Test Report: Kaplan

Jen Garner

April 20, 2019

1 Revision History

Date	Version	Notes
December 10, 2018 (Monday)	1.0	write review

2 Symbols, Abbreviations and Acronyms

See the Reference Tables in the System Requirements Specification (SRS) (Section 2) documentation for the repository: https://github.com/PeaWagon/Kaplan.

symbol	description
T	Test
NFR	Non-functional requirement
R	Requirement
HPC	high-performance computing
FT	Functional test
NFT	Non-functional test
pmem	Population member
BS	basis set
QCM	quantum chemical method
CI	continuous integration

Contents

This document is a review of the system tests that have been performed on Kaplan. This report is a partner to the Unit VnV Report that is located in the docs/VnVReport/UnitVnVReport directory.

3 Review of Planning and Future Work

This section will review the System VnV Plan (found in the SystVnVPlan Section 4). Since not all testing has been done to validate and verify the code, this section serves the purpose of being a "to-do" list for future work.

3.1 SRS

The plan is still to test various fitness functions to see if an optimal form exists (or at least to optimize the coefficients C_E and C_{RMSD}) for a few given input molecules. The author's supervisor has provided feedback on the validity of the assumptions that were made in the SRS. This feedback can be found in the github issue tracker. From this feedback, dihedral angles are not fully representative of the molecule if taken from a zmatrix file. Therefore, the plan to test different orders of atomic inputs and search for conformers may be thrown out in favour of rewriting a module to use proper dihedral angles. Most of the SRS issues that were raised by Brooks MacLachlan were addressed and closed.

3.2 Design

Many changes were made to the module guide (MG) document, including closing github issues raised by classmates Malavika Srinivasan, Oluwaseun Owojaiye, and Karol Sekis. The original set of 11 modules was increased to 13 to avoid multiple secrets being contained by a single module - specifically, the pmem and geometry modules were added. The module instance specification (MIS) document must be changed such that the state variables reflect the definition of a state variable.

The design has yet to be reviewed by the author's supervisor; a review will be done after all github issues have been resolved from classmates and Dr. Spencer Smith. Still to implement in the design - use of fit_form (ga_input parameter) and prog (mol_input parameter), since these are currently place-holder variables that do not do anything in the code.

The ring module will be updated (meaning its design will be updated in the MIS) to include extinction operators. These extinction operators are designed to return the ring to a more exploratory state once it becomes filled. The code has been written such that addition of these extinction operators will be relatively simple to implement. Some examples of extinction operators are:

- **virus**: infect n parents and kill the segment +/- within those parents (if parents have fitness lower than x).
- **plague**: kill lowest fraction of population (arranged by fitness).
- **asteroid**: select random segment and delete all within that segment.
- **deluge**: set a "waterlevel" for the fitness; all below that waterlevel are killed.
- **agathic**: kill the oldest fraction of the population.

3.3 Implementation

Still to do for implementation verification is to fix all of the linting issues raised by the Travis build (see the UnitVnVReport for details). All tests for external libraries have been run, with the exception of rmsd and pubchempy. Other test results are discussed in Section 9 of this document.

4 Functional Requirements Evaluation

R1 is the requirement that the program can read inputs:

- n_G is given as num_geoms in the ga_input module.
- C_E is given as coef_energy in the ga_input module.
- C_{RMSD} is given as coef_rmsd in the ga_input module.
- BS is given as basis in the mol_input module.
- QCM is given as qcm in the mol_input module.

- The molecular geometry is specified using struct_input and struct_type in the mol_input module. The charge and multiplicity of the molecule are also given (as charge and multip) in the mol_input module.

R2 is the requirement that the inputs can be used to solve for the fitness function Fit_G. The initial geometry should be used to generate potential solutions (in the form of sets of dihedral angles). The mol_input and ga_input modules both have verification functions to ensure that the inputs can be used to calculate the fitness function. Furthermore, the pmem module is a data structure that generates dihedral angles randomly (so that potential solutions to the conformer optimization are generated).

R3 requires the energy for each conformer and the rmsd for each conformer pair be calculated - an energy module and an rmsd module have been written for these purposes. The fitg module combines these values such that the fitness function can be calculated.

R4 requires that the energy calculations converge. Part of the verification of the mol_input module is to ensure that the initial geometry can be run and that the given basis set and method are in the chosen program. Fit_G is required to be positive, and this requirement is satisfied by making the coefficients positive and by forcing the user to use fit_form = 0 (the same as the fitness instance model in the SRS).

R5 is that the output geometries are generated, which is satisfied by the use of the geometry module and the output module. External libraries that were used to help with this requirement include vetee, openbabel, and pubchempy. The ring data structure from the ring module has an attribute called zmatrix that keeps track of the original geometry specification, such that we can confirm the ordering has not changed.

R6 is that there are templates for the user to follow. They are provided in the test/testfiles directory and are called example_mol_input_file.txt and example_ga_input_file.txt.

R7 is that the conformers are optimized to maximize fitg. The tournament module selects a random number of solutions from the ring. Then it choses the two pmems from the tournament that have the highest fitness and uses those to generate new solutions for the ring. In this way, the value of fitg can become optimized, since only the best pmems are chosen for reproduction.

5 Nonfunctional Requirements Evaluation

NFR1 states that Kaplan should be robust with regards to changes made to the fitness function. The program has been written such that more fitness functions can be added via the fit_form input variable in the ga_input. The code maintainer would then have to add the fitness function to the fitg module and ensure that the inputs were constrained such that the fitness would be positive.

NFR2 states that the code should be maintainable. Having written the documentation in detail and provided a github repository should satisfy this requirement. The code has also been explained to two fellow group members in the author's lab, such that more people understand how the code should work.

NFR3 is that the program should be:

- parallelisable (not yet satisfied under current conditions) - this requirement could be satisfied by implementing python's multiprocessing module
- capable of running on high performance computing servers (not yet satisfied - in progress of being installed on Compute Canada's cluster, graham)
- without a difficult install process - so far this program has been successfully installed on 4 systems (Ubuntu on Windows, Ubuntu, Kbuntu) and tested on Ubuntu.

Originally, the program was planned to be made into a conda package. This is still the plan, but for now it is installed using pip install (since the build is not yet passing).

NFR4 is that the program should be easy to use and quick to explain to chemists. A discussion of this requirement is covered by Section 5.1.

NFR5 states that the program should work well with other quantum chemistry packages. Similar to the NFR1 regarding robustness with respect to fitness calculation, a placeholder input variable (called prog) in the mol_input module will be used to specify the program the energy module should use to perform quantum chemical calculations. Right now, the only acceptable input is psi4, which was chosen to ensure that proprietary software was avoided (thus making the package easy to install for all users). In

the future, Horton, Gaussian, and others may be added. These additions will change which functions are called from the energy module by the ring module (and more functions will therefore need be added to the energy module). Since the current state of the package can interact with openbabel, vetee, psi4, and pubchempy without issue, then it is considered to partially satisfy this non-functional requirement.

5.1 Usability

To make the program easy to use, the interaction with the source code has been removed. The user does not need to know any coding to run the program. Instead, the user must specify their inputs in two plain text files. The user can provide a variety of input types, which means the program can be used even if an exact geometry is not known (for example, a smiles string can be used to generate a geometry using openbabel).

5.2 Performance

The Travis Continuous Integration (CI) for Kaplan takes about 20 minutes to run (which includes installation and running the tests). The performance bottleneck is most likely the running of the psi4 program for the energy calculations. Performance is not yet a focus of the program.

6 Comparison to Existing Implementation

There is no existing implementation of Kaplan, therefore this section is not relevant.

7 Unit Testing

The unit testing will be covered in more details in the UnitVnVReport. The actual implementation of the unit tests can be found in the kaplan/test/ directory on the github repo https://github.com/PeaWagon/Kaplan.

8 Changes Due to Testing

The actual System VnV Plan tests will be discussed henceforth. FT1-1 has been mostly satisfied by the unit tests that were written for the input modules. FT1-2 will be improved; currently the tests for checking basis set and method availability have been by a try accept loop in python where (if the external program throws an error with the inputs) Kaplan throws an error. Instead, a complete list of available basis sets and methods are being prepared such that the user of Kaplan can reference these documents. FT2-1 has not been completed, but will be added to the unit tests (so far only values between 0 and 1 have been tested for the coefficients). FT2-2 and FT2-3 have been tested using jupyter notebooks (which in some cases have made it into the unit tests).

From the System VnV Plan, NFT1 can actually be disregarded. The original geometry is stored in the program to minimize the error when converting dihedral angles back into full geometry specifications. NFT2 has not yet been completed, but (as mentioned in the first paragraph of this section) the groundwork has been written into the code to enable such tests to be completed. NFT3 testing, as mentioned in the discussion of NFR3, for n cores can be completed when the code is parallelized. NFT3 testing on a HPC cluster can be completed once a successful install is performed on graham (still to do).

9 Automated Testing

The testing was done using nosetests https://nose.readthedocs.io/en/latest/. A Travis CI build for the code was made by committing the repository containing a .travis.yml file. Each time a commit is made, the build is started and the author gets progress reports by email. Currently, the builds are "failing", because none of the linters are passing. To see the builds, here is the link: https://travis-ci.org/PeaWagon/Kaplan.

The external testing for psi4 was run using nosetests. The test file was found in the psi4 directory for the kenv conda environment:

```
~/miniconda3/envs/kenv/lib/python3.6/site-packages/psi4/tests/test_psi4.py
```

The results from running this test file have also been added to the Kaplan repository under kaplan/test/external-tests in a file called nose-psi4-

test-result.txt. 2 out of 6 tests failed (errors for scf-property and dfmp2-1). Since the tests were run from a random file and not from the compiled version of the code, it is unknown as to their coverage and/or relevance.

For vetee, the tests were run using nosetests. The results from running these tests are in the kaplan/test/external-tests directory in a file called nose-vetee-test-result.txt. There were 5 failures over 43 tests total. Since this code is a work in progress by the author and another lab member, vetee is expected to improve its testing output.

For openbabel, the tests were run using make for openbabel version 2.4.1 that was compiled on the author's ubuntu machine. Of 160 tests, none failed. The results for the tests can be found under kaplan/test/external-tests/make-openbabel-test-result.txt.

It should be noted that psi4, openbabel, vetee, numpy, rmsd, and pubchempy are installed via a conda environment in order to run Kaplan. Therefore, these tests are not run using the same version that might be used for a separate installation. These results are meant to show that the external libraries in use have tests and that they can be run.

The tests for pubchempy can be found here: https://github.com/mcs07/PubChemPy/blob/master/tests/test_requests.py

The tests for rmsd can be found here: https://github.com/charnley/rmsd/blob/master/tests.py

10 Trace to Requirements

This section will be covered in the Unit VnV Report.

11 Trace to Modules

This section will be covered in the Unit VnV Report.

12 Code Coverage Metrics

Current code coverage metrics can be found in the Travis CI build. For example here: https://travis-ci.org/PeaWagon/Kaplan/builds/465865809 we have a code coverage of:

Name	Stmts	Miss	Cover
kaplan/__init__.py	12	0	100%
kaplan/energy.py	33	1	97%
kaplan/fitg.py	30	1	97%
kaplan/ga_input.py	46	1	98%
kaplan/gac.py	25	5	80%
kaplan/geometry.py	56	2	96%
kaplan/mol_input.py	59	5	92%
kaplan/mutations.py	30	0	100%
kaplan/output.py	31	1	97%
kaplan/pmem.py	11	0	100%
kaplan/ring.py	109	7	94%
kaplan/rmsd.py	11	0	100%
kaplan/tournament.py	30	0	100%
TOTAL	483	23	95%