GAPE: Genetic Algorithm for Pharmacophore Elucidation

Gareth Jones

Table of Contents

Table of Contents

1 Background

GAPE is a pharmacophore elucidation algorithm based on the GASP[1] program. GAPE[2] uses a Genetic Algorithm (GA) to align structures and a fitness score to optimize that alignment so as to identify a common pharmacophore. The pharmacophore is comprised of features such as donor hydrogens, acceptor atoms and planar rings. The fitness score includes terms for matching features and common molecular volume. In a GA a set of encoded problem representations called chromosomes compete over time. When the GA terminates the fittest chromosome is usually a good solution to the problem under investigation. Since the GA uses stochastic sampling multiple overlays and pharmacophore hypotheses may be created from sequential GA runs using different random number sequences. These may be ranked or sorted using the GAPE scoring function to create a final prediction.

2 Directory Structure

The GAPE distribution is merged with the DIFGAPE distribution. The directory structure for the distribution is as follows:

- target. Java build.
- bin. For convenience, Unix shell scripts for running common Java programs
- example_config_files. Example and stock configuration or parameter files.
- doc. Documentation.
- Examples/gape. Example results and configuration files for the test systems in the GAPE paper.

3 Required Software and Libraries

GAPE uses numerous Apache commons (http://commons.apache.org/) l libraries. These libraries are covered by the Apache 2 license which allows libraries to be incorporated into commercial products. GAPE no longer uses SVD or simplex code from Numerical Recipies- these have been replaced by routines from the Apache Commons Math library. The native libraries included in older releases have been replaced by FastMath functions from the Commons math library and calls to Java management beans for CPU timings. Also see the section about torsional distributions below. It would be desirable to commercialize with the GOLD torsional distributions included.

4 Installation

This section assumes that you are familiar with setting environment variables and navigating your file system. GAPE is a Java program, so you need to have Java installed. Use the Oracle/Sun standard JDK/J2SE version 1.8 or later. Though GAPE will run under a normal Java JRE, the standard edition is strongly recommended over the JRE as running the server VM (java -server) gives a 30% speed up for GAPE. The server VM is not normally available in the JRE.

GAPE no longer makes use of JNI (native code) for faster math operations and to get CPU timings.

5 Structure File Format

These programs handle both MOL2 and SDF structure files. Wherever a reference to a MOL2 file appears you can normally substitute an SDF file (and vice versa). GAPE expects input compounds to be 3-D structures. The software prefers that hydrogens be added, but if they are absent then GAPE will attempt to fill valence by adding protons. Following this step GAPE will automatically ionize groups which are normally charged at physiological pH and identify formal charges.

Output files are in the same format as input files. Multi-mol files are normally used to define all input structures and each overlay is written out to a mutli-mol file. The output file will include SD entries (or MOL2 comments) indicating pharmacophore points. Additionally, the output file will contain a molecule comprising of query or dummy atoms to represent the pharmacophore.

The GA is run a number of times. Given SDF file output, the hypothesis for the first run is

placed in the multi-mol file *GA_001.sdf*, the second result is placed in *GA_002.sdf* and so on. Upon completion of the job, the highest ranking hypothesis is placed in *GA_ranked_001.sdf*, the second rank solution is places in *GA_ranked_002.sdf* and so on. Thus the prediction from a GAPE job will always be found in *GA_ranked_001.sdf*.

The program has been extensively tested on SD files created by OpenEye and MOE software and MOL2 files created by SYBYL.

6 Configuration File

The main directory contains a configuration file called *superposition.conf*, which contains example GAPE settings (the same file can also be found in the main java jar file *gape.jar* at *com/cairn/gape/utils/superposition.conf*). The file contains considerable documentation for all the different settings. For the most part the user should be fine keeping current settings. However the following settings may be of interest:

- *n_runs*: the number of hypotheses generated. By default it is set to 10. For the validation[2] *n_runs* was set to 100 or 25.
- *guess_ga_parameters*: indicates that GAPE will override the values of the next three parameters (*popsize*, *n_iterations* and *n_islands*). These 3 parameters control the size of GA search space and the time taken by the algorithm. Setting *guess_ga_parameter* to true results in reasonable values for these parameters based on the number of compounds in the overlay. *popsize* is 100, *n_islands* is *n_molecules*+1 and *n_iterations* is *n_molecules*×15000. This setting was used in the validation[2].
- *n_islands*: The GA uses the island model where there are *n_islands* sub-populations.
- Each sub-population contains *popsize* chromosomes. So the total number of chromosomes in the system is *popsize*×*n_islands*.
- *n_iterations* is the total number of genetic operators applied over the length of the GA run. So each subpopulation will have *n_iterations*/*n_islands* operations applied.
- *directory*. By default GAPE is run in the current directory. Use this parameter to have GAPE run in a different directory. This is required if multiple GAPE jobs are running in a single process (see below).
- *tordist_file.* Specifies the torsional distribution file (see below).
- *user_features_file_1*: use this setting to extend GAPE with you own features. See the file *example_user_features.txt* in the main directory for more information.

7 Torsional Distribution File

Like GOLD[3], GAPE[2] is able to avoid sampling poor torsions by using a torsional distribution file so that, for defined torsions, only experimentally observed torsion angles are sampled. Two torsional distribution files are included in the *gape.jar* file: the distributions from MIMUMBA[4] (*com/cairn/gape/molecule/TORDIST.MIMUMBA.txt*) and the distributions used in GOLD (*com/cairn/gape/molecule/TORDIST.GOLD.txt).* By default *GAPE* uses the MIMUMBA

distributions. However, the GOLD distributions were found to give superior results in the validation. These distributions are property of the CCDC. Permission was kindly granted to use the GOLD distributions for non-commercial validation of the algorithm. Licensed users of GOLD or the CCDC may use the GOLD distributions for commercial use within GAPE. To do this extract *TORDIST.GOLD.txt* from the jar file to the file system and set the *tordist_file* parameter to point to it.

8 Command Line Usage

```
java -server com.cairn.ga.Superposition <configuration_file>
<structure_files>
```

Runs GAPE and creates results. Program progress is documented in the file *ga.out*.

9 Multi-Threaded Program Usage

GAPE is thread safe (though it is fair to say that this has not been rigorously tested) and thus you can run multiple GAPE jobs in a single process. The following test program is available for this:

```
java -server com.cairn.gape.utils.test.MultiThreadGape
<configuration_files>
```

This runs multiple GAPE threads: one for each configuration file argument. Each configuration file should use the *molecules* parameter to specify input structure files and the *directory* parameter to ensure each thread has its own unique output directory.

The multi-threaded model was created to allow the management of multiple GAPE jobs from a single supervisory program. A listener to monitor the progress of each thread can be attached to the GAPE job in that thread. The class *com.cairn.gape.utils.test.MultiThreadGape* contains example code.

10 Other Programs

.10.1 Pharm Search

An additional program is available to screen a database of compounds against a rigid query or queries. This is equivalent to multiple GAPE runs, each of which compares the same rigid and fixed queries against a different flexible database molecule. Once the last database molecule has been searched the GAPE target function can be used to rank database compounds against the queries.

```
java -server -Xmx1024M com.cairn.gape.PharmSearch <configuration_file> <query_file>
<structure_files>
```

The structures contained in *query_files* are the rigid and fixed queries. These can be GAPE overlays. The structures in *structure_files* are database compounds. Once the program is completed an ordered list of best target conformations is saved in *Matches.sdf*, while all GAPE overlays are contained in *All_Solutions.sdf* (or *Matches.mol2* and *All_Solutions.mol2* if

MOL2 file format is used in the input files).

An example configuration file *pharm_search.conf* can be found in the main directory or in the *gape.jar* file at *com/cairn/gape/utils/pharm_search.conf.*

.10.2 GRIPS (GAPE RIgid Pharmacophore Search)

```
java -server -Xmx1024M com.cairn.gape.RigidPharmSearch
<configuration_file> <query_file> <conformer_files>
```

A third program matches a query against a compressed conformer library. Clique detection is used to match features in the query against conformers. If at least three points match then the conformer can be fitted on the query and the overlay scored using the GAPE scoring function (using a feature match score and the volume overlay score).

The configuration file contains special settings for *GRIPS*. An example configuration file called *rigid_search.conf* can be found in the main directory and also in the *gape.jar* file at *com/cairn/gape/utils/rigid_search.conf*. The conformer files can be gzipped. Ordered hits are output in *Matches.sdf* (or *Matches.mol2* if MOL2 file format is used in the input files).

.10.2.1 GRIPS Queries

Three types of query are available. Firstly, you can use a single structure and not include any pharmacophore. In this case all features in the single query structure are used. To select this use the *no_pharmacophore* setting in the configuration file. This option will also be chosen automatically if you only have one query structure and it has no pharmacophore.

Secondly you can use the results of a GAPE run as a query. In this case the query features will be selected from the base molecule pharmacophore (you can change this by setting *base_molecule_only* to no- in this case pharmacophore features from all molecules in the overlay will be considered) and all molecules in the overlay contribute to the volume score. Obviously, using these options you don't have to use the complete GAPE overlay as a query. You could extract just the base molecule (or another molecule from the overlay) and use it as a query.

Finally, the query can just include a pharmacophore (in a .pharm file). It his case the scoring function will only include a feature match term- the volume score will be 0.

You can create pharmacophore queries from GAPE overlays. Use this to extract the pharmacophore information from a GAPE overlay:.

```
java com.cairn.gape.feature.Pharmacophore -extract-pharm
GA_ranked_001.sdf ga.pharm
```

.10.2.2 GRIPS Performance

Grips can spend much more time reading and preparing input structures than searching for matches. There are some things that can be done to improve performance. Firstly, it's best to solvate and add hydrogens before running GRIPS, rather than have GRIPS do this. So turn off these options in the configuration file. Conformer libraries can be prepared using this command:

```
java com.cairn.molecule.MoleculeUtils prepare mols13.sdf.gz
mols13_prepared.sdf.gz
```

Secondly, GRIPS will run a lot faster if in knows that one structure is just a new conformer of the previous structure. This is implemented for the OpenEye conformer naming scheme (conformer names are like conformer_1, conformer_2 or conformer_1_rconf, conformer_2_rconf etc) using the *openeye_conformers* setting.

11 Utilities

```
java com.cairn.gape.utils.config.ConfigEditor
```

Provides a graphical front end for editing configuration files.

```
java com.cairn.molecule.MoleculeUtils
```

Performs a number of useful transforms on structure files (for exmaple, add hydrogens, solvate or remove lone pairs)

```
java com.cairn.molecule.PatternMatch
```

Searches an sln pattern against a mol2 file.

```
java com.cairn.intranet.molecule.MolViewer
```

Takes mol2 or sdf files and displays them in a simple viewer.

12 References

GAPE was originally presented at CUP 2005. The abstract and presentation are in the documentation directory:
G Jones, “A Genetic Algorithm for Flexible Molecular Overlay and Pharmacophore Elucidation”, *OpenEye Scientific CUP User Group Meeting,* Santa Fe, NM (2005)

Other related algorithms of interest are DIFGAPE:

G. Jones, Y. Gao and C. R. Sage, “Elucidating Molecular Overlays from Pairwise Alignments Using a Genetic Algorithm”, *Journal of Chemical Information and Modeling* 49 (2009) 1847–1855.

Cited References

1. Jones, G.; Willett, P.; Glen, R. C. A genetic algorithm for flexible molecular overlay and pharmacophore elucidation. *J Comput. Aided Mol Des* **1995,** *9*, 532-549.

2. Jones, G. GAPE: an improved genetic algorithm for pharmacophore elucidation. *J Chem Inf. Model* **2010,** *50,* 2001-2018.

3. Jones, G.; Willett, P.; Glen, R. C.; Leach, A. R.; Taylor, R. Development and

validation of a genetic algorithm for flexible docking. *J Mol Biol* **1997,** *267*, 727-748.

4. Klebe, G.; Mietzner, T.; Weber, F. Methodological developments and strategies for a fast flexible superposition of drug-size molecules. *J Comput. Aided Mol Des* **1999,** *13*, 35-49.

Gareth Jones