Fullrmc, a Rigid Body Reverse Monte Carlo Modeling Package Enabled with Machine Learning and Artificial Intelligence

Bachir Aoun*

A new Reverse Monte Carlo (RMC) package "fullrmc" for atomic or rigid body and molecular, amorphous, or crystalline materials is presented. fullrmc main purpose is to provide a fully modular, fast and flexible software, thoroughly documented, complex molecules enabled, written in a modern programming language (python, cython, C and C++ when performance is needed) and complying to modern programming practices. fullrmc approach in solving an atomic or molecular structure is different from existing RMC algorithms and software. In a nutshell, traditional RMC methods and software randomly adjust atom positions until the whole system has the greatest consistency with a set of experimental data. In contrast, fullrmc applies smart moves endorsed with reinforcement machine learning to groups of atoms. While fullrmc allows running traditional RMC modeling, the uniqueness of this approach resides in its ability to customize grouping atoms in any convenient way with no additional programming efforts and to apply smart and more physically meaningful moves to the defined groups of atoms. In addition, fullrmc provides a unique way with almost no additional computational cost to recur a group's selection, allowing the system to go out of local minimas by refining a group's position or exploring through and beyond not allowed positions and energy barriers the unrestricted three dimensional space around a group. © 2016 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.24304

Introduction

RMC as a modeling technique, based on the standard Monte Carlo simulation method with Markov chain sampling,[1] is a variation of the standard Metropolis Monte Carlo (MMC) method.[2] A molecular model designated by a set of three dimensional atomic coordinates is refined by randomly moving its atom positions until they are consistent and agree within a certain total error with a set of experimentally measured observables. Inherently, RMC molecular systems solved structures suffer from uniqueness and therefore a RMC solution cannot be considered as correct per say.[3] However, RMC models are very useful for understanding static structural relationships devoid of any dynamical properties of the studied materials. Traditional RMC does not require interatomic and intra-molecular potentials, however physical constraints such as a system's density, interatomic distances cutoff, etc. are used in order to generate plausible solutions. In general, structural quantities such as Pair Distribution Function (PDF), Structure Factor (SF), etc. are commonly used in materials modeling and unraveling correlations from low to high ranges. Originally, RMC modeling was developed to derive structural models for atomic liquids and glasses based on experimental data.

In early 1990's, RMC modeling method was doubted and criticized shortly after it was published simply because one can always argue whether the produced model is correct or not. While the model uniqueness assumption will always hold true, it is worth to note the following. Experimental data are never complete or correct, but they are convoluted, distorted by nature and contain a non-negligible signal to noise ratio. Therefore, making a so called "unique" model from experimental data or deriving and solving the structure using any modeling technique does not mean that the solution is correct. In this respect RMC, modeling is not different. Besides, it is impossible to know the true structure and the exact positions of all atoms of any material simply because atomic structures and positions are not static and there is a difference between time average and exact atomic positions. In this regard, RMC modeling is advantageous and allows one to explore different plausible results by imposing different constraints and different initial configurations. Nowadays, RMC modeling techniques are commonly used because of their great success in producing three-dimensional structural models of ordered and disordered materials. Besides RMC are very useful for their importance in the understanding of some particular physical properties and to speculate on further experiments and likely structures and other relevant insights into materials. RMC modeling is becoming more popular as a complementary technique to other computational and analytical methods such as Monte Carlo or Molecular Dynamics. While other modeling methods may provide more reliable results in some particular cases, the main strength of RMC modeling resides in its general application to a wide variety of types of structures in numerous research areas such as in liquids and solids, crystalline materials,[4] disordered crystals where long-range average structure is very often different from short-range one[5] and

Bachir Aoun
Argonne National Laboratories - Joint Center for Energy Storage Research, 9700 South Cass Ave B109, Lemont, Illinois
E-mail: baoun@aps.anl.gov

disordered structures.[6] Also, RMC is not limited to modeling PDF data inputs, Gurman and McGreevy described the application of RMC technique to the analysis of EXAFS data[7] using no input information other than the density and the chemical composition of the sample. Keen and McGreevy[8] also described how RMC modeling may be used to analyze neutron diffraction data from a disordered magnetic material. They determined both atomic structure and magnetic structure and reproduced the diffraction data of D_yNi_3 binary alloy.

Not many implementations of the RMC method are publicly available. RMC++[9,10] written in native C++, is among the known implementations of RMC method. It is designed to specifically study liquids and amorphous materials using constraints and experimental data such as PDF, total scattering and EXAFS. RMCProfile[11,12] is another implementation of the original RMC code mainly written in Fortran 95. RMCProfile is mostly known for its capability of modeling crystalline materials by explicitly using the information contained within the Bragg diffraction data. Besides, RMCProfile is adept to modeling magnetic materials, using the magnetic component of total scattering data. HRMC,[13,14] which stands for Hybrid Reverse Monte Carlo (RMC), is a combination of traditional RMC and energy minimization MMC code written in Fortran 90. HRMC is capable of fitting simultaneously electron, X-ray and neutron diffraction and EXAFS experimental data and porosity information while minimizing the system's empirical interatomic energy potential.

As a reverse modeling method, the traditional RMC algorithm evolves by repeatedly sampling and modifying the three dimensional atomic coordinates model (configuration) upon minimizing the so called chi square (χ^2) as described in eq. (1). χ^2 is computed as the sum of all squared deviations of the model M_{ij} from each used experimental data E_{ij} normalized by the data variance V_i. Although any experimental data that can be computed directly from an atom coordinates may be used in RMC modeling, very often RMC refers to modeling the experimental diffraction SF S(Q) and PDF G(r) as defined in eq. (2):

$$\chi^2=\sum_{i=1}^{N}\frac{\text{Sum of Squared Deviations}}{\text{variance}_i}=\sum_{i=1}^{N}\frac{\sum_j\left(M_{i,j}-E_{i,j}\right)^2}{V_i} \tag{1.1}$$

$$G(r)=\frac{2}{\pi}\int_0^\infty Q[S(Q)-1]\sin(Qr)dQ=4\pi r[\rho(r)\text{-}\rho_0] \tag{2.1}$$

$$R(r)=4\pi r^2\rho(r)=\frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{b_ib_j}{\langle b\rangle^2}\delta\left(r-r_{ij}\right)$$

$$G(r)=\frac{R(r)}{r}-4\pi\rho_0 \tag{2.2}$$

where Q is the momentum transfer and r the real distance between two atoms, $\rho(r)$ is the spherical average defined as $\int n(r').n(r-r')dr$ and ρ_0 is the average number density of the samples. $R(r)$ is the radial distribution function, N the total number of atoms, b_i is the scattering length for atom denoted as i and $r_{i,j}$ the distance between i and j atoms.

As described by Evrard and Pusztai[9] Fig. 2, the flowchart of a traditional RMC algorithm runs as the following. A configuration simulation box of defined size and volume, containing a fixed number of atoms hence a fixed density is used as a model starter. The radial distribution histograms are initialized and computed for all atoms in the system. The experimental diffraction data, the SF and the PDF are computed and hence the initial model's χ^2 is calculated. Then, RMC main simulation loop starts. In every loop step, a random atom is selected and a random translation is applied to it. The total radial distribution function and the PDF and SF are updated and the new χ^2 value is computed and compared to the old one. If the value of χ^2 decreases, the new atom position is accepted. If χ^2 value increases, the atom new position is only accepted with a likelihood of $e^{-\Delta\chi^2}$. As a result, if the later loop step is repeated enough number of times, χ^2 decreases and consequently the model fits better to experimental data as the sum of the total computed squared deviations of the model from the experimental data gets smaller.

Methods

fullrmc is especially designed with the highest maneuverability for modeling complex molecular systems. A thorough class definitions inheritance diagram of fullrmc package is shown in Fig. 1. Firstly, fullrmc Engine is the main class definition that contains all the methods needed to perform the modeling by managing all the configuration's data and all other definitions and modeling parameters. fullrmc adopts protein data bank (pdb) file format[15] as the configuration file with the addition of a few header lines like the boundary conditions information for instance (example provided in supporting information). Unlike traditional RMC, fullrmc models a system's configuration by applying moves upon defined groups rather than atoms. A group is a list of atom indexes used to group atoms to rigid or quasi-rigid bodies in any random or desired and convenient way (e.g., functional group, fragment, residual, molecule, etc.) to modeling a system.

The first difference between fullrmc and a traditional RMC software is the concept of GroupSelector that is basically responsible for selecting a single group by its index at every RMC step. Two main branches, the random selectors and the defined order selectors bifurcate from GroupSelector the parent class definition of all defined selectors. The random selectors are composed of sequentially three subclass selectors, the RandomSelector, the WeightedRandomSelector, and the SmartRandomSelector. As its name indicates, a RandomSelector is merely a GroupSelector with its selections being totally random and unpredictable where all groups are selected with the same likelihood. WeightedRandomSelector is also a RandomSelector but unlike RandomSelector, it supports a static selection probability scheme that can be biased and not evenly weighted. Finally, a SmartRandomSelector is another WeightedRandomSelector with its selection scheme being implemented with machine learning algorithms, dynamically changing by automatically biasing and unbiasing the selector's probability scheme to selecting the groups that are more likely

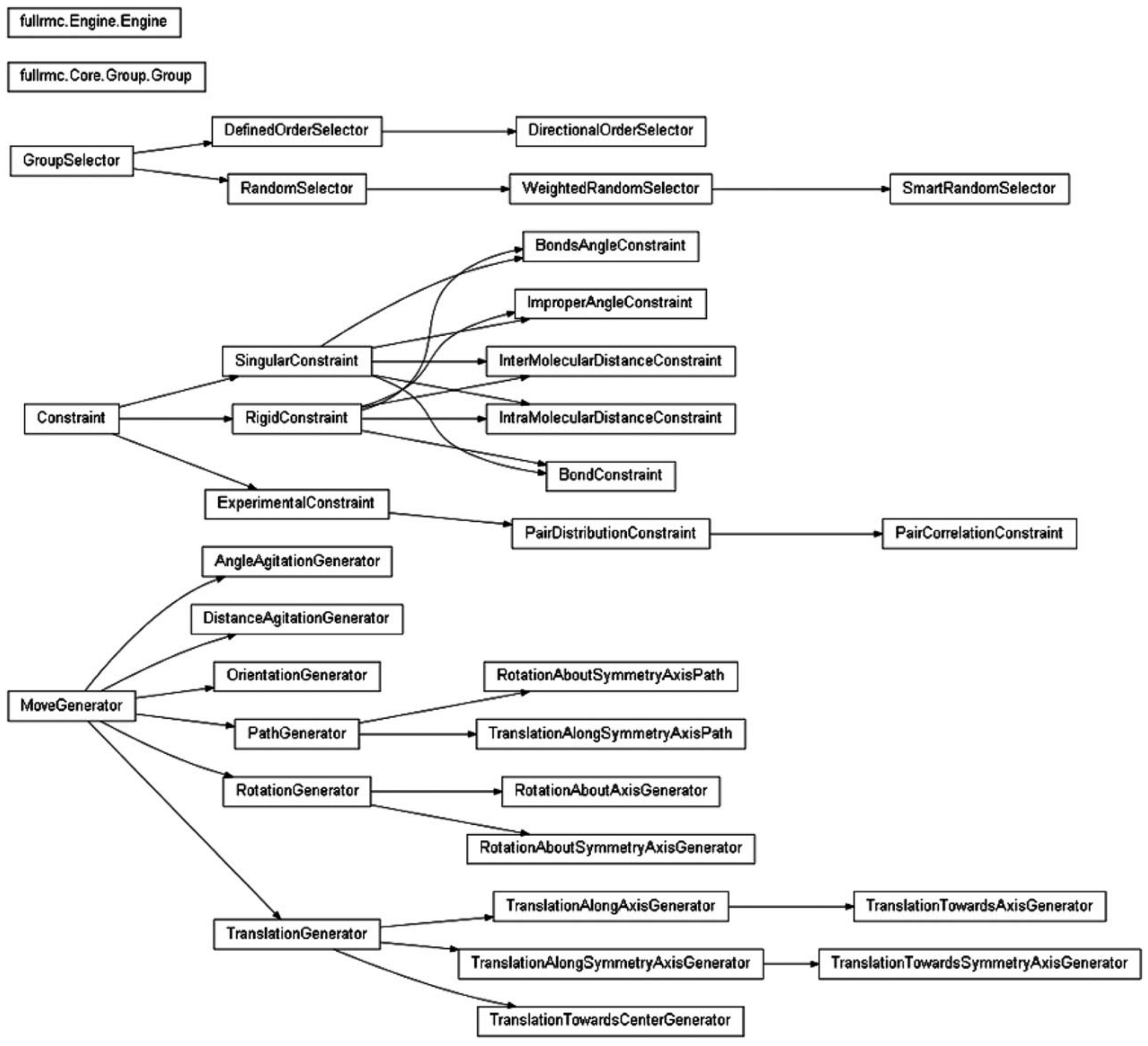
fullrmc.Engine.Engine
fullrmc.Core.Group.Group
GroupSelector
DefinedOrderSelector
DirectionalOrderSelector
RandomSelector
WeightedRandomSelector
SmartRandomSelector
Constraint
SingularConstraint
RigidConstraint
ExperimentalConstraint
BondsAngleConstraint
ImproperAngleConstraint
InterMolecularDistanceConstraint
IntraMolecularDistanceConstraint
BondConstraint
PairDistributionConstraint
PairCorrelationConstraint
MoveGenerator
AngleAgitationGenerator
DistanceAgitationGenerator
OrientationGenerator
PathGenerator
RotationAboutSymmetryAxisPath
TranslationAlongSymmetryAxisPath
RotationGenerator
RotationAboutAxisGenerator
RotationAboutSymmetryAxisGenerator
TranslationGenerator
TranslationAlongAxisGenerator
TranslationTowardsAxisGenerator
TranslationAlongSymmetryAxisGenerator
TranslationTowardsSymmetryAxisGenerator
TranslationTowardsCenterGenerator

Figure 1. fullrmc inheritance diagram. Five main base classes with their most important derived classes are visualized.

to achieve a successful move enhancing the total configuration structure, hence in some cases increasing substantially the total number of accepted moves. However, defined order selectors are those with no randomness in the selection but with a predefined order of selection. DefinedOrderSelector and DirectionalOrderSelector are the two defined subclasses of this branch of selectors. A DefinedOrderSelector is simply a GroupSelector with the order of selection being stored in advance and the groups being pulled or selected one after the other according to the selection order. A DirectionalOrderSelector is another DefinedOrderSelector, but with the only difference being that the order of selection is sorted automatically at Engine's runtime, then such groups are pulled according to their distance from a center point in space. DirectionalOrderSelector becomes very useful when modeling expanding or contracting systems allowing groups and therefore the atoms position to be refined in layers.

The use of constraints is a similarity between fullrmc and traditional RMC. Constraints play the role of a referee during the modeling process. They are what reflect our knowledge about the system being modeled, therefore in fullrmc, experimental data along with physical constraints are all defined as constraints. Three main branches, the singular constraints, the rigid constraints, and the experimental constraints derive from Constraint, the parent class definition of all defined constraints. SingularConstraint classes are all constraints that are not allowed to have multiple instances in the same engine. Basically, all constraints that require a definition input from the user (e.g., bonds) are defined as singular to avoid conflicts in cross definitions (e.g., same atom types different bond length). RigidConstraint classes are constraints that must always be satisfied for a RMC step to be accepted. Even if the total system's χ^2 value decreases upon a RMC group move, if any RigidConstraint total squared deviations increases, the move is rejected and therefore the step is not accepted. ExperimentalConstraint is the parent class definition of all constraints based on experimental data inputs. To avoid constraints cross definitions conflict, all physical constraints are defined as singular and rigid.

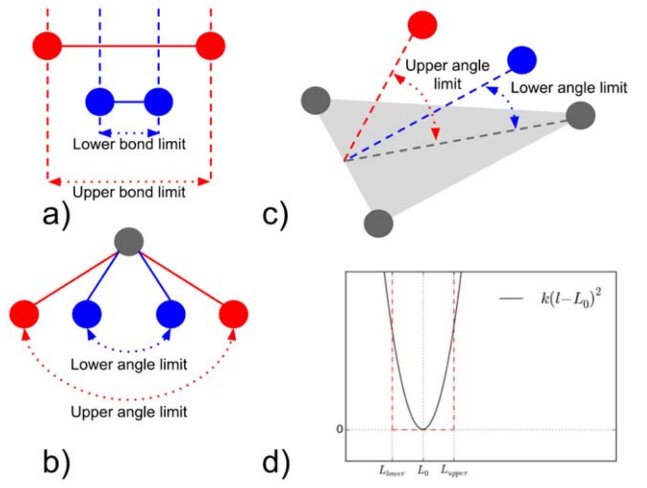

Figure 2. Intramolecular bonds and angles constraints. a) Bond constraint lower and upper bond distance limits. b) Bonded atoms angle constraint with lower and upper angle limits. c) Improper angle constraint used to coerce an atom to stay within lower and upper angle limits with a plane. d) Harmonic potential approximated with lower and upper limits as used in fullrmc.

In very few words, we will herein describe all existing constraints, for more details, readers are encouraged to refer to the online documentation.[16] Lennard Jones [eq. (3) like constraints such as distance constraints can be modeled with mere cutoff distances that can be roughly approximated equal to σ (Fig. 3).

$$V(r)=4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^{6}\right] \quad (3)$$

InterMolecularDistanceConstraint governs the intermolecular atomic distances by setting a cutoff to the minimum allowed value of the distance separating two atoms. In the same way, IntraMolecularDistanceConstraint governs only intramolecular atomic distances instead of intermolecular ones. On the other hand quasi-harmonic constraints generally computed as a pure harmonic of the form $U(r)=K(X-X_0)^2$ can be modeled simplistically with lower and upper limit cutoffs (Fig. 2d). BondConstraint is a purely intramolecular quasi-harmonic constraint that controls molecular covalent bonds between two atoms in a molecule. In this case the lower and upper bond's distance limit cutoffs are defined between intramolecular atoms thus imitating a rigid bond (Fig. 2a). BondsAngleConstraint is another quasi-harmonic purely intramolecular constraint that coerces the angle between three intramolecular atoms by also using lower and upper angle value limits (Fig. 2b). ImproperAngleConstraint is the last so far defined quasi-harmonic intramolecular constraint. It is used to force an atom to stay within lower and upper angle value limits formed between the atom and a plane of three independent atoms (Fig. 2c).

The second main difference between fullrmc and a traditional RMC software is the concept of move generator that is basically responsible for moving a group by applying a transformation matrix to its atoms coordinates. By definition, every group has its own move generator. Generators derived from MoveGenerator base class definition are classified into geometrical subgenerators.

TranslationGenerator is a move generator that performs random translations upon a group of atoms. TranslationAlongAxisGenerator is another translation generator with random translations being performed along a user predefined axis. TranslationAlongSymmetryAxisGenerator is also a translation generator where translation vectors are computed along one of the symmetry axes of the group. TranslationGeneratorsTowards is another branch of translation generators where translation moves are computed towards a certain point or along a certain axis within an angle of tolerance. TranslationTowardsCenterGenerator is a translation towards generator that generates translation moves upon a group towards a center point. The point can be given as fixed coordinates or computed on the fly as the geometrical center of a set of atoms. TranslationTowardsAxisGenerator is another towards generator that generates moves towards a predefined axis. TranslationTowardsSymmetryAxisGenerator is also a translation towards generator where translations are computed towards one of the symmetry axes of the group.

RotationGenerator is another class of move generators that performs random rotations upon a group of a minimum of two atoms or more. RotationAboutAxisGenerator is a rotation generator where the rotation axis is predefined. RotationAboutSymmetryAxisGenerator is also a rotation generator where the rotation axis is one of the symmetry axes of the group computed on the fly at engine runtime.

Agitation generators form another branch of generators that agitate or shake a group about a certain equilibrium point. DistanceAgitationGenerator generates random agitations to two atoms by applying translation moves to any or each of them along the direction axis separating them. Hence, shortening or enlarging the distance between the two atoms. This is mainly used to adjust bond lengths in a molecular structure. AngleAgitationGenerator is another agitation generator that agitates the angle defined between three atoms. Move will be performed by translating any or all of the atoms to increasing or decreasing the angle. This is mainly used to adjust bonded atoms angles in a molecular structure.

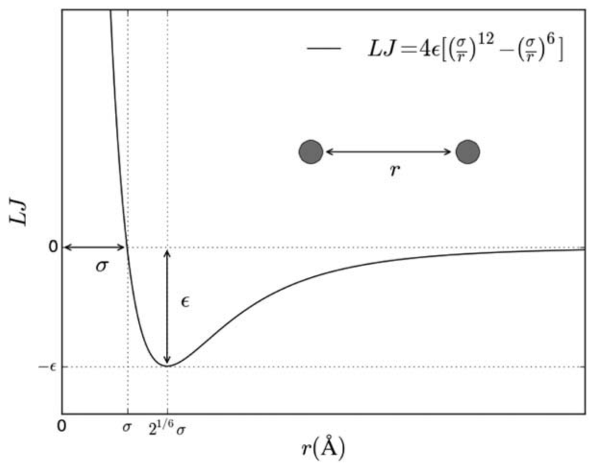

Figure 3. The lennard Jones potential approximated with a hard cutoff limit as used in fullrmc.

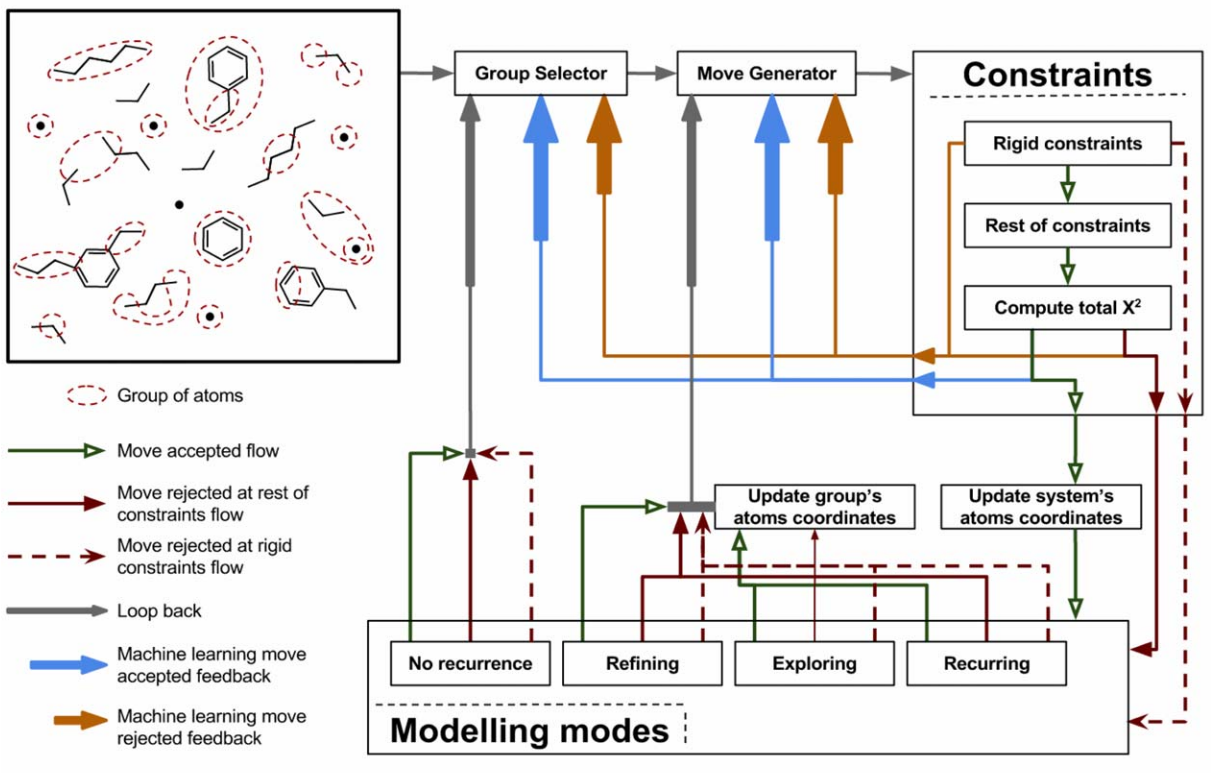
Group Selector
Move Generator
Constraints
Rigid constraints
Rest of constraints
Compute total X²
Group of atoms
Move accepted flow
Move rejected at rest of constraints flow
Move rejected at rigid constraints flow
Loop back
Machine learning move accepted feedback
Machine learning move rejected feedback
Update group's atoms coordinates
Update system's atoms coordinates
No recurrence
Refining
Exploring
Recurring
Modelling modes

Figure 4. fullrmc algorithm flow chart. Including a configuration sketch and all fullrmc main algorithm flows including the reinforcement machine learning ones.

The last so far defined generators branch is the so called path generators. Path generators store a list of moves in advance and pull them one move after another. TranslationAlongSymmetryAxisPath and RotationAboutSymmetryAxisPath are respectively path generators where the translation and rotation amplitudes are pre-stored in advance and pulled one after the other every time the generator is called.

Although not shown in the inheritance diagram (Fig. 1), a group can have a collection or a combination of move generators through respectively MoveGeneratorCollector and MoveGeneratorCombinator. A MoveGeneratorCollector is basically a list of move generators from which a generator will be randomly used during engine runtime. A MoveGeneratorCombinator, however, is a list of move generators but at every runtime step, the generated move is a serial combination or a convolution of all the generators moved. Therefore, more complex moves such as translation combined with a rotation can be performed by simply combining different generators.

Implementation

fullrmc is another reverse modeling package but it is more than a traditional RMC. Python 2.7[17] is the programming language chosen to write fullrmc merely because it has a human readable syntax and it is among the simplest and most versatile programming languages with a very active community of developers. The way fullrmc is programmed, complies with all modern programming practices. The programming approach is fully object oriented and structured with templates and inheritance. The same coding conventions are used in all the modules and the code is thoroughly commented and documented.[16]. The core modules are continuously tested with automated tests to cut down the number of possible bugs during the development and the addition of features by potential users. As in traditional RMC codes, the main modeling loop is only parallelizable at the expense of accuracy, by adding some long distances precision errors and substantial complex overhead to the code by partitioning the model into lists of neighbors or splitting the whole space into a grid of small domains. However, in fullrmc the main loop is always executed in serial, one step following another, but by decomposing each step into parallel tasks when it is possible. Also, all time consuming tasks like histogram computations for instance are smartly optimized and only computed when necessary. Vectorization with numpy[18] is applied all over the code and where speed and computation performance are needed, C and C++ compiled Cython[19,20] is used and called transparently in fullrmc main code.

A simplified flowchart of fullrmc is shown in Fig. 4 decomposing the code into its main blocks of computation. Atomic and molecular three dimensional configuration with any shape of periodic boundary conditions is used as an initial model. Nonperiodic boundary conditions are not implemented in the current version of fullrmc but they are going to be included in the coming distributions. As one can see, groups or rigid bodies are represented with red dashed domains in the

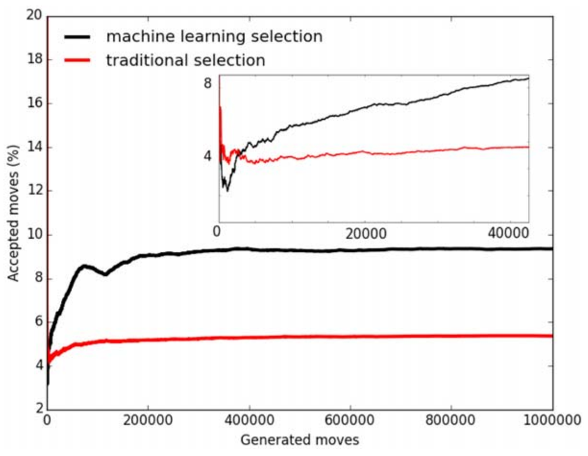

Figure 5. fullrmc percentage of accepted moves percentage. Smart selector implemented with reinforcement machine learning compared to traditional RMC selection. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

configuration representation (Fig. 4). Atoms can be part of different groups at the same time and can also be left outside of any group which is equivalent to fixed atoms in traditional RMC. Grouping atoms into clusters does not require any molecular information as groups are merely lists of atom indexes. However, every group has its own customizable move generator and can be composed of a whole molecule's atom indexes, a subset of a molecule atom indexes or even a mixture of atom indexes between different molecules. The flowchart is read through the gray arrow path, which can change color sometimes according to different paths the algorithm might take. During fullrmc engine runtime, the group selector selects a group from the defined list of groups, updates its atom's coordinates to the last accepted ones from the current configuration state. A move gets computed and applied to the group's atoms then the resulting structure is passed to the constraints box.

First, rigid constraints are computed and checked sequentially one after the other. If any of the rigid constraints rejects the move, the step gets rejected automatically, otherwise the new structure proceeds to computing all the rest of non-rigid constraints and the new total after move. At this stage, only three possible outcomes from the constraints box are likely to happen. The move is either accepted or rejected after computing the new χ^2, or the move is rejected prematurely upon encountering an unsatisfied rigid constraint before computing the new χ^2. In all of the later three cases, an accepted move (blue arrow), rejected or prematurely rejected move (orange arrow) feedback is sent to the group selector and the move generator respectively updating the machine learning selection and move generation schemes. At the very output of the constraints box, the configuration's atoms coordinates get updated only if the step is accepted and then the engine proceeds to the modeling modes box.

Unlike traditional RMC, fullrmc concedes different modeling and fitting modes. The no-recurrence mode is equivalent to a traditional RMC where at every step a new group of atoms is selected by looping back to the group selector. Group selection recurrence is allowed in fullrmc, the same group will be selected a recur number of times before a new group gets selected. The three different types of recurrence modes are "recurring," "refining," and "exploring." For recurrence modes, it is noteworthy to differentiate between group atoms position and the configuration position of the same atoms. It becomes important to clarify that the way fullrmc works is that when a group is selected its atoms positions are the same as the configuration's positions. In addition, move generator applies its moves upon the selected group atoms position and changes their state, but the configuration's position of the same atoms is not updated and modified to the after move positions until the move is accepted.

The first recurrence mode is the so called "recurring" mode. It is the simplest and most straightforward among all recurrence modes. During "recurring" mode, no new group is selected and both configuration and group's atoms position are updated when a move gets accepted. Hence, the move generator will always apply moves upon the last accepted group's atoms positions.

The second recurrence mode is the "refining" mode that is used to refine the position of a group of atoms. During "refining"' mode, configuration's atoms will be updated to the last accepted move but the group's atoms will always be reset to their initial position. Therefore, move generator will always apply its moves upon the same atoms position, which are those of when the group was selected. This way 'refining' allows a group to refine its atoms position by finally accepting the best generated move starting from the same position of atoms.

The last recurrence mode is "exploring." This mode is a very important mode in fullrmc modeling because it allows a group of atoms to explore the space around it and go out of local minima and beyond energy barriers to finding the global minimum. Like in all other modes, configuration's atoms will be updated to the last accepted move. However, in "exploring" mode, group's atoms position will always be updated to the last generated move whether it is accepted or rejected with the exception of rigid constraint rejection. Therefore, during "exploring" mode, the group wanders in the allowed three dimensional space and the configuration's atoms position gets updated only when a better position is found.

Insight into Machine Learning in fullrmc

Machine learning as known nowadays, has been given many definitions covering a wide range of applications. In fullrmc, we will be using machine learning for optimization reasons only. Therefore, we would like to give our own definition of machine learning and artificial intelligence as computationally applied statistics and learning over time, based on experience and exploitation. Machine learning can be generally categorized into three main types. The supervised machine learning[21] where learning is a function induction or approximation that reproduces up to a certain level of accuracy a set of so called training data. The second type is the unsupervised learning,[22] which is more about describing the data. Unsupervised learning is generally used in classification and identification problems. The third type is called reinforcement machine learning[23] and is the one used in fullrmc. Reinforcement machine learning is a trial and error learning which fundamentally reminds RMC modeling. In contrast with supervised and unsupervised learning where the machine is taught explicitly by creating a model or a classifier from a set of data, in reinforcement learning the machine is taught from the consequences of its actions and interaction with the data and past experience or exploitation. Therefore, a RMC simulation is a reinforcement machine learning problem by definition and construction. As described earlier, the whole mechanism of a RMC simulation step can be summarized to, applying a coordinates' transformation upon certain selected atoms, and then accepting or rejecting the step by verifying whether it enhances or not the whole system agreement with a set of constraints and experimental data. In traditional RMC, randomly selecting atoms and moves, without reference to an estimated probability distribution, is known to give rise to very poor performance in accepted to generated moves ratio. Thanks to the reinforcement machine learning implementation at the level of the group selection and move generation in fullrmc, the number of accepted to generated moves can be considerably improved in comparison to traditional RMC. The method used in fullrmc consists in biasing and unbiasing the group selector and later the move generator towards selecting a group that most likely needs to be refined by having its atoms moved. In Fig. 5, we show a comparative example between traditional RMC random selection and fullrmc SmartRandomSelector implemented with reinforcement machine learning dynamic selection probability distribution scheme. In this example, a random molecular system is created with 80% of its groups' move generators set to perform unrealistic moves leading inevitably to rejection. Clearly in the graph, after few thousand steps, the ratio of accepted to generated moves in the traditional RMC approach saturates around 5%. However, using fullrmc SmartRandomSelector, the ratio of accepted to generated moves almost doubles and saturates around 9%. As one can see, using machine learning can be very beneficial especially after a certain amount of generated selections and moves. Like in any machine learning problem, there is a learning curve which can be slow sometimes or fast other times depending on the system, but in general the selector needs to gather enough experience to start gaining in the ratio of accepted moves. In this particular example, below a couple of thousands moves, traditional RMC selection seems to be more effective than the machine learning implementation one in fullrmc, this is called the warming up time or number of selection. We can conclude that if the simulation is not meant to run more than the warming up time, the approach becomes ineffective and it is better to run traditional RMC selection.

Tetrahydrofuran (THF) Simulation Example

fullrmc is an easy scripting package where everything is comprehensive and self-explanatory. The herein code is a small THF liquid molecular simulation. Thanks to the Joint Center for Energy Storage Research (JCESR) efforts, we obtained the THF X-ray diffraction PDF pattern collected at 11-ID-B beamline at the Advanced Photon Source of Argonne National Laboratories. THF based solvents have been lately heavily investigated as potential candidates for multi-valence batteries electrolyte.[24] More complete and elaborate examples are delivered with fullrmc distribution and thoroughly explained in fullrmc online documentation.[16] For the sake of the discussion we will here explain in details every step in the following simulation script.

```
# SECTION: DEFINITIONS IMPORTATION
from fullrmc.Engine import Engine
from fullrmc.Constraints.PairDistributionConstraints import PairDistributionConstraint
from fullrmc.Constraints.DistanceConstraints import InterMolecularDistanceConstraint
from fullrmc.Constraints.BondConstraints import BondConstraint
from fullrmc.Constraints.AngleConstraints import BondsAngleConstraint
from fullrmc.Constraints.ImproperAngleConstraints import ImproperAngleConstraint
from fullrmc.Core.MoveGenerator import MoveGeneratorCollector
from fullrmc.Generators.Translations import TranslationGenerator
from fullrmc.Generators.Rotations import RotationGenerator
# SECTION: CREATING ENGINE
ENGINE = Engine(pdb="thf_conf.pdb", constraints=None)
# create constraints
PDF = PairDistributionConstraint(engine=None, experimentalData="thf_pdf.dat", weighting="atomicNumber")
EMD = InterMolecularDistanceConstraint(engine=None)
BON = BondConstraint(engine=None)
BAN = BondsAngleConstraint(engine=None)
IAN = ImproperAngleConstraint(engine=None)
# add constraints to engine
ENGINE.add_constraints([PDF, EMD, BON, BAN, IAN])
# create constraints definitions
```

```
BON.create_bonds_by_definition (bondsDefinition={"THF": [('O', 'C1', 1.20, 1.70),
('O', 'C4', 1.20, 1.70), . . .]})
BAN.create_angles_by_definition (anglesDefinition={"THF": [('O', 'C1', 'C4', 105, 125),
('C1', 'O', 'C2', 100, 120), . . .]})
IAN.create_angles_by_definition (anglesDefinition={"THF": [('C2','O','C1','C4', -15, 15),
('C3','O','C1','C4', -15, 15), . . .]})
# SECTION: RUNNING ATOMIC RMC
ENGINE.set_groups_as_atoms()
ENGINE.run(numberOfSteps=10e6, saveFrequency=10e3, savePath="thf_engine.rmc")
# SECTION: RUNNING MOLECULAR RMC
ENGINE.set_groups_as_molecules()
for g in ENGINE.groups:
  TG = TranslationGenerator(amplitude=0.2)
  RG = RotationGenerator(amplitude=2)
  GC = MoveGeneratorCollector(collection=[TG, RG], randomize=True))
  g.set_move_generator(GC)
ENGINE.run(numberOfSteps=10e6, saveFrequency=10e3, savePath="thf_engine.rmc")
```

As in any python script the needed packages and modules have to be imported prior to any initialization. Hence, all the needed definitions from fullrmc and other needed python modules and packages are imported in the beginning of the script in the section "definitions importation." The very next section "creating engine" is for initializing fullrmc RMC simulation engine and to adding all the configuration attributes and known information about the structure such as experimental and molecular constraints.

The engine is initialized with the initial pdb configuration and no constraints. Next all the constraints are initialized one after the other and added to the engine using the engine's method "add_constraints." Then all constraints definitions are created, all the bonds and bond angles in the THF molecule as well as the improper angles definition to keeping the molecule more or less planar within $\pm 15°$.

In the next "running atomic rmc" section, the groups are created using the automatic creation method "set_groups_as_ atoms," which will automatically go through all the configuration atoms and create single atom groups for all and each atom in the system. Upon creating a group, fullrmc insures that the group has its own move generator instance and for keeping the generality of the approach the simple TranslationGenerator is attributed automatically. Because in this section all groups contain one single atom, therefore only random translation moves make sense and therefore no further modifications are needed. Since now everything is well set, an engine run is performed in the next line using the engine's method "run." It is important to note that "run" method can take multiple arguments. Among others "numberOfSteps" argument is used to specify the number of steps to perform. "saveFrequency" and "savePath" are used to indicate the frequency of saving the engine to the disk and the absolute path of the file where to save it.

After finishing this section, the script will proceed to the next section, which is "running molecular rmc." In this section, we intend to show how easy is to perform a molecular RMC with fullrmc. By calling the engine method "set_groups_as_ molecules"' all the existing defined groups will be deleted and new groups of rigid bodies THF molecules will be created. The way fullrmc knows a molecule is by parsing the pdb file contents and more precisely all of the following attributes "Residue name," "Sequence number," and "Segment identifier" of every and each record in the pdb file. pdb files are widely known and very common molecular files mainly used to describe proteins and big molecules. For more information about pdb files readers are encouraged to look at the online documentation of pdb coordinate files format.[15] Now that all the groups are set to molecules, it's time to set the move generator of every group. Therefore, a loop of all the defined engine's groups is created. In every step of this loop and for the sake of this example, we chose to simply attribute to every group the same type of move generator, which is a MoveGeneratorCollector of a random translation through TranslationGenerator and a random rotation through RotationGenerator. We must note that the attributes amplitude in the instantiation of TranslationGenerator sets the maximum allowed translation vector amplitude in Angstrom and in RotationGenerator it sets the maximum allowed rotation angle in degrees. Finally, the same engine's "run" method is called to continue the modeling of the same system but this time with molecular groups instead of atomic.

Although the purpose of this article is not to discuss about the solved structure but in Fig. 6 we show in short the simulation progress and resulting structure. The initial PDF comparison with experimental XRD data is shown in Fig. 6a. Running the simulation on groups of single atoms, was enough to model the intramolecular atoms distances (<3 Å) as shown in Fig. 6b. The range of long distances (>3 Å) predominantly coming from the inter-molecular correlations is very poorly fitted even after 1,000,000 steps of atomic RMC simulation. In Fig. 6c, we show the final result of the simulation after enabling the molecular and rigid body moves upon groups of THF molecules. Free rotations and rotations about symmetry axes as well as free translation and translations along symmetry axes of THF molecules were needed in order to reproduce the inter-molecular peaks at ~5, 10, and 15 Å.

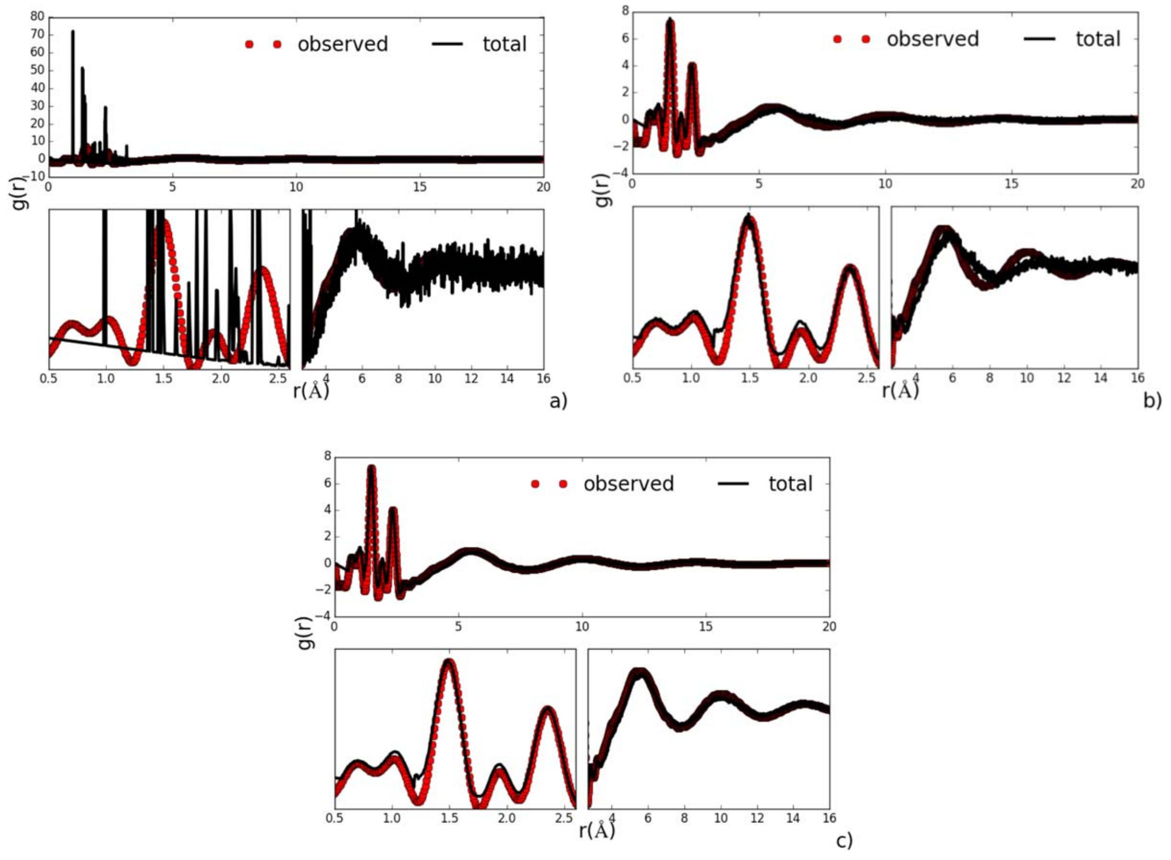

Figure 6. THF fullrmc modeling PDF. a) Initial configuration. b) Atomic refinement with no molecular moves. c) Molecular and atomic refinement. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Benchmark

In this section, some of fullrmc components and functionalities performance benchmarks is discussed. In Fig. 7a, the mean time per single RMC step is computed as a function of group's number of atoms for all of PDF constraint (pdf), intermolecular distance constraint (vdw), bond constraint (bond), bonded atoms angle constraint (angle), improper angle constraint (improper) as well as all and none of the later constraints.

In every benchmark experiment, 100,000 RMC steps are performed on the same initial THF molecular system. This study clearly shows where most of RMC computation time is spent. Firstly, as one can see, each and every constraints' computation time is linearly dependent to the number of atoms per

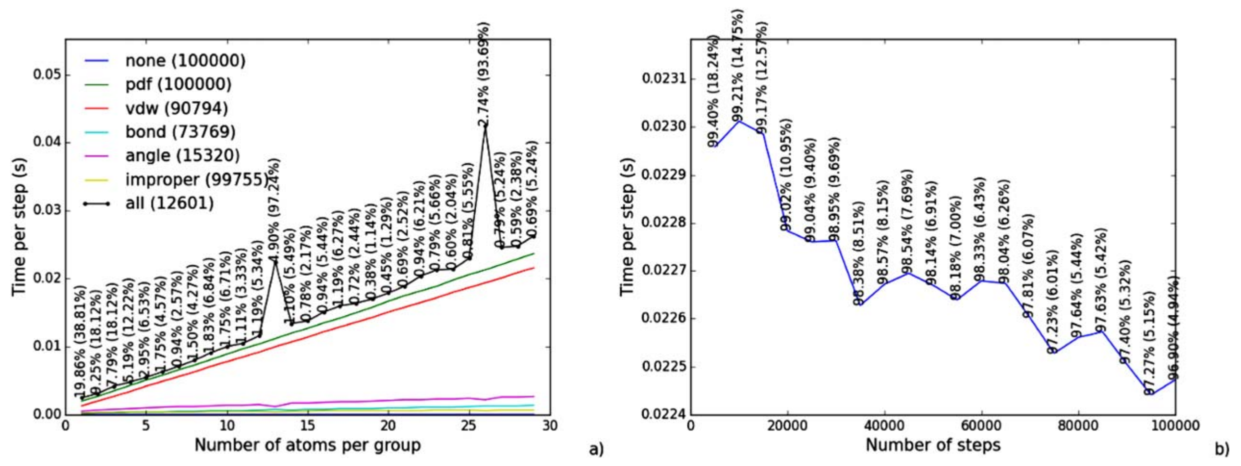

Figure 7. fullrmc mean time per step benchmark. a) Benchmarking constraints' mean time per step as a function of the number of atoms per group. The legend is formatted as the following: "constraint (mean accepted moves number)." The used constraints are "none" for no constraints, "pdf" for PairDistributionFunction, "vdw" for InterMolecularDistanceConstraint, "bond" for BondConstraint, "angle" for BondsAngleConstraint, "improper" for ImproperAngleConstraint and finally "all" for using all the above constraints. b) Benchmarking the time per step as a function of the total number of steps for 13 atoms group size. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

group. Secondly, all of intra-molecular constraints such as BondConstraint, BondsAngleConstraint, ImproperAngleConstraint involve relatively small amount of atoms at every move and therefore require less of computation time in comparison to intermolecular constraints. PairDistributionFunction and InterMolecularDistanceConstraint because they involve the whole system's atom pairs distance computation at every RMC step, are mostly the constraints that cause the damage to the performance. All constraints benchmark shows that as the number of atoms per group increases, the mean time spent per step increases linearly except for groups of single THF molecule equal to 13 atoms or double THF molecules equal to 26 atoms. At those particular numbers, the ratio of Tried/Generated moves as well as Accepted/Generated moves increase drastically and therefore the mean time spent per step takes higher values accordingly. In addition, fullrmc engine is smartly optimized to reducing the mean total computation time per step when multiple constraints are used. Even though "all" constraints benchmark computation contains "vdw" and "pdf" constraints, the mean time per step of 'all' constraints is smaller than the sum of both "vdw" and "pdf" benchmarks.

The second benchmarking is done over the total number of steps per simulation (Fig. 7b). In this benchmark experiment, THF molecular groups are used and all constraints are applied to the system. As one can see, the mean time per step slightly decreases by increasing the number of steps. The reason why is because the number of tried and accepted moves decreases with the number of total performed steps. Therefore, the computation performance artificially gets better when the system's evolution slows down.

Conclusion

This article describes fullrmc, a new RMC package for refining an atomic or molecular system by reverse modeling a set of experimental data and constraints. fullrmc computation core is written in Cython, compiled in C and C++ and totally interfaced with python. The package provides a wide set of definitions and allows to easily setup almost any kind of refining engine for all kinds of applications. A particular advantage of fullrmc in comparison to other RMC solutions is the usage of reinforcement machine learning that substantially increases the refinement speed and accuracy in some cases. fullrmc class hierarchy and implementation are quite innovative and concepts such as Group, GroupSelector and MoveGenerator and RMC modeling modes (recurring, refining and exploring) stand out from all other RMC software.fullrmc has already been used in multiple projects including molecular liquids, crystalline materials, alloy phase transformation, and nanoparticles in solution. The code is being continuously developed. Functionalities and constraints are being added along with every new scientific challenge.

Future developments of the code will include the addition of molecule recognition and reconstruction during the fitting process.

Acknowledgments

This research used resources of the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science User Facility operated for the DOE Office of Science by Argonne National Laboratory under Contract No. DE-AC02-06CH11357. The author would like to thank B. Reinhart (ANL), P. Chupas (ANL) and K. Chapman (ANL) for the technical assistance and scientific inputs. The Tetrahydrofuran example was provided by the Joint Center for Energy Storage Research (JCESR) efforts an Energy Innovation Hub funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences.

Keywords: reverse Monte Carlo · rigid body · machine learning · pair distribution function · modeling

How to cite this article: B. Aoun. *J. Comput. Chem.* **2016**, *37*, 1102–1111. DOI: 10.1002/jcc.24304

Additional Supporting Information may be found in the online version of this article.

[1] R. L. McGreevy, L. Pusztai, *Mol. Simul.* **1988**, *1*, 359.
[2] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, *J. Chem. Phys.* **1953**, *21*, 1087.
[3] R. L. McGreevy, *J. Phys. Condens. Matter* **2001**, *13*, R877.
[4] V. M. Nield, D. A. Keen, R. L. McGreevy, *Acta Cryst.* **1995**, *A51*, 763.
[5] D. A. Keen, M. G. Tucker, M. T. Dove, *J. Phys. Condens. Matter* **2005**, *17*, S15.
[6] R. L. McGreevy, M. A. Howe, *Annu. Rev. Matter. Sci.* **1992**, *22*, 217.
[7] S. J. Gurman, R. L. McGreevy, *J. Phys. Condens. Matter* **1990**, 9463.
[8] D. A. Keen, R. L. McGreevy, *J. Phys. Condens. Matter* **1991**, 7383.
[9] G. Evrard, L. Pusztai, *J. Phys. Condens. Matter.* **2005**, *17*, S1.
[10] RMC++ available at: http://www.szfki.hu/~nphys/rmc++/opening.html.
[11] M. G. Tucker, D. A. Keen, M. T. Dove, A. L. Goodwin, Q. Hui, *J. Phys.: Condens. Matter* **2007**, *19*, 335218.
[12] RMCProfile available at: http://www.rmcprofile.org/Main_Page.
[13] G. Opletal, T. C. Petersen, S. P. Russo, *Commun. Phys. Commun.* **2014**, *185*, 1854.
[14] Hybrid Reverse Monte Carlo available at: http://www.hrmccode.com/.
[15] Protein Data Bank file format available at: http://deposit.rcsb.org/adit/docs/pdb_atom_format.html.
[16] (a) fullrmc's source code available at: https://github.com/bachiraoun/fullrmc (b) fullrmc's online documentation available at: http://bachiraoun.github.io/fullrmc/.
[17] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org.
[18] S. van der Walt, S. C. Colbert, G. Varoquaux, *Comput. Sci. Eng.* **2011**, *13*, 22.
[19] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, K. Smith, *Comput. Sci. Eng.* **2011**, *13*, 31.
[20] S Behnel, R Bradshaw, D Seljebotn, In Cython tutorial in Proceedings of the 8th Python in Science conference (SciPy); G. Varoquaux, S. van der Walt, J. Millman, Eds.; **2009**; pp. 4–14.
[21] S. B. Kotsiantis, *Informatica* **2007**, *31*, 249.
[22] A. K. Jain, M. N. Murty, P. Flynn, *ACM Comput. Surv.* **1999**, *31*, 264.
[23] A. G. Barto, R. Sutton, Introduction to Reinforcement Learning; MIT Press, **1997**. https://mitpress.mit.edu/books/reinforcement-learning.
[24] O. Mizrahi, N. Amir, E. Pollak, O. Chusid, V. Marks, H. Gottlieb, L. Larush, E. Zinigrad, D. Aurbach, *J. Electrochem. Soc.* **2008**, *155*, A103.

Received: 21 September 2015
Revised: 4 November 2015
Accepted: 17 December 2015
Published online on 22 January 2016