AiZynthFinder: A Fast Robust and Flexible Open-Source Software for Retrosynthetic Planning

Samuel Genheden, Amol Thakkar, Veronika Chadimova, Jean-Louis Reymond, Ola Engkvist, Esben Jannik Bjerrum

Submitted date: 11/06/2020 • Posted date: 15/06/2020
Licence: CC BY-NC-ND 4.0
Citation information: Genheden, Samuel; Thakkar, Amol; Chadimova, Veronika; Reymond, Jean-Louis; Engkvist, Ola; Bjerrum, Esben Jannik (2020): AiZynthFinder: A Fast Robust and Flexible Open-Source Software for Retrosynthetic Planning. ChemRxiv. Preprint. https://doi.org/10.26434/chemrxiv.12465371.v1

We present the open-source AiZynthFinder software that can be readily used in retrosynthetic planning. The algorithm is based on a Monte Carlo tree search that recursively breaks down a molecule to purchasable precursors. The tree search is guided by an artificial neural network policy that suggests possible precursors by utilizing a library of known reaction templates. The software is fast and can typically find a solution in less than 10 seconds and perform a complete search in less than 1 minute. Moreover, the writing of the code was guided by a range of software engineering principles such as automatic testing, system design and continuous integration leading to robust software. The object-oriented design makes the software very flexible and can straightforwardly be extended to support a range of new features. Finally, the software is clearly documented and should be easy to get started with. The software is available at http://www.github.com/MolecularAI/aizynthfinder.

File list (1)
Genheden2020 - AiZynthFinder- a fast robust and flexib... (654.41 KiB)  view on ChemRxiv  » download file
AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning

Samuel Genheden*, Amol Thakkar1,2, Veronika Chadimová1, Jean-Louis Reymond2, Ola Engkvist1, Esben Bjerrum1*

1 Hit Discovery, Discovery Sciences, R&D, AstraZeneca Gothenburg, Sweden
2 Department of Chemistry and Biochemistry, University of Bern, Freiestrasse 3, 3012 Bern, Switzerland.

* Corresponding authors: samuel.genheden@astrazeneca.com, esben.bjerrum@astrazeneca.com

Abstract

We present the open-source AiZynthFinder software that can be readily used in retrosynthetic planning. The algorithm is based on a Monte Carlo tree search that recursively breaks down a molecule to purchasable precursors. The tree search is guided by an artificial neural network policy that suggests possible precursors by utilizing a library of known reaction templates. The software is fast and can typically find a solution in less than 10 seconds and perform a complete search in less than 1 minute. Moreover, the writing of the code was guided by a range of software engineering principles such as automatic testing, system design and continuous integration leading to robust software. The object-oriented design makes the software very flexible and can straightforwardly be extended to support a range of new features. Finally, the software is clearly documented and should be easy to get started with. The software is available at http://www.github.com/MolecularAI/aizynthfinder.

Introduction

Synthesis planning is the process by which a chemist or a computer determines how to synthesize a target compound. This is typically carried out by retrosynthesis where the desired compound is iteratively broken down into intermediates or smaller precursors until known or purchasable molecules have been found. Such analysis was pioneered by Corey and was traditionally carried out by hand or by using expert systems utilizing hand-encoded rules [1, 2, 3]. With the rise of deep learning, in the last couple of years, the field of retrosynthetic software tools has undergone a swift change. Now, sophisticated and automatic algorithms have the potential to provide retrosynthetic analysis to a broader application domain with better accuracy [4, 5, 6].

Retrosynthesis planning algorithms can be divided into template-based and template-free approaches. In template-based approaches, reaction templates or rules that describe chemical transformations are manually encoded or derived from a database of known reactions, and subsequently applied to other compounds to create plausible reaction outcomes. Segler et al. showed that it was possible to train a neural network to prioritize templates, and subsequently use this as a policy to guide a Monte Carlo tree search algorithm that suggests synthetic pathways for a given compound [7, 8]. Template-free approaches, on the other hand, do not rely on such templates but typically treat the chemical reaction as a natural language problem, where one set of words (the reactants) are transformed into another set of words (the products) [9, 10].
There are several tools available for retrosynthesis planning but to our knowledge only two are fully open source, i.e. the ASKCOS suite of programs from MIT [11] and LillyMol from Eli Lilly and Company [12]. The tools Chemical AI [13] and IBM RXN [14] are free for registered users, but only the algorithm of the latter has been reported in the literature. Other tools [15, 16, 17, 18, 19] are closed commercial applications where the algorithm is partly undisclosed. This is partly a problem of data availability – most of the reaction databases or manually encoded rules are commercial and limits the way a free and open source software can use them. The same applies to the database of purchasable precursors that is used as a stop criterion in several programs. However, we believe that the scientific community would benefit from an open source implementation that provides algorithmic transparency and promotes reproducible research with sustainable software. Therefore, we present the AiZynthFinder tool in this paper that can be used for retrosynthesis planning. An early version of this tool has been used previously to determine the influence of the reaction database on the retrosynthesis predictions [20], but the code base has been re-engineered to make it more flexible, robust and maintainable. We provide a trained neural network policy as well as tools to make a database of purchasable precursors so that the tool can be used directly. In addition, we provide extensive documentation to lower the learning curve for new users. We envisage that by providing this tool free and open-source, other researchers can use it for benchmarking, contribute to a continuous development and, suggest synthetic routes for novel compounds.

Implementation

The AiZynthFinder software is written in Python 3 and is distributed on GitHub under the MIT license [21]. It is dependent on approximately 15 freely available Python packages such as TensorFlow [22], RDKit [23] and NetworkX [24]. We highly recommend installing it in an anaconda environment due to the dependency on RDKit.

The central algorithm of the AiZynthFinder software has been described elsewhere [8, 20] and therefore, we only provide a brief outline here: The input is a molecule that will be broken down to purchasable precursors. The outcome will be a list of precursors that can be purchased or molecules that cannot be broken down by the algorithm. The software is based on a Monte Carlo tree search [25], where each node in the tree corresponds to a set of molecules that can or cannot be broken down further. At each iteration a leaf node is selected that is deemed to be most promising to exploit further using upper confidence bound statistics [25]. A neural network policy is then used to shortlist reaction templates and prioritize which child to create by applying a reaction template to create the new precursors. This procedure is repeated until a terminal state has been reached, i.e., a precursor that is purchasable has been found, or the tree has reached a maximum depth. At this point the score of the leaf node is back propagated up to the root of the tree (the input molecule), and the next iteration commences. The tree search is terminated either after a fixed number of iterations or a time-limit has passed.

The structure of the AiZynthFinder package is shown in Figure 1a. The main interface to the algorithm is in the aizynthfinder.py module, which brings classes from the mcts sub-package together to perform the tree search. However, for the end-user we provide two interfaces: one command-line interface (CLI) and one graphical user interface (GUI) that is to be used in a Jupyter notebook. These two interfaces, which reside in the interface sub-package, are installed together with the package. The sub-package training contains tools to train the policy neural network, and the sub-package tools contains other useful CLIs.

The overall design follows principles from object-oriented programming such that each component is implemented as a class. The main classes for the tree search and their relationships are shown in Figure 2.
1b. The AiZynthFinder class load a user configuration from file as a Configuration object, which includes the creation of a Policy and a Stock object. This configuration is then used to control the tree search. The actual tree search is then carried out by the TreeSearch class that creates a Node object representing a node in the tree search that can be expanded to create new Nodes. The molecules on each Node are represented by a State object that holds a list of TreeMolecule objects. A Reaction class encapsulate a chemical reaction on TreeMolecule objects and is used to apply the reaction templates to create new precursors.

![Figure 1](image_url) – the AiZynthFinder package. a) The python package structure, outlining top-level modules and sub-packages. b) the main classes involved in the tree search and the relationships. A line ending with a solid diamond indicates an "owns"-relation, and a line ending with arrow indicates an "uses"-relationship.

The Policy class encapsulates a recommendation engine based on a trained neural network. Given a molecule object, it will return a sorted list of reaction templates and the probability of each template. The templates are sorted based on the probability as given by the neural network. We have trained neural networks based on several template libraries and provide one based on the publicly available US patent office data (USPTO) set [26] for anyone to use. We also provide tools to train the neural network, in case one has their own or in-licensed reaction database. These tools can for instance be used with RDChiral [27] and our previously described procedure [20] for extracting templates.

The Stock class requires a little further explanation: it is an abstraction around a collection of compounds that serves as stop-conditions for the tree search. Typically, this is a list of purchasable compounds, but could in principle be an abstract collection based on some rules, e.g. compounds with
less than 7 carbon atoms are considered purchasable. To support different kinds of collections, the Stock class uses one or more query classes that given a molecule object returns whether that compound is “in stock”. The package comes with two query classes, one that holds a set of InChI keys [28] in memory and one that holds a connection to a Mongo database with InChI keys. We also provide examples how one could create a rule-based query class. We are using lists of purchasable compounds from several commercial vendors, but it is just as straightforward to create a list from open source databases such as ZINC [29]. To simplify this process, we provide a tool to make a stock in a suitable format for the tree search from files containing SMILES strings [30].

More than 85% of the code is covered by automatic unit and integration tests, which we execute on each commit. Furthermore, the code is pep8 compliant, autoformatted and code complexity is monitored automatically on each commit. All of this contributes to the robustness and maintainability of the code base and provides the basis for continuous integration. Extensive API documentation is autogenerated from docstrings and is complemented by hand-written tutorials.

Results and discussion
As described in the Implementation section, there are two main interfaces to the tool: a CLI and a GUI to be used with a Jupyter notebook. Here, we exemplify the usages of the tool with the GUI and then proceed with a comparison using the CLI. In the example below we have used the policy trained on USPTO data [20]. Furthermore, we created a stock from compounds available in the ZINC database [29]; we only downloaded tranches including fragment compounds (molecular weight up to 250 D and log P up to 3.5) that had reactivity labeled as “standard” or “reactive”, resulting in 17,422,831 compounds.

Graphical user interface
To use the GUI (and the CLI), a configuration file needs to be created in YAML-format. This configuration file must at least contain the path to files for the policy and instructions how to setup the stock. The policy files are 1) the saved neural network model and, 2) a list of reaction templates. Multiple stocks and policy networks can be specified in the configuration and selected in the GUI before running the algorithm. The user is also free to fine tune the search algorithm using a set of properties. For the GUI, they serve as default values whereas for the CLI they are used in the search algorithm. If not provided in the configuration file, recommended settings are automatically applied.

To start the GUI, open a Jupyter notebook and execute two to three lines of Python code. Alternatively, these lines can be autogenerated the first time. When these lines of code are executed, the GUI is built and displayed to the user. A Jupyter notebook might seem like a suboptimal choice for non-technical researchers, but we believe that the amount of commands one must enter and the amount of python code you must write is very minimal. A Jupyter notebook is also ideal as a working environment for researchers that want to dig a little deeper in to the algorithm and the result of the tree search. Because, a Jupyter notebook provides the full python environment, one can easily customize the algorithm and fully inspect the predicted routes. Projects such as voilá [31] aim to make it easy to create interactive webpages directly from Jupyter notebooks, which can provide a fast way to provide an interface for users that primarily wants to use the tool to find suggestions for synthesis plans.

In Figure 2, we have input the SMILES string for the antiviral drug Amenamevir. Furthermore, the user can then select the stock and neural policy they want to use, as well as some options for the tree search. For most users the default options are recommended.
Figure 2 – the input section of the AiZynthFinder GUI. A user has entered the SMILES string for the drug Amenamevir and selected the ZINC stock.

When the tree search is completed, the user can view the predicted reaction routes. The GUI allows browsing through the top-ranked routes, but it is possible with a few lines of python code to display all routes. Figure 3 shows an example for the Amenamevir drug. First, the results show whether the route is solved or not, i.e. if all precursors are in stock, and the score of the route. Currently, the score reflects the fraction of solved precursors and the number of reactions required to synthesize the target compound. Second, the results clearly display what precursors to procure in order to synthesize the target compound. Lastly, it shows the predicted route with precursors in stock highlighted with a green rectangle, and precursors not in stock highlighted with orange. In the example shown in Figure 3, we see that suggested route is very similar to the reported route for Amenamevir [32], with the exception that the alkylated aniline is available to purchase and does not need to be synthesized.
Comparison with ASKCOS
The CLI comes with some additional features that are lacking from the GUI. Foremost, it allows compounds to be processed in batch, i.e. the user can submit predictions on hundreds or thousands of compounds with one command. Secondly, detailed results are stored to disc that later can be processed or viewed. For instance, one can calculate statistics on search tree, or one can produce images of the top-ranked routes. Lastly, the CLI allows a finer detail of debugging information, which could be invaluable to software developers.

As mentioned above, there exists several other retrosynthesis tools, but unfortunately very few of them are open source or even well described in the literature. The software that is closest for a comparison is the Tree builder module in the ASKCOS suite of programs [11]. First the algorithm underlying the Tree builder module is similar to the algorithm of AiZynthFinder, although different expansion policies are used, and the search tree constructed differently. The software is written in Python and the code is available on Github. However, it is foremost intended for end-users and the interface is web-based. LillyMol [12], which is the other open-source code, uses a completely different and much simpler algorithm and is thus less interesting to compare to. To make a rough baseline comparison between ASKCOS and AiZynthFinder we selected 100 random compounds from the ChEMBL database and submitted them to the Tree builder module of the public ASKCOS web server [33]. Even though this might not represent the latest version of the codebase, it is intuitively the
interface that most people would use. We set a max depth of 6, an expansion time of 120 s and used a fast filter; otherwise default values were applied. We used the AiZynthFinder CLI together with the ZINC stock and the USPTO policy to predict routes for the same 100 compounds. Some statistics on source code and the route finding are collected in Table 1 and the full data is available as supporting information.

Table 1 – Statistics of AiZynthFinder and ASKCOS predictions on 100 compounds from ChEMBL.

|                          | AiZynthFinder | ASKCOS |
|--------------------------|---------------|--------|
| Number of core statements | 1095          | 2336   |
| Number of total statements| 1495          | 9987   |
| Reaction database        | USPTO [26]    | Reaxys [34] |
| Stock                    | ZINC [29]     | Sigma and eMolecules [6] |
| Average search time (s)  | 38.7          | 151.0  |
| Average solution time (s)| 7.1           | 14.3   |
| Number of solved routes  | 54            | 62     |
| Average number of steps  | 2.4           | 3.3    |
| Average number of precursors | 2.7          | 3.2    |

1 The number of python statements in the modules that are used by the AiZynthFinder CLI and tree builder module, respectively. 2 The total number of python statements in the aizynthfinder and makeit (ASKCOS) package, respectively. 3 The average time to find one solution.

We see that AiZynthFinder and ASKCOS find routes for 54 and 64 compounds, respectively. There were 47 compounds for which both tools found a route, 15 compounds where ASKCOS found a solution and AiZynthFinder did not, and 8 compounds where AiZynthFinder found a solution and ASKCOS did not. There were 30 compounds that neither tool found a solution for. In our experience, route finding capabilities depends a lot on the stock that is used as stop criteria in both tools. The example stock created from a subset of the ZINC database is for instance much less extensive than some of the commercial stocks we typically use. If we for instance include the readily available Enamine building blocks in the stock, we could find routes for an additional 10 compounds. The ASKCOS tool from the public webserver is employing a commercial database consisting of 107,000 compounds with list prices less than $100/g from Sigma and eMolecules [6]. The other factor that determines if a solution is found is the template library – here we used USPTO policy for AiZynthFinder, whereas ASKCOS is based on the more extensive Reaxys database [34]. Furthermore, the capability to find a route for both tools is closely related to the complexity of the synthesis. This can be seen in Figure 4, where we have plotted the distribution of the synthetic accessibility (SA) score [35] for four sets of data. We see that for both AiZynthFinder and ASKCOS, the SA score is generally lower for compounds that the tools were able to find a solution for. Similar observations have been discussed previously in the literature [36]. It seems that ASKCOS is slightly better at finding solutions with a mid-range SA score, but this could be due to the lack of some scaffolds in the ZINC stock. Moreover, it seems that AiZynthFinder predicts slightly shorter reaction routes, with fewer purchasable precursors, although it is unclear if the difference is statistically significant given the rather small test set.

We see that AiZynthFinder is much faster than ASKCOS, both in terms of total search time and the time to find one solution, although this difference could be partially attributed to the environment in which
the test was executed, a local Linux computer in the case of AiZynthFinder and a webserver in the case of ASKCOS. Lastly, we want to point out that the AiZynthFinder is a much slimmer code base than ASKCOS, with at least half the number of python statements in the core modules (the part of the code necessary to execute the tree search). The large difference in total statements of the package could to a large extent be attributed to the fact that ASKCOS has a lot more features than AiZynthFinder. However, we believe that the difference in the number of core statement is because we re-engineered the AiZynthFinder package such that it is a better designed package than the previously released code. This is an advantage if one wants to extend the package with new functionality – the less code there is to understand, the faster it will be to extend it.

![Figure 4](image)

**Figure 4** – distribution of the synthetic accessibility score of the 100 Chembl compounds, grouped by whether a synthetic route was found with AiZynthFinder or ASKCOS.

This is far from a comprehensive comparison that is more intended to highlight the similarities and differences between the two tools. As mentioned above, it is very hard to compare the software on equal footing. Different researchers have different priorities when it comes to retrosynthesis, and it is not entirely clear how to make a good comparison. We have for instance not discussed the quality of the predicted routes, but this is an ill-defined metric. For instance, we submitted Amenamevir to the ASKCOS webserver and did not recover the expected literature route, but that does not mean that the route suggested by ASKCOS is incorrect. The only fair way to find out is to synthesize the compounds according to the proposed route. As such, a comprehensive comparison of tools is out of scope for this paper.

**Future developments**

In our view, the AiZynthFinder software provides a framework for research and development of novel retrosynthesis algorithms. Therefore, we have designed the software so that it is easy to extend with new features. Currently, it contains a solid foundation, i.e., the Monte Carlo tree search algorithm that has shown promising results in finding routes for a range of compounds. And we provide interfaces that suits this core activity. However, it does not yet provide a fully integrated solution. For instance, we are working on improving the accuracy of the predicted routes by implementing a scoring framework. It also of interest to augment the predictions with an information retrieval system for the used templates, so that chemists can e.g. look up similar reactions. Finally, we are working on improving the recommendation policy, by for instance utilizing the “ring breaker” policy [37]. All such
extensions are rather straightforward to implement in the current codebase. If the features are useful and freely usable by a larger community, they will be made available when we publish new research findings. We expect minor releases with new features to happen several times a year, whereas patch releases fixing bugs and trivial code updates will be released continuously.

Conclusions
We have presented the AiZynthFinder tool for retrosynthesis planning. In our experience, it can suggest synthetic routes for most compounds in a very short time. We hope that it will be perceived as user-friendly and with a low learning curve, because we provide extensive documentation. Furthermore, the software is robust and flexible and lend itself to easy extension with novel features. Although it does not provide a complete and integrated solution to synthesis planning, we believe that we have with this software provided a framework and platform where novel algorithms can be tested and integrated in the future. We hope that by releasing the software to the public that researchers interested in retrosynthesis can use it to explore synthetic route prediction and provide suggestion how it can be improved. By providing open source code and algorithmic transparency, we aim to promote collaboration around a sustainable reference software. We encourage users to contribute ideas or code so that the tool can be incrementally improved and thereby provide more accurate and useful predictions of reaction routes.

Availability and requirements
- Project name: AiZynthFinder
- Project home page: http://www.github.com/MolecularAI/aizynthfinder
- Operating system(s): Platform independent
- Programming language: Python 3
- Other requirements: approximately 15 open source python packages
- License: MIT

Any restrictions to use by non-academics: none

Data availability
The complete output from the tree search underlying Table 1 is available as Supporting material. The ZINC stock as well as the trained USPTO policy is available to download from Figshare.

Funding
Amol Thakkar was supported financially by the European Union’s Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement no. 676434, “Big Data in Chemistry” (“BIGCHEM,” http://bigchem.eu).

Acknowledgements
We thank Dr. Michael E. Fortunato, Dr. Conner W. Coley and Prof. Klavs F. Jensen for helpful comments and clarifications regarding the ASKCOS software.
References

1. Corey EJ, Todd Wipke W (1969) Computer-assisted design of complex organic syntheses. Science 166:178–192. https://doi.org/10.1126/science.166.3902.178
2. Pensak DA, Corey EJ (1977) LHASA—Logic and Heuristics Applied to Synthetic Analysis. In Computer-Assisted Organic Synthesis, American Chemical Society. 61:1–32
3. Ihlenfeldt W-D, Gasteiger J (1996) Computer-Assisted Planning of Organic Syntheses: The Second Generation of Programs. Angew Chemie Int Ed English 34:2613–2633. https://doi.org/10.1002/anie.199526131
4. Engkvist O, Norrby O, Selmi N, et al (2018) Computational prediction of chemical reactions: current status and outlook. Drug Discov Today 23:1203-1218. https://doi.org/10.1016/j.drudis.2018.02.014
5. Coley CW, Green WH, Jensen KF (2018) Machine Learning in Computer-Aided Synthesis Planning. Acc Chem Res 51:1281–1289. https://doi.org/10.1021/acs.accounts.8b00087
6. Coley CW, Thomas DA, Lummis JAM, et al (2019) A robotic platform for flow synthesis of organic compounds informed by AI planning. Science 365, eaax1566. https://doi.org/10.1126/science.aax1566
7. Segler MHS, Preuss M, Waller P (2018) Planning chemical syntheses with deep neural networks and symbolic AI. Nature. 555:604-610. https://doi.org/10.1038/nature25978
8. Schwaller P, Laino T, Gaudin T, et al (2019) Molecular Transformer: A Model for Uncertainty-Calibrated Chemical Reaction Prediction. ACS Cent Sci 5:1572–1583. https://doi.org/10.1021/acscentsci.9b00576
9. Zheng S, Rao J, Zhang Z, et al (2020) Predicting Retrosynthetic Reactions Using Self-Corrected Transformer Neural Networks. J Chem Inf Model 60:47–55. https://doi.org/10.1021/acs.jcim.9b00949
10. Coley CW, Barzilay R, Jaakkola TS, et al (2017) Prediction of Organic Reaction Outcomes Using Machine Learning. ACS Cent Sci 3:434–443. https://doi.org/10.1021/acscentsci.7b00646
11. Watson IA, Wang J, Nicolaou CA (2019) A retrosynthetic analysis algorithm implementation. J Cheminform 11:1. https://doi.org/10.1186/s13321-018-0323-6
12. https://Chemical.AI
13. https://rxn.re.rib.com/
14. https://opensource.org/licenses/MIT
15. https://www.elsevier.com/solutions/reaxys/how-reaxys-works/synthesis-planner
16. https://rdkit.org
17. https://molecule.one/
18. https://www.infochem.de/synthesis/ic-synth
19. https://www.sigmaaldrich.com/chemistry/chemical-synthesis/synthesis-software.html
20. Thakkar A, Kogej T, Reymond J-L, et al (2019) Datasets and their influence on the development of computer assisted synthesis planning tools in the pharmaceutical domain. Chem Sci. 11:154–168. https://doi.org/10.1039/C9SC04944D
21. https://opensource.org/licenses/MIT
22. Abadi M, Agarwal A, Barham P, et al (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems
23. RDKit: Open-source cheminformatics, http://www.rdkit.org.
24. Haberg AA, Schult DA, Swart PJ (2008) Exploring Network Structure, Dynamics, and Function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy2008), ed. G. Varoquaux, T. Vaught and J. Millman, Pasadena, CA USA. pp. 11–15.
25. Browne CB, Powley E, Whitehouse D, et al (2012) A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games 4:1–43
26. D. Lowe, Chemical reactions from US patents, 1976–Sep 2016, https://figshare.com/articles/Chemical_reactions_from_US_patents_1976-Sep2016_/5104873, accessed 31-04-2018.
27. Coley CW, Green WH, Jensen KF (2019) RDChiral: An RDKit Wrapper for Handling Stereochemistry in Retrosynthetic Template Extraction and Application. J Chem Inf Model 59:2529–2537. https://doi.org/10.1021/acs.jcim.9b00286
28. Heller SR, McNaught A, Pletnev I, et al (2015) InChI, the IUPAC International Chemical Identifier. J Cheminform 7:23. https://doi.org/10.1186/s13321-015-0068-4
29. Sterling T, Irwin JJ (2015) ZINC 15 - Ligand Discovery for Everyone. J Chem Inf Model 55:2324–2337. https://doi.org/10.1021/acs.jcim.5b00559
30. Weininger D (1988) SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules. J Chem Inf Comput Sci 28:31–36. https://doi.org/10.1021/ci00057a005
31. https://voila.readthedocs.io/en/stable/index.html
32. Flick AC, Leverett CA, Ding HX, et al (2019) Synthetic Approaches to the New Drugs Approved during 2017. J. Med. Chem. 62:7340–7382
33. http://askcos.mit.edu/, accessed 2020-04-27 to 2020-04-29
34. Reaxys®, Copyright © 2019 Elsevier Limited except certain content provided by third parties, Reaxys is a trademark of Elsevier
35. Ertl P, Schuffenhauer A (2009) Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. J Cheminform 1:8 https://doi.org/10.1186/1758-2946-1-8
36. Gao W, Coley CW (2020) The Synthesizability of Molecules Proposed by Generative Models. J Chem Inf Model. https://doi.org/10.1021/acs.jcim.0c00174
37 Thakkar A, Selmi N, Reymond J-L, et al (2020) ‘Ring Breaker’: Neural Network Driven Synthesis Prediction of the Ring System Chemical Space. J Med Chem https://doi.org/10.1021/acs.jmedchem.9b01919
### Supporting information:

**AiZynthFinder:** a fast, robust and flexible software for retrosynthetic planning

Samuel Genheden, Amol Thakkar, Veronika Chadimová, Ola Engkvist, Esben Bjerrum

| SMILES | SA | Search time | Solution time | Solved |解决了with BB* | Number of steps | Number of precursors | Search time | Solution time | Solved | Number of steps | Number of precursors |
|--------|----|-------------|---------------|--------|---------------|----------------|---------------------|-------------|---------------|--------|----------------|---------------------|
| O=C(NCC1ccc(c)c1ccccc2(2)c1)OCO2 | 1.7 | 4.6 | 0.0 | Yes | Yes | 1 | 2 | 121.7 | 10.1 | Yes | 1 | 2 |
| CCl3=CC1=c2(ncc3ccc2[nH]3)c2n1 | 2.9 | 41.2 | No | No | | | | 10.2 | No |
| CCl3=CC1=CC=CC(=O)C2 | 3.7 | 58.6 | No | No | | | | 120.9 | No |
| O=C(NCC1ccc(c)c1ccccc2(2)c1)OCO2 | 2.3 | 21.3 | 0.4 | Yes | Yes | 1 | 2 | 123.7 | 10.1 | Yes | 2 | 2 |
| O=C(NCC1ccc(c)c1ccccc2(2)C(CCN)1(c1)ccc(3)CCC(=O)N1CC1N(C(=O)C(CCC(=O)O)NC(=O)c3N1CC1C2N1C(=S)Nc1ccccc1COC12C(=COC(N)=O)c3c(O)c(N)c(C)c(O)Cn1ccc2ccc3c(c21)CCN3C(=O)Nc1cccnc1)C1CCN(C(=O)C | 3.6 | 53.5 | No | No | | | | 122.8 | No |
| CCl3=CC1=CC=CC(=O)C2 | 2.5 | 36.8 | 13.3 | Yes | Yes | 2 | 3 | 126.3 | 10.1 | Yes | 3 | 3 |
| O=C(NCC1ccc(c)c1ccccc2(2)c1)OCO2 | 2.1 | 13.3 | 0.0 | Yes | Yes | 1 | 2 | 152.6 | 10.1 | Yes | 2 | 3 |
| CCl3=CC1=CC=CC(=O)C2 | 2.3 | 15.9 | 0.0 | Yes | Yes | 1 | 2 | 123.1 | 45.2 | Yes | 5 | 4 |
| CCl3=CC1=CC=CC(=O)C2 | 4.0 | 81.0 | No | Yes | | | | 270.1 | 16.1 | Yes | 7 | 5 |
| COCl=CC1=CC=CC(=O)C2 | 2.2 | 33.3 | 28.1 | Yes | Yes | 3 | 3 | 132.3 | 10.1 | Yes | 1 | 2 |
| CCl3=CC1=CC=CC(=O)C2 | 3.7 | 67.5 | No | No | | | | 123.3 | No |
| CCl3=CC1=CC=CC(=O)C2 | 3.8 | 78.3 | No | No | | | | 121.6 | No |
| Nc1=CCCC1=O | 2.3 | 34.5 | 14.3 | Yes | Yes | 3 | 2 | 147.8 | 10.1 | Yes | 1 | 2 |
| COCl=CC1=CC=CC(=O)C2 | 2.6 | 57.7 | No | No | | | | 121.5 | No |
| CCl3=CC1=CC=CC(=O)C2 | 3.2 | 30.9 | 14.9 | Yes | Yes | 3 | 3 | 132.0 | 10.1 | Yes | 7 | 4 |
| O=C(NCC1ccc(c)c1ccccc2(2)C(CCN)1(c1)ccc(3)CCC(=O)N1CC1N(C(=O)C(CCC(=O)O)NC(=O)c3N1CC1C2N1C(=S)Nc1ccccc1COC12C(=COC(N)=O)c3c(O)c(N)c(C)c(O)Cn1ccc2ccc3c(c21)CCN3C(=O)Nc1cccnc1)C1CCN(C(=O)C | 2.1 | 21.7 | 0.0 | Yes | Yes | 2 | 2 | 126.9 | 10.1 | Yes | 1 | 2 |
| CCl3=CC1=CC=CC(=O)C2 | 2.1 | 33.4 | 0.0 | Yes | Yes | 4 | 5 | 132.1 | 10.1 | Yes | 5 | 3 |
| CCl3=CC1=CC=CC(=O)C2 | 2.9 | 59.7 | No | No | | | | 10.1 | No |
| CCl3=CC1=CC=CC(=O)C2 | 2.7 | 56.0 | No | Yes | | | | 121.3 | No |
| COCl=CC1=CC=CC(=O)C2 | 2.5 | 44.2 | No | No | | | | 121.4 | No |
| CCl3=CC1=CC=CC(=O)C2 | 2.9 | 57.3 | No | No | | | | 121.4 | No |
| CCl3=CC1=CC=CC(=O)C2 | 1.9 | 15.7 | 2.3 | Yes | Yes | 1 | 1 | 10.1 | 10.1 | Yes | 1 | 1 |
| O=C(NCC1ccc(c)c1ccccc2(2)C(CCN)1(c1)ccc(3)CCC(=O)N1CC1N(C(=O)C(CCC(=O)O)NC(=O)c3N1CC1C2N1C(=S)Nc1ccccc1COC12C(=COC(N)=O)c3c(O)c(N)c(C)c(O)Cn1ccc2ccc3c(c21)CCN3C(=O)Nc1cccnc1)C1CCN(C(=O)C | 2.6 | 29.3 | 9.7 | Yes | Yes | 4 | 5 | 120.9 | No |
| CCl3=CC1=CC=CC(=O)C2 | 2.2 | 24.3 | 0.0 | Yes | Yes | 1 | 2 | 128.7 | 10.1 | Yes | 1 | 2 |
| CCl3=CC1=CC=CC(=O)C2 | 2.5 | 32.7 | 9.3 | Yes | Yes | 6 | 3 | 121.4 | No |
| CCl3=CC1=CC=CC(=O)C2 | 4.9 | 51.0 | No | No | | | | 121.2 | No |
| O=C(NCC1ccc(c)c1ccccc2(2)c1)OCO2 | 2.6 | 8.9 | 0.0 | Yes | Yes | 1 | 2 | 10.1 | 10.1 | Yes | 1 | 2 |
| CCl3=CC1=CC=CC(=O)C2 | 3.5 | 54.0 | No | No | | | | 292.0 | 209.5 | Yes | 11 | 6 |
| CCl3=CC1=CC=CC(=O)C2 | 3.6 | 62.5 | No | No | | | | 121.3 | No |
| O=C(NCC1ccc(c)c1ccccc2(2)C(CCN)1(c1)ccc(3)CCC(=O)N1CC1N(C(=O)C(CCC(=O)O)NC(=O)c3N1CC1C2N1C(=S)Nc1ccccc1COC12C(=COC(N)=O)c3c(O)c(N)c(C)c(O)Cn1ccc2ccc3c(c21)CCN3C(=O)Nc1cccnc1)C1CCN(C(=O)C | 2.4 | 15.2 | 0.0 | Yes | Yes | 1 | 2 | 292.0 | 10.1 | Yes | 5 | 5 |
| O=C(NCC1ccc(c)c1ccccc2(2)c1)OCO2 | 2.4 | 17.6 | 0.5 | Yes | Yes | 2 | 2 | 122.1 | 10.1 | Yes | 3 | 3 |
| CCl3=CC1=CC=CC(=O)C2 | 2.7 | 68.5 | 59.1 | Yes | Yes | 3 | 3 | 124.4 | 10.1 | Yes | 2 | 3 |
| CID | Molecular Formula | MW | LogP | Charged | Ring | 1H NMR | 13C NMR | Lipinski | 2D LogP | 3D LogP | 3D SA | QED | 2D Lipinski | 3D Lipinski | 3D SA Lipinski | 3D QED | 3D QED Lipinski |
|-----|------------------|-----|------|---------|------|---------|---------|----------|---------|---------|-------|------|--------|---------|----------------|--------|--------------------|
| 1   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 3.2 | 192.3 | Yes     | Yes   | 10.1   | Yes     | Yes      | 1       | 2       |       | 3     | 3      | 10.1   | Yes             | 10.1  | Yes               |
| 2   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 4.2 | 306.1 | Yes     | Yes   | 10.1   | Yes     | Yes      | 4       | 4       | 128.6 | Yes  | 1      | 10.1   | Yes             | 10.1  | Yes               |
| 3   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 5.2 | 316.9 | Yes     | Yes   | 10.1   | Yes     | Yes      | 2       | 2       | 163.3 | Yes  | 2      | 10.1   | Yes             | 10.1  | Yes               |
| 4   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 6.2 | 328.1 | Yes     | Yes   | 10.1   | Yes     | Yes      | 3       | 3       | 174.2 | Yes  | 3      | 10.1   | Yes             | 10.1  | Yes               |
| 5   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 7.2 | 339.9 | Yes     | Yes   | 10.1   | Yes     | Yes      | 4       | 4       | 205.0 | Yes  | 4      | 10.1   | Yes             | 10.1  | Yes               |
| 6   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 8.2 | 351.1 | Yes     | Yes   | 10.1   | Yes     | Yes      | 5       | 5       | 221.5 | Yes  | 5      | 10.1   | Yes             | 10.1  | Yes               |
| 7   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 9.2 | 362.3 | Yes     | Yes   | 10.1   | Yes     | Yes      | 6       | 6       | 221.5 | Yes  | 5      | 10.1   | Yes             | 10.1  | Yes               |
| 8   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 10.2| 373.5 | Yes     | Yes   | 10.1   | Yes     | Yes      | 7       | 7       | 221.5 | Yes  | 5      | 10.1   | Yes             | 10.1  | Yes               |
| 9   | O=C(O)CCCN(2ccccc3[O]=O)N3CCCN(2ccccc3[O]=O)C1CC   | 11.2| 384.7 | Yes     | Yes   | 10.1   | Yes     | Yes      | 8       | 8       | 221.5 | Yes  | 5      | 10.1   | Yes             | 10.1  | Yes               |

* Solution found if the enamine building blocks were added to the ZINC stock.
Genheden2020 - AiZynthFinder: a fast robust and flexible... (654.41 KiB)  view on ChemRxiv  download file