

Martinize2 and Vermouth: Unified Framework for Topology Generation

Peter C Kroon, Fabian Grunewald , Jonathan Barnoud, Marco van Tilburg, Chris Brasnett, Paulo Cesar Telles de Souza, Tsjerk A Wassenaar, Siewert-Jan J Marrink 

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands • Heidelberg Institute for Theoretical Studies (HITS), Heidelberg, Germany • Interdisciplinary Center for Scientific Computing, Heidelberg University, Heidelberg, Germany • CITIUS Intelligent Technologies Research Centre,, Santiago de Compostela, Spain • Laboratoire de Biologie et Modélisation de la Cellule, CNRS, UMR 5239, Inserm, U1293, Université Claude Bernard Lyon 1, Ecole Normale Supérieure de Lyon, Lyon, France • Centre Blaise Pascal de Simulation et de Modélisation Numérique, Ecole Normale Supérieure de Lyon, Lyon, France

 https://en.wikipedia.org/wiki/Open_access

 Copyright information

eLife Assessment

The authors present an **important** multi-scale computational platform, which aims to automate the workflow for coarse-grained simulations of biomolecules in the framework of the popular MARTINI model. The capability of the platform has been **convincingly** demonstrated by the application to a large number of proteins as well as macrocycles and polymers. This work will be of interest to both computational biophysicists and chemists.

<https://doi.org/10.7554/eLife.90627.3.sa4>

Abstract

Ongoing advances in force field and computer hardware development enable the use of molecular dynamics (MD) to simulate increasingly complex systems with the ultimate goal of reaching cellular complexity. At the same time, rational design by high-throughput (HT) simulations is another forefront of MD. In these areas, the Martini coarse-grained force field, especially the latest version (*i.e.* v3), is being actively explored because it offers an enhanced spatial-temporal resolution. However, the automation tools for preparing simulations with the Martini force field, accompanying the previous version, were not designed for HT simulations or studies of complex cellular systems. Therefore, they become a major limiting factor. To address these shortcomings, we present the open-source *Vermouth* python library. *Vermouth* is designed to become the unified framework for developing programs, which prepare, run, and analyze Martini simulations of complex systems. To demonstrate the power of the *Vermouth* library, the *Martinize2* program is showcased as a generalization of the *martinize* script, originally aimed to set up simulations of proteins. In contrast to the previous version, *Martinize2* automatically handles protonation states in proteins and post-translation modifications, offers more options to fine-tune structural biases such as the elastic network (EN), and can convert non-protein molecules such as ligands. Finally, *Martinize2* is used in two high-complexity benchmarks. The entire I-TASSER protein template database as well as a subset of 200,000 structures from the AlphaFold Protein Structure Database are converted to

CG resolution and we illustrate how the checks on input structure quality can safeguard high-throughput applications.

Introduction

Molecular dynamics (MD) has grown to be a valuable and powerful tool in studying a variety of systems in molecular detail. Advances in force fields and computer hardware have enabled the use of MD in increasingly complex systems, exemplified by recent simulations of, *e.g.* realistic cell membranes^{1,2}, virus particles^{2,3}, and even complete aerosol droplets⁴. However, there is a growing interest in studying systems of even greater complexity, culminating in molecularly detailed simulations of whole organelles^{5,6} and the set goal of simulating entire cells^{7–9}.

Moreover, the growing demand for computer-aided rational design relies on high-throughput simulations with millions of systems simulated in parallel.^{10–12} Currently, the computational demand of MD methods representing all atoms explicitly severely limits the access to the spatial-temporal resolution needed to simulate the aforementioned systems. Coarse-grained (CG) MD methods overcome this challenge by grouping several atoms into one effective interaction site called a bead and thus reduces the number of degrees of freedom that have to be simulated.

Among the most popular CG methods is the Martini force field.^{13,14} Within the scope of the Martini force field about 2-5 non-hydrogen atoms are grouped into one bead. Non-bonded interactions between beads are defined in a discrete interaction table calibrated to reproduce thermodynamic data, whereas bonded interactions are matched to underlying atomistic reference simulations. Molecule parameters created following this approach are transferable between different systems and chemical contexts.^{13,14} This transferability-based approach allows Martini simulations to easily reach the aforementioned complexity scale. However, to really prepare the Martini force field for the high-throughput and whole cell scale simulation era, automated workflows that enable fast and efficient setup of complex systems are of fundamental importance.

The Martini community has a long-standing history of easy-to-use and freely accessible scripts and programs, which helped researchers to set up, run, analyze, and backmap simulations. A non-exhaustive overview can be found in our recent review of the 20-year history of Martini.¹⁵ However, the codes and scripts developed to date share no common framework or backend even though they share many common operations such as resolution transformation or mapping of coordinates. In addition, input files which define molecule parameters or fragments thereof, are not transferable between the tools, with each one of them often defining their own input file formats. We consider that unifying these operations as well as input streams into a single framework will speed up program development and also the robustness of code design to bugs. In addition, it will allow the implementation of modern software techniques such as code review, continuous integration (CI) testing, and version control which generally improve code quality and resilience.¹⁶ It is also simpler to adopt a single framework to new developments such as the recently proposed CGsmiles.¹⁷ line notation. CGsmiles strings can describe molecules at multiple different resolutions and the interconversion between these resolutions. Thus, they offer a more robust way for sharing, storing, and applying resolution transformations compared to previous data formats.

Designing and coding a unified framework to support general Martini software development is a massive undertaking with many facets as the original scripts and programs deal with different stages of MD simulations. To start the development, we focused the design of the framework on topology generation. A topology lies at the heart of each simulation and defines the starting coordinates as well as input parameters for the simulation. For example, to run protein simulations within Martini, a script called *martinize*¹⁸ takes atomistic protein coordinates, maps

them to the CG resolution, and generates the protein molecule definitions from building blocks. This workflow is quite classic and underlies many scripts and programs for topology generation both at the CG and at the all-atom (AA) level.^{18–37} With the latest release of version 3 of Martini, proteins have been thoroughly reparametrized.¹³ The new capabilities of Martini 3 proteins are exemplified by their use of high-throughput drug binding assays^{11,38}, which are an essential step in computer-aided drug design (CADD). Part of the improved protein properties come from the redefined Martini interaction table. However, another part of the improvement is the result of protein-specific methods such as the use of structure-biased dihedrals³⁹ (often referred to as side-chain corrections), specific ENs⁴⁰, or integration of Gō-like models.^{41–43} All these features are additional specific biasing steps applied after the generation of the original topology file for the protein and are not part of the capabilities of the previous *martinize* script. Hence, we chose to co-develop a unified framework for topology generation together with a new *martinize* version, *Martinize2*.

In this paper, we present the VERsatile MODular Universal Transformation Helper (*Vermouth*) library, a general python framework aiding in the design of programs that can create topologies for complex systems at AA, united-atom (UA), and CG resolution. On top of *Vermouth*, we built the *Martinize2* program, as the successor of the *martinize* script.^{18,35} The goal of *Martinize2* is to encompass all functionality required to generate Martini protein parameters (supporting the older versions Martini 2^{18,40,44} as well as the latest Martini 3) and be compatible with high-throughput workflows as needed in CADD approaches based on Martini. In addition, both *Vermouth* and *Martinize2* are designed to have sufficient flexibility and robustness to ready Martini for the era of high-throughput high complexity simulations.

Finally, we note that much of the progress of Martini has resulted from an active community of researchers contributing scripts, programs, and parameters. However, as is the case for most research software in the field they often fail to adhere to the principles of FAIR: findability, accessibility, interoperability, and reusability.^{45–47} The FAIR principles⁴⁶, originally designed to improve data management and reproducibility in science, have recently been extended to research software in a more general sense. This extension is aimed at fostering more sustainable software development in science.⁴⁵ To meet these standards the software tools we present here are distributed under the permissive open-source Apache 2.0 license on GitHub and are developed using contemporary software development practices, such as continuous integration testing. To make adoption as easy as possible, they have few dependencies, are distributed through the Python Package Index, and can be installed using the Python package manager *pip*. Other researchers are encouraged and welcome to contribute parameters and code as outlined in our contribution workflow.

Results

In this section, we first outline the design and API of the *Vermouth* library. Then we discuss how the *Vermouth* library is used to construct a pipeline for generating protein input parameters for the Martini force field. This pipeline constitutes the new *Martinize2* program. Finally, we present some benchmarks and selected test cases to demonstrate the capabilities of *Martinize2* and assess its fitness for generating complex system topologies and high-throughput workflows, surpassing the capabilities of the previous *martinize* script.

In the literature, many scripts and programs have been described that can create topologies for linear molecules and some specific software exists that also handles branched molecules such as carbohydrates²⁵, or dendritic polymers.²⁶ However, to the best of our knowledge, there is at present not a general program that can create topologies from atomistic structures for any type of system, and at any resolution, presenting an extendable and stable API. Based on the existing software, we can, however, define a number of required and desirable features for such a general

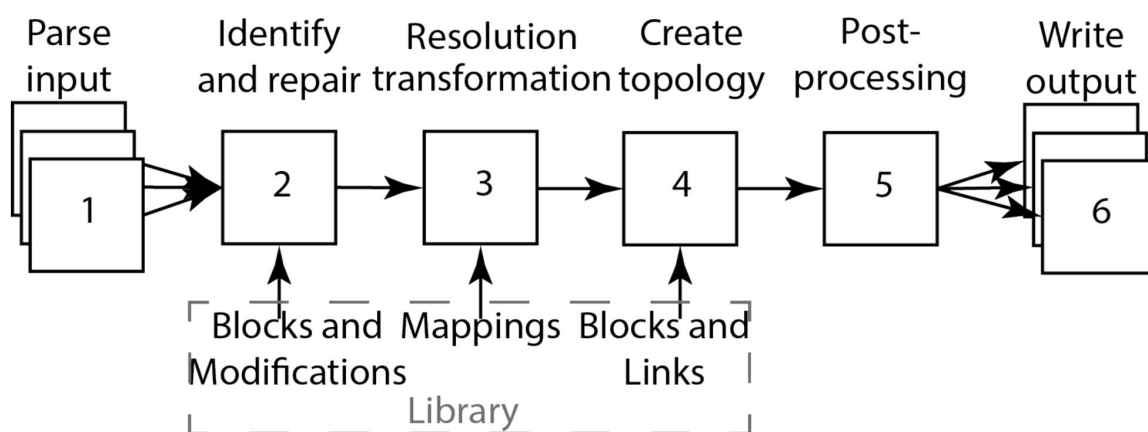


Figure 1

Fundamental stages in topology generation from atomistic structures.

First, the provided input is parsed (step 1). Second, for every parsed residue its atoms are identified and, if needed, atom names are corrected and missing atoms are added (step 2). Third, mappings are taken from the library and a resolution transformation to the required output resolution is performed (step 3). Fourth, intra-residue interactions are added from blocks taken from the library, and inter-residue interactions are added from links taken from the library (step 4). Fifth, optionally, post-processing is performed to add e.g. an EN (step 5). Finally, the produced topology is written to output files (step 6).

program and library to have: 1) it must be force field and resolution agnostic; 2) it must be MD engine agnostic; 3) it must use data files that can be checked, made and modified by users, and 4) it must be able to process any type of molecule or polymer, be it linear, cyclic, branched, or dendrimeric, and mixtures thereof.

To start designing a library that can fulfill the above requirements we note that most workflows used for topology generation can be decomposed into six fundamental stages (**Figure 1** [↗](#)): First, reading input data, usually an atomistic coordinate file (e.g. from the protein data bank); second, identifying the parsed atoms, to find how they correspond to the atoms in the data files describing the building blocks; third, optionally a resolution transformation step; fourth, the generation of the actual topology and assigning particle types and bonded interactions; fifth, any type of post-processing; and finally, sixth, writing the required output files. Even though these stages are generally shared for topology generation pipelines, they also apply to other workflows commonly encountered in Martini programs. Especially, stages 1, 3, 5, and 6 can be found in almost all Martini programs, which generate simulation input files in the broader sense.^{18 [↗](#), 48 [↗](#)–50 [↗](#)} Separating these stages, therefore, helps to define an API with data structures and independent processes, which optimally support such workflows. In addition, the clear distinction in stages helps to externalize any data files, which can be edited by the user or force field developers. *Vermouth* is built on the idea and definition of processors, which are tasks arranged in a pipeline. This design was inspired by the ubiquitous workflow managers available in the field.^{51 [↗](#)} We formalize the idea of processors by introducing an abstract base class the **Processor**. New pipeline stages can be created as subclasses of this base class. All **Processors** operate on the central data structure class **System**, which contains any number of **Molecule** data structures (see **Figure 2** [↗](#)). A **Molecule** is defined as the graph of a molecule or assembly of molecules, which are connected by bonded interactions. The nodes of a **Molecule** usually correspond to atoms or CG beads but can be any form of particle as defined by the force field.

Nodes can have attributes that describe additional information such as a residue name or charge. However, only the atom name, residue name, and residue number are required as attributes. In addition, the edges of the **Molecule** follow the connectivity as defined by bonds, angles, or other bonded interactions. For example, two protein chains connected by a disulfide bridge would be considered a single **Molecule**. In contrast, a cofactor, which is only interacting via non-bonded interactions, would be a separate **Molecule**. Operations on **Molecules** usually add or remove bonded interactions or change node attributes. For convenience, **Processors** can also operate on a collection of molecules, which are defined by the **System** class (see **Figure 2** [↗](#)). A list of all available processors is given in the documentation.

Processors operate on **Molecules**. However, often additional data is required to perform the pipeline as defined by the **Processor**. The additional data can be provided in the form of one of the four other main data structures (**Blocks**, **Links**, **Modifications**, **Mappings**) or arguments of the processors that can be set in a script or via the command line interface. These four other data structures contain all molecular level information required to fully define and/or modify a topology for any type of MD code (e.g., atom types, bonded interactions, and positions) as well as enable transformations between topologies. For example, a **Mapping** consists of two molecular fragments at different resolutions and a correspondence between their particles. In contrast, **Blocks**, **Links**, and **Modifications** are graphs, which describe these molecular fragments, the links between those, and possible changes to fragments respectively. They are all subclasses of a **Molecule** and an extension of the graph class from the networkx library.^{52 [↗](#)} (see **Figure 2** [↗](#)).

As shown in **Figure 2** [↗](#), to make the data structures that are force field specific (**Blocks**, **Links**, **Modifications**) easier to use, *Vermouth* offers a second collection class called a **ForceField**. Every molecule must have a **ForceField** associated with it. Additional information on the data structures is given in the documentation.

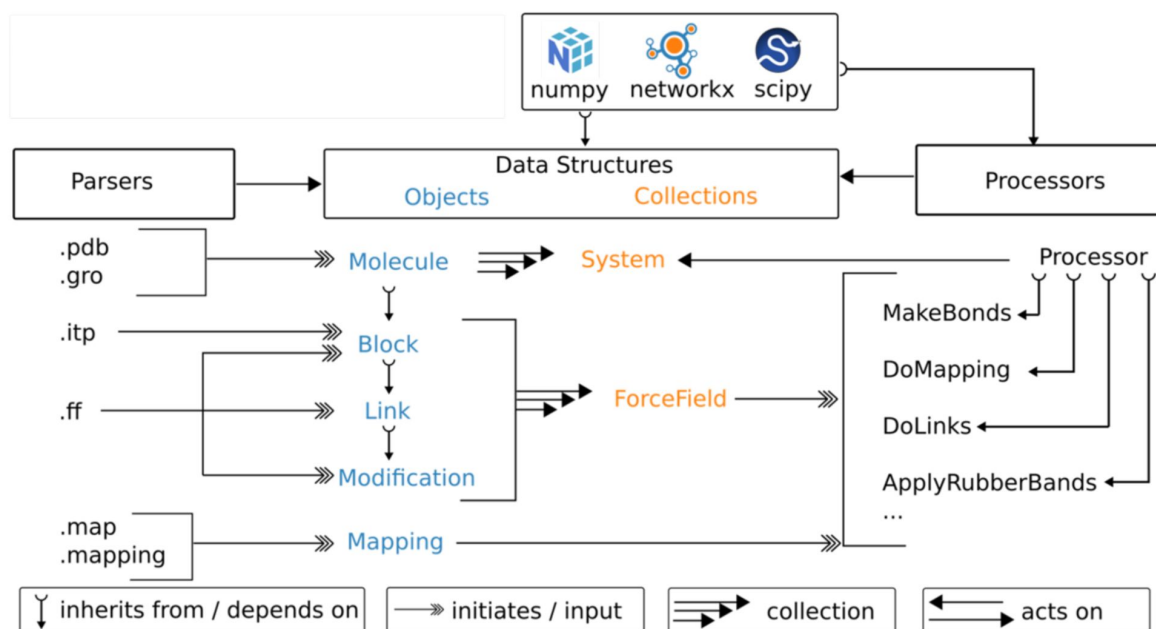


Figure 2

Organization of the Vermouth library.

The vermouth library defines 5 types of data structures (blue) to store molecular information and force field information. For convenience, it also defines two collection classes (orange) composed of several data-structure instances. Data structures are initiated or get input from parsers, which read 6 types of data files (see [Table S1](#) for more details on file types). The central data structure(s) are **Molecule** and **System**. These are changed, updated, or transformed by so-called **Processor** classes, which take force field data as input. Parsers, data structures, and **Processors** only depend on three Python libraries as shown. At the moment vermouth also exposes four types of writers (not shown here) to go along with the parsers (see [Table S2](#)).

Finally, the *Vermouth* library also contains a number of parsers that return instances of the data structures from common input file formats. For example, the in-house ff format defines **Blocks**, **Links**, and **Modifications**, while the backwards style mapping format can be read to return an instance of the **Mapping** class. **Table S1** [in the Supporting Information](#) summarizes all input parsers as well as the format and data structure they return. We note that *Vermouth* is also able to read content associated with the '[molecules]' directive of the GROMACS topology file, which is colloquially referred to as included topology file (itp) file. This allows users to directly manipulate GROMACS molecule files within *Vermouth*. We note that as neither parsing nor the **Molecule** itself depends on GROMACS code, the library can easily be extended to other MD engines.

Martinize2

Martinize2 is a pipeline constructed of *Vermouth* **Processors** with a command line interface (CLI), with the purpose of transforming atomistic structure data to a CG Martini topology including both coordinates and simulation parameters. *Martinize2* is the successor of the *martinize* script, which was used for generating input parameters for Martini 2 proteins, DNA, or RNA. However, different branches had to be used for proteins and DNA (*martinize.py*¹⁸ [in the Supporting Information](#), *martinize-dna.py*³⁵ [in the Supporting Information](#)) or RNA³⁷ [in the Supporting Information](#). In contrast, *Martinize2* is designed to generate topologies for the Martini force field for proteins, DNA, and in principle any other arbitrarily complex molecule.

Martinize2 consists of different **Processors** which fulfill the basic stages of topology generation as shown in **Figure 1** [in the Supporting Information](#). We note that the design of *Martinize2* is general and applies to arbitrarily complex polymers consisting of arbitrary monomeric repeat units (MRUs). However, to increase the readability of the following sections the layout of the program is described in terms of residues in proteins.

The *Martinize2* pipeline starts by reading an atomistic structure, which describes a single molecule (e.g., protein) or assembly of any size. Subsequently, bonds between the atoms are inferred either by distance calculation, atom names within residues, or using CONECT records of the PDB file. All atoms that are connected by bonds form a **Molecule**. Thus, *Martinize2* creates a **System of Molecules** at the atomistic resolution at the end of the input reading stage. In stage 2, *Identify and Repair*, each residue of each molecule is compared against its canonical definition. Canonical definitions are selected by residue name from the library files. This comparison identifies missing or additional atoms on a residue and fixes all atom names (**Figure 3a** [in the Supporting Information](#)). To efficiently do these comparison operations, *Martinize2* relies on a number of algorithms coming from graph theory (e.g. subgraph isomorphism), which reduces the dependence on accurate atom names, since these occasionally differ based on the source of the input structure. Which algorithms are used in the code is described in more detail in the Supporting Information. Once the residues have their canonical atom names, *Martinize2* checks if the missing or additional atoms are described by any of the modification files (**Figure 3b** [in the Supporting Information](#)). Modifications describe changes in residues from their canonical form, e.g. different protonation states, termini, or post-translational modifications (PTMs), and the effect these have on the topology.

After completing the repair stage everything is in place to perform the mapping to CG resolution. The mapping descriptions are read from the mapping input files in the library and tie together residue definitions at the AA and CG level and the correspondence between them (**Figure 3c** [in the Supporting Information](#)). Mapping to CG level in *Martinize2* is done with a multistep subgraph isomorphism procedure, which is general enough to cover edge cases such as when mappings span multiple atomistic residues. A detailed description is provided in the Supporting Information. The mapping Processor provides a **System of Molecules** at the CG level. These molecules already define all bonded interactions within the residues as well as the coordinates of the CG system. To generate the interactions that link the residues, a simple graph matching with library link definitions is done in the create-topology stage (**Figure 3d** [in the Supporting Information](#)). Finally, after that, we end up with the full CG topology, which is ready for post-processing steps. Post-processing summarizes all biases and modifications

that have to be done on the CG molecule and its CG coordinates. For example, an EN is needed to keep the tertiary structure of the protein and is applied in the post processing stage. Finally, *Martinize2* writes out the CG coordinates and the CG topology file that are production-ready.

Custom Protonation States and PTMs

Of the 20 common amino acids, there are four (GLU, ASP, LYS, HIS), which can readily change their protonation state as a function of pH or environment. Whereas commonly those amino acids are still considered to be in their pH 7 protonation state, it is more appropriate to determine their local pKa from for example continuum electrostatics.⁵³ Subsequently the appropriate charge of the amino-acid can be determined from that pKa and set for the simulation. Even though recently more advanced methods became available for dynamically treating protonation states^{54–56} – also at the Martini level^{57,58} – the fixed charge approach is still the most common and for Martini most computationally efficient. However, the previous *martinize* version lacked the functionality to treat protonation states for all amino acids. Only histidine protonation states could be set interactively but only for two of three possible protonation states.

Other protonation states as defined by the atomistic structure coordinates or residue names were ignored without warning. In addition, the interactive setting of protonation states becomes very cumbersome for large protein complexes.

To overcome this problem and make protonation state handling easier and more robust, we utilize a dual strategy in *Martinize2* to identify and correctly set the protonation states (see **Figure 4**). In route a) the user provides atomistic structure coordinates with AA residue names including those of non-default protonation states corresponding to the naming conventions used in CHARMM⁵⁹ or AMBER⁶⁰. Protonation states can be obtained from online servers such as H++⁶¹ or propKa⁶², for example. If the residue names are correctly given, they can be matched against the parameters in the library and the CG residue obtains the correct protonation state. In the alternative route b), the residue name is simply that of the default pH 7 amino acid, however, the structure file contains an additional hydrogen. In the repair and identify step the chemical graph of the amino acid is compared to the building blocks in the library and any unexpected atoms are flagged. For example, in the case of protonated histidine, the additional hydrogen is labeled (see **Figure 4**). Subsequently, *Martinize2* checks if there are any modifications that would match the complete input graph if added to the original block. In the Histidine example, the modification contains the additional hydrogen which together with the original Histidine block make up a protonated Histidine. The modification also changes the mapping such that the correct protonation state is set at the CG level. This route is more appropriate for example when processing crystal structure files, which are not necessarily named according to any force field convention. We have tested this feature on two protein structures taken from the PDB (1MJ5, 3LZT) and processed as described in the Methods section. In 1MJ5 there are six Histidine residues of which one is predicted to be charged at pH 7. The others are neutral. However, they are divided between the ϵ -tautomer (3 residues) and the δ -tautomer (2 residues). Martini 3 parameters are different for the two tautomers in contrast to Martini 2, which is accordingly recognized by *Martinize2*. In addition, for Lysozyme, we have considered residue GLU35 protonated, which would be appropriate at pH 6 or lower. For both examples, the appropriate protonation states and tautomers are generated at the CG level.

The same procedure used for setting protonation states also applies to identifying any other common post-translational modification (PTMs). Using this procedure, lipidation, phosphorylation, amination or acetylation can be taken into account automatically. To demonstrate that *Martinize2* can handle PTMs, we have implemented dummy parameters for testing of Tyrosine phosphorylations in the Martini 2 force field and generated a Martini topology for the EGFR kinase as an example (PDB 2GS2). Residue TYR845 (see **Figure 5**), which is located in the activation loop of the EGFR kinase, is phosphorylated when the kinase is activated.⁵⁶ *Martinize2* was able

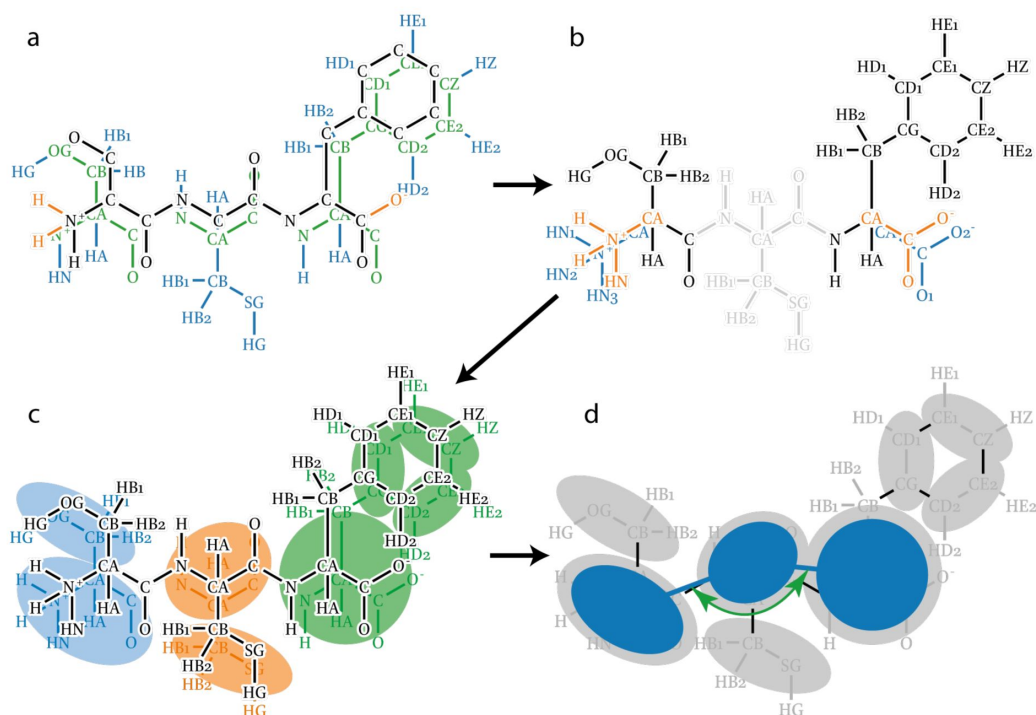


Figure 3

Illustration of atom recognition, mapping, and linking in topology generation.

a) To identify all atoms in the input molecule (black and orange) every residue in the molecule is overlaid with its canonical reference (blue and green). Atoms are recognized when they overlap with atoms in the reference (green). Atoms not present in the molecule are also identified (blue) and will be added later. Finally, atoms in the molecule not described by the canonical references are also labeled (orange) so that they may be identified later. b) Identifying the terminal atoms that are not part of the canonical residues. The modification templates are depicted in blue and the atoms they match in orange. The cysteine does not participate since it does not carry any unexpected atoms, and is depicted in grey for clarity. c) Mappings (blue, orange, and green) describe a molecular fragment at two different resolutions and a correspondence between their particles. The correspondence is depicted approximately here. The mappings are applied to the molecule (black). d) Example of applying a Link. The link depicted (dark blue) adds an angle potential over CG backbone beads.

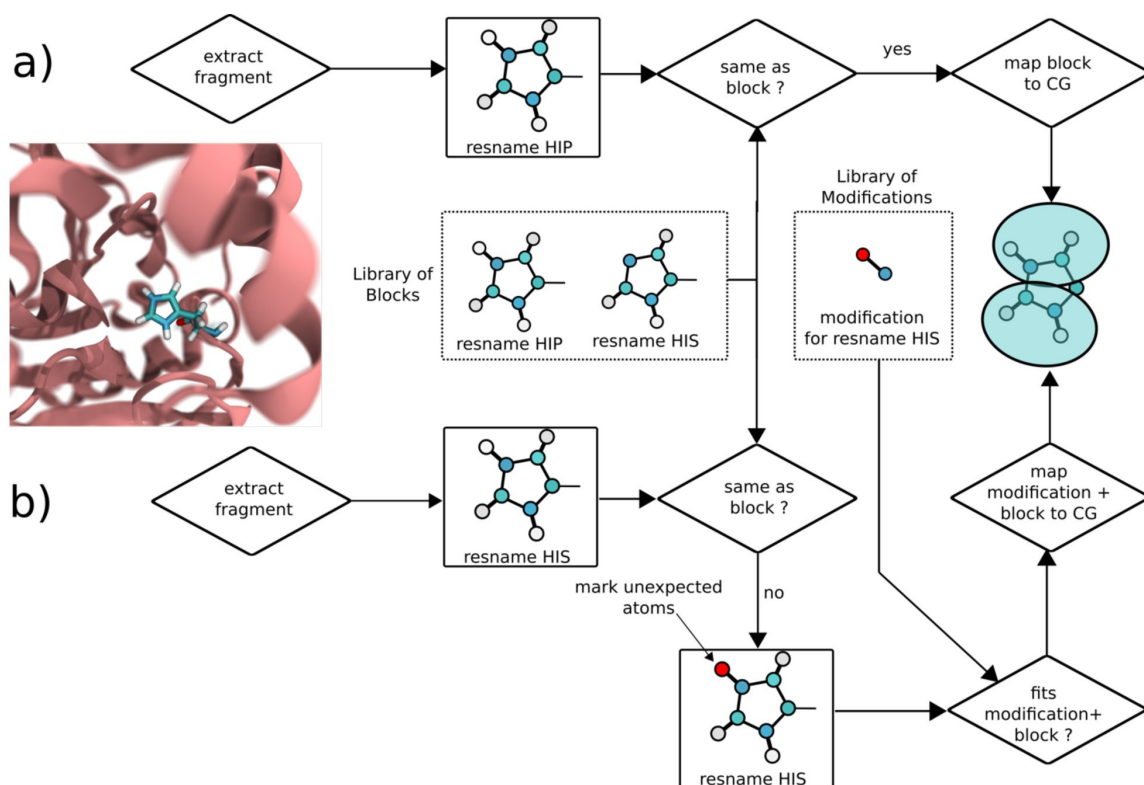


Figure 4

Workflows for identifying protonation states or PTMs exemplified on protonated histidine.

In route a) the residue name of the protonated histidine extracted from the atomistic coordinates matches the residue name in the library and matches the fragment. Hence the protonation state is correctly picked up. In route b) the residue name matches that of neutral Histidine in the library. A mismatch of the fragments is recognized and the extra hydrogen is labelled. Subsequently by matching the extra hydrogen to a modification of the Histidine block the protonated Histidine is recognized as neutral Histidine plus protonation modification and the correct parameters for protonated Histidine are generated.

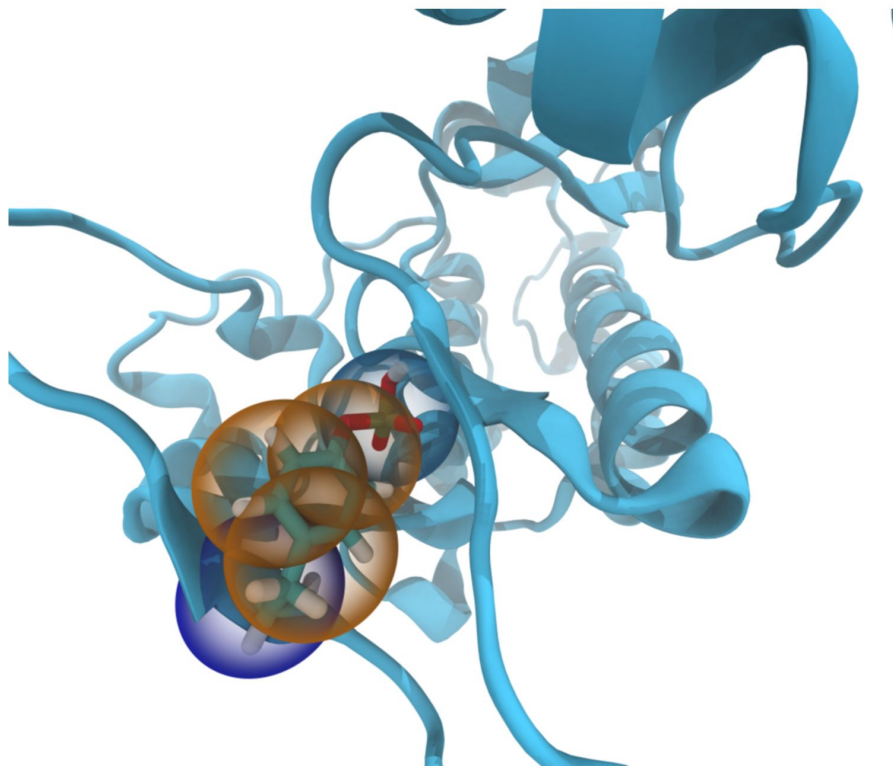


Figure 5

Example of automated identification of PTMs.

CG Martini model of phosphorylated Tyrosine found in the EGFR kinase activation loop. The mapped structure of the phosphorylated residue is shown as beads overlying the atomistic structure.

to convert the structure in one go to Martini 2 resolution. We note that at the time of writing the Martini 3 force field is lacking parameters for these PTMs and they are therefore not implemented in *Martinize2* yet. In this case, a warning is issued by the program.

Expanding the Options of Elastic Network Fine-Tuning

Due to the limitations in most CG protein models (e.g. lack of explicit hydrogen bonding directionality), the tertiary structure has to be enforced with a structural bias called EN.⁶³ An EN for Martini proteins consists of weak harmonic bonds between backbone beads of residues (within a chosen cut-off distance) and is generated after the resolution transformation as a postprocessing step.^{40,44} *Martimize* offered only two types of EN options, the regular model and the Elnedyn⁴⁰ approach, both of which are also implemented in *Martimize2*. However, as the EN fixes the tertiary structure, changes in the structure upon, e.g., ligand binding are not captured. To improve protein models in this aspect recently Gō-like models have been applied to Martini.⁴¹ In a Gō-like model the harmonic bonds are substituted by custom Lennard-Jones interactions that can dissociate, thereby allowing for some tertiary structure changes. Within the scope of Martini, a workflow is available to replace the EN with a Gō model that is generated from a provided contact map.

Even though Gō models offer better flexibility, they are currently limited to single monomeric protein units and require some fine-tuning to get the optimal performance.⁴¹ Especially for high-throughput workflows the EN approach is therefore the preferred option. To further improve upon the ENs generated by the old *martinize*, *Martimize2* offers several options to fine-tune the EN and get better behavior within the constraints of the EN approach. Besides the cut-off and force-constant, *Martimize2* now implements a residue minimum distance (RMD). The RMD is defined as a graph distance and dictates how far residues need to be apart in order to participate in elastic bonds. Defining the RMD as a graph distance means that no bonds are generated between residues that are for example bound by a disulfide bridge. It thus presents a more rigorous implementation than in the previous version. Usually, the residue minimum distance is 3 in order to avoid the EN competing with the bonds, angles, and dihedrals between the backbone beads.

We note that this is part of the Martini protein model and should not be changed. Additionally, *Martimize2* allows to select which beads to generate the EN between. This option is needed for Martini 2 DNA³⁵, for example. Martini 2 DNA offers a stiff EN version, where also sidechain beads are included. Furthermore, *Martimize2* allows to define where in the protein to apply the EN. This is done with the EN unit option. The EN unit can be a molecule, chain, all, or ranges of residue indices. The most trivial option is *all* in which case an EN is applied between all protein molecules in the system. The option molecule and chain yield the same network, if distinct molecules are also distinct chains. However, when two chains are connected by a disulfide bridge, for example, they would be one **Molecule** in the *Martimize2* sense. On the other hand, if the interface is not very well defined or more flexible, biasing the two chains separately could improve the EN. In that case, the *chain* option can be used. This use-case is shown for the human insulin dimer in **Figure 6a** and **Figure 6b**. The human insulin dimer consists of two chains, which are connected by two disulfide bridges. If the molecule or all option is used an EN is generated within the chains and between the chains (**Figure 6b**). However, to avoid generating the EN between the two chains the chain option can be supplied in which case the EN is only generated within chains. As the zoom-in on the tail part shows there are no more bonds between the two chains in **Figure 6b** whereas there are in **Figure 6a**.

Furthermore, *Martimize2* allows the definition of regions of residue IDs where an EN should be generated. This feature gives maximum flexibility and allows to bias structural regions of proteins whereas an EN in intrinsically disordered regions (IDRs) can be avoided. For example, **Figure 6c** and **Figure 6d** show the FtsZ protein of E-coli as predicted by alpha-fold.^{64,65} FtsZ possesses

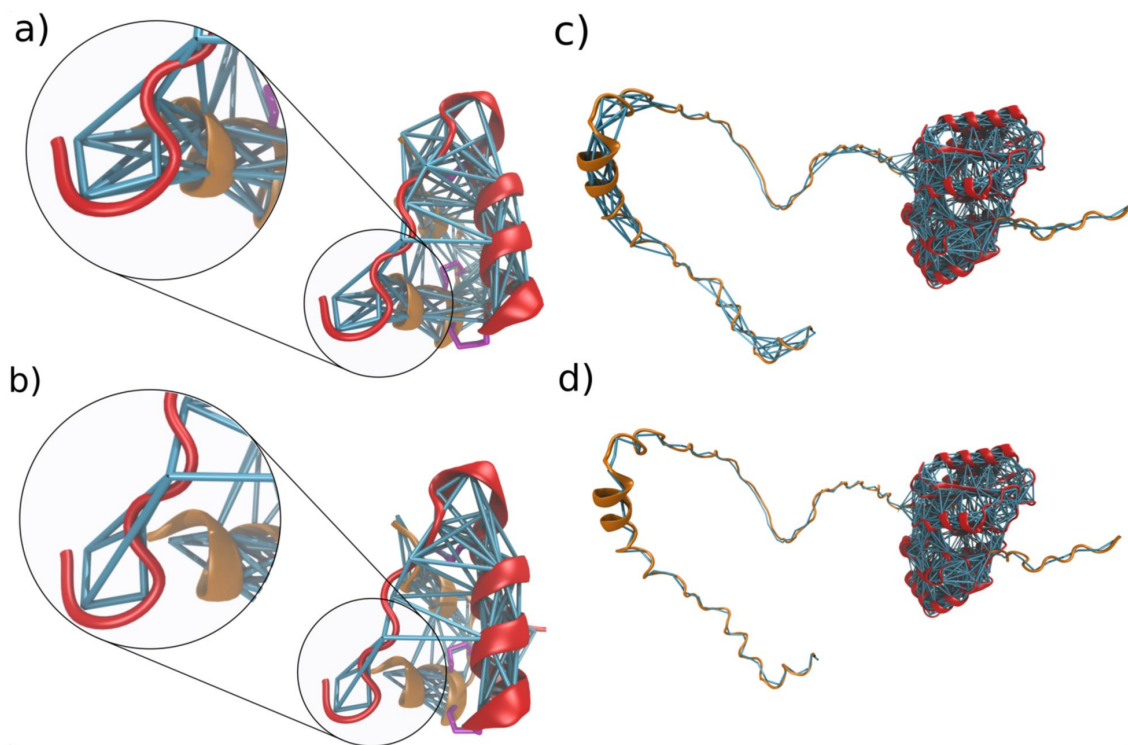


Figure 6

Fine-tuning options for the elastic network.

a) ENs and backbone bonds within the human insulin dimer when generated with the molecule or all-option. The dimer consists of two chains colored in red and orange, which are connected by two disulfide bridges shown in purple. EN bonds are generated between the two chains and within the chains. b) EN and backbone bonds within the insulin dimer when generated with the chain option. In this case, no elastic bonds are generated between the two chains. They are only connected by the disulfide bridge and non-bonded interactions. c) EN within the Ftsz protein, when generated for both the intrinsically disordered tail domains (orange) and structural domain (red) d) EN within the Ftsz protein when the EN is only generated within the structural domain by defining the EN unit as going from resid 12 to 320.

a structural unit and two disordered tail domains. With the region option, *Martinize2* allows the generation of an EN only for the structural domain. Within the old *martinize* superfluous bonds needed to be removed manually.

Finally, we note that *Martinize2* is now implemented in the Martini Data Base (MAD), which offers a further utility to remove certain elastic bonds selectively.⁶⁶ We note that ENs networks can only be applied within protein molecules at the moment.

Beyond proteins; incorporating other molecules in Martinize2

Legacy *martinize* is only applicable to one category of molecule (*i.e.* proteins or DNA), which is one of its biggest drawbacks even for setting up simple protein simulations. *Martinize2* allows the inclusion of new classes of molecules without adjusting the codebase. For instance, proteins frequently have other molecular units associated such as ligands, cofactors, metal ions, or lipids. The general workflow of *Martinize2* allows us to convert these systems in one go provided that the library files are present. Having a single step for topology generation greatly facilitates high-throughput workflows such as protein-ligand binding, one of the cornerstones of CADD.

We test this on two protein complexes. The first test case concerns Flavin Reductase (see **Figure 7a**), which consists of two chains that have flavin mononucleotide ligands (FMN) and one NAD cofactor bound (2BKJ). Martini 2 parameters and mappings from the GROMOS force field were previously published.⁶⁷ Parameters and mappings have been added to the Vermouth database. Subsequently, the system could be converted in one step. During a short simulation, the cofactors remain well bound, indicating that no inappropriate parameters or faulty geometries were generated. Next, we created topologies and starting structures for Lysozyme with a benzene molecule bound (1L84), using the Martini 3 force field (**Figure 7b**). The protein and ligand were again converted in one step and then simulated for a short period. As previously the ligand stays bound, showing that the protocol generates reasonable starting structures and correct parameters.

To fully leverage this new feature, ligand data files are required to be present. Thus, we implemented mappings and parameters from a previously published small molecule database for the Martini 3 force field.⁶⁸ The set comprises 43 small molecules, which are often part of drugs or drug precursors. All small molecules have corresponding parameters in the CHARMM ligand database, which allows users to directly convert atomistic CHARMM simulations to Martini. Mapping directly from crystal structures as present for example in the PDB or other force fields is also possible. In these cases, the residue names may have to be adjusted to be the same as in the CHARMM naming convention. However, *Martinize2* is also able to handle topologically more complex molecules. For example, crown ether (**Figure 7c**) consists of six polyethylene glycol (PEO) repeat units and is cyclic. To test whether *Martinize2* can handle cyclic molecules of multiple repeat units it was converted to Martini 2 resolution applying the latest PEO parameters.⁶⁹ The second example is branched Polyethylene, where we chose a sequence that begins with two linear units followed by three branched ones and two linear units after. Also, this molecule is converted to Martini 2 resolution⁷⁰ by *Martinize2*.

Finally, we have set up instructions on how researchers can submit parameters to the database allowing it to grow and support other researchers. In addition, *Martinize2* facilitates dynamic linking of citations to parameters. With this mechanism, citations are printed at the end of the run that dynamically includes citations to all parameters used in the final topology. Such a system also allows researchers to easily receive credit for contributed parameters.

To assess the robustness of *Martinize2* in a high-throughput use case, we processed the template library used by the I-TASSER⁷¹ protein prediction software (**Figure 8**). At the time of download (26 March 2021), the dataset contained 87084 protein structures. We processed each of

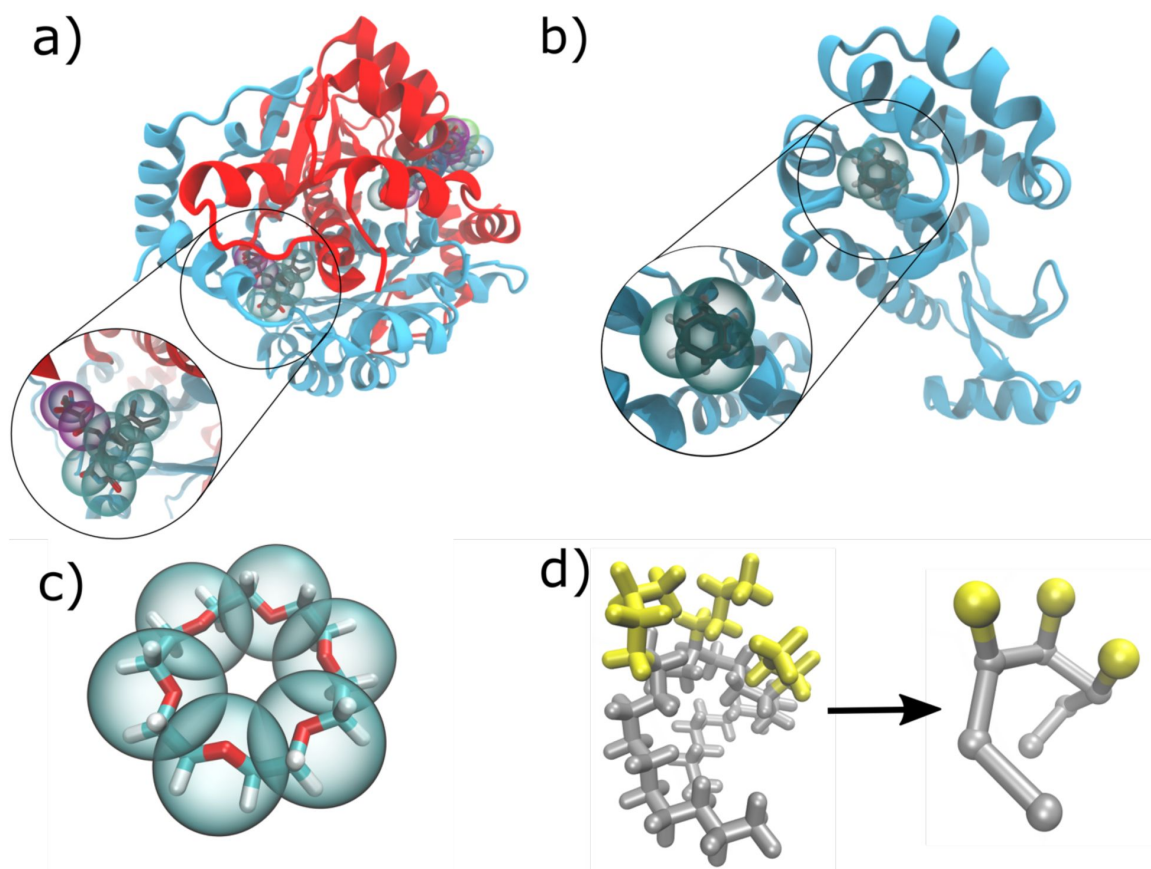


Figure 7

Ligands, cofactors, and polymers transformed to CG Martini level.

a) Flavin Reductase with two FMNs and one NDP cofactor bound in the reference AA state and mapped to Martini CG as indicated by the spheres. The inset shows a zoom onto FMN; b) Lysozyme with benzene ligand bound in the reference AA structure and mapped to Martini CG resolution; c) Crown ether with Martini beads shown on top of the AA structure; d) Branched polyethylene at AA resolution (left) and Martini resolution (right) with the linear chain part shown in gray and the branches in yellow.

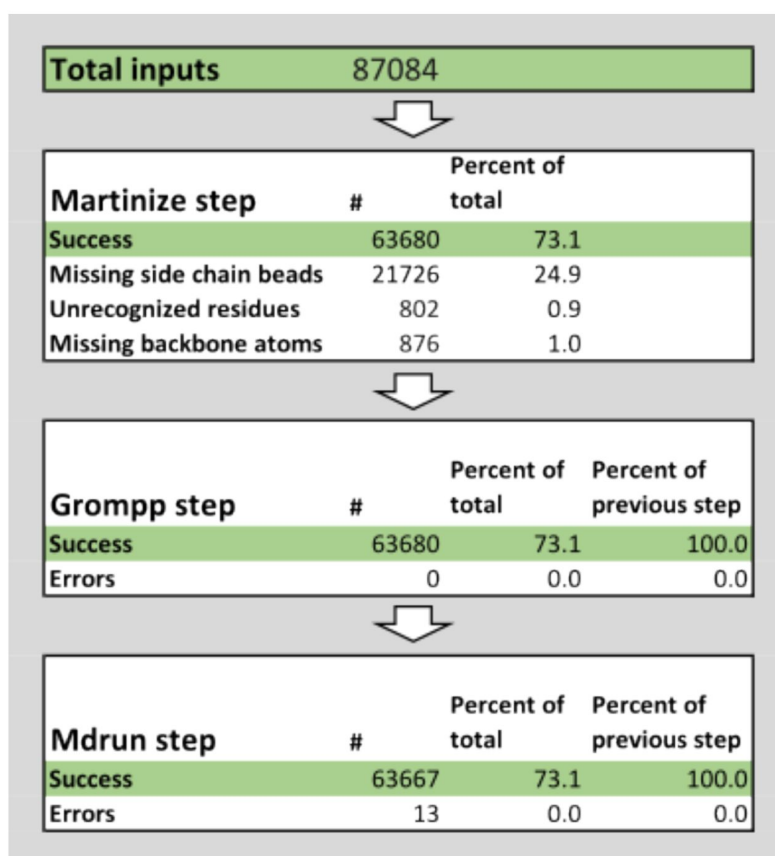


Figure 8

Summary of the successes and failures of the high-throughput pipeline.

We ran the pipeline on the 87084 structures from the template library used by the I-TASSER⁷¹ protein prediction software of which 73% could be converted with Martinize2. The other 26.4% failed mostly due to missing coordinates, and unrecognized residues. For 100% of the converted structures, a GROMACS run input file (i.e. tpr-file) could be generated, and on all but 13 of the converted structures, an energy minimization could be performed.

these structures with *Martinize2* to get Martini 2.2 models with ENs. We then energy minimized the CG protein in a vacuum to validate that the generated structures and topology could be processed by GROMACS 2022.3.

Of the 87084 structures in the dataset, 63680 (73%) could be processed through the whole workflow without error. The main cause of failure (25% of the structures) was missing coordinates in the input structures. When all the atoms that compose a bead are missing from the input, *Martinize2* can generate a topology but it cannot generate coordinates for the bead. Note that if only some atoms are missing, then Vermouth does estimate the position of the bead. 876 structures (1%) had missing coordinates in the backbone that prevented the use of DSSP⁷²[72](#),⁷³[73](#). Finally, 802 input structures (1%) had at least one residue that was inconsistent with the library. Upon further inspection, most of these structures contain malformed glycine residues with an unexpected C β atom. *Martinize2* detected these inconsistencies and emitted a warning for each of them; warnings can be explicitly and selectively ignored, if they are not no output is written to avoid subsequent workflow steps working with corrupted files.

All the 63680 input structures that were successfully processed by *Martinize2* could be processed by the GROMACS pre-processor (*grompp*). However, 13 structures failed the energy minimization. A visual inspection of some of these failing inputs shows the input atomistic structures can be problematic. Erroneous interatomic distances (steric clashes or extended bonds) lead to high energies in the CG systems, which causes a failure in the energy minimization routine. Likely these starting structures are also not numerically stable in a CG simulation.

As a second test case to assess the robustness of *Martinize2*, we processed a subset of the AlphaFold Protein Structure Database.⁶⁴[64](#),⁶⁵[65](#) 200,000 randomly chosen unique protein structures (see Supporting Information) were given to *Martinize2* and subsequently an energy minimization was performed, if the structure could successfully be converted to CG representation. Of the 200,000 structures in the dataset, 7 structures (see Supporting Information) raised an error during the conversion step. Upon further (visual) inspection of the problematic structures, we concluded that all errors were caused by inaccurate initial atomistic coordinates. These inaccurate atomic positions caused bonds to not be identified or additional superfluous bonds to be detected (**Figure 9**[9](#)). In these cases, the unrecognizable residues were detected and caused *Martinize2* to emit a warning. The remaining 199,993 successfully converted structures could be processed by the GROMACS pre-processor (*grompp*) and it was possible to perform an energy minimization.

Finally, as a proof of concept we tested if the topology generation workflow underlying *Martinize2* is also applicable to generating topologies at the AA level. We selected 100 structures from the AlphaFold database and generated topologies for the CHARMM36 forcefield. To facilitate the process a new reader for the .rtp file format, which is the GROMACS protein topology file format, was implemented. In addition, we manually added modifications files to pick up end-terminal modifications, protonation states, and histidine tautomers.

Martinize2 was able to successfully generate topologies for all proteins. Their accuracy was verified by comparing single point energies against topology files generated using *pdb2gmx*. All data and commands for this test case are available from the GitHub examples repository.

Discussion

In the previous section, we have presented the Vermouth python library for facilitating topology generation and manipulation. For researchers to use Vermouth as a framework for software development it presents a clear API separated into data structures, parsers, and processors. In addition, the library relies on only three permissibly licensed open-software projects namely

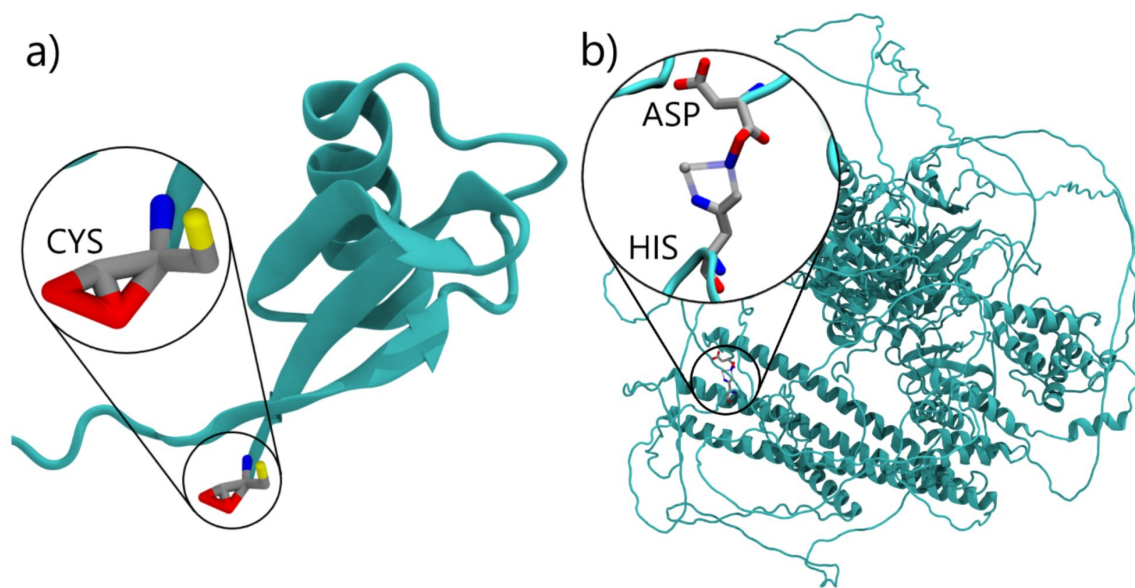


Figure 9

Two examples of problematic atomistic protein structures flagged by Martinize2.

a) the cysteine residue with too small O-O and O-C distances leads to superfluous bonds being recognized. b) the incorrect interatomic distances in the histidine ring led to missing bonds (transparent), an erroneous O-N bond connecting the histidine to a neighboring asparagine. Additionally, a nitrogen atom is switched for an oxygen atom in asparagine.

numpy⁷⁴, scipy⁷⁵, and networkx⁵². This allows researchers more freedom in licensing their code and reduces the potential for bugs introduced by dependency changes. Furthermore, the central data structure represents molecules as graphs. Representing molecules as graphs allows to leverage algorithms from graph theory. Using graph theory for many of the workflows underlying the **Processors** makes them faster and more robust towards edge cases. Even though applying graph theory to molecules is not a new idea^{76–78}, Vermouth is specifically designed to also handle CG level molecule transformation focusing on the Martini force field. Therefore, Vermouth presents additional functionality often lacking from other packages. For example, the handling of virtual-sites, which are ubiquitous in many Martini 3 molecules, is rigorously handled in all **Processors**. As another more general example, the **Processor** applying interactions between residues can automatically compute structural biases from the mapped coordinates. Finally, the *Vermouth* library adheres to the FAIR principles^{43,45,46} to allow adoption by non-experts and ensure quality control. In particular, for both the Vermouth library and *Martinize2*, continuous integration testing is implemented and code review is required. The software is also semantically versioned, and it is distributed through established channels, most notably the Python Package Index, and hosted openly on GitHub.

We have shown how Vermouth was used to shape the *Martinize2* program. However, *Martinize2* is not the only program leveraging the power of the Vermouth library. The *polyply* python suite is another library and collection of command line programs built upon *Vermouth*. *Polyply* enables users to generate both AA and CG simulation input data, i.e. structures and topologies, from sequence information. As such, it allows building system coordinates for arbitrarily complex macromolecular systems and nanomaterials.⁴⁸ Furthermore, the *martini-sour* package⁵⁷, which is currently under development, utilizes Vermouth to convert topology files from regular Martini to titratable Martini simulations. *MartiniGlass* uses *Vermouth* to prepare CG topologies for visualisation, to further aid evaluation and validation of simulation input topologies.⁷⁹ These examples already illustrate that Vermouth has the potential to indeed become the central framework for Martini software development and possibly for other scientific software developments.

Martinize2 enables researchers to prepare simulation input files for arbitrary (bio)polymers, starting from an atomistic structure. We have shown in-depth examples focusing on protein-specific applications, given that they are the most important target for *Martinize2*. However, more complex molecules such as cyclic crown ether and branched PE were showcased to demonstrate the capabilities of *Martinize2*. Furthermore, the user has complete control over the data files used. The abstraction of force field data into Blocks, Modifications, and Links allows researchers to reason about model intricacies in a structured manner. This helps the development of optimized models and parameters for complex (polymeric) molecules, as well as clearly defining in which combinations these are validated. The new program uses algorithms from graph theory to identify atoms and assign the appropriate interactions. This makes the program more tolerant towards its input so that the users have to worry less over details such as atom names, or ensuring that all residues are in order and appropriately numbered. In particular, *Martinize2* is capable of detecting and using protonation states, PTMs, and capping groups automatically. In addition, *Martinize2* allows the fine-tuning of the EN and—as it is not limited to proteins—can also generate parameters for ligands, cofactors, or lipids.

In practice, there are decisions a user needs to make when using Vermouth and *Martinize2*, especially for high-throughput pipelines. *Martinize2* detects but does not reconstruct atoms that are missing from the input structures; these missing atoms can have adverse effects on the result. In the most harmless cases, they only shift the position of a particle in the output structure. When all the atoms for a particle are missing, then the program cannot compute a position for that particle leading to an incomplete output where a particle does not have coordinates. Also, some workflows depend on DSSP^{72,73} to assign secondary structures and some specific missing atoms can prevent DSSP from working properly. In those cases, *Martinize2* issues a warning

whenever it cannot automatically take care of pitfalls. Handling of these cases is a central difference between the new and old version. The old version either terminates with an undefined error or, probably worse, runs and gives output that does not correspond to the atomistic structure given as input. To illustrate the robustness of *Martinize2* towards problematic input, we applied the program to the complete I-TASSER database (~87k structures) as well as a subset of the AlphaFold Protein Structure Database (~200k structures). For the two benchmark cases, *Martinize2* was able to issue a warning or error for all structures which contained seriously malformed residues. Of the first database only 13 structures failed in the energy minimization due to problematic starting coordinates but not obviously malformed residues. In the second benchmark set only 7 seriously malformed residues were identified, and all other structures were successfully energy minimized. Thus, we consider *Martinize2* more robust and fit for high-throughput and high-complexity tasks. In addition, *Martinize2* was able to successfully generate CHARMM36 topologies for proteins selected from the AlphaFold database. This proof-of-concept shows that the workflow underlying *Martinize2* can handle force-fields other than Martini.

Ultimately, the robustness comes at a price. *Martinize2* uses subgraph isomorphism to identify atoms based on their connectivity, and then issues a warning or repair the input. However, subgraph isomorphism is an NP-complete problem⁸⁰. As a result, *Martinize2* is significantly slower than *martinize*. Nevertheless, considering the flexibility the new program offers, in addition to the fact that it is still fast enough to process all entries in the I-TASSER data bank⁷¹, this is deemed to be acceptable. Even though *Martinize2* will most likely never be as fast as *martinize* we note that many of the processes can still be optimized to yield further performance increases. Currently, *Martinize2* is about one order of magnitude slower than its predecessor (Figure S2). However, even for large proteins it takes less than 2 minutes to generate the input file, which is still much faster than any MD simulation even at the CG level. Aside from the performance limitations, *Vermouth* and *Martinize2* present some other limitations as well. Both are currently only capable of writing topologies in GROMACS format. However, our library does not use the MD parameters of the produced topologies or call GROMACS functions, so support for other MD engines can be added in the future. In addition, since *Vermouth* defines an API, it could even be integrated with existing software such as OpenMM.⁸¹ Furthermore, the processor pipeline underlying *Martinize2* is currently hardcoded. Future improvements will focus on making the workflow defined by *Martinize2* more flexible, in order to include the processor pipeline as part of the force field definitions. This would enable the use of different pipelines for different force fields, allowing for easier force field-specific post-processing. In addition, implementing CGsmiles¹⁷ as unified representation of mappings will help offer a broader scope of molecules and make the pipeline more robust with respect to naming conventions in atomistic force fields.

Methods

Preparation of protein input files

Crystal structures were obtained from the RCSB for the following proteins (3LZT; 2GS2; 2BKJ; 1L84; 3I40; 3IGM, 1MJ5) or the Alpha Fold Data Bank⁶⁴ for FtsZ with the ID A0A7Y6D765. Hydrogens and missing heavy atoms were reconstructed using the PRAS package, if appropriate.⁸² For 3LST and 1MJ5, the pKa and half-way titration point were estimated using the propka package.⁶² For 3LST the GLU35 was protonated using the CHARMM-GUI solution builder.^{36,83} The HIS-tag of 1MJ5 was removed.

All-atom simulations

For 2GS2 and 1L84 CHARMM parameters⁵⁹ were created using the CHARMM-GUI solution builder^{36,83} and a small equilibration simulation (20ns) was run before the structures were converted with *Martinize2*. The AA simulation used the recommended non-bonded force settings as for CHARMM with GROMACS⁸⁴. The temperature was maintained using the v-rescale thermostat by Bussi *et al.*⁸⁵ at 310K and pressure was maintained at 1 bar using the Parrinello-Rahman⁸⁶ barostat ($\tau = 12$ ps) after initial equilibration with the Berendsen⁸⁷ barostat. For the CHARMM36 test case, we subsampled 100 structures from the AlphaFold database and generated CHARMM36 reference itp files using *pdb2gmx*. Subsequently, *Martinize2* was used to generate the same itp file from the coordinates output by *pdb2gmx* to ensure the coordinates are exactly the same. Both itp files were compared by computing a single point energy on the common structure.

Coarse-grained simulations

All CG MD simulations were run using GROMACS 2021.5⁸⁸ and the recommended mdp parameters for Martini 2⁸⁹ and Martini 3¹³ respectively. In particular, the Lennard-Jones interactions were cut-off at 1.1 nm and electrostatics were treated with reaction-field (cut-off 1.1 nm, dielectric constant 15). The time-step was 20 fs in all cases and the production trajectories were run with the standard leap-frog integrator. Temperature was maintained using the v-rescale thermostat by Bussi *et al.*⁸⁵ at 310K with ($\tau = 6$ ps) and separate coupling groups for solvent and proteins. The pressure was maintained at 1 bar using the Berendsen barostat for equilibrations ($\tau = 6$ ps). The initial systems were solvated using the *polyply*⁴⁸ package or *gmx solvate* utility.

Complexity benchmark

The (Swiss-Prot) subset of the AlphaFold protein structure database used for the complexity benchmark contained 542,378 pdb structure files at the time of download (22 December 2022). The testing pipeline we used was written in Python and randomly picked 200,000 structures which were given to *Martinize2*. Possible errors during conversion or the subsequent *grompp* and energy minimization steps were captured.

Supplementary Information

1 – Input Parsers & Output Writers

Extension	Data Class	Parser Name	Input Format
.ff	Links Block Modifications	read_ff	in house force-field format
.itp	Block	read_itp	GROMACS topology file; all [molecule] directive content
.map	Mapping	read_mapping	mapping file as defined using backwards style
.pdb	System	read_pdb	canonical PDB format
.gro	Molecule	read_gro	Gromacs .gro file

Table S1

Data Parsers object returned as well as format definition and extension

Table S2

Data Writers and the object returned as well as format definition and extension

Input Format	Data Class	Parser Name	Output Format
.gro	System	write_gro	G96 gro file
.pdb	System	write_pdb	PDB file
.top	System	write_top	Pseudo topology file
.itp	System	write_itp	GROMACS topology file; all [molecule] directive content

2 – Related Tools

Name	Force field	Type of system	External data files	Notes
pdb2gmx ^{19,20}	Any AA/UA	Linear polymers	Yes	
LEaP ²¹	Any AA/UA	Linear polymers	Yes	
CHARMM ¹⁰³	Any AA/UA	Linear polymers	Yes	
psfgen ²³	Any AA/UA	Linear polymers	Yes	
Martinize 1 ^{18,35}	Martini	Proteins, DNA	No	
Sirah Tools ²⁴	Any CG	Linear polymers	Yes	Performs mapping only
DoGlycans ²⁵	AMBER, OPLS	Sugars	Yes	Builds coordinates

HOOBAS ²⁶	Multiple	Multiple	Yes	No user interface, builds coordinates
CHARMM-GUI ^{27-28, 36}	Multiple	Multiple	No	Web server, builds coordinates
VerMoUTH/Martinize2	Multiple	Multiple	Yes	This work
ATB ^{29,30}	GROMOS54a7 GROMOS54a8	Small molecules	N/A	Automatic de novo parametrization
LigParGen ³¹⁻³³	OPLS-AA	Small molecules	N/A	Automatic de novo parametrization
CGenFF ³⁴	CHARMM General Force Field	Small molecules	N/A	Automatic de novo parametrization

Table S3

Limited overview of selected competing tools capable of generating MD topologies. “Force Field” lists the force fields for which this tool can generate topologies without changing the source code. “Type of system” describes the type of system this tool can generate topologies for. “External data files” means whether the force field parameters used are included in separate data files, making it possible to easily change them. “Notes” lists additional remarks and comments, “builds coordinates” means it is capable of constructing coordinates for complete systems, rather than only for e.g. missing sidechains.

3 – Martinize2 Pipeline

In this section, we describe the pipeline underlying the *martinize2* program in more depth highlighting the algorithms used.

Step 1 Parse input. Reading different input file formats is trivial, and all that is needed is to select the correct parser based on the file name provided. At the time of writing parsers are available for **pdb**, **cif**, and **gro** files (coordinate files in Gromacs format). The input is commonly a list of atoms with associated properties such as atom names, coordinates, and monomeric repeat unit (MRU) names. Sometimes the input also provides information about bonds in the system, such as PDB ‘CONNECT’ records. These will be used if available. Otherwise, bonds will be added between the atoms based on simple geometric criteria. At the very least we require MRU names and numbers, elements, and either coordinates or bonds. In the end, the input has been parsed and transformed into an undirected graph with atoms as nodes and bonds as edges.

Step 2 Identify and Repair. To identify the parsed atoms the current generation of tools takes the combination of atom name and MRU name as leading, even though this is the most variable between models. For instance, the atom names assigned in the experimental data often do not match the atom names expected by the force field causing existing tools to either throw an error, or even produce incorrect output. We identify atoms based on their MRU names, connectivity, and their elements by overlaying the MRU with its canonical form (**Figure 3** [↗](#) main paper).

Doing so allows us to identify deviations from the canonical structure such as PTMs, different protonation states and capping groups. In addition, this method reveals which atoms are missing in the input data, allowing us to reconstruct them. We rely on graph theory to perform the overlaying of input and reference structures (see the dedicated section on graph algorithms below).

In order to do this, every MRU in the input molecule is overlaid with its canonical reference structure with the constraint that the elements of corresponding atoms must be the same. To get the relevant canonical structure it is assumed the MRU names in the input molecule are correct and that for each MRU a corresponding block can be found in the library. If the corresponding block cannot be found an error is raised and execution is terminated. Since the library files are designed to be human readable and writable, users can add any data to the library they need.

In the best case finding the overlay is an induced subgraph isomorphism problem where $M_I \sqsubseteq R_I$ with M_I an MRU of the input molecule and R_I the corresponding canonical form. However, this is treated as a largest common induced subgraph problem (see below) since M_I can contain “unexpected” atoms not described by R_I such as PTMs or capping groups. If there are multiple solutions, the solution where most atom names correspond is taken. Either way, a correspondence between the input molecule and its canonical form is obtained. This correspondence is used to a) identify and add missing atoms, b) correct the atom names for the atoms that are there, and c) find which atoms are not described by the canonical MRUs. It should be noted that in this paradigm PTMs, non-standard protonation states, termini, and capping groups are all considered unexpected atoms and treated the same way.

Next, we try to identify all these unexpected atoms by overlaying them with modification template graphs from the library (**Figure 3b** [↗](#) main paper). This is a graph covering problem where we aim to find a minimal combination of templates that covers all unexpected atoms (see below). This does mean that unless there is clear additional metadata there can be no missing atoms in the found modifications since it is not known what they should look like beforehand. The found correspondences are then used to correct the atom names. The MRUs these atoms are part of are labeled so that the correct mappings and interactions can be applied later on. In the end, the input

molecule is complete, has correct atom names, and MRUs that deviate from the reference are labelled. At this point, all information contained in the atom definitions in the input file and their connectivity has been used. Any atoms that could not be recognized will be removed. A warning is issued to the user if this is the case.

Step 3 Resolution Transformation. The resolution transformation step maps the input molecule to the desired output resolution (**Figure 3c** [main paper](#)). We must assume that these mappings are many-to-many correspondences and that in a mapping from e.g. AA to CG a single AA atom can be mapped to multiple CG beads. Unfortunately, this generalization prevents the use of methods developed in graph theory for this problem so far^{91,92}. Instead, we perform the transformation using the same type of overlay we used to identify atoms in the input molecule. This requires a ‘Mapping’ object, which consists of two molecular fragments at different resolutions, and a correspondence between their particles. These Mapping objects are taken from a library. Including this resolution transformation step in the pipeline makes vermouth resolution agnostic, capable of also generating CG topologies.

Mappings from the input force field to the required output force field are taken from the library. However, since these mappings can cross MRU boundaries this is a graph covering problem. This is a variant of the exact cover problem and therefore an NP-hard problem^{80,93}. Because in this case it applies to the full polymer, this is intractable. We sidestep this problem by approaching it as if it were an induced subgraph isomorphism problem where all possible places a mapping fits on the input graph are found, respecting the constraints that atom and MRU names must match. In addition, the mapping may only cross MRU boundaries where it is explicitly allowed by the mapping. If mappings overlap an error is raised. For every mapping that is applied interactions from the corresponding Block are added to the output graph.

Once done, the found modifications can be mapped. First, the modifications are grouped together by connectivity with their MRUs. This is done because with multiple modifications for a single MRU their interactions may influence each other, e.g. (partial) charges in zwitterionic amino acids. Based on these groups the modification mappings that apply to most of those modifications at once are found by solving the exact set covering problem. The found modifications are then applied by finding the corresponding subgraph isomorphisms. Warnings are issued if multiple modification mappings affect the same particle or interaction.

Step 4 Create Topology. Left then is generating the topology. Generating the inter-MRU interactions by applying the appropriate Links is a series of induced subgraph isomorphism problems where all possible ways a link fits on the produced output graph are found. A link can be applied multiple times, and can overlap with other links. Whenever a link is applied the interactions it defines are added to the output graph. In addition to adding interactions, links can also change interactions already set by blocks. For example, the particle type or partial charge may depend on neighboring MRUs. Because of this, links are non-commutative, and the order in which they are applied matters. To resolve this, we solve the subgraph isomorphism problems in the order the links are defined in the library (**Figure 3d** [main paper](#)).

At this point the output graph represents a molecule at the desired resolution with most interactions defined and coordinates can be generated. Usually, these can be trivially taken from the input coordinates. However, in case atoms were missing in the input this might not be possible. In those cases, we generate coordinates based on the coordinates of the neighboring atoms.

Step 5 Post-Processing. Post-processing can consist of any number of steps, and can perform all sorts of force field specific dress-up. For example, it can create an elastic network⁴⁰, or generate the parameters required for Gō interactions^{41,94}. These steps have access to the complete molecule with coordinates and canonical atom names, even if they were missing in the input, and

they have access to the full topology with all associated interactions. Separating these steps out into separate Processors helps to keep them independent of each other, and allowing for any type of post-processing helps in making the program force field agnostic. There can be any number of this kind of processors depending on what was requested by the user.

Step 6 Write Output. Lastly, the output topology and coordinate files have to be written. Since this is just a matter of file formatting, this is trivial. Separating it out from the rest of the pipeline makes the program agnostic of the MD engine used. At the time of writing vermouth is capable of writing Gromacs compatible topologies.

4 – Graph algorithms

Steps 2-4, which form the core of Vermouth rely heavily on graph algorithms, because molecules and polymers can be very naturally described as undirected graphs.^{76–78,95} In our case nodes correspond to atoms, and edges to bonds between atoms. In addition, polymers can also be described as a coarser graph, where nodes correspond to MRUs and edges to bonds between MRUs. Graph theory is a subfield of mathematics that deals with graphs, making it a particularly powerful tool in the context of this work. We primarily use methods from graph theory to identify atoms. First when curating the provided input data (Step 2), but also when performing the resolution transformation (Step 3) and when applying links (Step 4). Our primary tools for this are algorithms for finding induced subgraph isomorphisms,^{96–99} and for finding largest common induced subgraphs.^{100,101}

Largest Common Induced Subgraph. When repairing the provided molecule correspondences between the MRUs in the input molecule (M_r) and their canonical forms (R_r) are needed. In the case where M_r is not a subgraph of R_r and contains atoms that are not described by R_r , this is a largest common induced subgraph (LCIS) problem. The solution to this problem is the largest graph G that is an induced subgraph of both M_r and R_r , and the correspondences between the nodes in G and M_r ; and between the nodes in G and R_r . This problem belongs to the class of NP hard problems.^{80,93} A possible solution to the LCIS problem is to approach it as a repeating subgraph isomorphism problem where initially $G = M_r$, and nodes are removed from G in a breadth-first manner until an induced subgraph isomorphism $G \sqsubseteq R_r$ is found.¹⁰⁰ Once a subgraph isomorphism between G and R_r is found the subgraph is not shrunk further since that would always result in a smaller common subgraph. We have based our implementation on the ISMAGS subgraph isomorphism algorithm^{97,102} since, generally, molecules can be described as very sparse and (locally) symmetric graphs. The ISMAGS algorithm exploits these properties and produces only symmetrically distinct answers which reduces the runtime significantly compared to both other subgraph isomorphism algorithms, such as VF2⁹⁷ and other LCIS algorithms, such as Koch's.¹⁰¹ Since our implementation of the ISMAGS is more generally applicable than just in the context of vermouth we have collaborated with the authors of the popular Python graph library NetworkX⁵² to include our implementation.

We extended our implementation of the ISMAGS algorithm to also solve the LCIS problem in order to further exploit the symmetry breaking constraints used in the subgraph isomorphism problem. The symmetry breaking constraints are used when finding subgraph isomorphisms (see ⁹⁷) and when shrinking the subgraph: when nodes are equivalent the node with the highest index is removed from G preferentially. In addition, to ensure common subgraphs are preferentially found using nodes with a lower index (analogous to the ISMAGS algorithm), the candidate subgraphs are sorted by their node indices. In this way we obtain good performance because in our case it is generally true that: a) there are only a few nodes not part of the reference, and b) those nodes have the highest node index. Because of this we can terminate the algorithm after the first common subgraph is found.

To demonstrate how this works we consider an example where we will try to find all LCISs between graph X and subgraph Y. The example is illustrated in [Figure S1](#). Note that at this point the distinction between “graph” and “subgraph” is arbitrary, except for symmetry detection and performance. Nodes are represented by a letter that reflects their underlying attributes (e.g. atom type). We will consider nodes compatible if they have the same letter, and we distinguish nodes with the same letter by subscripts. First all symmetries in subgraph Y are found. This reveals A_1 to be equivalent to A_2 . In the first iteration we try to find a subgraph isomorphism between X and Y (Iteration 1). Since none can be found, subgraph Y is shrunk. This yields the subgraphs in box “Iteration 2”. Since the subgraph made from the nodes $\{A_1, B, E, F\}$ is symmetry equivalent to the subgraph made from nodes $\{A_2, B, E, F\}$, only the first is taken into consideration. Because no subgraph isomorphism can be found between X and any of these four subgraphs for this iteration, they are shrunk further, resulting in seven subgraphs with unique symmetries consisting of three nodes each. These are depicted in box “Iteration 3”. Of these seven subgraphs, at least one is isomorphic to a subgraph of X ($\{A_1, A_2, B\}$), therefore all subgraph isomorphisms between X and these seven subgraphs are exported in order and the algorithm is terminated.

The algorithm presented is not without faults however: symmetry of X is not taken into consideration, which could reduce the search space dramatically depending on the graphs in question. In addition, some operations are performed multiple times. As an example, many of the subgraphs in [Figure S1](#) contain the motif $\{A_1, B\}$ (in bold). This results in the subgraph isomorphism algorithm reaching the conclusion that $\{A_1, B\}$ is isomorphic to $\{A_1, B\}$ and $\{A_2, B\}$ multiple times. This can be avoided by starting the algorithm using small subgraphs, and growing them as the algorithm progresses. The results of the smaller isomorphism problems can be used to restrict the search space of the larger ones. Since in most of our cases M_r contains only a few nodes that are not isomorphic to nodes in the R_r we do not expect a (large) performance gain. It may be worthwhile to implement an adaptive algorithm that switches strategy after a few iterations of either strategy however.

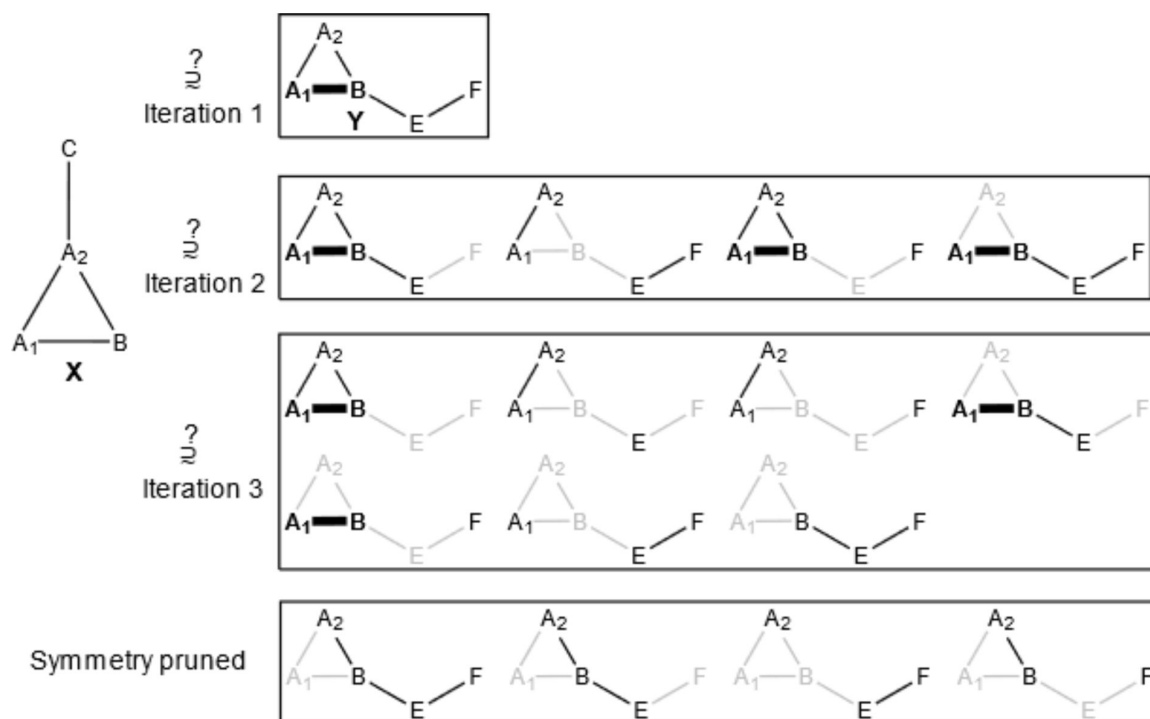


Figure S1

Example of finding all LCISs between graphs X and Y.

Greyed out nodes are not used (they are excluded from the comparison by the shrinking step), but are depicted for clarity. Since nodes A₁ and A₂ in Y are symmetry equivalent not all subgraphs are taken into account. Those that are excluded due to symmetry reasons are depicted in the box Symmetry pruned. Iteration 1: we try to find a subgraph isomorphism between X and Y. None is found. Iteration 2: Y is shrunk to produce the graphs depicted. We try to find subgraph isomorphisms between these and X. None are found. Iteration 3: all graphs from iteration 2 are shrunk further. Since a subgraph isomorphism can be found between at least one of these $\{A_1, A_2, B\}$ and X, the algorithm terminates afterwards. To highlight how often the algorithm discovers that $\{A_1, B\}$ is subgraph isomorphic to X, it is shown in bold.

Graph Covering. To identify unexpected atoms, we need to cover all those atoms with known fragments describing **e.g.** PTMs. We aim to find the solution where all unexpected atoms are covered exactly once, preferentially using fragments with a lower index. In vermouth we sort the fragments by size so that larger fragments are used preferentially. This is a variant of the exact cover problem, making it NP hard^{80,93}. We solve this problem by a recursive backtracking algorithm: in order, we try to fit the fragments on the unexpected atoms until all are covered. If applying a fragment result in atoms that can no longer be covered, the solution is rejected, and the next fit is tried.

4 – AlphaFold Benchmark

The following 7 structures from the AlphaFold benchmarks produced an error, which led *Martinize2* to abort the input file generation:

- AF-O80995-F1-model_v3.pdb
- AF-Q58295-F1-model_v3.pdb
- AF-B1GZ76-F1-model_v3.pdb
- AF-A1ZA47-F1-model_v3.pdb
- AF-J9VQ06-F1-model_v3.pdb
- AF-F1QWK4-F1-model_v3.pdb
- AF-P64653-F1-model_v3.pdb

A list of all surveyed models is available at https://github.com/marrink-lab/martinize-examples/blob/master/AlphaFoldBenchmark/surveyed_models.txt.

5 – Benchmark timings

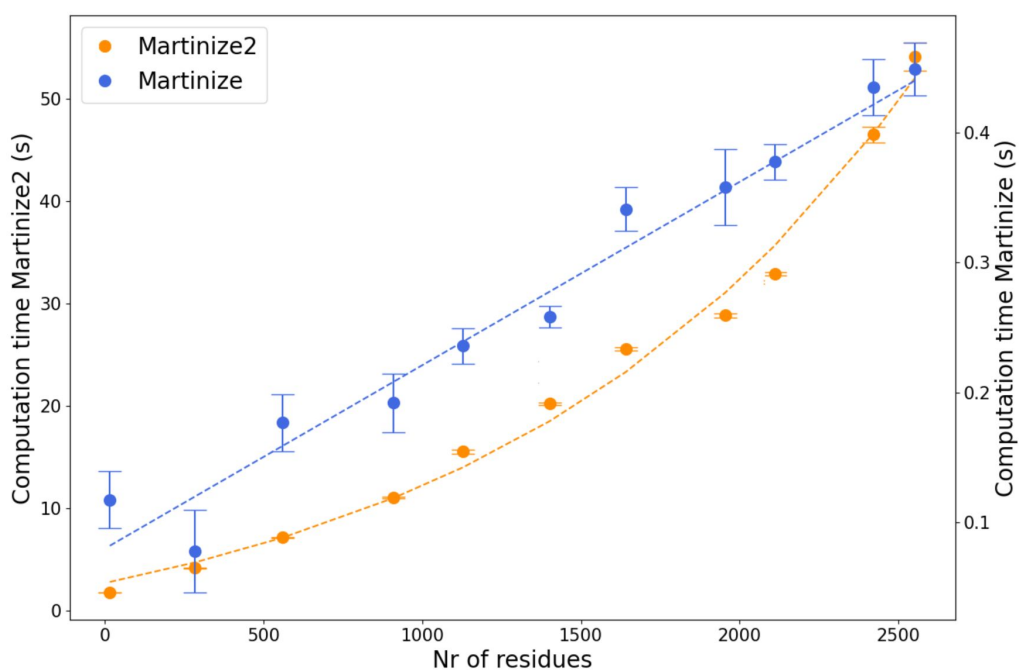


Figure S2

Comparison of processing speeds between martinize and Martinize2.

11 Protein structures with increasing size from a dataset of 200,000 structures were processed with martinize and Martinize2 to record the computation time. The protein structures were spaced evenly by residue count.

Data availability

Input files and commands required to reproduce the example test cases from this paper are available on GitHub at <https://github.com/marrink-lab/martinize-examples>. MD trajectories and benchmark data are available upon reasonable request from the corresponding authors.

Acknowledgements

We would like to thank all users that tested the development versions and provided valuable feedback, in particular the members of the SJM group and the participants of the Martini Workshop 2021. We also thank Melanie König for her feedback on the manuscript and figures. Work is supported by an ERC Advanced Grant (“COMP-MICR-CROW-MEM”) to SJM. PCTS acknowledges the support of the French National Center for Scientific Research (CNRS) and the research collaboration with PharmCADD. PCTS also thank the PSMN (Pôle Scientifique de Modélisation Numérique) and the Centre Blaise Pascal’s IT test platform at ENS de Lyon (Lyon, France) for access to their computing facilities. The platform operates the SIDUS solution developed by Emmanuel Quemener.⁹⁰ JB acknowledges financial support from the Agencia Estatal de Investigación (Spain), the Xunta de Galicia - Consellería de Cultura, Educación e Universidade (Centro de investigación de Galicia accreditation 2019-2022 ED431G-2019/04 and Reference Competitive Group accreditation 2021-2024, CÓDIGO AXUDA). The European Union (European Regional Development Fund - ERDF) and the European Research Council through consolidator grant NANOVR 866559. CB and SJM acknowledges funding from Novo Nordisk Foundation grant NNF20OC0063808, ‘BOUNDLESS’.

Additional information

Code availability

All code can be found online at <https://www.github.com/marrink-lab/vermouth-martinize>. In addition, all released versions are also published on the Python Package Index at <https://www.pypi.org/project/vermouth>. The documentation is available at <https://vermouth-martinize.readthedocs.io/en/latest/index.html>.

Author contributions

PCK and SJM conceived the project; PCK, JB, CB and FG implemented the described software; PCK, JB, TAW designed the program structure; PCTS & FG designed the benchmark tests used along the development of the code to guarantee the accuracy of the models; MvT ran the protein benchmark test-case using the alpha-fold database, while JB ran and analyzed the iTasser database; FG ran and analyzed all other test-cases. CB and FG conducted the all-atom proof of concept. PCTS helped to implement the force field files, and managed feedback from beta testers; PCK and FG wrote the manuscript, with contributions from all authors. SJM provided guidance and supervision in the project.

References

- (1) Marrink S. J., Corradi V., Souza P. C. T., Ingólfsson H. I., Tieleman D. P., Sansom M. S. P. (2019) **Computational Modeling of Realistic Cell Membranes** *Chem Rev* [Google Scholar](#)
- (2) Yu A., Pak A. J., He P., Monje-Galvan V., Casalino L., Gaieb Z., Dommer A. C., Amaro R. E., Voth G. A. (2021) **A Multiscale Coarse-Grained Model of the SARS-CoV-2 Virion** *Biophys J* **120**:1097–1104 [Google Scholar](#)
- (3) Pezeshkian W., Grünewald F., Narykov O., Lu S., Arkhipova V., Solodovnikov A., Wassenaar T. A., Marrink S. J., Korkin D. (2023) **Molecular Architecture and Dynamics of SARS-CoV-2 Envelope by Integrative Modeling** *Structure* **31**:492–503 [Google Scholar](#)
- (4) Dommer A., Casalino L., Kearns F., Rosenfeld M., Wauer N., Ahn S.-H., Russo J., Oliveira S., Morris C., Bogetti A., Trifan A., Brace A., Sztain T., Clyde A., Ma H., Chennubhotla C., Lee H., Turilli M., Khalid S., Tamayo-Mendoza T., Welborn M., Christensen A., Smith D. G., Qiao Z., Sirumalla S. K., O'Connor M., Manby F., Anandkumar A., Hardy D., Phillips J., Stern A., Romero J., Clark D., Dorrell M., Maiden T., Huang L., McCalpin J., Woods C., Gray A., Williams M., Barker B., Rajapaksha H., Pitts R., Gibbs T., Stone J., Zuckerman D. M., Mulholland A. J., Miller T., Jha S., Ramanathan A., Chong L., Amaro R. E. (2023) **#COVIDisAirborne: AI- Enabled Multiscale Computational Microscopy of Delta SARS-CoV-2 in a Respiratory Aerosol** *Int J High Perform Comput Appl* **37**:28–44 [Google Scholar](#)
- (5) Pezeshkian W., König M., Wassenaar T. A., Marrink S. J. (2020) **Backmapping Triangulated Surfaces to Coarse-Grained Membrane Models** *Nat Commun* **11**:1–9 [Google Scholar](#)
- (6) Autin L., Barbaro B. A., Jewett A. I., Ekman A., Verma S., Olson A. J., Goodsell D. S. (2022) **Integrative Structural Modelling and Visualisation of a Cellular Organelle** *QRB Discov* **3**:e11 [Google Scholar](#)
- (7) Feig M., Sugita Y. (2019) **Whole-Cell Models and Simulations in Molecular Detail** *Annu Rev Cell Dev Biol* **35**:191–211 [Google Scholar](#)
- (8) Im W., Liang J., Olson A., Zhou H.-X., Vajda S., Vakser I. A. (2016) **Challenges in Structural Approaches to Cell Modeling** *J Mol Biol* **428**:2943–2964 [Google Scholar](#)
- (9) Stevens J. A., Grünewald F., van Tilburg P. A. M., König M., Gilbert B. R., Brier T. A., Thornburg Z. R., Luthey-Schulten Z., Marrink S. J. (2023) **Molecular Dynamics Simulation of an Entire Cell** *Front Chem* :11 [Google Scholar](#)
- (10) Buch I., Harvey M. J., Giorgino T., Anderson D. P., De Fabritiis G. (2010) **High-Throughput All-Atom Molecular Dynamics Simulations Using Distributed Computing** *J Chem Inf Model* **50**:397–403 [Google Scholar](#)
- (11) Souza P. C. T., Limongelli V., Wu S., Marrink S. J., Monticelli L. (2021) **Perspectives on High-Throughput Ligand/Protein Docking With Martini MD Simulations** *Front Mol Biosci* :8 [Google Scholar](#)
- (12) Kutzner C., Knip C., Cherian A., Nordstrom L., Grubmüller H., de Groot B. L., Gapsys V. (2022) **GROMACS in the Cloud: A Global Supercomputer to Speed Up Alchemical Drug Design** *J*

- (13) Souza P. C. T., Alessandri R., Barnoud J., Thallmair S., Faustino I., Grünewald F., Patmanidis I., Abdizadeh H., Bruininks B. M. H., Wassenaar T. A., Kroon P. C., Melcr J., Nieto V., Corradi V., Khan H. M., Domański J., Javanainen M., Martinez-Seara H., Reuter N., Best R. B., Vattulainen I., Monticelli L., Periole X., Tieleman D. P., de Vries A. H., Marrink S. J. (2021) **Martini 3: A General Purpose Force Field for Coarse-Grained Molecular Dynamics** *Nat Methods* **18**:382–388 [Google Scholar](#)
- (14) Marrink S. J., Risselada H. J., Yefimov S., Tieleman D. P., De Vries A. H. (2007) **The MARTINI Force Field: Coarse Grained Model for Biomolecular Simulations** *Journal of Physical Chemistry B* **111**:7812–7824 [Google Scholar](#)
- (15) Marrink S. J., Monticelli L., Melo M. N., Alessandri R., Tieleman D. P., Souza P. C. T. (2023) **Two Decades of Martini: Better Beads, Broader Scope** *WIREs Computational Molecular Science* **13** [Google Scholar](#)
- (16) Abraham M. J., Melquiond A. S. J., Ippoliti E., Gapsys V., Hess B., Trellet M., Rodrigues J. P. G. L. M., Laure E., Apostolov R., de Groot B. L., Bonvin A. M. J. J., Lindahl E. (2018) **BioExcel Whitepaper on Scientific Software Development** *Zenodo* [Google Scholar](#)
- (17) Grünewald F., Seute L., Alessandri R., König M., Kroon P. C. (2025) **CGsmiles: A Versatile Line Notation for Molecular Representations across Multiple Resolutions** *J Chem Inf Model* **65**:3405–3419 [Google Scholar](#)
- (18) de Jong D. H., Singh G., Bennett W. F. D., Arnarez C., Wassenaar T. A., Schäfer L. V., Periole X., Tieleman D. P., Marrink S. J. (2013) **Improved Parameters for the Martini Coarse-Grained Protein Force Field** *J Chem Theory Comput* **9**:687–697 [Google Scholar](#)
- (19) Abraham M. J., Murtola T., Schulz R., Páll S., Smith J. C., Hess B., Lindahl E. (2015) **GROMACS: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers** *SoftwareX* **1-2**:19–25 [Google Scholar](#)
- (20) Páll S., Abraham M. J., Kutzner C., Hess B., Lindahl E. (2015) **Tackling Exascale Software Challenges in Molecular Dynamics Simulations with GROMACS** :3–27
- (21) Case D. A., Cheatham T. E., Darden T., Gohlke H., Luo R., Merz K. M., Onufriev A., Simmerling C., Wang B., Woods R. J. (2005) **The Amber Biomolecular Simulation Programs** *J Comput Chem* **26**:1668–1688 [Google Scholar](#)
- (22) Brooks B. R., Brooks C. L., Mackerell A. D., Nilsson L., Petrella R. J., Roux B., Won Y., Archontis G., Bartels C., Boresch S., Caflisch A., Caves L., Cui Q., Dinner A. R., Feig M., Fischer S., Gao J., Hodoscek M., Im W., Kuczera K., Lazaridis T., Ma J., Ovchinnikov V., Paci E., Pastor R. W., Post C. B., Pu J. Z., Schaefer M., Tidor B., Venable R. M., Woodcock H. L., Wu X., Yang W., York D. M., Karplus M. (2009) **CHARMM: The Biomolecular Simulation Program** *J Comput Chem* **30**:1545–1614 [Google Scholar](#)
- (23) Phillips J. C., Braun R., Wang W., Gumbart J., Tajkhorshid E., Villa E., Chipot C., Skeel R. D., Kalé L., Schulten K. (2005) **Scalable Molecular Dynamics with NAMD** *J Comput Chem* **26**:1781–1802 [Google Scholar](#)

- (24) Machado M. R., Pantano S. (2016) **SIRAH Tools: Mapping, Backmapping and Visualization of Coarse-Grained Models** *Bioinformatics* **32**:1568–1570 [Google Scholar](#)
- (25) Danne R., Poojari C., Martinez-Seara H., Rissanen S., Lolicato F., Rog T., Vattulainen I. (2017) **DoGlycans – Tools for Preparing Carbohydrate Structures for Atomistic Simulations of Glycoproteins, Glycolipids, and Carbohydrate Polymers for GROMACS** *J Chem Inf Model* **57**:2401–2406 [Google Scholar](#)
- (26) Girard M., Ehlen A., Shakya A., Bereau T., de la Cruz M. O. (2019) **Hoobas: A Highly Object-Oriented Builder for Molecular Dynamics** *Comput Mater Sci* **167**:25–33 [Google Scholar](#)
- (27) Jo S., Cheng X., Lee J., Kim S., Park S.-J., Patel D. S., Beaven A. H., Lee K., Rui H., Park S., Lee H. S., Roux B., MacKerell A. D., Klauda J. B., Qi Y., Im W. (2017) **CHARMM-GUI 10 Years for Biomolecular Modeling and Simulation** *J Comput Chem* **38**:1114–1124 [Google Scholar](#)
- (28) Qi Y., Ingólfsson H. I., Cheng X., Lee J., Marrink S. J., Im W. (2015) **CHARMM-GUI Martini Maker for Coarse-Grained Simulations with the Martini Force Field** *J Chem Theory Comput* **11**:4486–4494 [Google Scholar](#)
- (29) Malde A. K., Zuo L., Breeze M., Stroet M., Poger D., Nair P. C., Oostenbrink C., Mark A. E. (2011) **An Automated Force Field Topology Builder (ATB) and Repository: Version 1.0** *J Chem Theory Comput* **7**:4026–4037 [Google Scholar](#)
- (30) Canzar S., El-Kebir M., Pool R., Elbassioni K., Malde A. K., Mark A. E., Geerke D. P., Stougie L., Klau G. W. (2013) **Charge Group Partitioning in Biomolecular Simulation** *Journal of Computational Biology* **20**:188–198 [Google Scholar](#)
- (31) Jorgensen W. L., Tirado-Rives J. (2005) **Potential Energy Functions for Atomic-Level Simulations of Water and Organic and Biomolecular Systems** *Proceedings of the National Academy of Sciences* **102**:6665–6670 [Google Scholar](#)
- (32) Dodda L. S., Vilseck J. Z., Tirado-Rives J., Jorgensen W. L. (2017) **1.14*CM1A-LBCC: Localized Bond-Charge Corrected CM1A Charges for Condensed-Phase Simulations** *J Phys Chem B* **121**:3864–3870 [Google Scholar](#)
- (33) Dodda L. S., Cabeza de Vaca I., Tirado-Rives J., Jorgensen W. L. (2017) **LigParGen Web Server: An Automatic OPLS-AA Parameter Generator for Organic Ligands** *Nucleic Acids Res* **45**:W331–W336 [Google Scholar](#)
- (34) Vanommeslaeghe K., MacKerell A. D. (2012) **Automation of the CHARMM General Force Field (CGenFF) I: Bond Perception and Atom Typing** *J Chem Inf Model* **52**:3144–3154 [Google Scholar](#)
- (35) Uusitalo J. J., Ingólfsson H. I., Akhshi P., Tieleman D. P., Marrink S. J. (2015) **Martini Coarse-Grained Force Field: Extension to DNA** *J Chem Theory Comput* **11**:3932–3945 [Google Scholar](#)
- (36) Jo S., Kim T., Iyer V. G., Im W. (2008) **CHARMM-GUI: A Web-Based Graphical User Interface for CHARMM** *J. Comput. Chem* **29**:1859–1865 [Google Scholar](#)
- (37) Uusitalo J. J., Ingólfsson H. I., Marrink S. J., Faustino I. (2017) **Martini Coarse-Grained Force Field: Extension to RNA** *Biophys J* **113**:246–256 [Google Scholar](#)
- (38) Souza P. C. T., Thallmair S., Conflitti P., Ramírez-Palacios C., Alessandri R., Raniolo S., Limongelli V., Marrink S. J. (2020) **Protein-Ligand Binding with the Coarse-Grained Martini Model** *Nat*

- (39) Herzog F. A., Braun L., Schoen I., Vogel V. (2016) **Improved Side Chain Dynamics in MARTINI Simulations of Protein-Lipid Interfaces** *J Chem Theory Comput* 12:2446–2458 [Google Scholar](#)
- (40) Periole X., Cavalli M., Marrink S.-J., Ceruso M. A. (2009) **Combining an Elastic Network With a Coarse-Grained Molecular Force Field: Structure, Dynamics, and Intermolecular Recognition** *J Chem Theory Comput* 5:2531–2543 [Google Scholar](#)
- (41) Poma A. B., Cieplak M., Theodorakis P. E. (2017) **Combining the MARTINI and Structure-Based Coarse-Grained Approaches for the Molecular Dynamics Studies of Conformational Transitions in Proteins** *J Chem Theory Comput* 13:1366–1374 [Google Scholar](#)
- (42) Korshunova K., Kiuru J., Liekkinen J., Enkavi G., Vattulainen I., Bruininks B. M. H. (2024) **Martini 3 OliGomers: A Scalable Approach for Multimers and Fibrils in GROMACS** *J Chem Theory Comput* 20:7635–7645 [Google Scholar](#)
- (43) Pedersen K. B., Borges-Araújo L., Stange A. D., Souza P. C. T., Marrink S. J., Schiøtt B. OLIVES (2024) **A Gō-like Model for Stabilizing Protein Structure via Hydrogen Bonding Native Contacts in the Martini 3 Coarse-Grained Force Field** *J Chem Theory Comput* [Google Scholar](#)
- (44) Monticelli L., Kandasamy S. K., Periole X., Larson R. G., Tieleman D. P., Marrink S.-J. (2008) **The MARTINI Coarse-Grained Force Field: Extension to Proteins** *J Chem Theory Comput* 4:819–834 [Google Scholar](#)
- (45) Chue Hong N. P., Katz D. S., Barker M., Lamprecht A.-L., Martinez C., Psomopoulos F. E., Harrow J., Castro L. J., Gruenpeter M., Martinez P. A., Honeyman T., Struck A., Lee A., Loewe A., van Werkhoven B., Jones C., Garijo D., Plomp E., Genova F., Shanahan H., Leng J., Hellström M., Sandström M., Sinha M., Kuzak M., Herterich P., Zhang Q., Islam S., Sansone S.-A., Pollard T., Atmojo U. D., Williams A., Czerniak A., Niehues A., Fouilloux A. C., Desinghu B., Goble C., Richard C., Gray C., Erdmann C., Nüst D., Tartarini D., Rangelova E., Anzt H., Todorov I., McNally J., Moldon J., Burnett J., Garrido-Sánchez J., Belhajjame K., Sesink L., Hwang L., Tovani-Palone M. R., Wilkinson M. D., Servillat M., Liffers M., Fox M., Miljković N., Lynch N., Martinez Lavanchy P., Gesing S., Stevens S., Martinez Cuesta S., Peroni S., Soiland-Reyes S., Bakker T., Rabemanantsoa T., Sochat V., Yehudi Y., WG R. F. (2022) **FAIR Principles for Research Software (FAIR4RS Principles)**
- (46) Wilkinson M. D., Dumontier M., Aalbersberg Ij. J., Appleton G., Axton M., Baak A., Blomberg N., Boiten J.-W., da Silva Santos L. B., Bourne P. E., Bouwman J., Brookes A. J., Clark T., Crosas M., Dillo I., Dumon O., Edmunds S., Evelo C. T., Finkers R., Gonzalez-Beltran A., Gray A. J. G., Groth P., Goble C., Grethe J. S., Heringa J., 't Hoen P. A. C., Hooft R., Kuhn T., Kok R., Kok J., Lusher S. J., Martone M. E., Mons A., Packer A. L., Persson B., Rocca-Serra P., Roos M., van Schaik R., Sansone S.-A., Schultes E., Sengstag T., Slater T., Strawn G., Swertz M. A., Thompson M., van der Lei J., van Mulligen E., Velterop J., Waagmeester A., Wittenburg P., Wolstencroft K., Zhao J., Mons B. (2016) **The FAIR Guiding Principles for Scientific Data Management and Stewardship** *Sci Data* 3:160018 [Google Scholar](#)
- (47) Alibay I., Barnoud J., Beckstein O., Gowers R. J., Naughton F., Wang L. (2022) **MDAKits: Supporting and Promoting the Development of Community Packages Leveraging the MDAnalysis Library**
- (48) Grünwald F., Alessandri R., Kroon P. C., Monticelli L., Souza P. C. T., Marrink S. J. (2022) **Polyply; a Python Suite for Facilitating Simulations of Macromolecules and Nanomaterials** *Nat Commun* 13:68 [Google Scholar](#)

- (49) Empereur-Mot C., Pesce L., Doni G., Bochicchio D., Capelli R., Perego C., Pavan G. M. (2020) **Swarm-CG : Automatic Parametrization of Bonded Terms in MARTINI-Based Coarse-Grained Models of Simple to Complex Molecules via Fuzzy Self-Tuning Particle Swarm Optimization** *ACS Omega* **5**:32823–32843 [Google Scholar](#)
- (50) Wassenaar T. A., Pluhackova K., Böckmann R. A., Marrink S. J., Tieleman D. P. (2014) **Going Backward: A Flexible Geometric Approach to Reverse Transformation from Coarse Grained to Atomistic Models** *J Chem Theory Comput* **10**:676–690 [Google Scholar](#)
- (51) Marx V. (2020) **When Computational Pipelines Go ‘Clank.’** *Nat Methods* **17**:659–662 [Google Scholar](#)
- (52) Hagberg A. A., Schult D. A., Swart P. J. (2008) **Exploring Network Structure, Dynamics, and Function Using NetworkX** In: *Proceedings of the 7th Python in Science Conference* pp. 11–15 [Google Scholar](#)
- (53) Bashford D., Karplus M. (1990) **pKa's of Ionizable Groups in Proteins: Atomic Detail from a Continuum Electrostatic Model** *Biochemistry* **29**:10219–10225 [Google Scholar](#)
- (54) Huang Y., Chen W., Wallace J. A., Shen J. (2016) **All-Atom Continuous Constant PH Molecular Dynamics with Particle Mesh Ewald and Titratable Water** *J Chem Theory Comput* **12**:5411–5421 [Google Scholar](#)
- (55) Donnini S., Tegeler F., Groenhof G., Grubmüller H. (2011) **Constant PH Molecular Dynamics in Explicit Solvent with λ -Dynamics** *J Chem Theory Comput* **7**:1962–1978 [Google Scholar](#)
- (56) Bennett W. F. D., Chen A. W., Donnini S., Groenhof G., Tieleman D. P. (2013) **Constant PH Simulations with the Coarse-Grained MARTINI Model — Application to Oleic Acid Aggregates** *Can J Chem* **91**:839–846 [Google Scholar](#)
- (57) Grünewald F., Souza P. C. T., Abdizadeh H., Barnoud J., de Vries A. H., Marrink S. J. (2020) **Titratable Martini Model for Constant PH Simulations** *J Chem Phys* **153**:024118 [Google Scholar](#)
- (58) Aho N., Buslaev P., Jansen A., Bauer P., Groenhof G., Hess B. (2022) **Scalable Constant PH Molecular Dynamics in GROMACS** *J Chem Theory Comput* **18**:6148–6160 [Google Scholar](#)
- (59) Huang J., Rauscher S., Nawrocki G., Ran T., Feig M., de Groot B. L., Grubmüller H., MacKerell A. D. (2017) **CHARMM36m: An Improved Force Field for Folded and Intrinsically Disordered Proteins** *Nat Methods* **14**:71–73 [Google Scholar](#)
- (60) Lindorff-Larsen K., Piana S., Palmo K., Maragakis P., Klepeis J. L., Dror R. O., Shaw D. E. (2010) **Improved Side-Chain Torsion Potentials for the Amber Ff99SB Protein Force Field** *Proteins: Structure, Function, and Bioinformatics* **78**:1950–1958 [Google Scholar](#)
- (61) Anandakrishnan R., Aguilar B., Onufriev A. V. (2012) **H++ 3.0: Automating PK Prediction and the Preparation of Biomolecular Structures for Atomistic Molecular Modeling and Simulations** *Nucleic Acids Res* **40**:W537–W541 [Google Scholar](#)
- (62) Olsson M. H. M., Søndergaard C. R., Rostkowski M., Jensen J. H. (2011) **PROPKA3: Consistent Treatment of Internal and Surface Residues in Empirical p K a Predictions** *J Chem Theory Comput* **7**:525–537 [Google Scholar](#)

- (63) Kmiecik S., Gront D., Kolinski M., Wieteska L., Dawid A. E., Kolinski A. (2016) **Coarse-Grained Protein Models and Their Applications** *Chem Rev* **116**:7898–7936 [Google Scholar](#)
- (64) Varadi M., Anyango S., Deshpande M., Nair S., Natassia C., Yordanova G., Yuan D., Stroe O., Wood G., Laydon A., Žídek A., Green T., Tunyasuvunakool K., Petersen S., Jumper J., Clancy E., Green R., Vora A., Lutfi M., Figurnov M., Cowie A., Hobbs N., Kohli P., Kleywegt G., Birney E., Hassabis D., Velankar S. (2022) **AlphaFold Protein Structure Database: Massively Expanding the Structural Coverage of Protein-Sequence Space with High-Accuracy Models** *Nucleic Acids Res* **50**:D439–D444 [Google Scholar](#)
- (65) Jumper J., Evans R., Pritzel A., Green T., Figurnov M., Ronneberger O., Tunyasuvunakool K., Bates R., Žídek A., Potapenko A., Bridgland A., Meyer C., Kohl S. A. A., Ballard A. J., Cowie A., Romera-Paredes B., Nikolov S., Jain R., Adler J., Back T., Petersen S., Reiman D., Clancy E., Zielinski M., Steinegger M., Pacholska M., Berghammer T., Bodenstein S., Silver D., Vinyals O., Senior A. W., Kavukcuoglu K., Kohli P., Hassabis D. (2021) **Highly Accurate Protein Structure Prediction with AlphaFold** *Nature* **596**:583–589 [Google Scholar](#)
- (66) Hilpert C., Beranger L., Souza P. C. T., Vainikka P. A., Nieto V., Marrink S. J., Monticelli L., Launay G. (2023) **Facilitating CG Simulations with MAD: The MARTINI Database Server** *J Chem Inf Model* **63**:702–710 [Google Scholar](#)
- (67) Sousa F. M., Lima L. M. P., Arnarez C., Pereira M. M., Melo M. N. (2021) **Coarse-Grained Parameterization of Nucleotide Cofactors and Metabolites: Protonation Constants, Partition Coefficients, and Model Topologies** *J Chem Inf Model* **61**:335–346 [Google Scholar](#)
- (68) Alessandri R., Barnoud J., Gertsen A. S., Patmanidis I., de Vries A. H., Souza P. C. T., Marrink S. J. (2022) **Martini 3 Coarse-Grained Force Field: Small Molecules** *Adv Theory Simul* **5**:2100391 [Google Scholar](#)
- (69) Grünewald F., Rossi G., de Vries A. H., Marrink S. J., Monticelli L. (2018) **Transferable MARTINI Model of Poly(Ethylene Oxide)** *J. Phys. Chem. B* **122**:7436–7449 [Google Scholar](#)
- (70) Panizon E., Bochicchio D., Monticelli L., Rossi G. (2015) **MARTINI Coarse-Grained Models of Polyethylene and Polypropylene** *Journal of Physical Chemistry B* **119**:8209–8216 [Google Scholar](#)
- (71) Yang J., Yan R., Roy A., Xu D., Poisson J., Zhang Y. (2015) **The I-TASSER Suite: Protein Structure and Function Prediction** *Nat Methods* **12**:7–8 [Google Scholar](#)
- (72) Kabsch W., Sander C. (1983) **Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features** *Biopolymers* **22**:2577–2637 [Google Scholar](#)
- (73) Touw W. G., Baakman C., Black J., te Beek T. A. H., Krieger E., Joosten R. P., Vriend G. (2015) **A Series of PDB-Related Databanks for Everyday Needs** *Nucleic Acids Res* **43**:D364–D368 [Google Scholar](#)
- (74) Harris C. R., Millman K. J., van der Walt S. J., Gommers R., Virtanen P., Cournapeau D., Wieser E., Taylor J., Berg S., Smith N. J., Kern R., Picus M., Hoyer S., van Kerkwijk M. H., Brett M., Haldane A., del Río J. F., Wiebe M., Peterson P., Gérard-Marchant P., Sheppard K., Reddy T., Weckesser W., Abbasi H., Gohlke C., Oliphant T. E. (2020) **Array Programming with NumPy** *Nature* **585**:357–362 [Google Scholar](#)

- (75) Virtanen P., Gommers R., Oliphant T. E., Haberland M., Reddy T., Cournapeau D., Burovski E., Peterson P., Weckesser W., Bright J., van der Walt S. J., Brett M., Wilson J., Millman K. J., Mayorov N., Nelson A. R. J., Jones E., Kern R., Larson E., Carey C. J., Polat \.Ilhan, Feng Y., Moore E. W., VanderPlas J., Laxalde D., Perktold J., Cimrman R., Henriksen I., Quintero E. A., Harris C. R., Archibald A. M., Ribeiro A. H., Pedregosa F., van Mulbregt P. (2020) **SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python** *Nat Methods* **17**:261–272 [Google Scholar](#)
- (76) Engler M. S., Caron B., Mark A. E. (2018) **Multiple-Choice Knapsack for Assigning Partial Atomic Charges in Drug-Like Molecules** In: *18th International Workshop on Algorithms in Bioinformatics (WABI 2018)* pp. 1–13 [Google Scholar](#)
- (77) Engler M. S., El-kebir M., Mulder J., Mark A. E., Geerke D. P., Klau G. W. (2017) **Enumerating Common Molecular Substructures** *PeerJ Prepr* :1–10 [Google Scholar](#)
- (78) Cao Y., Jiang T., Girke T. (2008) **A Maximum Common Substructure-Based Algorithm for Searching and Predicting Drug-like Compounds** *Bioinformatics* **24**:i366–i374 [Google Scholar](#)
- (79) Brasnett C., Marrink S. J. (2025) **MartiniGlass: A Tool for Enabling Visualization of Coarse-Grained Martini Topologies** *J Chem Inf Model* **65**:3137–3141 [Google Scholar](#)
- (80) Cook S. A. (1971) **The Complexity of Theorem-Proving Procedures** In: *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71* pp. 151–158 [Google Scholar](#)
- (81) Eastman P., Swails J., Chodera J. D., McGibbon R. T., Zhao Y., Beauchamp K. A., Wang L.-P., Simmonett A. C., Harrigan M. P., Stern C. D., Wiewiora R. P., Brooks B. R., Pande V. S. (2017) **OpenMM 7: Rapid Development of High Performance Algorithms for Molecular Dynamics** *PLoS Comput Biol* **13**:e1005659 [Google Scholar](#)
- (82) Nnyigide O. S., Nnyigide T. O., Lee S.-G., Hyun K. (2022) **Protein Repair and Analysis Server: A Web Server to Repair PDB Structures, Add Missing Heavy Atoms and Hydrogen Atoms, and Assign Secondary Structures by Amide Interactions** *J Chem Inf Model* **62**:4232–4246 [Google Scholar](#)
- (83) Lee J., Cheng X., Swails J. M., Yeom M. S., Eastman P. K., Lemkul J. A., Wei S., Buckner J., Jeong J. C., Qi Y., Jo S., Pande V. S., Case D. A., Brooks C. L., MacKerell A. D., Klauda J. B., Im W. (2016) **CHARMM-GUI Input Generator for NAMD, GROMACS, AMBER, OpenMM, and CHARMM/OpenMM Simulations Using the CHARMM36 Additive Force Field** *J Chem Theory Comput* **12**:405–413 [Google Scholar](#)
- (84) Bjelkmar P., Larsson P., Cuendet M. A., Hess B., Lindahl E. (2010) **Implementation of the CHARMM Force Field in GROMACS: Analysis of Protein Stability Effects from Correction Maps, Virtual Interaction Sites, and Water Models** *J Chem Theory Comput* **6**:459–466 [Google Scholar](#)
- (85) Bussi G., Donadio D., Parrinello M. (2007) **Canonical Sampling through Velocity Rescaling** *J Chem Phys* **126**:14101 [Google Scholar](#)
- (86) Parrinello M., Rahman A. (1981) **Polymorphic Transitions in Single Crystals: A New Molecular Dynamics Method** *J Appl Phys* **52**:7182–7190 [Google Scholar](#)

- (87) Berendsen H. J. C., Postma J. P. M., van Gunsteren W. F., DiNola A., Haak J. R. (1984) **Molecular Dynamics with Coupling to an External Bath** *J Chem Phys* **81**:3684–3690 [Google Scholar](#)
- (88) Abraham M. J., Murtola T., Schulz R., Páll S., Smith J. C., Hess B., Lindahl E. (2015) **GROMACS: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers** *SoftwareX* **1-2**:19–25 [Google Scholar](#)
- (89) de Jong D. H., Baoukina S., Ingólfsson H. I., Marrink S. J. (2016) **Martini Straight: Boosting Performance Using a Shorter Cutoff and GPUs** *Comput Phys Commun* **199**:1–7 [Google Scholar](#)
- (90) Quemener E., Corvellec M. (2013) **SIDUS—the Solution for Extreme Deduplication of an Operating System** *Linux J* **2013** [Google Scholar](#)
- (91) Webb M. A., Delannoy J.-Y., de Pablo J. J. (2018) **Graph-Based Approach to Systematic Molecular Coarse-Graining** *Journal of Chemical Theory and Computation* *acs.jctc.8b00920* <https://doi.org/10.1021/acs.jctc.8b00920> | [Google Scholar](#)
- (92) Chakraborty M., Xu C., White A. D. (2018) **Encoding and selecting coarse-grain mapping operators with hierarchical graphs** *The Journal of Chemical Physics* **149**:134106 [Google Scholar](#)
- (93) Karp R. M. (1972) **Reducibility among Combinatorial Problems** In: Miller R. E., Thatcher J. W., Bohlinger J. D., editors. *Complexity of Computer Computations* Springer US pp. 85–103 https://doi.org/10.1007/978-1-4684-2001-2_9 | [Google Scholar](#)
- (94) Taketomi H., Ueda Y., Gō N. (1975) **Studies on protein folding, unfolding and fluctuations by computer simulation. I. The effect of specific amino acid sequence represented by specific inter-unit interactions** *International journal of peptide and protein research* **7**:445–459 [Google Scholar](#)
- (95) Chung F. (2010) **Graph Theory in the Information Age** *Notices of the AMS* **57**:726–732 [Google Scholar](#)
- (96) Bonnici V., Giugno R., Pulvirenti A., Shasha D., Ferro A. (2013) **A subgraph isomorphism algorithm and its application to biochemical data** *BMC Bioinformatics* **14**:S13 [Google Scholar](#)
- (97) Houbraken M., et al. (2014) **The Index-Based Subgraph Matching Algorithm with General Symmetries (ISMAGS): Exploiting Symmetry for Faster Subgraph Enumeration** *PLoS ONE* **9**:e97896 [Google Scholar](#)
- (98) Cordella L. P., Foggia P., Sansone C., Vento M. (2001) **An improved algorithm for matching large graphs** In: *Proceedings of the 3rd IAPR Workshop on Graph-Based Representations in Pattern Recognition* pp. 149–159 [Google Scholar](#)
- (99) Cordella L. P., Foggia P., Sansone C., Vento M. (2004) **A (sub)graph isomorphism algorithm for matching large graphs** *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**:1367–1372 [Google Scholar](#)
- (100) Krissinel E. B., Henrick K. (2004) **Common subgraph isomorphism detection by backtracking search** *Software: Practice and Experience* **34**:591–607 [Google Scholar](#)
- (101) Koch I. (2001) **Enumerating all connected maximal common subgraphs in two graphs** *Theoretical Computer Science* **250**:1–30 [Google Scholar](#)

- (102) Demeyer S., et al. (2013) **The Index-Based Subgraph Matching Algorithm (ISMA): Fast Subgraph Enumeration in Large Networks Using Optimized Search Trees** *PLoS ONE* 8:e61183 [Google Scholar](#)
- (103) Brooks B. R., et al. (1983) **CHARMM: A program for macromolecular energy, minimization, and dynamics calculations** *Journal of Computational Chemistry* 4:187–217 [Google Scholar](#)

Author information

Peter C Kroon

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands

Fabian Grunewald

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands, Heidelberg Institute for Theoretical Studies (HITS), Heidelberg, Germany, Interdisciplinary Center for Scientific Computing, Heidelberg University, Heidelberg, Germany

For correspondence: f.grunewald@rug.nl

Jonathan Barnoud

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands, CiTIUS Intelligent Technologies Research Centre,, Santiago de Compostela, Spain

Marco van Tilburg

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands

Chris Brasnett

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands

Paulo Cesar Telles de Souza

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands, Laboratoire de Biologie et Modélisation de la Cellule, CNRS, UMR 5239, Inserm, U1293, Université Claude Bernard Lyon 1, Ecole Normale Supérieure de Lyon, Lyon, France, Centre Blaise Pascal de Simulation et de Modélisation Numérique, Ecole Normale Supérieure de Lyon, Lyon, France

Tsjerk A Wassenaar

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands

Siewert-Jan J Marrink

Groningen Biomolecular Sciences and Biotechnology Institute, University of Groningen, Groningen, Netherlands

For correspondence: s.j.marrink@rug.nl

Editors

Reviewing Editor

Qiang Cui

Boston University, Boston, United States of America

Senior Editor

Qiang Cui

Boston University, Boston, United States of America

Reviewer #1 (Public review):

Summary:

In this study, the authors provide a new computational platform called Vermouth to automate topology generation, a crucial step that any biomolecular simulation starts with. Given a wide arrange of chemical structures that need to be simulated, varying qualities of structural models as inputs obtained from various sources, and diverse force fields and molecular dynamics engines employed for simulations, automation of this fundamental step is challenging, especially for complex systems and in case that there is a need to conduct high-throughput simulations in the application of computer-aided drug design (CADD). To overcome this challenge, the authors develop a programing library composed of components that carry out various types of fundamental functionalities that are commonly encountered in topological generation. These components are intended to be general for any type of molecules and not to depend on any specific force field and MD engines. To demonstrate the applicability of this library, the authors employ those components to reassemble a pipeline called Martinize2 used in topology generation for simulations with a widely used coarse-grained model (CG) MARTINI. This pipeline can fully recapitulate the functionality of its original version Martinize but exhibit greatly enhanced generality, as confirmed by the ability of the pipeline to faithfully generate topologies for two high-complexity benchmarking sets of proteins.

Strengths:

The main strength of this work is the use of concepts and algorithms associated with induced subgraph in graph theory to automate several key but non-trivial steps of topology generation such as the identification of monomer residue units (MRU), the repair of input structures with missing atoms, the mapping of topologies between different resolutions, and the generation of parameters needed for describing interactions between MRUs. In addition, the documentation website provided by the authors is very informative, allowing users to get quickly started with Vermouth.

Weaknesses:

Although the Vermouth library can work for different force fields, exhibiting certain generality, its application has been demonstrated only with GROMACS. The extension of the library to other major MD engines could be future directions for improvement but may not be needed for this study.

<https://doi.org/10.7554/eLife.90627.3.sa3>

Reviewer #2 (Public review):

This work introduces a Vermouth library framework to enhance software development within the Martini community. Specifically, it presents a Vermouth-powered program, Martinize2, for generating coarse-grained structures and topologies from atomistic structures. In addition to introducing the Vermouth library and the Martinize2 program, this paper illustrates how Martinize2 identifies atoms, maps them to the Martini model, generates topology files, and identifies protonation states or post-translational modifications. Compared with the prior version, the authors provide a new figure to show that Martinize2 can be applied to various molecules, such as proteins, cofactors, and lipids. To demonstrate the general application, Martinize2 was used for converting 73% of 87,084 protein structures from the template library, with failed cases primarily blamed on missing coordinates.

I appreciate the changes that the authors made to clarify the novelty. I have no doubt this paper will receive attention and citations.

<https://doi.org/10.7554/eLife.90627.3.sa2>

Reviewer #3 (Public review):

Summary:

The manuscript Kroon et al. described two algorithms, which when combined achieve high throughput automation of "martinizing" protein structures with selected protonation states and post-translational modifications.

The authors have addressed all of my concerns as provided previously. Specifically, Figure S2 will be a very useful guideline for future improvement (e.g., parallelization) of the code.

<https://doi.org/10.7554/eLife.90627.3.sa1>

Author response:

The following is the authors' response to the previous reviews.

Reviewer #1 (Public Review):

Summary:

In this study, the authors provide a new computational platform called Vermouth to automate topology generation, a crucial step that any biomolecular simulation starts with. Given a wide arrange of chemical structures that need to be simulated, varying qualities of structural models as inputs obtained from various sources, and diverse force fields and molecular dynamics engines employed for simulations, automation of this fundamental step is challenging, especially for complex systems and in case that there is a need to conduct high-throughput simulations in the application of computer-aided drug design (CADD). To overcome this challenge, the authors develop a programing library composed of components that carry out various types of fundamental functionalities that are commonly encountered in topological generation. These components are intended to be general for any type of molecules and not to depend on any specific force field and MD engines. To demonstrate the applicability of this library, the authors employ those components to re-assemble a pipeline called Martinize2 used in topology generation for simulations with a widely used coarse-grained model (CG) MARTINI. This pipeline can fully recapitulate the functionality of its original version

Martinize but exhibit greatly enhanced generality, as confirmed by the ability of the pipeline to faithfully generate topologies for two high-complexity benchmarking sets of proteins.

Strengths:

The main strength of this work is the use of concepts and algorithms associated with induced subgraph in graph theory to automate several key but non-trivial steps of topology generation such as the identification of monomer residue units (MRU), the repair of input structures with missing atoms, the mapping of topologies between different resolutions, and the generation of parameters needed for describing interactions between MRUs. In addition, the documentation website provided by the authors is very informative, allowing users to get quickly started with Vermouth.

Weaknesses:

Although the Vermouth library is designed as a general tool for topology generation for molecular simulations, only its applications with MARTINI have been demonstrated in the current study. Thus, the claimed generality of Vermouth remains to be examined. The authors may consider to point out this in their manuscript.

In order to demonstrate generality of the here proposed concepts for generating topologies for molecular dynamics simulations, we have now implemented and tested a workflow that will produce topologies for the popular CHARMM36 all-atom force field. To facilitate generation of all-atom topologies with Martinize2 a .rtp reader was introduced, which allows users to provide .rtp files that are the native GROMACS topology files for proteins instead of .ff files. These .rtp files exist for all major atomic protein forcefields. In addition, for CHARMM36 we also included modification files, which describe non-standard pH amino acids, histidine tautomers, and end terminal modifications. Thus, the current implementation unlocks all features available at the CG Martini level also for CHARMM36. We note that users must add the modifications files for other all-atom force fields e.g. AMBER.

We have added a new item in the main manuscript (p28) briefly describing this proof-of-concept implementation. However, we like to point out that there are many specialized tools for the various force fields adopted by the respective communities. Thus, an exhaustive discussion on the capabilities of Martinize2 for all-atom force fields seemed out of place.

Reviewer #2 (Public Review):

This work introduces a Vermouth library framework to enhance software development within the Martini community. Specifically, it presents a Vermouth-powered program, Martinize2, for generating coarse-grained structures and topologies from atomistic structures. In addition to introducing the Vermouth library and the Martinize2 program, this paper illustrates how Martinize2 identifies atoms, maps them to the Martini model, generates topology files, and identifies protonation states or post-translational modifications. Compared with the prior version, the authors provide a new figure to show that Martinize2 can be applied to various molecules, such as proteins, cofactors, and lipids. To demonstrate the general application, Martinize2 was used for converting 73% of 87,084 protein structures from the template library, with failed cases primarily blamed on missing coordinates.

I was hoping to see some fundamental changes in the resubmitted version. To my disappointment, the manuscript remains largely unchanged (even the typo I pointed out previously was not fixed). I do not doubt that Martinize2 and Vermouth are useful to the Martini community, and this paper will have some impact. The manuscript is very technical and limited to the Martini community. The scientific insight for the general

coarse-grained modeling community is unclear. The goal of the work is ambitious (such as high-throughput simulations and whole-cell modeling), but the results show just a validation of Martinize2. This version does not reverse my previous impression that it is incremental. As I pointed out in my previous review (and no response from the authors), all the issues associated with the Martini model are still there, e.g. the need for ENM. In this shape, I feel this manuscript is suitable for a specialized journal in computational biophysics or stays as part of the GitHub repository.

We apologize for not fixing the typo; it was fixed but unfortunately got reintroduced in the final resubmitted version. We politely disagree that the goal of the work itself is high-throughput simulations and whole-cell modeling, but the Martinize2 tool is certainly an important element in our ambitions to achieve this. Given the broad interest in these goals by the modeling community in general, we believe this work has a much wider impact beyond the (already large) group of Martini users. Addressing limitations of the Martini model itself, which are certainly there, is clearly not the scope of the current work.

Reviewer #3 (Public Review):

The manuscript Kroon et al. described two algorithms, which when combined achieve high throughput automation of "martinizing" protein structures with selected protonation states and post-translational modifications. After the revisions provided by the authors, I recommend minor revision.

The authors have addressed most of my concerns provided previously. Specifically, showcasing the capability of coarse-graining other types of molecules (Figure 7) is a useful addition, especially for the booming field of therapeutic macrocycles. My only additional concern is that to justify Martinize2 and Vermouth as a "high-throughput" method, the speed of these tools needs to be addressed in some form in the manuscript as a guideline to users.

We have added some benchmark timings in the manuscript SI and pointed to the data in the discussion part, which addresses the timing. Martinize2 is certainly slower than martinize version 1 as we already pointed out in the previous versions. However, even for larger proteins (> 2000 residues) we are able to generate topologies in about 60s. As Martinize2 runs on a single core, it can be massively parallelized. Keeping this in mind the topology file generation is likely to take up only a fraction in a high-throughput pipeline compared to the more costly simulations themselves.

<https://doi.org/10.7554/eLife.90627.3.sa0>