PIKACHU: a Python-based Informatics Kit for Analysing Chemical Units

Barbara Terlouw

Wageningen University & Research <https://orcid.org/0000-0002-4058-6718>

Sophie Vromans

Wageningen University & Research <https://orcid.org/0000-0002-7482-5721>

Marnix Medema (✉ marnix.medema@wur.nl)

Wageningen University & Research <https://orcid.org/0000-0002-2191-2821>

Research Article

Keywords: cheminformatics, Python, structure visualisation, in silico chemistry, molecular fingerprinting

Posted Date: January 18th, 2022

DOI: <https://doi.org/10.21203/rs.3.rs-1239072/v1>

License: This work is licensed under a Creative Commons Attribution 4.0 International License. [Read Full License](#)

Abstract

As efforts to computationally describe and simulate the biochemical world become more commonplace, computer programs that are capable of *in silico* chemistry play an increasingly important role in biochemical research. While such programs exist, they are often dependency-heavy, difficult to navigate, or not written in Python, the programming language of choice for bioinformaticians. Here, we introduce PIKACHU (Python-based Informatics Kit for Analysis CHemical Units): a light-weight cheminformatics toolbox implemented in Python. PIKACHU builds comprehensive molecular graphs from SMILES strings, which allow for easy downstream analysis and visualisation of molecules. While the molecular graphs PIKACHU generates are extensive, storing and inferring information on aromaticity, chirality, charge, hybridisation and electron orbitals, PIKACHU limits itself to applications that will be sufficient for most casual users and downstream Python-based tools and databases, such as Morgan fingerprinting, similarity scoring, substructure matching and customisable visualisation. In addition, it comes with a set of functions that assists in the easy implementation of reaction mechanisms. Its minimalistic design makes PIKACHU straightforward to use and install, in stark contrast to many existing toolkits, which are more difficult to navigate and come with a plethora of dependencies that may cause compatibility issues with downstream tools. As such, PIKACHU provides a perfect alternative for researchers for whom basic cheminformatic processing suffices, and can be easily integrated into downstream bioinformatics and cheminformatics tools. PIKACHU is available at <https://github.com/BTheDragonMaster/pikachu>.

Introduction

In a data-driven world where the discovery of novel natural and synthetic molecules is increasingly necessary, *in silico* chemical processing has become an essential part of biological and chemical research. Novel metabolites are compared or added to searchable chemical databases such as ChEBI [1], PubChem [2], NP Atlas [3], and COCONUT [4]; molecular structures are predicted from biological pathways [5, 6]; and bioactivities and pharmaceutical properties are predicted from chemical structure [7–9]. Such analyses rely on robust cheminformatics kits that can perform basic chemical processing, such as fingerprint-based similarity searches, substructure matching, molecule visualisation and chemical featurisation for machine learning purposes.

Typically, molecular processing by cheminformatics kits begins with the reading in of molecular data from chemical data formats, ranging from one-dimensional to three-dimensional molecular representations. Two such formats are SMILES (Simplified Molecular-Input Line Entry System) and InChI (International Chemical Identifier), which both represent a molecule as a one-dimensional string, describing atom composition, connectivity, stereochemistry, and charge. An important distinction is that the layer-based InChI strings and their associated hashed InChI keys are unique for any given molecule (and, depending on the number of layers, their tautomers and stereoisomers), while many different SMILES strings can be constructed for a single compound. More elaborate formats such as PDB and MOL use text files to store not just the abovementioned properties but also atom coordinates in three-dimensional space.

Depending on the application, different formats and subsequent processing are appropriate. Due to the vast number of possible chemical analyses, exhaustive cheminformatics kits have accumulated into software libraries that are so large that they are hard to navigate, and rely on so many dependencies that they are difficult to implement in software packages. As a result, the trade-off between time spent accessing and integrating these cheminformatics kits into a codebase and time spent on actual analyses is disproportionate for users that need to perform simple *in silico* analyses such as reading in SMILES, drawing a molecule, or visualising a substructure. One popular open-source cheminformatics kit that suffers from this problem is RDKit [10]. While RDKit is an incredibly fast and powerful library that supports an immense variety of possible chemical operations, its use of both Python and C++ as programming languages as well as the sheer number of dependencies it relies on frequently causes compatibility issues when integrating RDKit into other programs, and disproportionately increases the number of libraries that need to be installed. Therefore, while RDKit is great for heavy-duty *in silico* analyses such as computing 3D conformers for a compound or constructing electron density maps, it is a bit heavyweight for the basic operations that most researchers in bioinformatics and cheminformatics require.

A second widely-used cheminformatics kit is CDK [11], which is far more compact than RDKit and therefore does not suffer from the aforementioned problems. Written in Java, it is well-suited for implementation in web applications, and has successfully been used for molecular processing in the COCONUT database [4], the Cytoscape application chemViz2[12], and the scientific workflow platform KNIME (Konstanz Information Miner) [13]. However, with Python becoming the programming language of choice for many scientists [14], especially those working in the growing field of (Deep) neural networks, CDK is not always an ideal fit.

To make basic cheminformatics processing more accessible for Python programmers, we therefore introduce PIKACHU: a Python-based Informatics Kit for Analysing Chemical Units. PIKACHU is a fast, flexible and light-weight cheminformatics tool that can parse molecules from SMILES, visualise chemical structures and substructures in matplotlib, perform Extended Connectivity FingerPrinting (ECFP) [15] and Tanimoto similarity searches, and execute basic reactions with a focus on natural product chemistry. Performing comparably to existing cheminformatics kits and state-of-the-art drawing software, while only relying on the matplotlib library as a dependency, we expect that PIKACHU will become the preferred cheminformatics kit of choice for many Python-based bio- and cheminformatics tools and databases that only demand basic chemical processing.

Methods And Implementation

Software Description

PIKACHU is implemented in Python (v3.9.7). Its only dependency is the common Python package matplotlib (v3.4.3). PIKACHU can be run on Windows, MacOS, and Linux systems.

Parsing molecules from SMILES

PIKACHU takes a SMILES string as input and from it builds a graph object, in which nodes represent atoms and edges represent bonds (Figure 1). For each atom, PIKACHU initially stores information on chirality, aromaticity, charge, and connectivity. For each bond, it stores bond type (single, double, triple, quadruple, or aromatic), neighbouring atoms, and cis-trans stereochemistry for double bonds. Once all atoms, bonds, and their connectivities have been stored, implicit hydrogens are added to the structure, electron shells and orbitals are constructed for each atom, electrons are allocated to σ or π bonds, and atom hybridisation is determined. Next, all cycles in the graph are detected using an open-source Python implementation [16] of the simple cycle detection algorithm described by D. Johnson in 1975 [17]. From these cycles, PIKACHU identifies the set of unique cycles larger than two atoms, and defines cyclic systems, where cycles are considered part of a cyclic system if they share a bond with at least one other cycle in the system. Where aromaticity is not directly inferred from a SMILES string, it is determined for the atoms and bonds of each cycle and each cyclic system by applying Hückel's $4n + 2$ rule on planar ring systems [18]. Five-membered aromatic heterocycles where a lone pair of electrons occupies a p-orbital are considered on a case-to-case basis: if exactly four atoms are sp^2 -hybridised and exactly one sp^3 -hybridised heteroatom carries a lone pair, then the lone pair of the heteroatom is promoted to a p-orbital and the heteroatom's hybridisation is adjusted to sp^2 .

Finally, a structure object is returned which can be visualised, kekulised, analysed through substructure matching and molecular fingerprinting, and altered through an assortment of built-in and custom chemical reactions.

If a SMILES string yields a structure object that is chemically incorrect due to too many or too few bonds being attached to an atom or valence shells not being filled appropriately in the case of organic atoms, PIKACHU gives a `SmilesError`, informing the user that the SMILES is wrong, and what is wrong with it.

Visualisation and kekulisation

Prior to visualisation, aromatic systems within a structure are kekulised so that aromatic systems can be represented by alternating single and double bonds. PIKACHU kekulises aromatic systems using a Python implementation (Yorkyer 2020) of Edmonds' Blossom Algorithm for maximum matching [20]. Next, atoms are positioned using PIKACHU's powerful drawing software, capable of visualising complex molecules, such as heavily cyclised molecules, chiral centres, and cis- and trans double bonds. PIKACHU's python-based drawing algorithm was adapted and improved from `SmilesDrawer` [21], a JavaScript library that constitutes one of the best open-source molecular drawing software packages currently available. While written in different programming languages, the algorithms underlying the drawing software of PIKACHU and `SmilesDrawer` are virtually identical. We will briefly recap this algorithm below; more detailed descriptions of the algorithm's elements can be found in the `SmilesDrawer` paper [21].

First, if indicated, PIKACHU's drawing algorithm removes hydrogens from the graph. Next, it finds the smallest set of smallest rings in the structure graph and classifies them into one of three groups: simple rings, overlapping rings, and bridged rings. Simple rings are standalone rings that do not have any overlapping atoms with any other rings. Overlapping rings are rings that overlap with one or more other

rings, where the overlap between any two rings can comprise at most two atoms, any atom in the overlap is part of at most two rings, and no atoms in the ring overlap with bridged rings. Finally, bridged rings are rings that share more than two atoms with another ring, contain atoms that are part of three or more rings, or share atoms with another bridged ring (Figure 2A).

After ring systems have been identified, atoms are placed onto a 2D coordinate system. If the molecule contains rings, positioning starts with the placement of an atom in a ring, prioritising bridged rings over simple and overlapping rings. Then, the graph is traversed one atom at a time in depth-first fashion. If an atom is part of a ring, the entire ring or ring system get placed at once. In the case of simple and overlapping rings, ring placement can be done using simple polygon geometry. For bridged rings, atoms are positioned using the force-spring model described by Kamada and Kawai [22], where all atoms of the bridged system are initially placed in a circle, and then pulled towards their optimal positions by minimising the difference between the desired bond length and the distance between neighbouring atoms, and maximising distances between non-neighbouring atoms. Non-ring atoms are positioned a bond length away from the previous atom, where the angle with respect to the previous atom is determined by the number of neighbours the atom has (Figure 2B), and the size of the molecular subtree behind each neighbouring atom (Figure 2C). Stereochemically restricted double bonds are always forced into the appropriate cis- or trans conformation. Unlike SmilesDrawer, which directly infers bond stereochemistry from the SMILES string, PIKACHU draws this information from bond objects stored in the molecular graph.

Once all atoms have been assigned initial coordinates, atoms adjacent to rings are flipped outside of their ring where possible. Then, the drawing is checked for overlaps between atoms, and these overlaps are resolved by rotating branches of the molecule around single bonds. Finally, some bonds are replaced with backward and forward wedges around chiral centres. They are placed such that they do not neighbour more than one chiral centre where possible, they are not part of a ring, and point in the direction of the shortest branch leading from a chiral centre, in that order of priority. The resulting image is subsequently written to a .svg or .png file or displayed directly in matplotlib.

Substructure matching

PIKACHU detects occurrences of a substructure in a superstructure in five steps. In all steps, hydrogens are ignored. First, PIKACHU checks for each atom type in the substructure if enough atoms of these types are accounted for in the superstructure. Second, it assesses for each atom in the substructure whether an atom exists in the superstructure with the same connectivity, looking at directly neighbouring bonds and atoms. Third, using the atom with the most diverse connectivity as a seed, it finds matches of the substructure in the superstructure using a depth-first search algorithm, ignoring stereochemistry. By first looking at atom type and atom connectivity, and by using atoms of diverse connectivity as seeds for substructure matching, the number of calls to the computationally expensive depth-first search function is minimised. Fourth, for each match, it determines if all chiral centres in the substructure have the same orientation as corresponding chiral centres in the superstructure. Fifth, PIKACHU checks if cis-trans orientation of double bonds in the substructure matches that of double bonds in the superstructure. Chiral centre and double bond stereochemistry checks can be toggled by the user independently of one another. If

chirality of bonds and atoms are considered, substructures with undefined stereochemistry will still match to parent structures with defined stereochemistry. This does not apply in reverse: if a stereocentre or stereobond is defined for a substructure, it will not match to parent structures with undefined stereochemistry.

Substructures can be easily visualised through a range of functions in PIKACHU's general library.

Fingerprinting

PIKACHU uses an improved version of the classical Morgan fingerprinting, ECFP [15], to perform similarity searches and convert molecules to bit vectors for machine learning featurisation. Using Python's inbuilt hashlib library, PIKACHU initialises each atom to a 32-bit hash, derived from a tuple containing information on heavy neighbours, valence, atomic number, atomic weight, charge, hydrogen neighbours, and ring membership. Then, each atom hash is iteratively updated with hashes from its neighbours, as well as the distance from the neighbour to the atom and stereochemical information if the atom is a chiral centre. The number of iterations depends on a radius which can be set to any number (default = 2 for ECFP-4 fingerprinting). The ECFP algorithm was described in detail by Rogers and Hahn in 2010 [15]. Finally, duplicate hashes are removed, as well as different hashes representing the same substructure, yielding a set of 32-bit hashes that constitutes a molecule's fingerprint.

Using ECFP fingerprinting, PIKACHU can calculate Jaccard/Tanimoto distance and/or similarity between any two molecules. Furthermore, PIKACHU can convert molecule libraries into bit vectors of varying lengths (default = 1024) and an accompanying list of substructures represented by those bit vectors that can be used in downstream machine learning algorithms.

Defining reaction targets

In order to facilitate implementation of reactions and reaction pathways, PIKACHU lets users define target bonds or atoms within substructures with a set of dedicated functions. These functions take a SMILES string representing a substructure, and either one or two integers that define an atom or a bond between two atoms respectively. For example, the SMILES string C(=O)NC, accompanied by the integers 0 (pointing to the first C atom) and 2 (pointing to the N atom), represents a peptide bond. The occurrences of these bonds/atoms are then detected within a superstructure and returned as a list of bonds/atoms.

Subsequently, the returned bonds/atoms can be used as reaction targets, for instance for bond hydrolysis or atom methylation, using functions in PIKACHU for breaking or creating bonds and adding or removing atoms.

Characterisation and visualisation of the polyketide ketoreduction reaction

We demonstrated the implementation of reactions using PIKACHU by characterising and visualising a polyketide ketoreduction reaction. We built the ketoreduction reaction by first defining a reaction target as described above, in this case a β -keto bond, and detecting its position in a polyketide chain. Next, we wrote a function that reduces the double carbonyl bond to a single bond, which identifies and removes the π -

electrons in the double bond, sets the bond type to single, adjusts the hybridisation of the atoms involved and finally updates the structure object through PIKACHU's refresh functions. To finalize the reaction, two hydrogen atoms were added to the carbon and oxygen atoms of the former carbonyl bond using PIKACHU's add_atom function. Finally, to visualise the reaction, we highlighted the atoms and bonds of the newly formed hydroxyl group in red and drew the molecule.

Detailed instructions on how to make full use of PIKACHU's range of functionalities, as well as the script used to implement the ketoreduction reaction, can be found in the online documentation.

Speed assessment

For PIKACHU and RDKit, speed was assessed by timing the drawing of all molecules in NP Atlas. We used the Linux 'time' command with the calls to the Python scripts performing the visualisation. To assess SmilesDrawer's speed, we used the developers' online drawing portal, which also reports computational time for generating a drawing. For ChemDraw, drawing speed could not be assessed accurately, but for the molecules we tested image generation seemed instant.

Results And Discussion

PIKACHU is a light-weight cheminformatics kit implemented entirely in Python. With only matplotlib as dependency and an extensive readme, wiki, tutorials, and example scripts on its GitHub page, PIKACHU is easy to install, run, and integrate into bioinformatics and cheminformatics pipelines. Below, we will first assess PIKACHU's ability to correctly interpret SMILES, draw structures accurately and comprehensibly, detect and visualize substructures, and perform ECFP fingerprinting. Next, we demonstrate how PIKACHU can be used to implement and visualise reactions. Finally, we compare PIKACHU to the state-of-the-art cheminformatics kits/chemical drawing libraries RDKit, ChemDraw and SmilesDrawer.

PIKACHU correctly interprets organic compounds from SMILES

To determine if PIKACHU is able to correctly interpret SMILES syntax, we first used PIKACHU's SMILES reader on all SMILES strings from two natural product databases: NP Atlas and COCONUT, databases containing 32,552 and 406,747 natural product structures, respectively. NP Atlas is entirely contained in COCONUT; however, as the SMILES from NP Atlas describe stereochemistry while the SMILES in the COCONUT database do not, we decided to run PIKACHU on both databases to assess PIKACHU's performance on a large variety of both isomeric and canonical SMILES. PIKACHU failed to convert 23 SMILES from NP Atlas (~0.07%) and 1,325 SMILES from COCONUT (~0.33%) to structure graphs. Upon manual inspection, we observed that the vast majority of these structures were erroneous SMILES describing incorrect chemistry. One SMILES from NP Atlas was erroneous, incorrectly describing an aromatic system. The other 22 SMILES described nitrogen atoms with a valency of 5, which is impossible considering that nitrogen only has four electron orbitals available for bonding in its valence shell (Additional File 1). Many failed SMILES from the COCONUT database fell into the same category, based on

manual inspection of a random subset (Additional File 2). This demonstrates how the detailed graph-based, object-oriented encoding of chemical structures down to the electron level in PIKACHU intrinsically ensures that all structures that are loaded are chemically valid.

Next, we manually assessed the correctness of 22 SMILES-to-graph conversions by reading in and subsequently drawing the SMILES in PIKACHU. We chose the SMILES such that a variety of chemistries were represented, including rings, aromatic systems, charge, stereocentres and bond stereochemistry. Some SMILES describe the same structures but use a different syntax. PIKACHU handled all SMILES correctly, accurately detecting and visualising all aforementioned chemical properties (Figure 3).

The only molecules that PIKACHU struggles with are molecules with a high number of recursive cycles, such as buckminsterfullerene. As PIKACHU detects all possible cycles within a molecule to determine aromaticity of cyclic systems, this step takes so long to compute that the program appears to get 'stuck'. However, there exist only a handful of examples of such molecules, none of which have any real practical biological or chemical relevance.

PIKACHU draws accurate and readable structures

The most important aspect of automated visualisation is accuracy: users need to be able to rely on the correctness of drawing software, especially when processing a large number of structures at once making it impossible to inspect each image independently. To this purpose, we visualised a chemically diverse set of structures in PIKACHU and manually assessed their correctness. As PIKACHU largely relies on the same logic for molecule visualisation as SmilesDrawer, a high-end JavaScript SMILES drawing library, it was unsurprising that PIKACHU's chemical drawings were of high standard, visualising highly cyclised systems and introducing minimal overlaps between molecule branches. Stereocentres and bond stereochemistry are always drawn correctly, and aromatic systems are appropriately kekulised (Figure 4).

There is always a bit of debate regarding the visualisation of molecular macrocycles. Many organic chemists opt for a 'honeycomb' architecture, as employed by ChemDraw and CDK, to better represent the 3D architecture of a molecule, hinting at long-distance interactions that may take place within the compound (Figure S1A [Additional File 3]). However, this representation does not instantly draw the eye to sites of cyclisation, a drawback for natural product biologists and bioinformaticians who are often interested in the biosynthetic steps involved in a compound's assembly. Additionally, the 'honeycomb' algorithm does not always yield readable images. As PIKACHU was created with natural product chemistry in mind, we chose to use a polygon representation for macrocycles, which clearly shows cyclisation sites (Figure S1B [Additional File 3]).

PIKACHU facilitates straightforward detection and visualisation of substructures

A dedicated set of functions ensures that performing substructure searches using PIKACHU is straightforward. With a single line of code, users can visualise a single occurrence of a substructure, all occurrences of a substructure, or all occurrences of a range of substructures in a chemical compound

(Figure 5A). Substructure searches are fast due to several pre-processing steps, ensuring that the expensive graph matching algorithm is only executed when a match is likely. Stereochemistry matching, activated by default, can be toggled on and off.

With PIKACHU's substructure matching algorithm, we visualised the amino acid composition of the cyclic peptides daptomycin and vancomycin, using only a single line of code for each (Figure 5B). Colours are fully and easily customisable, and can be provided as hex codes or as colour names.

PIKACHU computes molecular similarity using ECFP fingerprinting

To quickly determine the approximate similarity between two molecules, PIKACHU employs ECFP fingerprinting, an evolved version of Morgan fingerprinting. PIKACHU hashes each molecule into a set of unique identifiers, each of which represents a substructure. Collectively, these identifiers make up a molecule's fingerprint. Then, PIKACHU calculates the Jaccard/Tanimoto similarity between two molecules by comparing their fingerprints, giving a measure of molecular similarity and/or distance.

Here, we showcase PIKACHU's ECFP fingerprinting by calculating and subsequently constructing a tSNE plot of the molecular distances between 36 calcium-dependent lipopeptides. Lipopeptides of the same family grouped together (Figure S2 [Additional File 3]), confirming that PIKACHU's ECFP fingerprinting yields reliable measures of chemical similarity.

Additionally, PIKACHU's ECFP fingerprinting makes it possible to generate bit vectors from molecule sets, where each element in the vector represents the presence/absence of a specific substructure. These can subsequently be used as interpretable molecular featurisations for machine learning.

Building in silico reactions using PIKACHU

PIKACHU provides an intuitive platform for the creation and visualisation of reaction mechanisms by providing a range of reaction functions that can be used to make or break molecular bonds, add or remove atoms and alter the chirality of stereocentres. In addition to these built-in reaction building blocks, PIKACHU allows users to easily define more complex reactions through the manipulation of atom- and bond object attributes. As a proof of principle, we used PIKACHU to define and visualise a polyketide ketoreduction reaction, catalysed by a ketoreductase polyketide synthase domain during polyketide synthesis, employing both built-in and custom reaction functions (Figure S3 [Additional File 3]). This example, as well as a comprehensive guide containing instructions on how to build reaction mechanisms using PIKACHU, can be found in the online documentation.

PIKACHU compares positively with state-of-the-art chemical drawing software

Finally, we assessed how PIKACHU performs compared to existing chemical drawing software. To this purpose, we visualised various structures in PIKACHU, RDKit, ChemDraw and SmilesDrawer, and manually

assessed drawing quality and correctness (Figure 6). Only SmilesDrawer occasionally produced an incorrect structure, confusing cis-trans stereochemistry when stereochemistry is defined in or after a branch (Figure 6A). In terms of drawing quality, PIKACHU consistently performs comparably to or better than RDKit, ChemDraw and SmilesDrawer (Figure 6B, 6C). It has an advantage over RDKit in drawing molecules of varying sizes, automatically adjusting the canvas size based on the size of the molecule to be drawn. While it is possible to manually adjust canvas size in RDKit, some extra coding steps are required to achieve this. Also, PIKACHU's visual output is far more customizable than that of SmilesDrawer, allowing for molecule rotation, drawing multiple molecules on a single canvas, and custom colouring of each individual bond and atom, supporting hex-codes as well as a range of descriptive strings. Compared to ChemDraw, PIKACHU has the advantage of scalability: while ChemDraw allows high-quality and highly customisable visualisation, it cannot do this automatically, drawing only a single molecule at a time. Moreover, in contrast to ChemDraw, PIKACHU is open source. This makes PIKACHU suitable for integration into automated pipelines required by many projects.

Finally, while PIKACHU is not the fastest of the drawing software packages, drawing about 8,000 molecules in an hour, it will still be fast enough for the purposes of almost all bioinformatics and cheminformatics researchers. What PIKACHU lacks for in speed, it more than makes up for in reliability, customisability and scalability.

Conclusions

We developed PIKACHU, a light-weight and virtually dependency free cheminformatics library for Python programmers. Having extensively tested our software, we conclude that it performs at least as well if not better than existing cheminformatics libraries for the most common chemical analyses. Backed by extensive online documentation, easy and straightforward installation, and state-of-the-art automated visualisation software, we expect that PIKACHU will become the package of choice for many chem- and bioinformaticians programming in Python.

Abbreviations

PIKACHU: Python-based Informatics Kit for Analysing Chemical Units

SMILES: Simplified Molecular-Input Line Entry System

InChI: International Chemical Identifier

ECFP: Extended Connectivity FingerPrinting

Declarations

Availability of data and materials

The PIKACHU software is made available under an open-source (MIT) license and can be found at <https://github.com/BTheDragonMaster/pikachu>. A wiki can be found at <https://github.com/BTheDragonMaster/pikachu/wiki>. Scripts used for the results section of this paper are made available at https://github.com/BTheDragonMaster/pikachu/tree/main/example_scripts. The NP Atlas and COCONUT databases used in our analyses can be downloaded at <https://www.npatlas.org/download> and <https://coconut.naturalproducts.net/download> respectively.

Competing interests

M.H.M. is a member of the Scientific Advisory Board of Hexagon Bio and co-founder of Design Pharmaceuticals.

Funding

This work was supported by the Novel Antibacterial Compounds and Therapies Antagonising Resistance program (NACTAR) from the Dutch Research Council (NWO) [project number 16440].

Authors' contributions

B.R.T.: Development and testing of the PIKACHU software, wrote the initial version of the manuscript and further edited it based on the suggestions of the other authors.

S.P.J.M.V.: Significant contributions to writing the manuscript, extensive testing and debugging of the PIKACHU software, writing and executing example scripts.

M.H.M.: Significant contributions to writing the manuscript, suggestions for software features, research supervision.

Acknowledgements

Daniel Probst for providing an in-depth explanation of the SmilesDrawer software; Rutger Bosch for reporting software bugs; Zach Reitz, Joris Louwen, Lotte Pronk, David Meijer, Hannah Augustijn, Kumar Singh, Huali Xie, Jiayi Jing, Mohammad Alanjary, Chrats Melkonian, Justin van der Hooft, and Catarina Sales E Santos Loureiro for testing the PIKACHU software.

References

1. Hastings J, Owen G, Dekker A, et al (2016) ChEBI in 2016: Improved services and an expanding collection of metabolites. *Nucleic Acids Res* 44:D1214–D1219. <https://doi.org/10.1093/nar/gkv1031>
2. Kim S, Chen J, Cheng T, et al (2021) PubChem in 2021: New data content and improved web interfaces. *Nucleic Acids Res* 49:D1388–D1395. <https://doi.org/10.1093/nar/gkaa971>
3. Van Santen JA, Jacob G, Singh AL, et al (2019) The Natural Products Atlas: An Open Access Knowledge Base for Microbial Natural Products Discovery. *ACS Cent Sci* 5:1824–1833. <https://doi.org/10.1021/acscentsci.9b00806>

4. Sorokina M, Merseburger P, Rajan K, et al (2021) COCONUT online: Collection of Open Natural Products database. *J Cheminform* 13:1–13. <https://doi.org/10.1186/s13321-020-00478-9>
5. Skinnider MA, Johnston CW, Gunabalasingam M, et al (2020) Comprehensive prediction of secondary metabolite structure and biological activity from microbial genome sequences. *Nat Commun* 11:1–9. <https://doi.org/10.1038/s41467-020-19986-1>
6. Blin K, Shaw S, Kloosterman AM, et al (2021) AntiSMASH 6.0: Improving cluster detection and comparison capabilities. *Nucleic Acids Res* 49:W29–W35. <https://doi.org/10.1093/nar/gkab335>
7. Volkamer A, Kuhn D, Rippmann F, Rarey M (2012) Dogsitescorer: A web server for automatic binding site prediction, analysis and druggability assessment. *Bioinformatics* 28:2074–2075. <https://doi.org/10.1093/bioinformatics/bts310>
8. Stokes JM, Yang K, Swanson K, et al (2020) A Deep Learning Approach to Antibiotic Discovery. *Cell* 180:688-702.e13. <https://doi.org/10.1016/j.cell.2020.01.021>
9. Alvarsson J, Lampa S, Schaal W, et al (2016) Large-scale ligand-based predictive modelling using support vector machines. *J Cheminform* 8:1–9. <https://doi.org/10.1186/s13321-016-0151-5>
10. Landrum G (2021) RDKit: Open-source cheminformatics. <http://www.rdkit.org>. Accessed 7 November 2021
11. Willighagen EL, Mayfield JW, Alvarsson J, et al (2017) The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching. *J Cheminform* 9:1–19. <https://doi.org/10.1186/s13321-017-0220-4>
12. Morris J, Jiao D (2016) ChemViz2: Cheminformatics App for Cytoscape
13. Beisen S, Meinel T, Wiswedel B, et al (2013) KNIME-CDK: Workflow-driven cheminformatics. *BMC Bioinformatics* 14:2–5
14. Cass S (2021) Top Programming Languages 2021, IEEE Spectrum. <https://spectrum.ieee.org/top-programming-languages/>. Accessed 7 November 2021
15. Rogers D, Hahn M (2010) Extended-Connectivity Fingerprints. *J Chem Inf Model* 50:742–754
16. Miles LH (2019) Cycle detection. <https://github.com/qpwo/python-simple-cycles>. Accessed 21 August 2021
17. Johnson D (1975) Finding all the elementary cycles of a digraph. *SIAM J Comput* 4:77–84
18. Hückel E (1931) Quantentheoretische Beiträge zum Benzolproblem - I. Die Elektronenkonfiguration des Benzols und verwandter Verbindungen. *Zeitschrift für Phys* 70:204–286. <https://doi.org/10.1007/BF01339530>
19. GitHub (user: yorkyer) (2020) Python implementation of Edmonds' Blossom Algorithm. <https://github.com/yorkyer/edmonds-blossom>. Accessed 24 August 2021
20. Edmonds J (1965) Paths, trees, and flowers. *Can J Math* 17:449–467
21. Probst D, Reymond JL (2018) SmilesDrawer: Parsing and Drawing SMILES-Encoded Molecular Structures Using Client-Side JavaScript. *J Chem Inf Model* 58:1–7. <https://doi.org/10.1021/acs.jcim.7b00425>

Figures

Figure 1

Overview of the internal structure of PIKACHU's structure graphs. This example uses L-alanine, a small amino acid. The four bottom boxes in grey indicate attributes for each of PIKACHU's major classes: Structure, Atom, Bond and Electron.

Figure 2

PIKACHU's drawing algorithm. A. Examples of simple (blue), overlapping (red) and bridged (pink) rings. Note that the aromatic rings in pink become part of the bridged ring system because they overlap with

bridged rings. **B.** Determination of bond angles based on neighbouring atoms. If an atom has 3 or fewer non-hydrogen neighbours, the angles default to 120° (yellow). If an atom has 4 non-hydrogen neighbours, angles default to 90° if three or more of the branches have a depth more than 1, or three or four branches have a depth of exactly 1 (red). If however two of the branches have a depth of exactly 1 (blue), the angle is set to 120° between the two longest branches, 90° between any short branch and any long branch, and 60° between the shortest branches. **C.** Positioning of neighbouring branches depends on the depth of each branch: the two longest branches (red and dark yellow, depths 7 and 6 respectively) are always placed opposite one another.

Figure 3

Assessment of PIKACHU's SMILES reader. Structures were drawn from the SMILES written beneath the molecule depictions. All 22 structures were correctly drawn.

Figure 4

Various molecules drawn automatically by PIKACHU.

a

```
> daptomycin = smiles_from_file('daptomycin.smi')
> aspartate = r"N[C@H](C(=O)CC(O)=O"
> tryptophan = r"N[C@H](C(=O)CC1=CNC2=CC=CC=C21"

> highlight_substructure(aspartate, daptomycin, search_mode='single', colour='light blue')
> highlight_substructure(aspartate, daptomycin, search_mode='all', colour='light blue')
```

Figure 5

Substructure matching with PIKACHU. A. From left to right: examples of highlighting a single instance of a substructure, all instances of a substructure, or all instances of multiple substructures. In the example, occurrences of aspartic acid and tryptophan were searched in the superstructure daptomycin. The code used to generate the images is displayed underneath the panels. B. PIKACHU's substructure matching algorithm using to visualise all amino acid components of the antibiotics daptomycin (left) and

vancomycin (right). Code can be found at https://github.com/BTheDragonMaster/pikachu/blob/main/example_scripts/amino_acid_composition.py.

a

b

c

Figure 6

Comparison of PIKACHU to various other chemical drawing software packages. A. SmilesDrawer, ChemDraw, RDKit and PIKACHU drawings given the SMILES string 'C/C=C(\N)/C'. While ChemDraw, RDKit and PIKACHU all draw the cis-trans stereochemistry of the double bond correctly, with the amino group cis of the methyl group, SmilesDrawer draws the stereobond in the wrong orientation. B. SmilesDrawer, ChemDraw, RDKit and PIKACHU drawings of the heavily cyclised molecule vancomycin. C. SmilesDrawer, ChemDraw, RDKit and PIKACHU drawings of the molecule aplasmomycin B.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [graphicalabstract.svg](#)
- [supplementaryfigures.docx](#)
- [supplementaryfile1.txt](#)
- [supplementaryfile2.txt](#)