the GAP website, you may proceed to Section 3.2. Alternatively, the Ferret package may be installed using a separate archive, for example, for an update or an installation in a non-default location (see (**Reference: GAP Root Directories**)).

Below we describe the installation procedure for the `.tar.gz` archive format. Installation using other archive formats is performed in a similar way.

It may be unpacked in one of the following locations:

- in the `pkg` directory of your GAP 4 installation;
- or in a directory named `.gap/pkg` in your home directory (to be added to the GAP root directory unless GAP is started with `-r` option);
- or in a directory named `pkg` in another directory of your choice (e.g. in the directory `mygap` in your home directory).

In the latter case one one must start GAP with the `-l` option, e.g. if your private `pkg` directory is a subdirectory of `mygap` in your home directory you might type:

```
gap -l ";myhomedir/mygap"
```

where *`myhomedir`* is the path to your home directory, which (since GAP 4.3) may be replaced by a tilde (the empty path before the semicolon is filled in by the default path of the GAP 4 home directory).

3.2 Compiling Binaries of the Ferret Package

After unpacking the archive, go to the newly created `ferret` directory and call `./configure` to use the default `../..` path to the GAP home directory or `./configure` *`path`* where *`path`* is the path to the GAP home directory, if the package is being installed in a non-default location. So for example if you install the package in the `~/.gap/pkg` directory and the GAP home directory is `~/gap4r5` then you have to call

Example

```
./configure ../../../gap4r5/
```

This will fetch the architecture type for which GAP has been compiled last and create a `Makefile`. Now simply call

Example

```
make
```

to compile the binary and to install it in the appropriate place.

3.3 Loading the Ferret Package

To use the Ferret Package you have to request it explicitly. This is done by calling `LoadPackage` (**Reference: LoadPackage**):

Example

```
gap> LoadPackage("ferret");
true
```

If you want to load the Ferret package by default, you can put the `LoadPackage` command into your `gaprc` file (see Section (**Reference: The gap.ini and gaprc files**)).

Index