Computation for Drug Discovery

by

David Martindale and Jason Zou

A Major Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science

by

David Martindale Jason Zou

May 2021

APPROVED:

_____________________
Smith, Therese
Abstract

The goal of this project is to investigate inefficiencies in the functionality of Retrosim, chief among them being its inability to process large molecules. To accomplish this goal, we created a call tree to map out how functions are connected when Retrosim runs. We also conducted tests to measure the efficiency of all of Retrosim’s functions to determine their runtimes. Our work is intended to be the foundation for future improvements and eventual automation of Retrosim.
List of Tables

4.1 Timing Analysis ............................................. 7
Chapter 1

Problem Description

Retrosim is a code base for an automated approach to retrosynthesis, a technique for solving organic synthesis problems that involves breaking molecules down into simpler precursor structures recursively until simple or commercially available structures are reached. It evaluates where to perform disconnections with a similarity based approach circumventing the need of setting a generalization level from past approaches. This allows it to be a data-driven and deterministic method.

A few major issues exist in the current iteration of the Retrosim code. The first is that it is written in the now outdated Python 2. The second issue is the fact that it is incapable of running its functions on larger molecules. Lastly, since we were not the original authors of the code, we began the project unaware of the structure of the code, the code’s call tree, or the purpose of each individual function in the program.

The first issue: the fact that the code is written in Python 2, can have an impact on the code’s overall efficiency. Python 3 code is designed to be more streamlined and accessible than its predecessor. This, combined with the fact that Python 2 has reached the end of its life, are reasons that make converting to Python 3 in our best interest. In addition, Python 3 enables the use of various chemistry related functions, such as the Tanimoto metric. Because of these problems with Python 2 and the improvements provided by Python 3, converting the code to the newest version is integral.
The second issue is that the code simply fails to perform its function on larger molecules. The purpose of retrosynthesis is to break down large molecules into smaller, easier to find ones. If the code fails to break down a very large molecule, it has essentially failed in its purpose. The reason for this failure lies in the fact that the code times out on these large molecules due to long run time. Because of this issue, it is important to find out the run time of each function in the code, locate the ones that take the longest, examine them for possible runtime improvements, and then implement those improvements.

Lastly, we are not familiar with the code. We do not know how the code works, which functions call others, and how each portion of the code impacts other portions. The solution to this is to discover a call tree, which would help us see how the code interacts with itself, and help us learn more about how we can improve the code as a whole.
Chapter 2

Methodology

2.1 Investigating Retrosim

The first step in our strategy was to gather information about each function. This included how long it took to run as well as the call tree of the functions to understand whether and if so, where, any bottlenecks are being created. We accomplished this by adding timing and printing statements in the functions python code so whenever we ran a function we could take those timings and copy them over to a text file that we could then analyze to extract required information.

After investigating the timing and use of each function we were able to focus on optimizing those functions most important to reducing run time and increasing usability.

The development environment used to modify retrosim was pycharm inside Anaconda. Anaconda was used primarily to create a separate environment that is running in python 2.7 to modify the code and easily switch to contemporary versions of python and back during later stages of development. Anaconda Navigator was used to manage the development environment and its packages on a windows system vs linux. In order to run the retrosim code several packages needed to be installed. This included jupyterlab, matplotlib, tqdm, joblib and when the anaconda environment was created with this command conda create -c rdkit -n my-rdkit-env rdkit python=2.7 the rdkit package was included.
Chapter 3

Related Work

3.1 General

There are three kinds of (computer-aided syntheses planning) programs. The first is done by hand encoding rules, the second is algorithms that build rules from known reactions, and the third is sequence to sequence and similarity based methods. Retrosim focuses on this second approach building a set of rules after first extracting reaction sequences which is where RDchiral comes in.

3.2 RdChiral

RDChiral performs the step of getting templates from a list of reactions. This is important when you are doing computer-aided synthesis planning because there is currently no standardized way of doing it. This is a problem for reproducibility and determinism: one of the goals of Retrosim. RDChiral is a wrapper on the RDKit library which focuses on converting a set of atom-mapped SMILES strings to generalized SMARTS patterns.
Chapter 4

Results and Analysis

4.1 Timing Analysis

After successfully obtaining the data for each function in a convenient CSV file, we were able to analyze the data for each function, and all of the functions as a whole.

In total, Retrosim contains 30 functions. Of those thirty, four are run over two thousand times: vprint, bond_to_label, atoms_are_different, and find_map_num. This is important, as even small inefficiencies in run time for these functions will be magnified due to the sheer number of times they are called.

There are also certain functions that have abnormally long run-times. Of the times get_changed_atoms runs over .1 ms, the average run time is .07 seconds. Another function of concern is rdchiralRun, which has an average of .02 seconds. These functions require further examination, and should be the first to receive tweaks to improve efficiency. These functions are very likely to be the reason certain runs of retrosim eventually time out.

For further analysis we created a call tree dynamically, with pycallgraph. This tree gave us further insight into the problem, confirming that vprint and bond_to_label were the main sources of delay. Upon further investigation of these functions vprint almost exclusively prints out results. This is something we believe could be optimized, to only print the final result. We also realized that our code which was
printing out the timings of our functions for, our first analysis could be causing further delay.

Employing pycallgraph on a clean version of the code resulting in a more comprehensive call tree. The resulting calltree shows that most of the functions in Retrosim run rather quickly. There were two exceptions: vprint and bond_to_label. The vprint function is mainly used to print information to the console, and is called over twenty thousand times in a single run of Retrosim. Because printing is an extremely time consuming task in Python, it would be sensible to remove any references to vprint in the program before running it, as vprint does not lend any true functionality to Retrosim outside of debugging purposes. The bond_to_label function takes the important information about certain chemical bonds, such as atomic number, and generates a label for it. This function is likely to be inefficient due to the number of dot calls it makes to other functions. Simply getting the atomic number of a bond requires this function to reference four others. Unlike vprint, however, bond_to_label is used by the atoms_are_different function to determine if two bonds are the same. This functionality may be crucial when distinguishing certain molecules. The optimal solution would be to somehow replicate the functionality of bond_to_label without the numerous dot calls and references, or find a separate, more efficient way to distinguish bonds and omit the bond_to_label function altogether.
Table 4.1: Timing Analysis

<table>
<thead>
<tr>
<th>Function Description</th>
<th>Times Started</th>
<th>Times Ended</th>
<th>Mean</th>
<th>Stdv</th>
</tr>
</thead>
<tbody>
<tr>
<td>vprint</td>
<td>2509</td>
<td>2509</td>
<td>0.001072</td>
<td>0.001077</td>
</tr>
<tr>
<td>initialize_reactants_from_smiles</td>
<td>2</td>
<td>2</td>
<td>0.0025</td>
<td>0.000707</td>
</tr>
<tr>
<td>mols_from_smiles_list</td>
<td>200</td>
<td>200</td>
<td>0.001914</td>
<td>0.000612</td>
</tr>
<tr>
<td>get_tagged_atoms_from_mol</td>
<td>266</td>
<td>266</td>
<td>0.001017</td>
<td>0.000131</td>
</tr>
<tr>
<td>get_tagged_atoms_from_mols</td>
<td>200</td>
<td>200</td>
<td>0.001559</td>
<td>0.003092</td>
</tr>
<tr>
<td>bond_to_label</td>
<td>8352</td>
<td>8352</td>
<td>0.001163</td>
<td>0.001715</td>
</tr>
<tr>
<td>atoms_are_different</td>
<td>2061</td>
<td>2061</td>
<td>0.00218</td>
<td>0.003101</td>
</tr>
<tr>
<td>find_map_num</td>
<td>2061</td>
<td>2061</td>
<td>0.001123</td>
<td>0.001503</td>
</tr>
<tr>
<td>get_tetrahedral_atoms</td>
<td>100</td>
<td>100</td>
<td>0.01705</td>
<td>0.007698</td>
</tr>
<tr>
<td>set_isotope_to_equal_mapnum</td>
<td>266</td>
<td>266</td>
<td>0.001123</td>
<td>1.04E-07</td>
</tr>
<tr>
<td>clear_isotope</td>
<td>266</td>
<td>266</td>
<td>0.001217</td>
<td>0.001976</td>
</tr>
<tr>
<td>get_changed_atoms</td>
<td>100</td>
<td>100</td>
<td>0.07752</td>
<td>0.023042</td>
</tr>
<tr>
<td>get_strict_smarts_for_atom</td>
<td>400</td>
<td>400</td>
<td>0.001211</td>
<td>0.001835</td>
</tr>
<tr>
<td>get_fragments_for_changed_atoms</td>
<td>200</td>
<td>200</td>
<td>0.003625</td>
<td>0.002442</td>
</tr>
<tr>
<td>expand_changed_atom_tags</td>
<td>100</td>
<td>100</td>
<td>0.001</td>
<td>1.09E-07</td>
</tr>
<tr>
<td>canonicalize_template</td>
<td>200</td>
<td>200</td>
<td>0.001314</td>
<td>0.00224</td>
</tr>
<tr>
<td>canonicalize_transform</td>
<td>100</td>
<td>100</td>
<td>0.001901</td>
<td>0.003604</td>
</tr>
<tr>
<td>reassign_atom_mapping</td>
<td>100</td>
<td>100</td>
<td>0.09381</td>
<td>1.16E-07</td>
</tr>
<tr>
<td>process_an_example</td>
<td>100</td>
<td>100</td>
<td>0.09381</td>
<td>0.023818</td>
</tr>
<tr>
<td>initialize_rxn_from_smarts</td>
<td>100</td>
<td>100</td>
<td>0.001515</td>
<td>0.002062</td>
</tr>
<tr>
<td>get_template_fras_from_rxn</td>
<td>100</td>
<td>100</td>
<td>0.001</td>
<td>1.08E-07</td>
</tr>
<tr>
<td>template_atom_could_have_been_tetra</td>
<td>708</td>
<td>708</td>
<td>0.001232</td>
<td>0.001952</td>
</tr>
<tr>
<td>expand_atoms_to_use</td>
<td>166</td>
<td>166</td>
<td>0.001</td>
<td>1.04E-07</td>
</tr>
<tr>
<td>check_tetrahedral_centers_equivalent</td>
<td>10</td>
<td>10</td>
<td>0.001111</td>
<td>0.000333</td>
</tr>
<tr>
<td>atom_chirality_matches</td>
<td>103</td>
<td>103</td>
<td>0.001029</td>
<td>0.00017</td>
</tr>
<tr>
<td>copy_chirality</td>
<td>792</td>
<td>758</td>
<td>0.001006</td>
<td>8.06E-05</td>
</tr>
<tr>
<td>canonicalize_outcome_smiles</td>
<td>36</td>
<td>36</td>
<td>0.001389</td>
<td>0.002004</td>
</tr>
<tr>
<td>rdchiralRun</td>
<td>100</td>
<td>100</td>
<td>0.027127</td>
<td>0.043166</td>
</tr>
<tr>
<td>convert_atom_toWildcard</td>
<td>4</td>
<td>4</td>
<td>0.001</td>
<td>0</td>
</tr>
<tr>
<td>get_frag_around_tetrahedral_center</td>
<td>10</td>
<td>10</td>
<td>0.001</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 5

Further Work

Beyond the work we have accomplished, there are a variety of additional tasks that can be performed to further optimize and comprehend Retrosim. The most laborious task is to convert the entire program to Python 3. Python 3 is generally better optimized, and is still receiving regular updates. Python 3 also contains various libraries that may help optimize runtime speed further. The main issue with this approach is the extensive amount of code that would need to be converted, as well as the additional bug testing required to ensure the code maintains its original functionality.

An additional improvement that can be implemented is the optimization of Retrosim's loops, which can be inefficient in Python. One solution is to use lazy evaluation and generators so that evaluation is only performed when necessary. Another potential solution is to substitute C libraries in place of slower Python processes. Loops execute more quickly in C, allowing for even faster runtimes. There also might be opportunities for early pruning but this would need some expertise in chemistry to create a useful heuristic.
Appendices
Appendix A

Call Trees

A.1 Call Tree One

This is the link to the call tree, before we removed the timing print statements we added. The image is also shown below in sections. https://i.imgur.com/5isKPBY.png

A.2 Call Tree Two

This is the link to the call tree, after we removed the timing print statements we added. The image is also shown below in sections. https://i.imgur.com/05VdEaf.png
templates.get_changed_atoms: 99 calls; 50001s

check_tetrahedral_centers_equivalent: 21 calls; 2001s

retrosim.get_fragments_around_tetrahedral_center: 21 calls; 36999s

retrosim: 295 calls; time: 0.53s
retro_templates.process_an_example
calls: 81
time: 2.244000s

62

81

Is from smiles list
retrosim.utils.generate_retro_templates.example

calls: 81
time: 0.005000s

mplates.reassign_atom_mapping

calls: 81
time: 0.12999s

retrosim.utils.generate_retro

call
time: 0