JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim

Application Note

Creating Coarse-Grained Systems with COBY: Toward Higher Accuracy of Complex Biological Systems

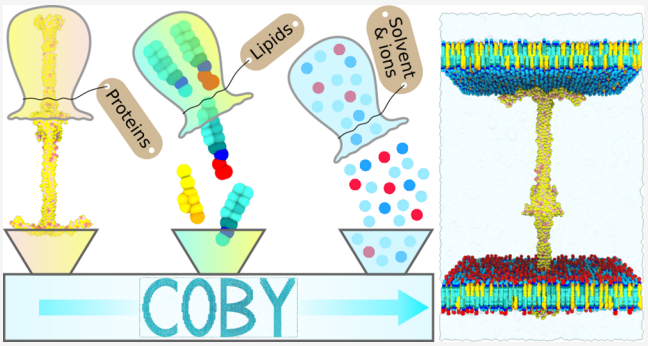
Mikkel D. Andreasen, Paulo C. T. Souza, Birgit Schiøtt, and Lorena Zuzic*

Cite This: J. Chem. Inf. Model. 2025, 65, 4760−4766

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Current trends in molecular modeling are geared toward increasingly realistic representations of the biological environments reflected in larger, more complex systems. The complexity of the system-building procedure is ideally handled by software that converts user-provided descriptors into system coordinates. This, however, is not a trivial task, as building algorithms use simplifications that result in inaccuracies in the system properties. We created COBY, a coarse-grained system builder that can create a large variety of systems in a single command call with an improved accuracy of the complex membrane and solvent building procedures. COBY also contains features for building diverse systems in a single step, and has functionalities aiding force field development. COBY is an open-source software written in Python 3, and the code, documentation, and tutorials are hosted at https://github.com/MikkelDA/COBY.



INTRODUCTION

Dynamics and function within a biological system are inextricably linked. Molecular dynamics (MD) simulations are a computational tool used to investigate motions within a macromolecular system on microsecond time-scales and in atomistic or near atomistic resolution. By using experiment-informed structures paired with force fields defined by realistic interatomic potentials, simulations complement the experimental method and allow for a detailed view of an individual system in motion.[1]

Continuously increasing computing resources, paired with the advancements in integrative biology, makes it possible to model systems of ever greater complexity. One particular area of interest is complex membranes, as membrane composition is linked to a plethora of biological functions, such as protein regulation through lipids,[2−8] modulated protein kinetics,[9] protein localization,[10] and biophysical properties of the membrane.[11] However, computational costs of achieving adequate sampling increase dramatically with system complexity. Simulating the complex membranes in a coarse-grained (CG) resolution, where multiple atoms are grouped together to form a bead, allows for longer sampling of bigger systems and ultimately a wider exploration of the conformational landscape[12,13]—a feat that is not easily achievable in atomistic simulations.

To run the MD simulations, one needs to start with system building, which is a user-involved process and can quickly become laborious in higher-complexity systems. The systems are therefore often built with the help of a software that automates the building process. It follows that system accuracy is directly dependent on the accuracy of the employed code.[1]

Membrane building in particular is reliant on the available software.[13] The procedure is nontrivial, as the membrane structure is not directly sourced from a given set of experimentally determined atom coordinates (as is the case with proteins); instead, the coordinates need to be built "from scratch", based only on few given parameters: membrane patch size, lipid types and their ratios, and the assigned area per lipid (APL). Ideally, all of the given parameters should exact to the given value in a built membrane; however, this is often not the case. Fundamentally, a membrane built for the purpose of MD simulations can only accept a whole number of lipids, which in consequence requires nontrivial consideration of the given parameters, as rounding can lead to value drift. This problem is exacerbated in complex membranes, where each lipid ratio (both within and between the leaflets) needs to be satisfied using a finite, integer number of lipids. If not considered, the resulting ratios will be offset from the requested values and produce an inaccurate model of a complex membrane.

Received: January 13, 2025
Revised: March 28, 2025
Accepted: April 15, 2025
Published: May 12, 2025

There exists a number of tools that can be used to automatize building of biomolecular systems (e.g., CharmmGUI,[14−16] TS2CG,[17] *insane*,[18] PACKMOL,[19] polyply[20]), and while each offers a bespoke set of functionalities, we identified the need for an increased accuracy in terms of lipid ratios in complex membrane systems, coupled with ease of use, speed, and customization ability.

Here we introduce a new system building software tool named COBY (Complex Biological System Builder), which is an open-source Python package for building Martini coarse-grained systems.[21] COBY was primarily designed to build flat complex membranes while accurately handling the given lipid ratios, and it has been broadened to handle the creation of systems without a membrane component. It was designed to be easy to use for a typical user, but also highly customizable in case of more specific system requirements. The software can be run both within a Python script or as a terminal command-line, and it uses a single command to specify all the system requirements.

The code, documentation, tutorials and license are available at github.com/MikkelDA/COBY.

THEORY

Code Organization. COBY is written in Python 3 and the commands can be used either within the Python script, or as a command line in the terminal window. The entire simulation system is specified in a single command call. The requested system elements are then built in a serial fashion (Figure S1), where individual elements are added in a predefined order.

Membrane and Protein Treatment. COBY treats leaflets as semi-independent components of the membrane and as such builds membranes on a leaflet-by-leaflet basis. If a protein is placed in the proximity of a leaflet, each protein bead is checked for potential overlap with the 3D leaflet space (Figure S2). By considering the bead overlap with the leaflets, rather than the whole membranes, COBY can better accommodate irregularities in protein shape. For large membranes, COBY performs dynamic segmentation of the leaflets for faster processing (Figure S3).

COBY contains tools for creating membrane patches and artificial membrane pores. These can be made either by using predefined shapes or by defining polygon points in the *xy*-dimension. It also features a specialized argument to create stacked membrane systems with customizable compositions of each individual membrane and solvent space.

Lipid Packing. Correct lipid packing is dependent on the user providing reasonable APL values for each leaflet. While APLs of homogeneous membranes can be based on the lipid type APL, complex asymmetric membranes benefit from a more careful APL prediction. We recommend building two symmetric membranes (each corresponding to the composition of one leaflet) and running short MD simulations until the APL values are reasonably equilibrated. These APLs can then be used as an input for building asymmetric complex membranes.

The lipid packing procedure starts with determining an absolute number of lipids in leaflets, followed by an initial lipid placement, and finally the optimization of lipid positions. The lipid occupancy area (A_{free}) is calculated by subtracting the area occupied by proteins or artificial pores from the total leaflet area. A_{free} is then used to calculate the highest allowed number of lipids ($N_{COBY,max}$) from the specified area per lipid (APL). The result is then rounded to the nearest integer:

$$N_{COBY,max} = \left\lfloor \frac{A_{free}}{APL} \right\rceil \quad (1)$$

$N_{COBY,max}$ is then allocated across the requested lipid types, taking their internal ratios into account, with the number of each lipid type being rounded down to the nearest integer. The resulting sum of lipids is considered the minimal allowed number of lipids ($N_{COBY,min}$) in the leaflet,

$$N_{COBY,min} = \sum_{i=1}^{N_{types}} \left\lfloor \frac{w_i}{\sum_{j=1}^{N_{types}} w_j} \cdot N_{COBY,max} \right\rfloor \quad (2)$$

where N_{types} is the number of lipid types, w is the interlipid ratio of a given lipid, and i and j are the indices of the lipid types. The default algorithm first adds $N_{COBY,min}$ lipids; then, $N_{COBY,max}$ is reached by iteratively adding lipids of the most underrepresented type relative to the requested ratio. Other optimization algorithms with different prioritisation schemes are also available in COBY.

In the lipid placement step, all lipids are represented as circles in the *xy* plane and with a diameter that corresponds to its lipid type. The lipid types are divided into groups based on size, with the larger lipids being semirandomly placed on a 2D-grid with spacing adjusted for lipid sizes. Then, the smaller lipids are placed on the *y*-directional lines within the nonoccupied area of the leaflet (Figure S2f-h). The number of lines and the corresponding number of lipids on the line are dynamically optimized until all requested lipids are placed in the leaflet.

Following the initial lipid placement, any overlaps between the circles are resolved by using a distance-based optimization algorithm (Figure S2h−i). The lipids are exerting a pushing "force" between each lipid pair within a cutoff distance, with the direction of the force informed by the vector between the two lipids, and the magnitude by the interlipid distance, optimization step (as the force is linearly scaled with each step), and the lipid size. (for details, see SI). The pushing is also applied from the edges of the membrane patch. Optimisation is completed when the maximum force on any lipid is smaller than a set tolerance value, or when the maximum number of steps is reached.

Solvation. Solvation steps are independent of the solvent molecule of choice, and COBY can use any molecule as a solvent, as long as its structure and the molarity information are provided.

Solvation and flooding steps are performed in a 3D box space, where the box is rasterised, with the grid resolution determined from the length of the largest solvent molecule. The grid points that overlap with other system beads or the hydrophobic volume of a membrane are marked as unavailable for solvent insertion. Based on a given concentration, the total number of solvent particles is calculated using only the solvent-occupied volume.

Solvent neutralization is carried out in two steps, first by adding the desired concentration of ions, after which the solvation box is neutralized. COBY offers three neutralization algorithms, where (i) counterions are added, (ii) co-ions are removed to reach neutralization, or where (iii) the addition of counterions and removal of co-ions is combined. In mixed solvent cases, the intersolvent ratios can be interpreted either

https://doi.org/10.1021/acs.jcim.5c00069
J. Chem. Inf. Model. 2025, 65, 4760−4766

as ratios based on the CG bead numbers, or as ratios that take into consideration the underlying atom-to-bead mapping.

Topology Handling. COBY reads topology files and uses them to obtain charge information for imported molecules. Topologies are unambiguously linked to the imported molecules by using the names specified in the [moleculetype] segment of the GROMACS topology file. The #include statements within the topology file are recursively processed, and the output topology file is updated with the specifications of the built systems and can directly be used in the simulation preprocessing steps in GROMACS.

Parameter Libraries, Molecule Building, and Import. Molecular structures are organized in parameter libraries. Multiple libraries of the same type can coexist, and can be used to differentiate between multiple implementations of the same molecule. The library contents can be accessed interactively using the COBY.Library command. This feature is particularly interesting in a development context, where molecule parameters can be sourced from different development libraries.

Inspired by *insane*,[18] we designed COBY so it can use internal fragment libraries to build the structures of molecules of interest (e.g., lipids with specific head, linker, and tail combinations). An existing set of fragments is provided as an in-built library, but the fragments can also be imported. The molecules are dynamically built by joining the building blocks at the specific attachment points. Depending on the type, lipids can contain a custom number of tails, which allows the user to import their own complex lipid components and use them to create custom lipids.

Molecules can be imported or created directly in the system-building command, or in a separate COBY.Crafter command call, which only outputs the structure of a requested molecule. By having an option to skip the system-building step, COBY can be used to straightforwardly provide structure files of the desired lipids, which is a routine requirement in the force field development process.

METHODS

System Showcase. Eleven systems have been created to showcase the flexibility and versatility of COBY. The code used to create the systems can be found in the Github repository under the Tutorial section (https://github.com/MikkelDA/COBY), and the system details are listed in Table S1.

CODE ACCURACY COMPARISON

Total Number of Lipids. We compared the accuracy of COBY and *insane* algorithms of reproducing the N_{ideal} by monitoring the deviation from the ideal of the number of lipids within a leaflet (Figure 1a) or the lipid ratios between the leaflets (Figure 1b) as a function of membrane size. The input for both programs was the membrane size in *xy* dimension and APL.

$$N_{ideal} = \frac{A_{free}}{APL} \tag{3}$$

The constraints of the build—namely, that (i) the system is built with an integer number of lipids (ii) placed within a user-defined leaflet area and (iii) with consideration of a given APL—necessitate a compromise between the given properties so that the resulting deviation ($N_{software}/N_{ideal}$) is minimal.

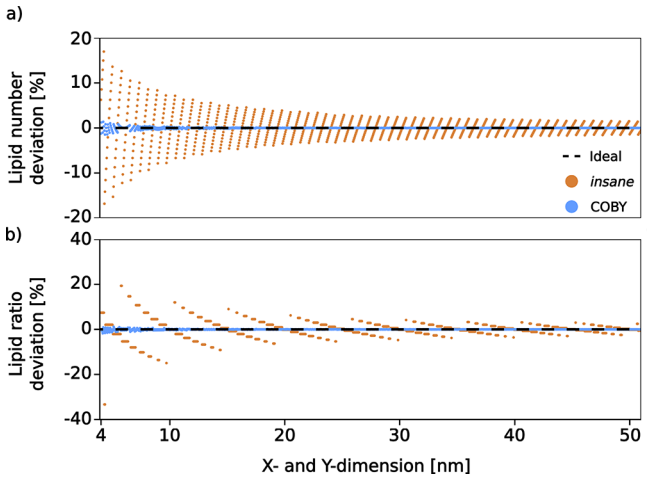


Figure 1. Comparison of accuracy in the membrane building step. The deviation of the number of lipids from the ideal values for COBY (in blue) and *insane* (in orange). Ideal deviation (always zero) is shown as a black dashed line. (a) The deviation of the number of lipids per leaflet compared to an ideal number ($N_{software}/N_{ideal}$) as a function of membrane size. (b) The deviation of interleaflet lipid ratios in asymmetric membranes ($N_{upper,software}$:$N_{lower,software}$/$N_{upper,ideal}$:$N_{lower,ideal}$) as a function of membrane size.

The number of lipids inserted by *insane* in a leaflet (N_{insane}) relies on two rounding operations and is calculated by

$$N_{insane} = \left\lfloor \frac{x}{\sqrt{APL}} \right\rfloor \cdot \left\lfloor \frac{y}{\sqrt{APL}} \right\rfloor \tag{4}$$

where *x* and *y* are the side lengths of a given membrane. The number of lipids inserted by COBY ($N_{COBY,max}$) is calculated by eq 1.

The comparison of software accuracy based on the deviation from the requested lipid ratios was based on concurrent command calls to COBY and *insane* with an increasing box size (between 4 and 51 nm and by 0.05 nm increments) and with both leaflets assigned APLs of 0.6 nm^2. We also assessed the deviation of interleaflet lipid ratios relevant for the asymmetric membranes (APL_{upper} = 0.60 nm^2; APL_{lower} = 0.45 nm^2), expressed as lipid ratio deviation ($N_{upper,software}$: $N_{lower,software}$/$N_{upper,ideal}$: $N_{lower,ideal}$).

Speed Tests. Speed of COBY and *insane* was assessed by calculating the mean time to create a system over 5 attempts and over three system series (membrane-only, solvent-only, and with both membrane and solvent). The membranes were assigned POPC lipids (APL = 0.6 nm^2), and the aqueous solvent contained 0.15 M NaCl. The *z* box dimensions were kept constant at 10 nm, while the *x* and *y* box size lengths were serially increased from 4 to 51 nm.

Molecular Dynamics Simulations. All shown examples (Figure 2) have been minimized, equilibrated and simulated for a short period to confirm that the systems perform well in simulations without exhibiting any major instabilities. The details of the simulation procedure are available in SI.

RESULTS

Accuracy of Membrane Builds. The constraints of the membrane building procedure—the need to accommodate APL, lipid ratios, and a membrane patch size with an integer number of lipids—inherently lead to an error within one or more of the requested values describing the membrane

https://doi.org/10.1021/acs.jcim.5c00069
J. Chem. Inf. Model. 2025, 65, 4760−4766

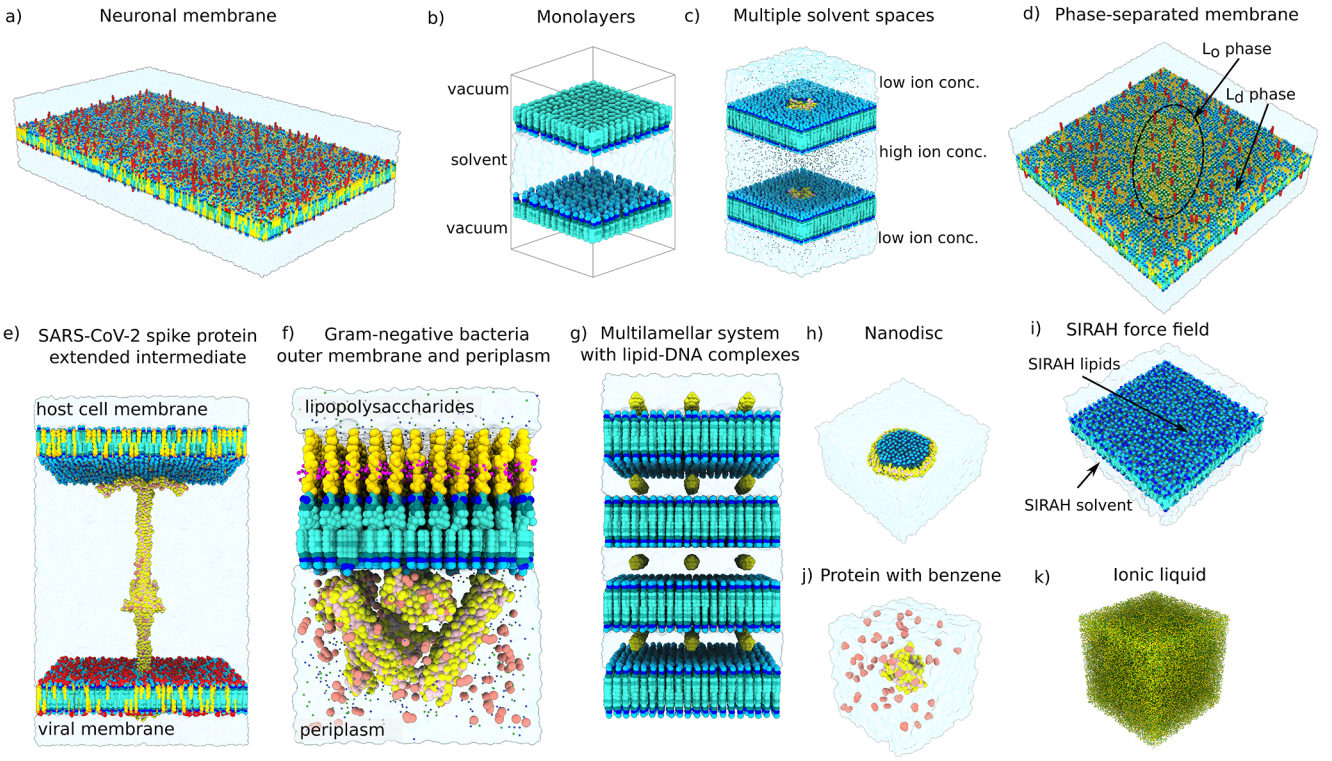


Figure 2. Showcase of systems built with COBY and using a single command-line call. (a) Complex asymmetric neuronal membrane consisting of 58 different lipid types using Martini 2 mapping scheme. Recreated from Ingólfsson et al.[22] (b) Monolayer simulation setup. (c) Voltage-gated potassium channels (PDB: 3F5W)[23] with modeled ion gradients contributing to the resting membrane potential.[24] (d) A phase-separated giant unilamellar vesicle membrane with an ellipsoid liquid-ordered (L_o) phase patch and a surrounding liquid-disordered (L_d) membrane. Based on Hammond et al.[25] (e) Extended intermediate of a SARS-CoV-2 spike protein[26] spanning the viral[27] and host cell membrane.[22] (f) Complex outer membrane and periplasm of Gram-negative bacteria containing an asymmetric bilayer, four types of periplasmic proteins, and four types of solutes: spermidine, putrescine, glycerol and urea. Based on Pedebos et al.[28] (g) Multilamellar lipid−DNA complexes used as artificial transfection agents, comprised of four vertically stacked membranes with dsDNA molecules occupying each intermembrane space.[29] (h) A nanodisc in solvent.[30] (i) Implementation of SIRAH molecules in COBY by importing solvent, ions, and lipids from the SIRAH force field.[31] (j) A benzene flooding setup surrounding a human K-Ras protein (PDB: 4OBE).[32] (k) An ionic liquid containing four types of cations and a corresponding number of tetrafluoroborate anions.

properties. COBY was designed to minimize the build errors with a protocol described in the Theory section.

We examined the effect of membrane patch size on the deviation from the ideal number of inserted lipids, as dictated by the leaflet APL. We compared the performance of COBY with a commonly used CG membrane building software, *insane*. The deviation from the ideal number of lipids was appreciably higher for the membranes built by *insane* than it was for the membranes generated in COBY (Figure 1a). The deviation for *insane* membranes was especially large in small lipid patches and it tangentially reduced with the membrane size, but remained above 2% even in the patches that are larger than 50 nm × 50 nm. On the other hand, the deviation from the ideal lipid number in the membranes built by COBY dropped below 1% for the 6 × 6 nm^2 membrane patch and remained under that value for all larger membranes.

We also assessed how the two algorithms compare in terms of preserving lipid ratios between the two leaflets in cases where the requested leaflet APLs are not the same (Figure 1b). While COBY effectively minimized the errors with only slight deviations present for the smaller membrane patches, *insane* deviations were appreciable even in larger membranes.

These differences in accuracy are due to the architecture of the underlying algorithms. COBY's algorithm ensures that the number of lipids is as close to N_{ideal} as possible, and the variations are mainly confined within the decimal values. As the number of inserted lipids is an integer, these deviations are only observable in very small membrane patches. *insane* is comparatively more sensitive to the membrane size effects, as it assigns the number of lipids within a strictly defined 2D-grid informed by the *x*- and *y*-axis patch dimensions, and with an increasing patch size, adds whole rows and columns of lipids at one time, producing larger deviations from the ideal value.

An additional handling error specific to *insane* appears when handling intraleaflet lipid ratios, as the integer number of lipids is achieved by using a rounding down operation. This artifact is not present in COBY.

COBY is inferior to *insane* when it comes to speed (Figure S4). In particular, solvent insertion in COBY is slower than in *insane*, primarily because COBY takes into consideration free volume and solvent molarity in order to calculate the number of solvent molecules. In comparison, *insane* creates a simple 3D grid with spacing that roughly corresponds to the density of regular water beads. The advantage of COBY's approach is that solvation is not restricted to water—instead, any solvent molecule can be used to solvate the system, provided solvent-specific molarity. We deemed this trade-off between accuracy and versatility vs speed a necessity, particularly when considering that even under these constrains, the creation of

https://doi.org/10.1021/acs.jcim.5c00069
J. Chem. Inf. Model. 2025, 65, 4760−4766

the larger systems (Figure 2a,d,e,f) took under a minute on an office workstation.

Versatility of the Built Systems. The increase in system complexity has been a trend in the modeling community as a way to better reflect the native biological environment.[33] Commonly, the system-building process is being carried out by nonstandardized procedures based on manual "system-stitching". We structured COBY in a way that all the elements of the system can be specified in a single command call, meaning that the program can optimize the building procedure with regards to the requested components. COBY uniquely offers considerations of molarity when using nonaqueous solvents, functionalities to build stacked membranes, phase-separated membranes, and multiple solvent spaces, all of which can be handled in combination within the same command call. These, in effect, can produce a great variety of reproducible high-complexity systems in a single step (Figure 2). Following is a brief overview of the diverse COBY functions that were used to create the models; for a more comprehensive overview of the systems' biological relevance, code syntax, relevant considerations and limitations, the reader is referred to SI.

COBY can create of a broad spectrum of flat membrane-based systems: bilayers (Figure 2a,d,e) and monolayers (Figure 2b) of any given complexity, phase-separated membranes (Figure 2d), stacked membranes (Figure 2c,g), or membrane patches and pores of various shapes (Figure 2d,h).

Besides the membrane, COBY also handles placement of other system elements: proteins, solutes, and solvents. The system built by COBY can contain any number of proteins (Figure 2c,e,f) that can either be membrane-inserted or soluble (Figure 2j). Solvation can be controlled by the user to be either complete or partial (Figure 2b,c), and the solvents do not necessarily need to be aqueous, but instead use any other user-defined solvents (Figure 2k). Solute-flooding can also be performed in COBY (Figure 2f,j).

The package includes developer-friendly features, such as easy handling of multiple parameter libraries, importing lipid structures (Figure 2f,i) that can be used in the membrane building step, building molecules from fragments (Figure 2f), and importing solute molecules for the purposes of flooding (Figure 2f,j). Finally, while COBY defaults to "best practice" calculation methods in order to build the desired system, it also offers a set of alternative algorithms that can be used instead (i.e., interleaflet lipid optimization options, multiple approaches to system neutralization and salting, and handling of mixed solvent ratios).

DISCUSSION

We created COBY, an open-source software for building CG systems with a primary focus on the accuracy of the building procedure, followed by the versatility of the produced output. The code has been designed to create systems with Martini CG models; however, the underlying code infrastructure can also be expanded to include other CG models (e.g., SIRAH,[31] as shown in Figure 2i).

COBY demonstrates a greatly improved accuracy in system building, employing error minimization algorithms, which produce systems that reflect the properties requested by the user. Otherwise, a system-building software could create errors that are difficult to predict, as the deviations from the ideal are neither well documented nor linear. This also affects the replicability of simulation studies, as unassuming changes in the starting conditions (e.g., simulation box size) can lead to unexpected errors in the built systems (incorrect APL values of the membrane). The improvements implemented in COBY allow for an accurate creation of multicomponent systems, and alleviate the burden on the user to check, detect, and correct the errors generated by the software.

We designed COBY with complex systems in mind and, as such, the software is able to handle a great variety of user requests within a single command line. This is crucial in high-throughput protocols that use MD simulations, as they rely on automated procedures to execute the entire pipeline,[34] and should ideally not be restricted by the limited complexity of the available system-building methods. The automated procedure also provides a straightforward approach to reliably replicate the simulation experiment. Note that the package is best suited to build the systems in 100 nm-size range, as bigger system sizes exponentially increase the computation time (Figure S4) or have high memory requirements, especially concerning the solvent creation step. As COBY is designed to only handle flat membranes, the curved membrane systems could be created with the TS2CG software instead.[17]

Force field development is reliant upon extensive testing of the new parameters in a streamlined manner. COBY parameter libraries are organized in a way that makes it possible to use multiple libraries within the same command call, allowing the user to mix and match the parameters of choice. Users can also create new libraries either by directly importing them, or by importing individual molecules from structure files and optional topology files. In addition, a big portion of lipids can be built in COBY from fragments available from the in-built libraries.

COBY is a tool that is meant to have a broad user base, as the simple systems can be built with a very few mandatory arguments—namely, box determining the size of the system, and at least one other element (e.g., membrane, solvent, protein, or solute). At the same time, it offers a great deal of flexibility to the experienced user wishing to employ it for complex system building or systematic parametrization procedures. Possible implementation of COBY within a graphical user interface environment, such as MAD,[35] could further simplify the system-building procedure for many users.

ASSOCIATED CONTENT

Data Availability Statement

The open-source code, basic and advanced tutorials, code for creating showcased systems, cheat sheet, and detailed documentation are hosted on the project page at https://github.com/MikkelDA/COBY.

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jcim.5c00069.

Figures illustrating COBY workflow, membrane creation process, membrane segmentation procedure, speed test comparison, a table detailing the showcase systems (for Figure 2), a description of the pushing force used in the optimization algorithm, the MD simulation protocol, and additional information for showcased systems (PDF)

https://doi.org/10.1021/acs.jcim.5c00069
J. Chem. Inf. Model. 2025, 65, 4760−4766

AUTHOR INFORMATION

Corresponding Author

Lorena Zuzic − *Department of Chemistry, Aarhus University, 8000 Aarhus C, Denmark;* orcid.org/0000-0002-7834-612X; Email: lorena.zuzic@chem.au.dk

Authors

Mikkel D. Andreasen − *Department of Chemistry, Aarhus University, 8000 Aarhus C, Denmark*

Paulo C. T. Souza − *Laboratoire de Biologie et Modélisation de la Cellule, CNRS, UMR 5239, Inserm, U1293, Universite Claude Bernard Lyon 1, Ecole Normale Supérieure de Lyon, 69364 Lyon, France; Centre Blaise Pascal de Simulation et de Modélisation Numérique, Ecole Normale Supérieure de Lyon, 69364 Lyon, France;* orcid.org/0000-0003-0660-1301

Birgit Schiøtt − *Department of Chemistry, Aarhus University, 8000 Aarhus C, Denmark;* orcid.org/0000-0001-9937-1562

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jcim.5c00069

Author Contributions

M.D.A.: Conceptualization, code writing and maintenance, test simulations and system building; manuscript—original draft and editing, user guides—original draft and editing. P.C.T.S.: Supervision support, manuscript—review and editing. B.S.: Funding acquisition; supervision, review and feedback on manuscript. L.Z.: Conceptualization, supervision, project administration, manuscript—original draft and editing, user guides—original draft and editing.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We thank Lisbeth R. Kjølbye for helpful discussions regarding desired software functionalities, Anna L. Duncan for continuous support and advice, Kasper B. Pedersen for encouragement and useful advice concerning lipid treament, Luís Borges-Araújo and Fabian Grünewald for astute discussion points in terms of desired developer features and software naming, Guillaume Launay for helpful advice towards improving code organisation, Helgi I. Ingólfsson for providing the original lipid dataset for creating a complex neuronal membrane, and Syma Khalid for providing the bacterial periplasm system descriptors. M.D.A. is supported by the funding grant from the AUFF NOVA (AUFF-E-2020-9-1 0). L.Z. is supported by grants from the Lundbeck Foundation (R346-2020-1944) and Novo Nordisk Foundation (NNF18OC0032608 and NF20OC0065431). The simulation computational resources were provided by the Grendel cluster of the Centre for Scientific Computing Aarhus and the Resource for Biomolecular Simulations supported by the Novo Nordisk Foundation (NNF18OC0032608 and NNF24OC0087976). P.C.T.S is thankful for the support of the French National Center for Scientific Research (CNRS) and the funding from research collaboration agreements with PharmCADD and Sanofi. P.C.T.S. also acknowledges the support of the Centre Blaise Pascal's IT test platform at ENS de Lyon (Lyon, France) for the computer facilities. The platform operates the SIDUS solution developed by Emmanuel Quemener.[36]

REFERENCES

(1) Hollingsworth, S. A.; Dror, R. O. Molecular Dynamics Simulation for All. *Neuron* **2018**, *99*, 1129−1143.

(2) Kervin, T. A.; Overduin, M. Regulation of the Phosphoinositide Code by Phosphorylation of Membrane Readers. *Cells* **2021**, *10*, 1205.

(3) Niggli, V. Regulation of protein activities by phosphoinositide phosphates. *Annu. Rev. Cell Dev. Biol.* **2005**, *21*, 57−79.

(4) Borges-Araújo, L.; Souza, P. C. T.; Fernandes, F.; Melo, M. N. Improved Parameterization of Phosphatidylinositide Lipid Headgroups for the Martini 3 Coarse-Grain Force Field. *J. Chem. Theory Comput.* **2022**, *18*, 357−373.

(5) Medoh, U.; Abu-Remaileh, M. The Bis(monoacylglycero)-phosphate Hypothesis: From Lysosomal Function to Therapeutic Avenues. *Annu. Rev. Biochem.* **2024**, *93*, 447−469.

(6) Duncan, A. L.; Corey, R. A.; Sansom, M. S. Defining how multiple lipid species interact with inward rectifier potassium (Kir2) channels. *Proc. Natl. Sci. U. S. A.* **2020**, *117*, 7803−7813.

(7) Kalinichenko, L. S.; Kornhuber, J.; Sinning, S.; Haase, J.; Müller, C. P. Serotonin Signaling through Lipid Membranes. *ACS Chem. Neurosci.* **2024**, *15*, 1298−1320.

(8) Drew, D.; Boudker, O. Ion and lipid orchestration of secondary active transport. *Nature* **2024**, *626*, 963−974.

(9) Nilsson, T.; Lundin, C. R.; Nordlund, G.; Ädelroth, P.; von Ballmoos, C.; Brzezinski, P. Lipid-mediated protein-protein interactions modulate respiration-driven ATP synthesis. *Sci. Rep.* **2016**, *6*, 24113.

(10) Jiang, Y.; Thienpont, B.; Sapuru, V.; Hite, R. K.; Dittman, J. S.; Sturgis, J. N.; Scheuring, S. Membrane-mediated protein interactions drive membrane protein organization. *Nat. Commun.* **2022**, *13*, 7373.

(11) Levental, I.; Levental, K. R.; Heberle, F. A. Lipid Rafts: Controversies Resolved, Mysteries Remain. *Trends Cell Biol.* **2020**, *30*, 341−353.

(12) Corradi, V.; Sejdiu, B. I.; Mesa-Galloso, H.; Abdizadeh, H.; Noskov, S. Y.; Marrink, S. J.; Tieleman, D. P. Emerging Diversity in Lipid−Protein Interactions. *Chem. Rev.* **2019**, *119*, 5775−5848.

(13) Marrink, S. J.; Corradi, V.; Souza, P. C. T.; Ingólfsson, H. I.; Tieleman, D. P.; Sansom, M. S. P. Computational modeling of realistic cell membranes. *Chem. Rev.* **2019**, *119*, 6184−6226.

(14) Jo, S.; Kim, T.; Iyer, V. G.; Im, W. CHARMM-GUI: A web-based graphical user interface for CHARMM. *J. Comput. Chem.* **2008**, *29*, 1859−1865.

(15) Qi, Y.; Ingólfsson, H. I.; Cheng, X.; Lee, J.; Marrink, S. J.; Im, W. CHARMM-GUI Martini Maker for Coarse-Grained Simulations with the Martini Force Field. *J. Chem. Theory Comput.* **2015**, *11*, 4486−4494.

(16) Hsu, P.-C.; Bruininks, B. M. H.; Jefferies, D.; Souza, P. C. T.; Lee, J.; Patel, D. S.; Marrink, S. J.; Qi, Y.; Khalid, S.; Im, W. CHARMM-GUI Martini Maker for modeling and simulation of complex bacterial membranes with lipopolysaccharides. *J. Comput. Chem.* **2017**, *38*, 2354−2363.

(17) Pezeshkian, W.; König, M.; Wassenaar, T. A.; Marrink, S. J. Backmapping triangulated surfaces to coarse-grained membrane models. *Nat. Commun.* **2020**, *11*, 2296.

(18) Wassenaar, T. A.; Ingólfsson, H. I.; Böckmann, R. A.; Tieleman, D. P.; Marrink, S. J. Computational Lipidomics with insane: A Versatile Tool for Generating Custom Membranes for Molecular Simulations. *J. Chem. Theory Comput.* **2015**, *11*, 2144−2155.

(19) Martínez, L.; Andrade, R.; Birgin, E. G.; Martínez, J. M. PACKMOL: A package for building initial configurations for molecular dynamics simulations. *J. Comput. Chem.* **2009**, *30*, 2157−2164.

(20) Grünewald, F.; Alessandri, R.; Kroon, P. C.; Monticelli, L.; Souza, P. C. T.; Marrink, S. J. Polyply; a python suite for facilitating simulations of macromolecules and nanomaterials. *Nat. Commun.* **2022**, *13*, 68.

(21) Marrink, S. J.; Monticelli, L.; Melo, M. N.; Alessandri, R.; Tieleman, D. P.; Souza, P. C. T. Two decades of Martini: Better beads, broader scope. *WIREs Comput. Mol. Sci.* **2023**, *13*, No. e1620.

(22) Ingólfsson, H. I.; Bhatia, H.; Zeppelin, T.; Bennett, W. F. D.; Carpenter, K. A.; Hsu, P.-C.; Dharuman, G.; Bremer, P.-T.; Schiøtt, B.; Lightstone, F. C.; Carpenter, T. S. Capturing Biologically Complex Tissue-Specific Membranes at Different Levels of Compositional Complexity. *J. Phys. Chem. B* **2020**, *124*, 7819−7829.
(23) Cuello, L. G.; Jogini, V.; Cortes, D. M.; Perozo, E. Structural mechanism of C-type inactivation in K+ channels. *Nature* **2010**, *466*, 203−208.
(24) Kutzner, C.; Köpfer, D. A.; Machtens, J.-P.; de Groot, B. L.; Song, C.; Zachariae, U. Insights into the function of ion channels by computational electrophysiology simulations. *Biochim. Biophys. Acta* **2016**, *1858*, 1741−1752.
(25) Hammond, A. T.; Heberle, F. A.; Baumgart, T.; Holowka, D.; Baird, B.; Feigenson, G. W. Crosslinking a lipid raft component triggers liquid ordered-liquid disordered phase separation in model plasma membranes. *Proc. Natl. Acad. Sci. U. S. A.* **2005**, *102*, 6320−6325.
(26) Su, R.; Zeng, J.; Marcink, T. C.; Porotto, M.; Moscona, A.; O'Shaughnessy, B. Host Cell Membrane Capture by the SARS-CoV-2 Spike Protein Fusion Intermediate. *ACS Cent. Sci.* **2023**, *9*, 1213−1228.
(27) Saud, Z.; et al. The SARS-CoV2 envelope differs from host cells, exposes procoagulant lipids, and is disrupted in vivo by oral rinses. *J. Lipid Res.* **2022**, *63*, 100208.
(28) Pedebos, C.; Smith, I. P. S.; Boags, A.; Khalid, S. The hitchhiker's guide to the periplasm: Unexpected molecular interactions of polymyxin B1 in E. coli. *Structure* **2021**, *29*, 444−456.
(29) Corsi, J.; Hawtin, R. W.; Ces, O.; Attard, G. S.; Khalid, S. DNA Lipoplexes: Formation of the Inverse Hexagonal Phase Observed by Coarse-Grained Molecular Dynamics Simulation. *Langmuir* **2010**, *26*, 12119−12125.
(30) Marcink, T. C.; Simoncic, J. A.; An, B.; Knapinska, A. M.; Fulcher, Y. G.; Akkaladevi, N.; Fields, G. B.; Van Doren, S. R. MT1-MMP Binds Membranes by Opposite Tips of Its β Propeller to Position It for Pericellular Proteolysis. *Structure* **2019**, *27*, 281−292.
(31) Klein, F.; Soñora, M.; Santos, L. H.; Frigini, E. N.; Ballesteros-Casallas, A.; Machado, M. R.; Pantano, S. The SIRAH force field: A suite for simulations of complex biological systems at the coarse-grained and multiscale levels. *J. Struct. Biol.* **2023**, *215*, 107985.
(32) Hunter, J. C.; Gurbani, D.; Ficarro, S. B.; Carrasco, M. A.; Lim, S. M.; Choi, H. G.; Xie, T.; Marto, J. A.; Chen, Z.; Gray, N. S.; Westover, K. D. In situ selectivity profiling and crystal structure of SML-8−73−1, an active site inhibitor of oncogenic K-Ras G12C. *Proc. Natl. Acad. Sci. U. S. A.* **2014**, *111*, 8895−8900.
(33) Gupta, C.; Sarkar, D.; Tieleman, D. P.; Singharoy, A. The ugly, bad, and good stories of large-scale biomolecular simulations. *Curr. Opin. Struct. Biol.* **2022**, *73*, 102338.
(34) Wassenaar, T. A.; Pluhackova, K.; Moussatova, A.; Sengupta, D.; Marrink, S. J.; Tieleman, D. P.; Böckmann, R. A. High-throughput simulations of dimer and trimer assembly of membrane proteins. The DAFT approach. *J. Chem. Theory. Comput.* **2015**, *11*, 2278−2291.
(35) Hilpert, C.; Beranger, L.; Souza, P. C. T.; Vainikka, P. A.; Nieto, V.; Marrink, S. J.; Monticelli, L.; Launay, G. Facilitating CG Simulations with MAD: The MArtini Database Server. *J. Chem. Inf. Model.* **2023**, *63*, 702−710.
(36) Quemener, E.; Corvellec, M. SIDUS—the solution for extreme deduplication of an operating system. *Linux J.* **2013**, *2013*, 101−110.

https://doi.org/10.1021/acs.jcim.5c00069
J. Chem. Inf. Model. 2025, 65, 4760−4766