VRIJE UNIVERSITEIT AMSTERDAM

FACULTY OF SCIENCE

Thesis Bachelor Project Pharmaceutical Sciences

Unlocking Docking: Open & Modular Molecular Docking with Pyrite

Author:

Mart J. van der Lugt
m.j.van.der.lugt@student.vu.nl
Student number: 2782775

April 2025–June 2025
Bachelorproject FAR
XB_0140
Credits: 24 EC
Honours Extension

Internship host
Department of Chemistry & Pharmaceutical Sciences
Division of Biomolecular Simulation
Vrije Universiteit Amsterdam
Daily supervisor: dr. D. A. Poole III

First assessor:
Dr. D. A. Poole III
d.a.poole@vu.nl

Second assessor:
Dr. D. P. Geerke
d.p.geerke@vu.nl

Abstract

Pyrite is a new, open, extensible, and accessible, Python-based framework for molecular docking that aims to allow straightforward testing and development of existing and novel docking methods. We introduce (i) a grid-based binding pocket detection and scoring algorithm that guides searching to plausible binding sites, (ii) a crowding and niching strategy that enforces pose diversity, and (iii) seamless integration with existing biochemical tools and optimization suites such as *rdkit*, *SciPy*, and *pygmo*.

Benchmarking on the 2024 PDBbind demo set shows that using the Pocket function accelerates searching 2-6 × compared to the Vina and Piecewise Linear Potential functions while maintaining, or in blind docking even surpassing, their accuracy. An example reference workflow — pocket + crowding for search with ant colony optimization, Piecewise Linear Potential for refinement — shows an accuracy of 37 ± 6 % of poses within 2 Å RMSD for autoboxed docking, and 16 ± 3 % in whole-protein mode. Error analysis reveals that most errors stem from searching errors, rather than scoring problems. Overly stringent pocket-depth filters and insufficient niching result in accuracy loss in the searching phase. Both problems are addressed in Pyrites development roadmap.

Beyond research, a pilot practicum with third-year Pharmaceutical Sciences students demonstrated Pyrite's educational value: all participants, despite having minimal coding background, implemented custom physics-based scoring functions within one lab day and reported heightened interest in both docking and programming.

Taken together, Pyrite offers a transparent and easily extensible package that lowers the barrier to innovation in molecular docking methods and teaching. Ongoing efforts focus on improving the performance and accuracy of Pyrite, and introducing additional functionalities that improve the development and learning experience.

Keywords: Pyrite, Molecular Docking, Binding Pocket Detection, Scoring Functions, Optimization, Niching, Computational Chemistry Education, Pharmaceutical Sciences Education.

Pyrite

Contents

1 Introduction

In pharmaceutical research, we rarely suffer from a lack of ideas — only from knowing which are worth pursuing. Millions of molecules might look promising on paper, but only a couple hundred will ever make it past a pipette — and only one will earn the honor of fulfilling its promise to patients. Every failed guess comes at a cost: time, money, and lab resources.

Suppose you have found a molecule with potential. It fits Lipinski's rule of five[i], it is easy to synthesize, and perhaps it is even structurally similar to known active compounds. Now you are left with a question: will it actually bind to your biological target? And if so, how? Answering this question experimentally could take weeks, and take up a large part of your budget. Do you commit, or keep searching? Do you exploit, or keep exploring?

This is the kind of problem molecular docking aims to help address. Instead of taking our hypothesis straight to the lab, we try to simulate binding using a computer; fitting our molecule inside its target, like a key into a lock. In this way, we can determine whether our molecule fits, where it binds, and in what orientation.

Solving this is anything but straightforward. A protein is not a stiff lock, and our key is highly flexible. Furthermore, the forces that determine binding are highly complex and computationally expensive to calculate. However, getting the answers even mostly right can save thousands of euros and months of lab work. When drug discovery timelines stretch for years and budgets reach billions of euros, even narrowing down the chemical space that needs to be explored in the lab by a few percent makes a big difference.

Moreover, the answers provided by molecular docking allow researchers and students to quickly better their understanding of the molecular basis and pharmacodynamics of drug action [1]. This, in turn, allows us not only to obtain faster answers, but also to ask better questions.

Molecular Docking

At its core, molecular docking is an optimization problem, where the interaction score between two molecules is optimized. This is coordinated by the feedback loop illustrated in Figure 1. Given a guest and a host, an initial set of poses is created. These poses are then scored using a scoring function, on the basis of which new poses are created. This loop is executed until we find that the score does not improve further, indicating a minimum in the interaction score. The poses with the best score are then selected and presented as possible binding modes.

This process is often executed in two phases: a searching, or sampling, phase, followed by a refinement phase. In the first phase, the goal is to effectively sample the searching space, often using an optimization algorithm that is tuned for finding the global minimum of a rugged function, using a fast and approximate scoring function. In this process, care needs to be taken to ensure a variety of diverse poses are returned to be able to find multiple possible binding modes.

[i] A set of four rules that predict whether a drug can be orally active.

This set of poses is then passed into the refinement phase, where a more complicated scoring function is used to optimize each pose to a local energy minimum. Generally, local optimizers such as gradient-based methods are employed here. After refinement, we rank poses by score and cluster them to make sure that the final ten poses are unique.

Ideally, this results in several diverse poses, where the one that is best ranked is highly similar to an experimentally determined pose. This is assessed using a Root Mean Square Deviation (RMSD) criterion. This method describes the distance between the atoms of two poses. Two poses with an RMSD lower than 2 Å are considered to be equal.

Existing Tools

Molecular docking has been a staple in the biochemical research community for over forty years [2]. As a result, there are many molecular docking tools out on the market today, each with their own perks and quirks. Furthermore, different tools employ different scoring functions and optimizers. For example, GOLD (Genetic Optimization for Ligand Docking) [3] uses a Genetic Algorithm (GA) to search for poses, together with its own Goldscore scoring function, while PLANTS (Protein-Ligand ANT System) [4] instead uses Ant Colony Optimization (ACO) and the CHEMPLP function for this purpose.

The highly different workflows applied by these tools highlight the complexity of the molecular docking problem, which is exacerbated by a vast variance in possible input data and large number of edge cases. This, combined with financial incentives, has historically led to monolithic or closed software, including the tools mentioned above.

While this approach has its benefits, as the monolithic design eases initial implementation and allows for simpler optimization, it is far from ideal for fundamental research. Current tools are difficult to modify and extend, as source code is either not available, or written in the relatively archaic C or Fortran programming languages and lacking in documentation. This greatly complicates the testing and development of novel docking methods — if you would want to develop a new scoring function or test a new searching method, you would have to create the entire program around it as well. Comparative analysis is also hindered by this approach to software design, as it is currently not possible to, for example, fairly compare the Goldscore function to the CHEMPLP

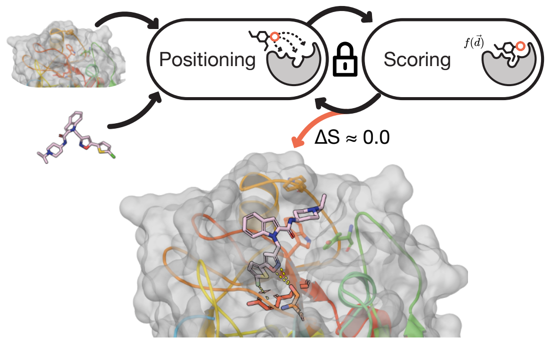


Figure 1: The general optimization workflow of a molecular docking program.

function, as using one of these scoring functions means you are locked to the specific corresponding workflow.

Furthermore, these tools function as a *black box*; you input your structures, and they output docking poses. This closed-off nature greatly interferes with the ability of researchers and students to gain a better understanding of the inner workings of the tool in question and the fundamental principles behind molecular docking.

Pyrite

On these grounds, we present *Pyrite*, an open, extensible, and accessible Python-based docking package. With Pyrite, we hope to establish a transparent open-source ecosystem that allows for completely custom molecular docking workflows, straightforward development and testing of existing and novel docking methods, and creation of other molecular optimization tools.

Through this, we hope to provide researchers with the tools needed to develop more efficient and performant docking methods and aid students in forming a better understanding of fundamental principles in computational chemistry.

2 Methods

The key principles of Pyrite, to be open, extensible, and accessible, greatly steer its software design. Pyrite is built on Python, which, according to [5], is the most used language in the world. Furthermore, many often-used tools and libraries in the field of biochemistry and cheminformatics are written in Python, allowing for easy interplay between these existing tools and Pyrite.

Molecular Representation

The molecular representation is a key part of any molecular docking program. The molecular representation decides how different poses are stored and therefore also what kind of operations can be carried out during the docking process. For example, PLANTS and GNINA represent a docking pose by translation (t_x, t_y, t_z), whole molecule rotation using Euler angles (α, β, γ), and torsional angles of rotatable bonds ($\phi_0 \cdots \phi_n$) [4, 6]. Only these variables can be changed by the optimization algorithm to change the pose of the ligand. Notably, internal molecular coordinates such as bond angles and lengths are omitted.

Other approaches use quaternions to represent whole molecule rotation instead [7, 8] to avoid the so-called gimbal lock problem when using Euler angles [9], where an axis of rotation is lost due to overlap with another axis. This, however, introduces another degree of freedom or other constraints to the optimization process.

Out of the box, Pyrite uses the same molecular representation as PLANTS and GNINA, encompassing translation, whole molecule rotation using Euler angles, and torsional angles of selected rotational bonds. This results in $6 + n$ degrees of freedom for a ligand with n rotatable bonds. This representation has been chosen for its relative simplicity, low number of degrees of freedom, and its similarity to state-of-the-art solutions.

By default, Pyrite considers all non-amide bonds rotatable, with the exception of bonds to a carbon atom bound to three hydrogen or halogen atoms, or methane groups.

Due to the open nature of Pyrite, however, changing the molecular representation is relatively straightforward. It is therefore feasible to experiment with, for example, the addition of bond angles in the representation or the impact of exchanging Euler angles for a quaternion-based rotation approach [9].

In Pyrite, the molecular representation is directly used as the vector describing a pose. Poses are stored as this feature vector, allowing for quick and easy data passing between the different Pyrite modules. Pyrite uses this vector to reconstruct the pose using a `Mol` object from the rdkit [10, 11]. Using this object, Pyrite can also quickly modify and transform the pose from a feature vector.

Furthermore, each atom in a molecule is assigned an atom type. Here, important physical information such as atom radius and whether the atom can be a hydrogen bond donor or acceptor is stored. By default, Pyrite uses Vina atom types [12], however, these can be customized if needed. Next to this, each atom is assigned a partial charge using Gasteiger charges [13]. These atom parameters can be utilized by scoring functions to calculate atom-specific scores.

Scoring Functions

A second essential part of a molecular docking program is a scoring function, a function that takes in the feature vector — the docking pose — and outputs a score. Ideally, this function is analogous to the true energy of the protein-ligand complex, such that the global minimum of the scoring function is equal to the global minimum of the actual potential energy surface (PES) of the protein-ligand interaction.

Due to the highly simplified nature of many scoring function models, this is often not the case. In these situations, it is possible that we identify a docking pose that has a better score than the "true" binding pose, which is often defined as the co-crystallized pose. This is called a *hard failure*, where we find a binding pose that is not equal to the co-crystallized pose, but has a better score [14]. The other type of failure is the *soft failure*. Here, we fail to find a successful pose, and only identify poses that have a worse energy than the co-crystallized pose.

Pyrite has multiple scoring functions built in. For example, the Vina and AutoDock functions that are implemented in GNINA [12, 6] are all emulated, and the original Piecewise Linear Potential (PLP) scoring function as described in [4] can also be used. Moreover, functions describing the internal energy of a molecule and the distance to configurable bounds have also been implemented.

Furthermore, to aid the extensible design of Pyrite, several abstract base scoring functions have been designed. For instance, all the logic for calculating the distances between ligand and nearby protein atoms has been implemented in one of these abstract functions, such that all you need to do to implement a new scoring function of this sort is transform these distances into a score. This is described in further detail in the Pyrite documentation [15].

Searching and Refining

As follows from Figure 1, all that is remaining is the loop itself. This role is fulfilled by an optimization algorithm. This algorithm takes in one or more feature vectors with corresponding scores, and decides on new feature vectors to try.

Conveniently, there are many existing optimizer solutions [16], with development having gained even more traction in recent years due to significant advances in the field of machine learning [17]. Due to the high presence of Python in this field and in the related fields of data science and analysis, there are many existing Python-based optimization packages. Pyrite is engineered in such a way that these existing implementations can conveniently be used in molecular docking workflows.

Perhaps one of the most well-known Python packages, *scipy* [18], for instance, implements many optimizers, such as the gradient-based *BFGS* and gradient-free *Nelder-Mead* [19] local optimization methods, and the basin hopping [20] and differential evolution [21] global optimization techniques.

More complicated optimization methods are implemented in the *pygmo* package [22]. Here, we can find techniques that are used for searching in state-of-the-art docking software, such as Ant Colony Optimization (ACO), as used in PLANTS [4], Genetic Algorithms (GA), as in GOLD [3], and Monte-Carlo methods, as used by Gnina [6].

Niching

Paramount in the success of a molecular docking program is the diversity of the poses. Next to the "best" generated pose, the analysis of the interactions and energies of other plausible poses is highly important, especially in binding mode prediction. The docking program should therefore not find only the global optimum, but multiple near-global optima. The relevance of this is also clear when searching using simplified scoring functions, where the global minimum might not equal the best binding pose — a *hard failure*.

Finding multiple near-global minima is a hard unsolved problem without a clear remedy [23]; it requires a careful balance between optimization performance — we want to find the global minimum — and diversity — we want multiple low-energy poses. One solution to this problem is running the optimization multiple times and penalizing proximity to existing poses for each subsequent run. This is implemented in Pyrite in the form of the `Crowding` function.

This function works based on RMSD, applying a penalty when this value is below a certain threshold. The penalty exponentially decays above this threshold to ensure that the search space is not disrupted. Poses generated using this method are forced to be different from earlier generated poses, effectively niching the search output.

Clustering Next to the explicit niching of poses, clustering is also an important tool to ensure a diverse output. During the clustering process, poses that are highly similar are grouped into clusters, after which only the top-ranking pose from each cluster is considered for the next step. This assures that in our output, no two poses are equal.

Pyrite clusters poses based on pairwise RMSD, using the rdkit [10] and the Butina algorithm [24]. Any two poses with an RMSD lower than 2 Å are considered as highly similar and clustered together.

Pocket Scoring Function

To demonstrate the academic value of Pyrite, we have created a novel scoring function aimed at efficient searching using binding pockets. This function scores a ligand based on either the overlap with, or the distance to this pocket.

Firstly, pockets are detected using a grid-based approach. A customizable grid is placed in and around the protein, after which the Van der Waals radii of the protein atoms are used to determine the `occupied` nature of the grid points. Points that are unoccupied are determined to be outside the protein, and thus possible binding pockets.

Several breath-first-searches are then executed on these points, "flooding" the grid with distances to the protein, an outside shell around the protein, and optionally selected protein residues. The connectedness of the grid points can be customized to 6, 18, or 26 neighbors, with 18 being the default value. Based on the grid size, the points are assigned a radius. This results in a set of spheres that are deemed to represent the binding pocket, as shown in Figure 2.

Furthermore, weights can be assigned to each sphere based on any combination of distance to the protein, to a shell around the protein, and to selected residues. The overlap scoring function then uses these weights in the score calculation; for every ligand atom, the weight of the closest overlapping sphere is summed.

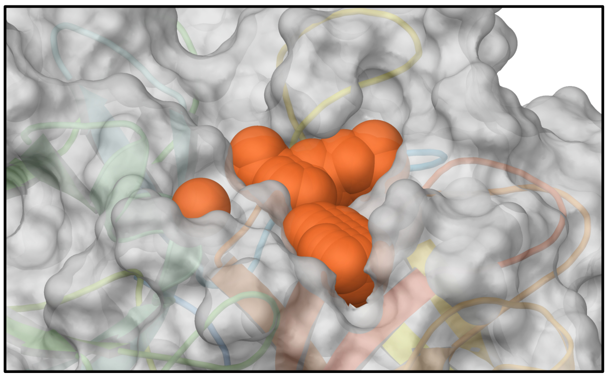


Figure 2: A binding pocket detected using Pyrite. The Pocket spheres are indicated in orange. In this case, the depth and distance-to-protein criteria are set to 14 and 4 respectively. PDB: 2BOH.

Docking with Pyrite

All of these methods can be combined to create a docking workflow. An example workflow, that is used for testing in the next chapter, has been implemented in a Jupyter Notebook and is published on the Pyrite GitHub [15].

To start, we use Pyrite to generate 100 000 initial positions and 4000 conformations, which are randomly combined. Whenever the Pocket function is used, a binding pocket is detected based on the protein structure using a minimum depth of 14 spheres, and a maximum distance to the protein of 4 spheres. If the search is autoboxed around the ligand, the pocket is intersected with an autobox with a padding of 2 Å.

The generated poses are then pre-screened by the algorithm that is to be used in the searching phase, and only the best 400 poses are kept.

We then search for 32 possible poses by feeding these initial variables into a global optimization algorithm. For the purposes of this paper, the number of searching evaluations per pose is limited to 20 000 wherever possible. During this optimization step we ensure the diversity of the poses by using a niching technique. When using the `Crowding` function, 4 groups of 8 poses are generated.

These poses are then fed into a local optimization algorithm, where a different scoring function can be used. In the example workflow, the *L-BFGS-B* algorithm [25] is used.

Lastly, the poses are clustered based on RMSD, and a final output of ten poses is constructed based on the clusters.

Analysis

Performance is validated using the PDBbind demo dataset [26] version 2024. The structures in the set are first prepared using *pdbfixer* [27] to replace missing and non-standard residues and add hydrogen, and then using *pdb4amber* [28] to remove all water molecules.

To determine if a pose is valid, an RMSD criterion is used. Any pose that has an RMSD lower than 2.0 Å to the co-crystallized pose of the PDBbind demo dataset is deemed as successful. Confidence intervals are determined using a two-sided proportion z-test, and p-values are calculated using a paired two-sided proportion z-test, both with an α-level of 0.05.

3 Results & Discussion

As Pyrite is a modular system, there is not one canonical docking workflow. Rather, Pyrite can be used to create any workflow, be it general or highly specific. Therefore, alongside directly benchmarking the performance of Pyrite against an existing docking program, we will compare different docking methods within Pyrite. In this way, the capabilities of Pyrite are more directly highlighted, and we can get a general feel of the performance as a bonus.

Scoring Functions for Searching

Firstly, the searching performance of several scoring functions was determined. We executed three simple docking workflows, one using the novel `DistanceToPocket` function, one using the Vina scoring function, and one using the PLP function for the searching phase. As can be seen in Figure 3a, using the pocket-based function yields significantly better results than searching with the Vina function ($p < 0.001$). The PLP function appears to yield even better results, though not to a statistically significant degree.

In blind docking, the Pocket scoring function performs substantially better than both the Vina and PLP functions (Figure 3b), with accuracy into the double digits where the other functions struggle.

This indicates that the Pocket function more effectively samples the ligand-protein interaction space.[ii] This could be attributed to the smoothness of the function, as it only contains two terms, whereas the Vina and PLP functions are made up of multiple terms (see Appendix A). Hereby, the Pocket function is more predictable, leading to a faster and easier optimization path. The Vina function also heavily punishes poses that intersect with the protein through its `Repulsion` term. This leads to a very small "well" in the potential energy surface for each valid pose, as a small deviation in the pose can lead to heavily punished overlap with the protein. In contrast, the Pocket function is forgiving, with more gradual slopes leading to broader wells. These are much easier to find.

Smoothness like this results in more varied and optimized poses, even though the Pocket function itself is not a good representation of the true potential energy surface of the ligand-protein interaction. This is compensated for by using a physically more complicated and accurate function in the refinement phase.

The strong performance of the Pocket function in blind docking can be attributed to its efficient filtering of potential binding sites. By identifying likely binding pockets and guiding poses toward them, the search space is substantially reduced compared to docking without the Pocket function, resulting in significantly improved search performance.

Furthermore, the pocket-based approach is substantially faster (Figure 3d; $p < 0.0001$), yielding an average speed-up of over 6× ($p < 0.0001$) compared to the Vina function, and over 2.8× ($p < 0.01$) compared to Plants' PLP function. This difference can be attributed to the number of pairwise comparisons that are calculated by each function. The pocket function only cares for the closest pocket-sphere, where the other functions use the k nearest protein points to determine a score. For a more detailed analysis of the computational complexity of the pocket function, and its comparison to Vina, see Appendix B.

These results indicate that the Vina scoring function performs significantly worse than the Pocket and PLP functions in the searching phase of docking, with the Pocket function completing the search in the shortest time. Furthermore, the Pocket function far outperforms both other functions when executing whole-protein blind docking.

Scoring Functions for Refining

The same three functions were tested during the refinement phase. While effective in the searching phase, the Pocket function shows reduced performance when used here, as shown in Figure 3e. This can be explained by the lower complexity of this function. While this allowed for efficient sampling during the searching phase, the absence of any modeling of physical interactions prevents the function to effectively score poses.

There is no significance difference between the Vina and PLP functions. The PLP function is, however, notably faster (Figure 3d), and would therefore be the better function to use on this specific dataset.

It is important to note that the performance of a scoring function is highly dependent on the group of targets that are

[ii]When combined with the `Crowding` function, see **Niching** below.

tested [29, 30]. The choice of scoring function should therefore carefully be made based on the characteristics of the target system and, where possible, supported by preliminary testing.

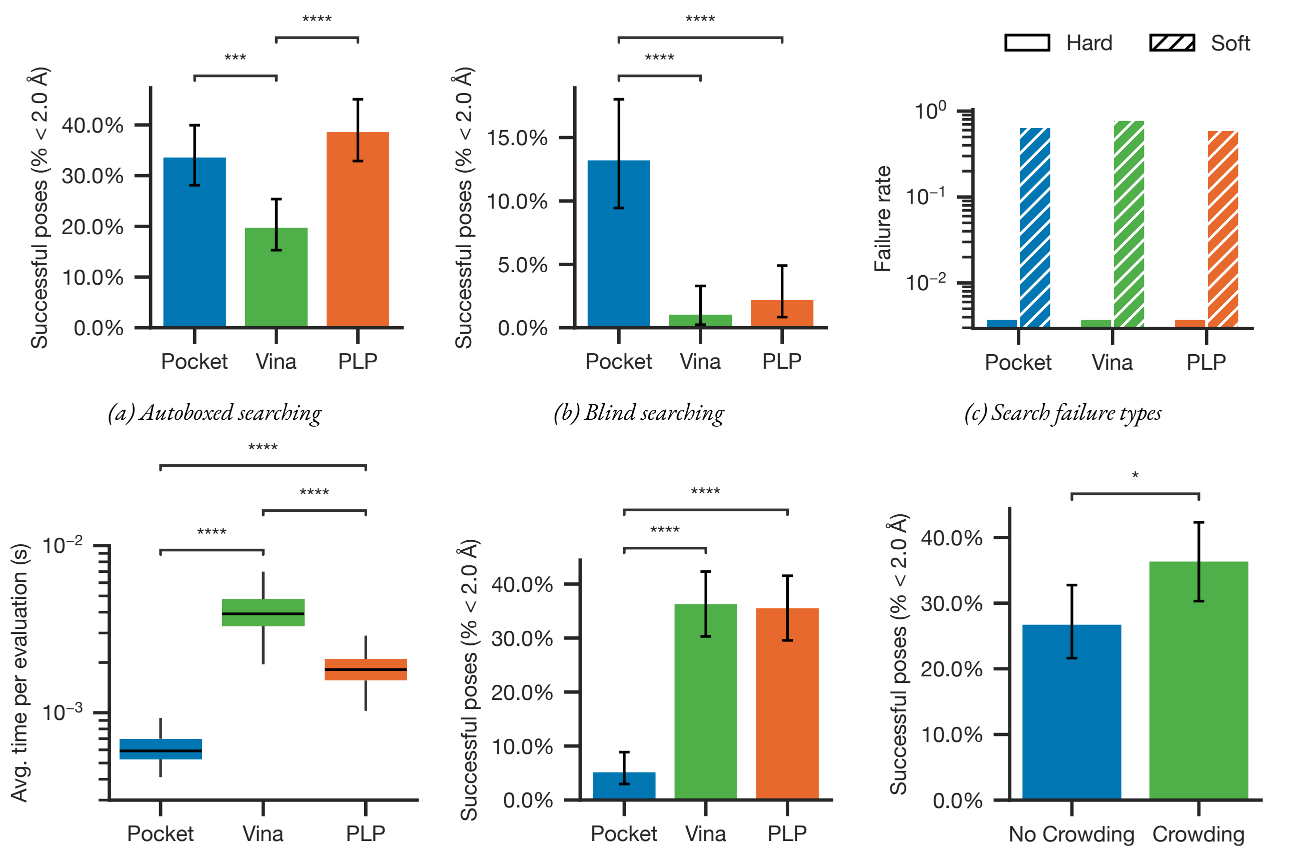
Searching

Alongside the scoring function, the searching algorithm is a key factor distinguishing one docking program from another. Therefore, we compared a range of different global optimization algorithms, all implemented in the *pygmo* Python package [22]. Among these are a GA, ACO and Simulated Annealing (SA), as used in tools like GOLD, PLANTS and Gnina [3, 4, 6], alongside other common optimization techniques.

Figure 4 shows the performance of the tested algorithms. Notably, ACO, GA, and SA — well-established methods in the field of molecular docking — performed slightly better than the rest, although the differences are not statistically significant. This suggests that the type of optimization algorithm used in the searching phase does not significantly affect the results, at least when using Pyrite's Pocket scoring function.

However, looking at Figure 3c, it follows that in nearly all cases where the correct pose is not found, a soft error is to blame. This indicates a failure to search the space effectively. As the error distribution is similar across different scoring functions, the Vina and PLP function show good performance in the refinement phase, and these are derived from existing docking implementations, the issue appears not to lie with the scoring functions, but with the searching algorithms or problem definition. This is explored further in section 3.4 Docking Results.

All functions tested in Figure 4 allow for the specification of a set number of scoring function evaluations per pose, except for SA. The number of function evaluations in this algorithm is dependent on the number of degrees of freedom in the feature vector, i.e., it will devote more iterations to more complex molecules with more dihedral angles. This results in similar per-



(a) Autoboxed searching *(b) Blind searching* *(c) Search failure types*

(d) Function runtime *(e) Refinement* *(f) Niching*

Figure 3: Performance of various docking methods. Performance was evaluated on the PDBbind demo dataset [26]. A result is considered successful if the top ranked pose has an RMSD to the co-crystallized ligand of less than 2.0 Å*. 95% confidence intervals are indicated by error bars. a, b, c) Docking performance when searching using Pyrite's Pocket scoring function, the Vina scoring function, and the PLP scoring function. ACO is used as the searching algorithm, diversity was ensured using the Crowding function in Pyrite. In all cases, the Vina function is used for refinement. d) Function runtime for all three tested functions. e) Docking performance when refining using Pyrite's Pocket scoring function, the Vina scoring function, and the PLP scoring function. Poses are first searched using ACO with the Pocket and Crowding scoring functions. f) Docking performance with and without the use of the Crowding function for diversification. ACO with the Pocket function is used for searching, and Vina is used for refining.*

$n = 260$; **** $p < 0.0001$, *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$. *RMSD: Root Mean Square Deviation; PLP: Piecewise Linear Potential; ACO: Ant Colony Optimization.*

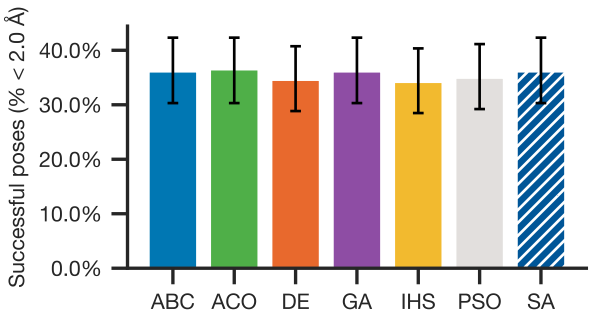


Figure 4: Docking performance using different searching algorithms with Pyrite's Pocket function for searching. During the searching, the Crowding function niching method is used, afterwards the Vina function is used for refinement. A result is considered successful if the top ranked pose has an RMSD to the co-crystallized ligand of less than 2.0 Å. *95% confidence interval is indicated by error bars. SA (hatched) does not result in the same number of function evaluations as the other algorithms (Supplementary Figure C.1). RMSD: Root Mean Square Deviation; ABC: Artificial Bee Colony; ACO: Ant Colony Optimization; DE: Differential Evolution; GA: Genetic Algorithm; IHS: Improved Harmony Search; PSO: Particle Swarm Optimization; SA: Simulated Annealing.*

formance to ACO and GA while using slightly fewer function evaluations (Supplementary Figure C.1).

Implementing this method of dynamic comprehensiveness in the workflow of other searching algorithms could lead to improved performance in the same time frame, as more time is invested in the ligands that are more complicated to dock. However, care should be taken when applying this method for virtual screening or other comparative ligand analyses to ensure that improved performance for more complex ligands does not come at the expense of reduced accuracy for simpler ones. This would counteract the torsional penalty typically applied in existing scoring functions, thereby neglecting the entropy cost associated with larger ligands.

Niching An important part of the searching phase is the diversification of generated poses. This is illustrated by Figure 3f, where the impact of the use of Pyrite's `Crowding` function during searching is shown. This function significantly improves the diversity of the poses generated during the searching process, allowing the identification of lower energy poses in the refinement phase. This is especially important when using the Pocket function to search, as its global minimum does not necessarily align with the true global energy minimum.

This crowding method, however, requires that the entire search loop runs to completion for every generated pose. This process is not parallelizable, as each generated pose needs to be added to the next run. Furthermore, it results in a lot of the searching steps and calculations being wasted. For example, when using population based optimization methods such as the algorithms in Figure 4[iii], entire populations are optimized in each run — in the case of Figure 4, 64 poses. Only the best pose is kept, the rest is discarded.

Ideally, we would want to optimize the populations to be diverse, such that a single optimization run is satisfactory. Niching optimization algorithms promise this, but they often require parameters that need to be very precisely balanced [31]. This can be prevented by using a more stochastic approach, by relying on a *ring* topology. In most regular population based optimization algorithms, a new generation is created by "breeding" within the entire population, every member can be influenced by every other member. This is called a *fully-connected* topology, and drives fast convergence. Contrary, in a ring topology every member of the population can only interact with its two neighbors, promoting more diverse local *neighborhoods*, leading to diverse output poses after only a single optimization run.

Contrary to the crowding method, however, this method does not enforce the niching. The ring topology promotes local neighborhoods, but nothing is stopping these neighborhoods from converging. A more recent development, multi-modal optimization, is able to enforce this [23]. This method requires a special optimization algorithm that is able to take multiple metrics into account when deciding on the next generation. It can therefore combine our scoring function with a diversity-preserving mechanism, ensuring diversity within a single optimization run.

Pyrite is able to employ all these niching methods, either through custom implementation or existing Python libraries, such as *pygmo* [22]. Implementation and testing of these techniques could lead to a significant improvement in both efficiency and accuracy.

Docking Results

All these techniques were combined in an example workflow, to gain an indication of the performance of Pyrite compared to Smina. Pyrite's Pocket function combined with the Crowding function and an ACO optimizer was chosen for the searching phase. The Pocket function shows good performance in both autoboxed docking and blind docking (Figure 3a & 3b), and is significantly faster than both other functions tested (Figure 3d). The Crowding function significantly improves searching performance (Figure 3f), and Ant Colony Optimization has shown to be a valid option for searching. Furthermore, it can easily be adapted to use a ring topology or multimodal optimization, allowing for straightforward prototyping and testing of these methods in the future.

For this run, the searching parameters are more comprehensive than earlier described in the methods. Instead of 4×8 poses, 4×12 poses are generated, and the number of function evaluations per pose is increased to 25 600. As the scoring function during refinement, PLP is used, as it is significantly faster than the Vina scoring function and exhibits similar performance (Figure 3d & 3e).

The docking results are compared to Smina [32], using the same structure preparation methods and default settings.

Figure 5 shows the outcome of this workflow. Pyrite is able to achieve an accuracy of 37 ± 6 % on the PDBbind demo dataset. This is significantly worse than Smina, which achieves an accuracy of 62 ± 6 %. The same trend holds for blind docking, where Pyrite achieves an accuracy of 16 ± 3 %, and Smina achieves 28 ± 5 %.

Several issues arise when inspecting the generated poses. In

[iii] All but simulated annealing use populations.

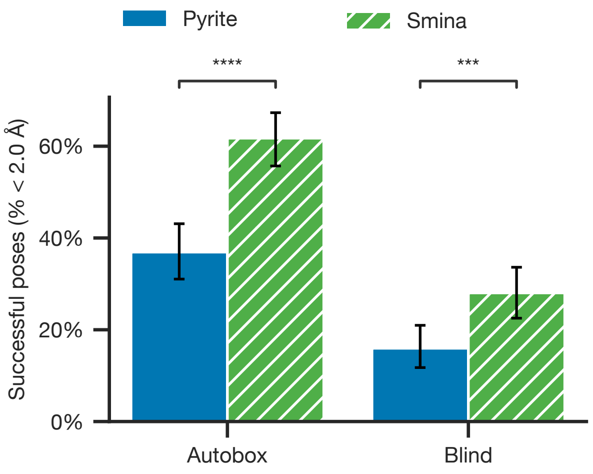
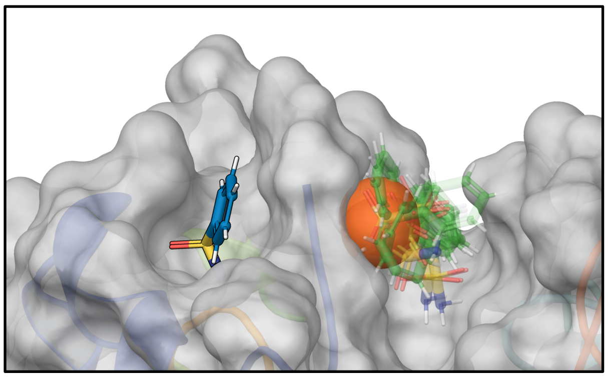
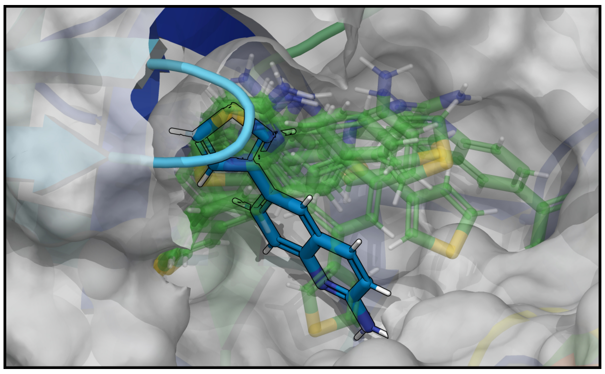


Figure 5: Docking performance of Pyrite compared to Smina*. The Pocket function combined with the Crowding function is used for Pyrite for searching. Ant Colony Optimization is used as searching algorithm. The PLP function is used for refinement. A result is considered successful if the top ranked pose has an RMSD to the co-crystallized ligand of less than* 2.0 Å*. Performance is evaluated on the PDBbind dataset, 95% confidence intervals are indicated as error bars.* PLP: Piecewise Linear Potential



(a) Pocket finding failure



(b) Searching depth failure

Figure 6: Two examples of soft failures in the Pyrite docking results. Pockets are shown in orange, the co-crystallized pose is shown in blue, and the output docking poses are shown in green. a) An example of a Pocket failure. Here, the pocket depth criterion is defined too strictly, resulting in erasure of the true binding pocket, which is relatively shallow. PDB: 4JSZ. b) An example of a searching depth failure. The pocket is defined correctly and the poses are placed in the right area, but the lowest energy pose is not found. PDB: 3RSX.

some cases, the Pocket finding algorithm fails to find the correct binding pocket (Figure 6a). This can be caused by a pocket depth filter that is too strict. When the true binding pocket is shallow, as in Figure 6a, this will result in it not being considered. This leads to a failed docking outcome, despite the apparent simplicity of the problem — only a single rotatable bond is present in the ligand.

Adjusting the depth criterion for pocket detection to allow for shallower pockets could resolve this issue, but would reduce the overall accuracy of the workflow. Shallower criteria would result in larger pockets, reducing the incentive for ligands to bury into the protein. A better solution would be to make the depth criterion dynamic, allowing for both shallow and deep pockets in the same run without enlarging the deeper pockets. Further investigation is necessary to assess the effectiveness of this method.

Other than the cases where the pocket is ill-defined, most errors appear to be caused by insufficient sampling of the space. In Figure 6b, for example, all poses are placed in a correctly defined pocket, and seem to be fairly diverse. However, no pose is similar to the co-crystallized binding pose. The degrees of freedom of this ligand are minimal, with only one dihedral angle. This once again indicates that current Pyrite workflows are limited by the searching and niching process.

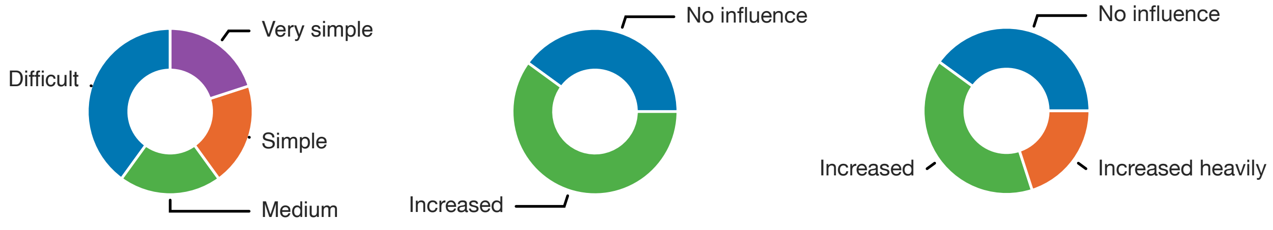
Important to note is that Pyrite was not able to successfully load some of the structures in the PDBbind dataset due to limitations in the bond-inference mechanism in the rdkit. The problematic structures ($N = 20; 7.0\%$) appear to be evenly spaced in chemical space (visual inspection) and not exclusive to certain protein classes, and are excluded in the results of both the Pyrite and Smina runs. Addressing this problem would greatly improve real-world reliability.

As previously mentioned, we hypothesize that the primary opportunities for improvement in Pyrite lie in optimizing the searching and niching processes. These efforts can be further supported by increasing the reliability and accuracy of the binding pocket identification algorithm. Addressing these limitations will be a critical step toward enhancing the performance and reliability of Pyrite, bringing it closer to established tools such as Smina.

Future Outlook

We developed Pyrite with the goal of creating an open, extensible, and accessible molecular docking framework. Our aim is not only to make it usable by anyone, but also maintainable by the broader community — enabling others to build upon this work through the collaborative power of open-source software. The entire source-code and documentation of Pyrite is therefore available on GitHub [15] under a non-commercial license with attribution, allowing modification and use for non-commercial purposes.

In addition, we will actively drive the development of Pyrite, by improving and optimizing its core functionality, creating



(a) Assignment difficulty *(b) Interest in docking* *(c) Interest in programming*

Figure 7: Survey result for students that completed the Pyrite assignment in the Current Topics in Computational Medicinal Chemistry & Toxicology course [33] ($N = 5$). Further data is shown in Table D.1.

more example workflows and user guides, and developing novel tools using Pyrite. Appendix E. Roadmap elaborates on this.

4 Educational Case Study

The value of Pyrite in an educational context has been evaluated in a pilot study among third-year bachelor students in Pharmaceutical Sciences via a practicum during the course *Current Topics in Computational Medicinal Chemistry & Toxicology* [33]. The students were tasked with completing a Pyrite Jupyter Notebook incorporating docking results they themselves gathered in earlier weeks of the course. This Jupyter Notebook is published on the Pyrite GitHub [15].

The code that was supplied to the students consisted of the loading of the receptor and docking results, the calculation of a score using an abstract scoring function, and the reporting of the results. During the guided assignment, the students were first asked to explain the code, and then devise a mathematical model that would describe the quality of a docking pose as a function of the distance between a certain ligand atom and a certain protein atom — most students were working on cytochrome P450 site of metabolism prediction, and were thus working with the distances between the possible sites of metabolism on the ligand and the iron of the cytochrome heme group. Students used literature on the transition states and ideal reaction distances of these different sites to create their model.

After formulating this model, the students were tasked with implementing it into the abstract scoring function. Running the notebook calculated the score for each of the supplied docking poses. During this process, the students had access to a Pyrite viewer, which directly visualized the docking poses and the corresponding calculated scores.

The assignment placed a heavy focus on the design of a scoring function, a fundamental docking concept that the students indicated was unfamiliar to them. By approaching this in a bottom-up fashion, we hope the underlying chemical and physical concepts are more effectively conveyed to the students.

Evaluation

The students, with little to no programming experience, were all able to complete the entire assignment within 6 hours, having successfully implemented a custom and highly specific physics-based scoring function using Pyrite. Several different approaches were taken, with most ending up with a Gaussian or linear function.

Student feedback was gathered using an evaluation survey ($N = 5$) of which the results are shown in Figure 7 and Appendix D. As can be seen in Figure 7a, the difficulty of the assignment was within an appropriate range, which is supported by the time in which the students completed the practical. Furthermore, the interest of students in docking and programming appears to generally have increased after the assignment (Figure 7b & 7c). We hypothesize that this is a result of a more hands-on, fundamental approach, which resulted in a more challenging and compelling assignment and a better understanding of molecular docking concepts.

An extended evaluation backs this hypothesis. Students reported that this assignment provided them with novel insights into the inner workings of molecular docking, as well as a deeper understanding of the mathematical and physical concepts underlying scoring functions. They also gained appreciation for scripting-based tools like Pyrite, and indicated that the workflow created during the assignment was of greater value in the data analysis during the rest of the course.

For other students, the coding was harder to master, and acted as a hurdle during the assignment. As Python currently has no prior place in the *Farmaceutische Wetenschappen* curriculum [34], a short Python introduction — a "crash course" — could help alleviate this. Students also indicated that they would have liked to complete an entire molecular docking workflow in Pyrite and to gain a deeper insight into pose generation.

Conclusively, preliminary results indicate that this kind of assignment would be a valuable addition to the *Farmaceutische Wetenschappen* curriculum, as it provided students with helpful and novel insights into fundamental computational chemistry concepts. However, students also report that the level of programming knowledge required was too high, and introductory programming concepts should either be integrated more in the *Farmaceutische Wetenschappen* curriculum, or into the assignment itself. This warrants a follow-up study with a bigger sample size and scope to validate if this inclusion is feasible and if it truly enriches the *Farmaceutische Wetenschappen* curriculum.

Future outlook

We believe Pyrite can play an instrumental role in biomolecular simulation education, especially around the topic of molecular docking. We therefore strive to assess the feasibility of inclusion in the *Farmaceutische Wetenschappen* curriculum, and aim to develop multiple educational resources using Pyrite. This is further elaborated on in Appendix E. Roadmap.

5 Conclusion

In this thesis, we presented Pyrite, an open, Python-based molecular-docking framework designed for openness, extensibility and accessibility. With Pyrite, we hope to lower the barrier for methodological research in molecular docking and related problems.

Our experiments show that Pyrite's modular design delivers tangible advantages. We developed a novel, pocket-based scoring function, which runs two- to six-times faster than the Vina and PLP functions, while matching or exceeding their search accuracy, especially in whole-protein docking. When the best poses are further refined with the PLP potential, Pyrite achieves 37 ± 6 % accuracy in autoboxed docking and 16 ± 3 % in whole-protein docking on the PDBbind demo set. While these figures trail behind existing docking implementations, we have shown that Pyrite can prove instrumental in the comparison of existing docking methods and the development of novel ones.

Error analysis revealed that most failures stem from searching rather than scoring. An overly strict pocket identification implementation results in the exclusion of shallow but biologically relevant binding pockets, and limitations in Pyrite's current niching algorithms result in insufficient sampling of viable poses. Addressing these points, through dynamic pocket filters and richer niching strategies, forms the first item on our roadmap.

Beyond research, Pyrite demonstrated its educational value in a pilot practicum. Third-year Pharmaceutical Sciences students with minimal coding experience implemented custom and novel physics-based scoring functions within hours and reported heightened interest in molecular docking and scientific programming. This suggests that Pyrite can serve as a valuable educational tool for teaching biomolecular simulation concepts.

In summary, Pyrite already offers a transparent test-bed for exploring new optimization algorithms and developing scoring functions. By resolving the identified search bottlenecks and encouraging community-driven contributions, we hope Pyrite can evolve into a versatile framework that accelerates both fundamental docking research and the training of the next generation of computational scientists.

6 References

[1] L. Pinzi and G. Rastelli, 'Molecular Docking: Shifting Paradigms in Drug Discovery', *International Journal of Molecular Sciences*, vol. 20, no. 18, p. 4331, 18 Jan. 2019, ISSN: 1422-0067. DOI: 10.3390/ijms20184331. [Online]. Available: https://www.mdpi.com/1422-0067/20/18/4331.

[2] I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge and T. E. Ferrin, 'A geometric approach to macromolecule-ligand interactions', *Journal of Molecular Biology*, vol. 161, no. 2, pp. 269–288, 25th Oct. 1982, ISSN: 0022-2836. DOI: 10.1016/0022-2836(82)90153-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002228368290153X.

[3] M. L. Verdonk, J. C. Cole, M. J. Hartshorn, C. W. Murray and R. D. Taylor, 'Improved protein–ligand docking using GOLD', *Proteins: Structure, Function, and Bioinformatics*, vol. 52, no. 4, pp. 609–623, 2003, ISSN: 1097-0134. DOI: 10.1002/prot.10465. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.10465.

[4] O. Korb, T. Stützle and T. E. Exner, 'PLANTS: Application of Ant Colony Optimization to Structure-Based Drug Design', in *Ant Colony Optimization and Swarm Intelligence*, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli and T. Stützle, Eds., Berlin, Heidelberg: Springer, 2006, pp. 247–258, ISBN: 978-3-540-38483-0. DOI: 10.1007/11839088_22.

[5] 'TIOBE Index', TIOBE. (), [Online]. Available: https://www.tiobe.com/tiobe-index/.

[6] A. T. McNutt, P. Francoeur, R. Aggarwal, T. Masuda, R. Meli, M. Ragoza, J. Sunseri and D. R. Koes, 'GNINA 1.0: Molecular docking with deep learning', *Journal of Cheminformatics*, vol. 13, no. 1, p. 43, 9th Jun. 2021, ISSN: 1758-2946. DOI: 10.1186/s13321-021-00522-2. [Online]. Available: https://doi.org/10.1186/s13321-021-00522-2.

[7] J. Fuhrmann, A. Rurainski, H.-P. Lenhof and D. Neumann, 'A new Lamarckian genetic algorithm for flexible ligand-receptor docking', *Journal of Computational Chemistry*, vol. 31, no. 9, pp. 1911–1918, 2010, ISSN: 1096-987X. DOI: 10.1002/jcc.21478. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21478.

[8] M. J. García-Godoy, E. López-Camacho, J. García-Nieto, A. J. Nebro and J. F. Aldana-Montes, 'Solving Molecular Docking Problems with Multi-Objective Metaheuristics', *Molecules*, vol. 20, no. 6, pp. 10 154–10 183, 6 Jun. 2015, ISSN: 1420-3049. DOI: 10.3390/molecules200610154. [Online]. Available: https://www.mdpi.com/1420-3049/20/6/10154.

[9] J. Fuhrmann, A. Rurainski, H.-P. Lenhof and D. Neumann, 'A new method for the gradient-based optimization of molecular complexes', *Journal of Computational Chemistry*, vol. 30, no. 9, pp. 1371–1378, 2009, ISSN: 1096-987X. DOI: 10.1002/jcc.21159. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21159.

[10] *RDKit: Open-source cheminformatics.* [Online]. Available: https://www.rdkit.org.

[11] G. Landrum, P. Tosco, B. Kelley, R. Rodriguez, D. Cosgrove, R. Vianello, sriniker, P. Gedeck, G. Jones, E. Kawashima, NadineSchneider, D. Nealschneider, A. Dalke, M. Swain, B. Cole, tadhurst-cdd, S. Turk, A. Savelev, A. Vaucher, M. Wójcikowski, I. Take, R. Walker, V. F. Scalfani, H. Faara, K. Ujihara, D. Probst, N. Maeder, J. Monat, J. Lehtivarjo and g. godin, *Rdkit/rdkit: 2025_03_4 (Q1 2025) Release*, version Release_2025_03_4, Zenodo, 30th Jun. 2025. DOI: 10.5281/zenodo.15773589. [Online]. Available: https://zenodo.org/records/15773589.

[12] O. Trott and A. J. Olson, 'AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading', *Journal of Computational Chemistry*, vol. 31, no. 2, pp. 455–461, 2010, ISSN: 1096-987X. DOI: 10.1002/jcc.21334. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21334.

[13] J. Gasteiger and M. Marsili, 'Iterative partial equalization of orbital electronegativity—a rapid access to atomic charges', *Tetrahedron*, vol. 36, no. 22, pp. 3219–3228, 1st Jan. 1980, ISSN: 0040-4020. DOI: 10.1016/0040-4020(80)80168-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0040402080801682.

[14] G. M. Verkhivker, D. Bouzida, D. K. Gehlhaar, P. A. Rejto, S. Arthurs, A. B. Colson, S. T. Freer, V. Larson, B. A. Luty, T. Marrone and P. W. Rose, 'Deciphering common failures in molecular docking of ligand-protein complexes', *Journal of Computer-Aided Molecular Design*, vol. 14, no. 8, pp. 731–751, 1st Nov. 2000, ISSN: 1573-4951. DOI: 10.1023/A:1008158231558. [Online]. Available: https://doi.org/10.1023/A:1008158231558.

[15] M. J. van der Lugt and D. A. Poole III, *PooleChem/Pyrite*, PooleChem, 3rd Jul. 2025. [Online]. Available: https://github.com/PooleChem/Pyrite.

[16] S. Bhandari, K. B. Sahay and R. K. Singh, 'Optimization Techniques in Modern Times and Their Applications', in *2018 International Electrical Engineering Congress (iEECON)*, Mar. 2018, pp. 1–4. DOI: 10.1109/IEECON.2018.8712308. [Online]. Available: https://ieeexplore.ieee.org/document/8712308.

[17] K. Karthick, 'Comprehensive Overview of Optimization Techniques in Machine Learning Training', *Control Systems and Optimization Letters*, vol. 2, no. 1, pp. 23–27, 1 13th Feb. 2024, ISSN: 2985-6116. DOI: 10.59247/csol.v2i1.69. [Online]. Available: https://ejournal.csol.or.id/index.php/csol/article/view/69.

[18] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa and P. van Mulbregt, 'SciPy 1.0: Fundamental algorithms for scientific computing in Python', *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020, ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: https://www.nature.com/articles/s41592-019-0686-2.

[19] F. Gao and L. Han, 'Implementing the Nelder-Mead simplex algorithm with adaptive parameters', *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 1st Jan. 2012, ISSN: 1573-2894. DOI: 10.1007/s10589-010-9329-3. [Online]. Available: https://doi.org/10.1007/s10589-010-9329-3.

[20] D. J. Wales and J. P. K. Doye, 'Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms', *The Journal of Physical Chemistry A*, vol. 101, no. 28, pp. 5111–5116, 1st Jul. 1997, ISSN: 1089-5639. DOI: 10.1021/jp970984n. [Online]. Available: https://doi.org/10.1021/jp970984n.

[21] R. Storn and K. Price, 'Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces', *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1st Dec. 1997, ISSN: 1573-2916. DOI: 10.1023/A:1008202821328. [Online]. Available: https://doi.org/10.1023/A:1008202821328.

[22] F. Biscani and D. Izzo, 'A parallel global multiobjective framework for optimization: Pagmo', *Journal of Open Source Software*, vol. 5, no. 53, p. 2338, 13th Sep. 2020, ISSN: 2475-9066. DOI: 10.21105/joss.02338. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.02338.

[23] X. Li, M. G. Epitropakis, K. Deb and A. Engelbrecht, 'Seeking Multiple Solutions: An Updated Survey on Niching Methods and Their Applications', *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 518–538, Aug. 2017, ISSN: 1941-0026. DOI: 10.1109/TEVC.2016.2638437. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7782373.

[24] D. Butina, 'Unsupervised Data Base Clustering Based on Daylight's Fingerprint and Tanimoto Similarity: A Fast and Automated Way To Cluster Small and Large Data Sets', *Journal of Chemical Information and Computer Sciences*, vol. 39, no. 4, pp. 747–750, 26th Jul. 1999, ISSN: 0095-2338. DOI: 10.1021/ci9803381. [Online]. Available: https://doi.org/10.1021/ci9803381.

[25] C. Zhu, R. H. Byrd, P. Lu and J. Nocedal, 'Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization', *ACM Trans. Math. Softw.*, vol. 23, no. 4, pp. 550–560, 1st Dec. 1997, ISSN: 0098-3500. DOI: 10.1145/279232.279236. [Online]. Available: https://dl.acm.org/doi/10.1145/279232.279236.

[26] R. Wang, X. Fang, Y. Lu, C.-Y. Yang and S. Wang, 'The PDBbind Database: Methodologies and Updates', *Journal of Medicinal Chemistry*, vol. 48, no. 12, pp. 4111–4119, 1st Jun. 2005, ISSN: 0022-2623. DOI: 10.1021/jm048957q. [Online]. Available: https://doi.org/10.1021/jm048957q.

[27] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, R. P. Wiewiora, B. R. Brooks and V. S. Pande, 'OpenMM 7: Rapid development of high performance algorithms for molecular dynamics', *PLOS Computational Biology*, vol. 13, no. 7, R. Gentleman, Ed., e1005659, 26th Jul. 2017, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005659. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1005659.

[28] D. A. Case, H. M. Aktulga, K. Belfon, D. S. Cerutti, G. A. Cisneros, V. W. D. Cruzeiro, N. Forouzesh, T. J. Giese, A. W. Götz, H. Gohlke, S. Izadi, K. Kasavajhala, M. C. Kaymak, E. King, T. Kurtzman, T.-S. Lee, P. Li, J. Liu, T. Luchko, R. Luo, M. Manathunga, M. R. Machado, H. M. Nguyen, K. A. O'Hearn, A. V. Onufriev, F. Pan, S. Pantano, R. Qi, A. Rahnamoun, A. Risheh, S. Schott-Verdugo, A. Shajan, J. Swails, J. Wang, H. Wei, X. Wu, Y. Wu, S. Zhang, S. Zhao, Q. Zhu, T. E. I. Cheatham, D. R. Roe, A. Roitberg, C. Simmerling, D. M. York, M. C. Nagan and K. M. J. Merz, 'AmberTools', *Journal of Chemical Information and Modeling*, vol. 63, no. 20, pp. 6183–6191, 23rd Oct. 2023, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.3c01153. [Online]. Available: https://doi.org/10.1021/acs.jcim.3c01153.

[29] W. Xu, A. J. Lucke and D. P. Fairlie, 'Comparing sixteen scoring functions for predicting biological activities of ligands for protein targets', *Journal of Molecular Graphics and Modelling*, vol. 57, pp. 76–88, 1st Apr. 2015, ISSN: 1093-3263. DOI: 10.1016/j.jmgm.2015.01.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1093326315000285.

[30] D. Jereva, P. Alov, I. Tsakovska, M. Angelova, V. Atanassova, P. Vassilev, N. Ikonomov, K. Atanassov, I. Pajeva and T. Pencheva, 'Application of InterCriteria Analysis to Assess the Performance of Scoring Functions in Molecular Docking Software Packages', *Mathematics*, vol. 10, no. 15, p. 2549, 15 Jan. 2022, ISSN: 2227-7390. DOI: 10.3390/math10152549. [Online]. Available: https://www.mdpi.com/2227-7390/10/15/2549.

[31] X. Li, 'Niching Without Niching Parameters: Particle Swarm Optimization Using a Ring Topology', *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 1, pp. 150–169, Feb. 2010, ISSN: 1941-0026. DOI: 10.1109/TEVC.2009.2026270. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5352335.

[32] D. R. Koes, M. P. Baumgartner and C. J. Camacho, 'Lessons Learned in Empirical Scoring with smina from the CSAR 2011 Benchmarking Exercise', *Journal of Chemical Information and Modeling*, vol. 53, no. 8, pp. 1893–1904, 26th Aug. 2013, ISSN: 1549-9596. DOI: 10.1021/ci300604z. [Online]. Available: https://doi.org/10.1021/ci300604z.

[33] D. A. Poole III. 'Current Topics in Computational Medicinal Chemistry and Toxicology', Vrije Universiteit Studiegids. (), [Online]. Available: https://studiegids.vu.nl/nl/vakken/2024-2025/XB_430547.

[34] 'Pharmaceutical Sciences', Vrije Universiteit Studiegids. (), [Online]. Available: https://studiegids.vu.nl/en/Bachelor/2024-2025/farmaceutische-wetenschappen.

[35] O. Korb, T. Stützle and T. E. Exner, 'Empirical Scoring Functions for Advanced Protein-Ligand Docking with PLANTS', *Journal of Chemical Information and Modeling*, vol. 49, no. 1, pp. 84–96, 26th Jan. 2009, ISSN: 1549-9596. DOI: 10.1021/ci800298z. [Online]. Available: https://doi.org/10.1021/ci800298z.

[36] S. Maneewongvatana and D. M. Mount. 'Analysis of approximate nearest neighbor searching with clustered point sets'. arXiv: cs/9901013. (26th Jan. 1999), [Online]. Available: http://arxiv.org/abs/cs/9901013, pre-published.

[37] C. B-Rao, J. Subramanian and S. D. Sharma, 'Managing protein flexibility in docking and its applications', *Drug Discovery Today*, vol. 14, no. 7, pp. 394–400, 1st Apr. 2009, ISSN: 1359-6446. DOI: 10.1016/j.drudis.2009.01.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1359644609000063.

[38] R. Quiroga and M. A. Villarreal, 'Vinardo: A Scoring Function Based on Autodock Vina Improves Scoring, Docking, and Virtual Screening', *PLOS ONE*, vol. 11, no. 5, e0155183, 12th May 2016, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0155183. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0155183.

[39] 'Zenodo'. (), [Online]. Available: https://zenodo.org/.

[40] 'Desmos | Beautiful free math.' (), [Online]. Available: https://www.desmos.com/.

Appendices

A Scoring Function Definitions using Pyrite

```
pocket = DistanceToPocket(ligand, pocket) + 1e-2 * InternalEnergy(ligand)

vina = ((-0.035579 * Gaussian(ligand, receptor, offset=0.0, width=0.5)
    + -0.005156 * Gaussian(ligand, receptor, offset=3.0, width=2.0)
    + 0.840245 * Repulsion(ligand, receptor, offset=0.0)
    + -0.035069 * Hydrophobic(ligand, receptor, good=0.5, bad=1.5)
    + -0.587439 * NonDirHBond(ligand, receptor, good=-0.7, bad=0.0)
    + 1e-2 * InternalEnergy(ligand))
    / (1 + ((0.1 * (1.923 + 1)) * (NumTors(ligand))) / 5))

plp = PlantsPLP(ligand, receptor) + 1e-2 * InternalEnergy(ligand)
```

The PlantsPLP function itself contains multiple terms, as described in [15] and [35].

B Runtime Complexity of the Pocket Scoring Function

Both the Pocket and Vina functions use *scipy*'s `KDTree` [18, 36], a query of which has a worst-case time complexity of $O(mn)$ and average time complexity of $O(m \cdot logn)$, where n is the number of points in the tree, in our case the number of spheres in the pocket or atoms in the protein, and m is the number of points to be queried, in our case the number of atoms in the ligand. For both functions, the number of dimensions is low ($d = 3$) and the problem is well defined, leading to a balanced tree. These are ideal circumstances, meaning the time complexity of this part of the functions is more inclined to the average case of $O(m \cdot logn)$.

Both functions process the queried data with a worst-case time complexity of $O(mk)$, where k is the number of nearest points queried. This leads to an expected time complexity for both functions of $O(m \cdot \log n + mk)$. As k is a constant, the true average time complexity for both functions is $O(m \cdot \log n)$.

Regardless, from this we can see that, due to the lower number of k, the pocket-based function performs better, even though theoretically both functions scale evenly.

C Supplementary Figures

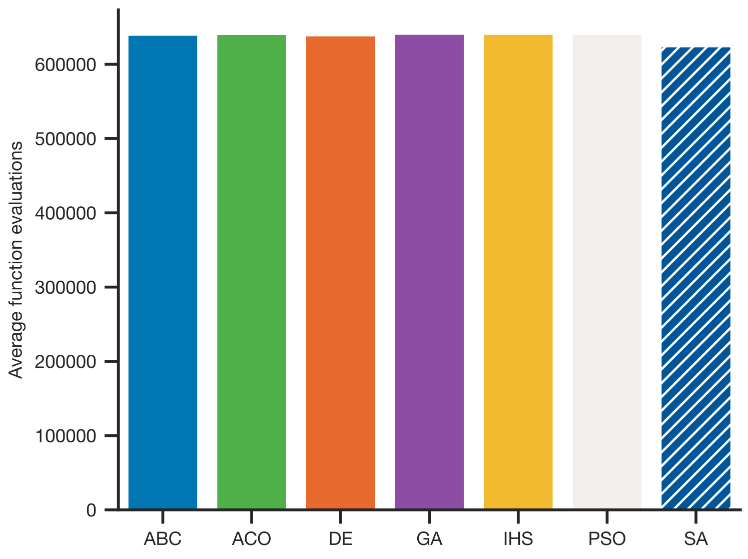


Figure C.1: The number of function evaluations executed during the searching phase of Figure 3a.

D Evaluation Survey CT-CMCT

Table D.1: Short primary evaluation of the Pyrite assignment of the CT-CMCT course.

What is your background in molecular docking?	What is your background in programming?	How simple do you feel MOE is to use?	How simple do you feel Pyrite (the program used in the assignment) is to use?	How much did you feel you learned about molecular docking from using MOE?	How much did you feel you learned about molecular docking from using Pyrite? (The program used in the assignment)	How has using MOE influenced your interest in molecular docking?	How has using the Pyrite tool influenced your interest in molecular docking?	How has using the Pyrite tool influenced your interest in programming?
3 – Bachelor-course	2 – A little experience	2 – Easy	1 – Very easy	3 – Medium	4	Equal	Increased	Increased
3 – Bachelor-course	2 – A little experience	3 – Medium	3 – Medium	4	3 – Medium	Equal	Equal	Equal
3 – Bachelor-course	3 – Experience with one programming language	3 – Medium	3 – Medium	4	3 – Medium	Increased	Increased	Equal
3 – Bachelor-course	2 – A little experience	3 – Medium	4 – Hard	3 – Medium	4	Equal	Equal	Equal
3 – Bachelor-course	1 – No experience	2 – Easy	4 – Hard	4	2	Increased heavily	Equal	Increased

Table D.2: *Long-answer secondary evaluation of the assignment of the CT-CMCT course. (In Dutch).*

Wat heb je geleerd door gebruik van een scripting-based tool (Pyrite), dat je niet eerder hebt geleerd door gebruik van een GUI-based tool (MOE, GOLD)?	Welke concepten binnen moleculaire docking zijn voor jou duidelijker geworden door het maken van het Pyrite practicum?	Welke onderdelen van moleculaire docking vind je nog steeds lastig om te begrijpen?	Wat voor toevoegingen of veranderingen aan Pyrite of aan het practicum zouden jouw leerproces vergemakkelijken?	Denk je dat Pyrite een toevoeging zou kunnen zijn op het FAR-curriculum, en zo ja, hoe zou je dat zien?	Is er nog iets anders wat je wil toevoegen over de Pyrite module, of de CT-CMCT cursus in het algemeen?
Door het gebruik van Pyrite zijn we actief bezig geweest met de data en met hoe we deze moesten interpreteren, waardoor ik meer inzicht kreeg in de onderliggende processen ten opzichte van de GUI-tools.	Dit sluit voor mij een beetje aan bij de vorige vraag: vooral het scoring-proces en de manier waarop ligand-receptor-interacties worden geëvalueerd.	De exacte berekeningen achter de scoringsfuncties vind ik nog lastig.	Dat we daadwerkelijk zouden kunnen docken in PyMOL zodat we de output van Pyrite direct kunnen vergelijken.	Ik denk dat het zeker een goede toevoeging is. Het laat studenten kennismaken met scripting en dwingt tot een dieper begrip.	Ik vond het een leuke module om uit te voeren. Een beknopte handleiding zou nog helpen.
Hoe je moet scripten. Dat je de waarschijnlijkheid van ligand-posities beter begrijpt.	De waarschijnlijkheid van de afstand van ligand-receptor-bindingen.	Begrijpen waarom sommige poses gunstiger zijn dan andere.	Eerst een kennismaking met Python, hoe scripting werkt, en wat achtergrondinformatie over docking.	Ja, het is een ontzettend leuke manier om kennis te maken met docking buiten de GUI om.	Voor de CT-CMCT is het wel handig om een syllabus te hebben waarin alle stappen duidelijk staan.
Ik heb geleerd dat er meerdere manieren zijn om docking uit te voeren en dat automatisering met scripts reproduceerbare workflows oplevert.	Zoals net benoemd, is dat de manier hoe scoringsfuncties tot stand komen me duidelijker is geworden.	Het kiezen van de juiste parameters voor een docking.	Misschien dat een uitleg per stuk code ervoor zorgt dat het leerproces soepeler verloopt.	Dat zou kunnen. Ik denk dat het dan een eenmalige workshop zou moeten zijn in plaats van een volledige module.	
Dat er veel meer flexibiliteit is met de data bij scripting-tools dan bij GUI-tools.	Dat de configuratie van de docking en het belang van parameterkeuze helderder zijn.	Waarom een bepaalde docking-pose soms slecht scoort ondanks visueel goede fit.	Er zou ergens tijdens de simulatie staan hoe alles teruggekoppeld wordt naar de resultaten.	Het zou wel een goede toevoeging zijn voor snelle docking berekeningen.	Niet echt nee

E Roadmap

Here, we will present a roadmap for the near future of Pyrite. For every feature or test, the requirements, relevance and value, and implementation details will succinctly be stated and explained. This section is divided into three parts, *Functional Enhancements*, which will describe updates and improvements to Pyrite's core feature set, *Testing and Workflow Optimization*, where recommendations stated in the Discussion and Conclusion regarding the workflow are elaborated on and novel methods are described, and *Documentation*, where ideas for the expansion of the documentation and education plans are noted.

Functional Enhancements

Consolidate Ligand and Receptor Classes Currently, Pyrite has two separate classes for storing and modifying molecules: the `Ligand` class, which can do atom-typing, transformation, dihedral angle modification, and placement; and the `Receptor` class, which handles atom-typing and has additional `Viewer` options for visualization. This split in functionality allows for separate functionality based on the "role" of the molecule in the docking workflow. However, no such functionality is currently implemented, except in scoring function space.

This distinction complicates docking(-adjacent) problems where there is no such clear distinction in roles, such as in protein-protein docking, docking into macromolecules such as cyclodextrin, or ligand shape-matching. Currently, `Scoring Functions` have to specify each argument to be either a `Receptor` or `Ligand`, disallowing the use of existing scoring function for these problems.

Furthermore, there is currently no functionality for the flexibility of protein side-chains. Consolidating the `Ligand` and `Receptor` class would allow for easy protein flexibility through `Ligands` dihedral angle mechanisms.

At the time of writing, the specific implementation of how `Ligand` and `Receptor` hold their `Mol` object is slightly different; `Ligand` extends the `Mol` class, allowing for the simple use of rdkit [10] methods on ligands, `Receptor` instead holds the `Mol` as an attribute. This change will therefore require quite a bit of refactoring, not only in the `Ligand` and `Receptor` classes, but also in the scoring functions.

This refactor will allow for more diverse problems, simpler interfacing — as the `Ligand` and `Receptor` will be able to use rdkit methods in the same way — and more general scoring function design.

Flexible Proteins As mentioned in the last section, Pyrite does not allow for protein flexibility. In real life, proteins are not rigid targets, and one way to mimic this flexibility is by allowing the side-chains of amino acids near the binding site to move. This can significantly improve docking performance [37].

Consolidating the `Ligand` and `Receptor` classes will put the mechanisms in place for this, but a way to easily integrate this into an optimization workflow and select which side-chains are flexible needs to be developed.

Custom Atom Parameters Pyrite currently uses Vina atom types [12], as mentioned in the main text. When designing custom physics-based scoring functions, or implementing existing functions such as the Vinardo [38] function, it is desired to easily be able to change these atom types. This way, scoring functions are not reliant on one specific atom parameterization.

There should therefore be a straightforward way to select and modify these atom types. We need to investigate whether this would be better to do on a molecule level, or on a scoring function level, as this will influence the software design.

The same should be possible for the charge model.

Structure Loading Robustness As mentioned in the main text, Pyrite has issues loading some protein structures. This is caused by the `PDBLoader` of the rdkit, which infers bonds based on distance. In rare cases, this can result in a physically (virtually-)impossible situation, such as a nitrogen with five bonds or a Texas carbon.[iv]

This problem hinders real world usability and reliability, as certain structures are currently not able to be loaded in Pyrite. A pragmatic solution to this issue is to save the file as a `mol2` file instead of a `pdb` file, but this is cumbersome, especially when using large datasets that only supply `pdb` files. Furthermore, protein preparation tooling is often built around `pdb` files, e.g. *pdb4amber* [28].

The real solution to this problem is updating the `pdb` loading mechanism in Pyrite. The bond-inferring mechanism should either be updated to intelligently infer bonds based on residues, like in OpenMM [27], or a wrongly inferred bond correction mechanism should be added. Ideally, Pyrite is able to load any `pdb` file straight from the Protein DataBank, possibly with help from OpenMM's *pdbfixer*.

Core & Pocket Optimization After any refactoring mentioned above, time and effort should be invested in optimizing the core feature set of Pyrite. In every docking workflow, Pyrite has to load and analyse the structure to determine which bonds are rotatable, assign atom types and charges, and possibly calculate binding Pockets. Furthermore, during each iteration of an optimization algorithm, a ligand has to be updated with a new position, rotation, and dihedral bond vector. In the end, about 30% of the time of the docking run is spent in the core feature set of Pyrite.

Further optimizing these core features would thus have a substantial impact on the runtime of the entire docking workflow.

Furthermore, the Pocket detection mechanism is currently based on multiple breath-first-search based flood-fill algorithms. These algorithms are all implemented in pure Python, and loop based. This makes them slow. Replacing these with C implementations, or accelerating them with numba, would result in a noticeable performance boost when searching for binding pockets.

[iv] Carbon with five bonds, like the five points of the Texas star. *Everything is bigger in Texas.*

Testing and Workflow Optimization

Additional Niching Mechanisms As mentioned in the discussion, the current Crowding function in Pyrite works well as a niching factor, but is far from ideal. Assessing and implementing additional niching methods, such as ring topologies and multimodal optimization, would be the best first step in optimizing Pyrite docking performance, especially as we hypothesize Pyrite is hindered by inadequate searching and niching.

Implementation of a workflow with a ring topology is straightforward, provided that a population based optimizer is used. Pyrite's scoring functions and ligand are thread-safe, and *pygmo* [22] supplies a `archipelago` class, where populations are "living" on islands that have limited communication and cross-breeding with each-other. Each island lives on a separate thread, such that the entire optimization is parallelized.

Using multimodal optimization is slightly more complicated. In *pygmo*, multimodal optimizers require a multimodal problem, meaning that the docking problem needs to be formulated in such a way that two variables are being optimized. Some *pygmo* multimodal optimizers have diversifying mechanisms built in, but still require a multimodal problem. Therefore, the docking problem needs to be adjusted using either a direct output variable describing the diversity of the population, or a "decoy" meta-variable combined with the optimizers niching mechanisms.

Implementation of these mechanisms hopefully allows Pyrite to find the global energy minimum more effectively, even in a very hilly energy landscape.

Other Refinement Optimizers In this paper, only the *L-BFGS-B* algorithm is used for refinement. This algorithm is very effective at sliding down the energy gradient into a minimum, but has little ability to escape local minima. This means that we are relying on our searching algorithm to have the best pose nearly right, as we otherwise have no way of finding it in the refinement stage. The *L-BFGS-B* algorithm can be tuned to take bigger initial steps, and thus escape local minima more effectively, but a better approach would be to use a different algorithm.

Gradient-based algorithms are very effective when optimizing smooth functions, but fall apart when optimizing functions that have more jagged edges. Using other optimization methods, such as during the searching stage, could solve this issue by allowing for more exploration around the minima.

We can experiment by feeding the output of the searching phase into the refinement phase as the population of an optimizer. However, this will steer the entire population towards the lowest energy pose, negatively affecting the diversity. Non-optimization based methods, such as simulated annealing, would perhaps work better, and allow the pose to nestle itself deeper into the local minimum without all poses consolidating to a single minimum.

Another option is to apply population based methods in combination with a niching parameter. The Crowding function is however not applicable here, as it penalized later poses more than earlier poses.

In any case, the algorithm used in the refinement stage should be tuned to exploit more than in the searching phase, where exploration is more important.

Dihedral Angle Quantization During the development of Pyrite, we shortly experimented with quantization of the dihedral angles. This allows optimizers that support integer optimization to drastically reduce the number of possible angles from virtually infinite to < 100. Due to time constraints, however, no extended testing has been executed using this method.

Quantization has the potential to drastically reduce searching time, as there are less possibilities to check. The precise impact of this technique on speed and accuracy should be assessed.

Dynamic Generation Count As mentioned in the main text, simulated annealing dynamically changes the total number of evaluations based on the number of degrees of freedom in is feature vector.

This can lead to more time and resource investment into more complicated ligands. The considerations described in the discussion, regarding torsional penalties, should be taken into account when designing this into the workflow.

Pocket Tuning As described in the method, the binding pocket generation algorithm is highly tunable, all depth filters are customizable. Furthermore, optional weights can be added based on this depth.

Preliminary testing shows that a depth of 14 is the sweet spot, but this should carefully be assessed and validated in further research. Other variables, such as grid-size and sphere size should also be taken into consideration.

It is important to note that the optimal values for these parameters are highly dependent on the specific target type. For example, targets with typical deep pockets, such as CYP450s, will fare better with a higher minimum depth, while targets with shallow pockets require lower minimum depths.

As mentioned in the main text, dynamically determining the optimal depth and distance to protein criteria could solve this issue and result in substantially better results. The impact of this technique should be assessed.

Additional Hyper-parameter Tuning Next to these variables, there are a lot of hyper-parameters that can be tuned for the workflow, not only in the searching and refining algorithms, but also in the scoring functions. For example, the size of the population used in the searching phase can greatly affect the exploration/exploitation ratio of the algorithm, and the number of neighbours considered for pairwise distances in the scoring functions determines the accuracy and speed of the function.

Careful tuning of these hyper-parameters can lead to improvements in accuracy and speed. Several hyper-parameter tuning algorithms exist — one of the few tangible outputs of the AI boom — and can readily adapted for this. However, care needs to be taken to ensure we are not overfitting on the dataset used. The PDBbind demo dataset, with under 300 structures, is orders of magnitude too small for effective use in this context. Furthermore, the computational time investment that tuning more than a couple parameters will take is astronomical in Pyrite's current state. General optimization and, ideally, GPU acceleration is required before this is feasible.

A knowledge-based approach to hyper-parameter tuning is currently more feasible, and can be executed by careful consideration and experimentation.

Documentation

Continuous Integration Pyrite has extensive documentation in the form of doc strings, that can easily be built using the Sphinx documentation system. Furthermore, we would like to publish Pyrite on package registries such as PyPI.

To simplify this process, we need to set-up a continuous integration pipeline within the Pyrite GitHub [15], such that any new software versions can automatically be propagated in documentation and package registries.

Furthermore, documentation should be version controlled, such that documentation for older versions of Pyrite is always accessible. Lastly, new software versions should be uploaded to Zenodo [39], such that they receive a DOI and can be cited.

Interactive Graphs The scoring functions in the Pyrite documentation all contain graphs that show how the output score is related to the pairwise distance between two atoms. Some functions, however, have many parameters. For example, the van der Waals function has 4 tunable parameters, that all drastically alter the shape of the function. Adding interactivity to these graphs, either directly in the documentation or in an external tool such as Desmos [40], will allow users and students to get a better grasp of the effect of these parameters, and allow for more straightforward knowledge- and physics-based tuning of these functions.

Function Usage Examples Even though the Pyrite documentation describes the workings, parameters and output of each of Pyrite's functions comprehensively, no usage examples are included. Adding these can help users get a grasp of how the usage of the function should look in the code.

User Guides The Pyrite documentation currently contains some rudimentary user guides, which shortly describe how Pyrite classes can be used. Expanding on these user guides with more complicated examples can help the comprehension of users.

Furthermore, adding examples specific to the implementation of novel scoring functions and methods using Pyrite will aid users in the development of novel docking methods.

Workflow Examples Next to the example workflow developed in this paper, no Pyrite example workflows exist. To demonstrate the versatility of Pyrite, we can develop several workflows each highlighting a different type of molecular docking (adjacent) problem.

These workflows could include different molecular docking methods. We can, for example, add an example workflow with multiple searching steps, or with only a single phase.

Furthermore, we can create a protein-protein docking workflow, or a workflow where a co-enzyme and ligand are docked simultaneously. This will highlight Pyrites capabilities beyond straightforward ligand-protein docking.

Tutorials Lastly, the educational value of Pyrite was highlighted in the pilot assignment described in main text. Adding assignment like tutorials to the Pyrite documentation or repository allows any user to walk through these assignments, and not only learn how to work with Pyrite, but also get a better understanding of molecular docking and its physical basis.