EASAL User/Developer Manual

Aysegul Ozkan, Rahul Prabhu, Ruijin Wu, Troy Baker,
James Pence, Jorg Peters, Meera Sitharam
University of Florida

December 23, 2016

This is a user/developer guide for the EASAL software described in the accompanying TOMS paper for generating, describing topology and exploring the assembly configuration space of two rigid sets of points in R^3 that are mutually constrained by distance intervals. Technical concepts and definitions that are required to use this software can be found in the main paper.

1 Introduction

EASAL generates and describes the key aspects of the topology and geometry of assembly configuration space of two rigid sets of points in R^3 [4, 2, 5]. EASAL implements algorithms that use new theoretical results, some of which are presented in [2, 3, 4]. EASAL is opensource and can be downloaded from `https://www.bitbucket.org/geoplexity/EASAL`. This user guide describes in depth the key conceptual functionalities, dependencies and installation, and the major classes and methods in EASAL. The user can find a detailed example of how to use the software in the README.md file which can be found along with the source code. A video presenting the theory, applications, and software components of EASAL is available at `http://www.cise.ufl.edu/~rprabhu/EASALvideo.mpg`.

Organization: The remainder of this guide is organized as follows. Section 2 discusses the main functionalities offered by EASAL. Section 3 gives a description of the software dependencies and installation instructions. Section 4 describes an example run to help the user understand how the software is used. Section 5 gives details of the major classes, attributes and methods to help developers gain an insight into how EASAL has been implemented.

2 Software Functionalities

This section introduces the main functionalities of EASAL.

2.1 Input/Output

2.1.1 Input

Input to EASAL is specified using the input window (see Fig. 3). The main input features are the following:

- **Two rigid point sets**: *the data for molecule (A/B)* field is used to specify the rigid point sets. This accepts the data in the pdb format. The user can either type in the location of the input point sets files or select it using the browse option. Once a file has been selected, the *set data* option allows the user to edit the input data before starting sampling.
- **The pairwise distance constraints (potential energy or enthalpy function)**: There are three types of constraints.

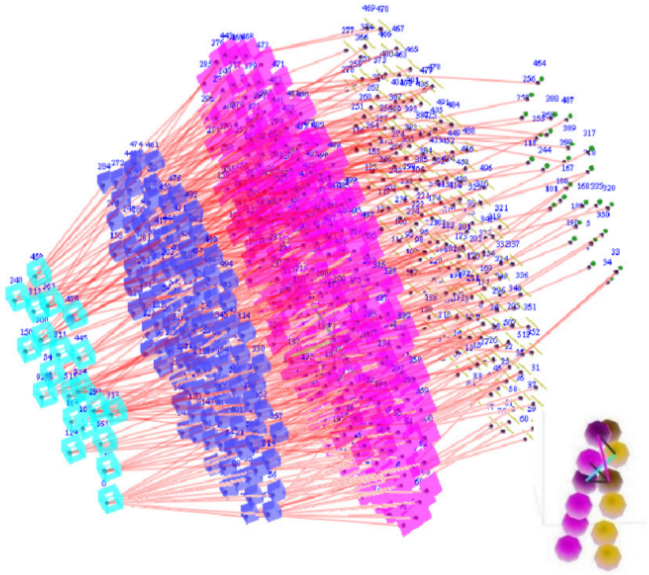

Figure 1: Stratification of an assembly constraint system with atlas nodes of dimension 4 (cyan), 3 (blue), 2 (purple), 1 (yellow), and 0 (green). Strata of each dimension of the assembly constraint system visualized in the lower right inset are shown as nodes of one color and shape in a directed acyclic graph. Each node represents an active constraint region. Edges indicate containment in a parent region one dimension higher.

- **Bonding threshold**: This is the range of distances between atoms where bond formation is feasible (i.e., the attractive forces dominate). this is given as $\lambda * (r_i + r_j) \pm \delta$ namely the Lennard-Jones well. Here, λ is specified by the lambda text field in the input window and δ is specified by the delta text box in the input window. r_i and r_j are the radii of the atoms participating in the reaction. When predefined interactions are specified, we use those for the bond formation and use the bonding threshold only for the hard-sphere potentials for the points not involved in the bond formation.
- **Collision threshold:** This is the minimum distance between points or atoms. This too is given as $\lambda * (r_i + r_j) \pm \delta$.
- **Predefined Interactions**: Only when the constraints between these pairs are active, are the corresponding configuration space regions explored.

2.2 Output

The output is the *atlas* (see Fig. 1), i.e., a topologically stratified set of sample feasible realizations or configurations of the two rigid point sets. In EASAL the output is visualized by what we call the *sweep view*. One point set is held fixed, while the other set is drawn many times to trace out the set of all feasible realizations.

2.3 Stratification

When EASAL first starts, the user is presented with the *atlas view* (see Fig. 5). The atlas view shows the stratification of the configuration space into regions of various dimensions, known as the active constraint regions. The stratification is a directed acyclic graph where each node represents an active constraint region and is associated with an active constraint graph and edges represent immediate containment of a region within its parent region. Here, the user can use the following functionalities.

2.3.1 Active Constraint Graph

When a node is clicked, EASAL automatically loads the active constraint graph (see Fig. 5) associated with that node in the bottom left corner. This active constraint graph shows only the participating points from each set. A solid edge between two points indicates a constraint and a dashed edge indicates a parameter between two points belonging to different sets. It has to be noted that the points in the same set already form a clique (not shown in the graph).

2.3.2 Finding Bounding Regions

Clicking *tree* (see Fig. 5) after selecting a particular node from the atlas shows us all the bounding regions of that node. In terms of the atlas, it shows the ancestors and descendants of that node . Since an edge in the graph represents containment, this feature allows us to inspect the containment of regions. Note that the bounding regions are also shown in the Cayley space view.

2.3.3 Finding Paths Between Nodes with 6 Active constraints

The atlas output by EASAL can be used to generate all the paths between any two active constraint regions along with their energies. Once the atlas has been generated, finding paths is *nearly instantaneous*. The path topology of the assembly configuration space is crucial for understanding assembly kinetics.

Of particular interest is finding paths between two lowest energy configurations with 6 active constraints or 0D nodes of the atlas with effectively rigid configurations. We are mainly interested in paths through active constraint regions with 5 or 6 active constraints (which are one step higher energy and have one fewer constraint). Such paths represent a continuous one degree of freedom motion.

Whenever the user clicks on two nodes in succession, EASAL finds the path between the two nodes, if it exists, and writes this path as a series of nodes to the "paths.txt" file. Once the sampling has been completed, EASAL computes the total number of paths of a particular length between every pair of vertices and writes the entire matrix to the "path_matrix.txt" file. The user can specify lengths by setting the "path_length" parameter in the settings.ini file.

2.3.4 Sampling Options

The user can at any point stop the current sampling mode and redirect the sampling. The various options available are:

- **Stop Sampling**: Clicking this button stops the sampling of the atlas and presents options to redirect the sampling.
- **The Constraint Selection Dialogue Box**: This dialogue box (see Figure 2) allows the user to select a particular node from the atlas and start sampling from that node. This feature gives users the flexibility to explore regions of the graph that are more relevant to them. The user can select a node by either specifying a node number or by specifying active constraints between the point sets. The spheres with the indices denote the atoms. A thick line between the atoms indicates a participating bond. Here, the first constraint is mandatory and hence does not have a *connect* check box next to it. The rest of the Active constraints are optional and the user may choose up to six.
- **CUR**: On clicking this, the sampling starts from the node that is currently selected and ends when the tree in which the node is present is fully sampled.
- **Cleanup**: Clicking this completes sampling on all partially sampled trees.
- **A-S**: A-S stands for Auto-Solve. This button starts sampling in the auto-solve mode. This is the default sampling mode. When starting afresh in this mode, EASAL generates all possible 4D and 5D root nodes depending on the user input. Then it proceeds to recursively sample all these nodes till the

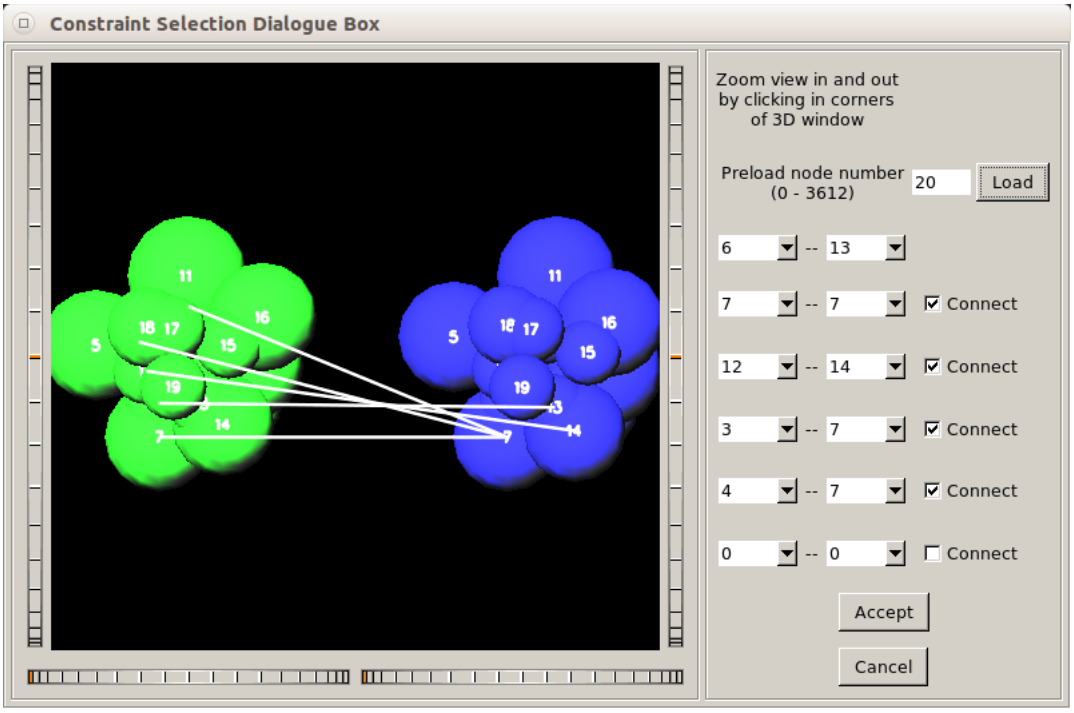

Figure 2: Constraint Selection Dialogue

atlas generation is complete. If there is a partially generated atlas, this mode of sampling completes the unfinished trees and then proceeds to sample all the other root nodes.

- **BFS**: Clicking this button forces EASAL to explore the atlas in a breadth first fashion which it otherwise does in a depth first fashion. In this mode of sampling, EASAL starts from the node currently selected and samples its subtree to completion.

2.3.5 Dimension of the root node

By default, the dimension of the root node of the atlas is 5, which means that the root node has one bond and 5 parameters. The *4D root node* option in the *advanced options* (see Fig. 4) allows the user to change this and make the root node a 4D node. With a 4D root node, there are 2 bonds and 4 parameters in the root node.

2.4 Convex Cayley Parametrization

After selecting a node, the user can view its Cayley configuration space shown in the *Cayley space view* (see Fig. 6). The Cayley space view shows the active constraint region in the Cayley parametrized chart representation for a particular node.

2.4.1 Inspect Cayley Points

The user can view the Cayley points in the Cayley Space View (see Fig. 6). The Cayley points are shown on a 3D grid. For nodes of dimension higher than 3d, one slider per extra dimension is given to view the points in the 4^{th} and 5^{th} dimensions.
Categorization of Cayley points: In the Cayley Space View (see Fig. 6), initially green points are shown which correspond to all the realizable points in the Cayley space. Clicking on the red square at the bottom

shows the points that have collision. Clicking the red square also shows two more options viz., cyan and pink. The cyan points represent the points which have angle violations and the pink points represent points which have distance violations. Clicking on the blue square shows all the points sampled. The blue points represent geometrically infeasible points.

2.4.2 View Boundaries

EASAL allows the user to inspect the points that form the boundary between a region and its parent/child. The boundaries option allows the user to inspect the boundaries of the Cayley space. The user can step through each of the boundaries using arrows provided.

2.4.3 Step Size

EASAL uses Cayley grid sampling to sample the Cayley space. The user can specify this step size in the input window (see Fig. 3). Another option available to the user is to choose the *dynamic step size* option in the advanced window (see Fig. 4). This tells EASAL to determine the step size based on the volume of the space it is sampling. Once a space has been sampled, there is always the option of refining the sampling. This can be done by selecting the *refine sampling* option in the atlas view whereupon EASAL halves the step size and re-samples all the sampled spaces.

2.5 Cartesian Realization

The realization view (see Fig. 7) in EASAL shows the Cartesian realization of the Cayley points.

2.5.1 View the sweep and flips of the Cartesian realization

The sweep feature of a realization keeps one of the point sets units fixed and draws all the possible orientations the other point set can take relative to the first one (see Fig. 7). The user can view the sweep. When the sweep is being displayed, the user can use the arrow keys to view the different flips.

2.5.2 View realization along boundaries

Clicking on the *boundaries* control allows the user to view the realizations along the boundary region. The user can view it walk through the boundary realizations using the arrows provided. Different colors are used to indicate the realizations along different boundaries.

2.5.3 View all realizations

The user can use the video controls at the bottom (see Fig. 7) to display all the realizations of a region one after the other. This differs from the sweep view in that sweep view shows all the realizations at the same time.

3 Dependencies and installation

This section discusses all the dependencies EASAL requires to run.

- Though technically, this should work on any UNIX variant platform with little to no modifications, it has been thoroughly tested to work on the following platforms.
  - Ubuntu 12.04 or higher.
  - Fedora 22 or higher.
  - OS X 10.8 Mountain Lion.

- We use Version 2.0 of the Eigen library for linear algebra computations. All necessary files pertaining to Eigen required by EASAL are provided with the source code in the *include* directory.
- For the GUI, we use OpenGL for visualization and the opensource fox-toolkit Version 1.6 for windowing. See the installation section for instructions on installing the necessary libraries.
- We use simpleini to read the settings from the settings.ini file. All necessary files pertaining to simpleini are provided with the source code in the include directory.
- EASAL has been tested on the NVIDIA graphics card with the NVIDIA proprietary driver (Version 331 or higher).
- EASAL is written in C++ and hence requires G++ Version 4.8 or higher or Clang Version 3.8 or higher to compile the source code.
- EASAL uses the GNU Make utility to compile the source files. Make Version 4.1 is required.

3.1 Installation

The software can be run either as a terminal application or with its associated GUI. The steps for running each of these versions has been explained here. You will need to download and install some third party libraries and update the makefile provided before you can successfully compile and run EASAL in the GUI mode.

- Install GLUT
  - On Ubuntu
    - sudo apt-get install freeglut3 freeglut3-dev binutils-gold
  - On Fedora
    - sudo yum install freeglut-devel mesa-dri-drivers mesa-libGL
- Install FOX-Toolkit (Version 1.6)
  - Download the fox library headers from `http://fox-toolkit.org/` and extract it.
  - On Ubuntu
    - sudo apt-get install libfox-1.6-0 libfox-1.6-dev
  - On Fedora
    - Download fox toolkit from `http://fox-toolkit.org/`
    - Extract it using "tar -xvzf libFox-1.6.X.tar.gz"
    - Run configure using "su ./configure"
    - Install using "su make install"
- Install GNU Make
  - On Ubuntu
    - sudo apt-get install make
  - On Fedora
    - yum install make
- Edit the Makefile to point to your versions of FOX.
  - Edit the line "include_dirs = -I /usr/lib/fox-1.6.50/include/ -I ./include/"
    and replace
    /usr/lib/fox-1.6.50 with the location you have downloaded and extracted the fox headers.
- Run *make* from the root/build directory.
- To run EASAL run *"bin/EASAL"* in a terminal from the root/build directory.

4 Sample Run

This section shows a sample run of EASAL in both the Terminal mode and the GUI mode.

4.1 Terminal Mode

When the software is run in the terminal mode, all input is specified using the settings.ini file.

4.2 GUI mode

- Run EASAL from the command line by running the following command. *$bin/EASAL*

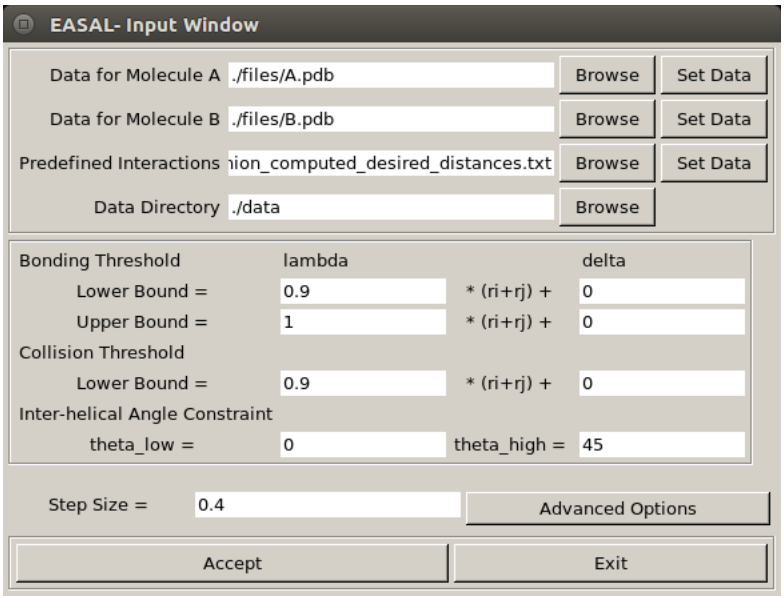

Figure 3: - In the input window, select the following either using the browse option or by entering the text in the text box provided
- Data for Molecule A - files/A.pdb
- Data for Molecule B - files/B.pdb
- Distance Data - files/source_files/union computed desired distances.txt
- Data Directory - data/
- Enter the values for Bonding Thresholds and step size .

- The user can then click on the *advanced options* button to set advanced user inputs. This opens a new pop-up where the user can enter either enter the data or choose to accept the default values.
- Click on *accept*. This opens the Atlas View.
- In the Atlas View, we initially see a root node at the center of a 3D grid. As and when more nodes are discovered, they are populated on the Atlas.
- When the tree view is on, it shows only the ancestors and descendants of the node selected. When it is off, it shows the entire atlas. This can also be achieved by clicking the Tree control at the bottom.
- In this view, the user can control how the sampling proceeds by using any of the controls on the left side of the atlas. The different controls available are (in the order they appear) - Stop Sampling. -

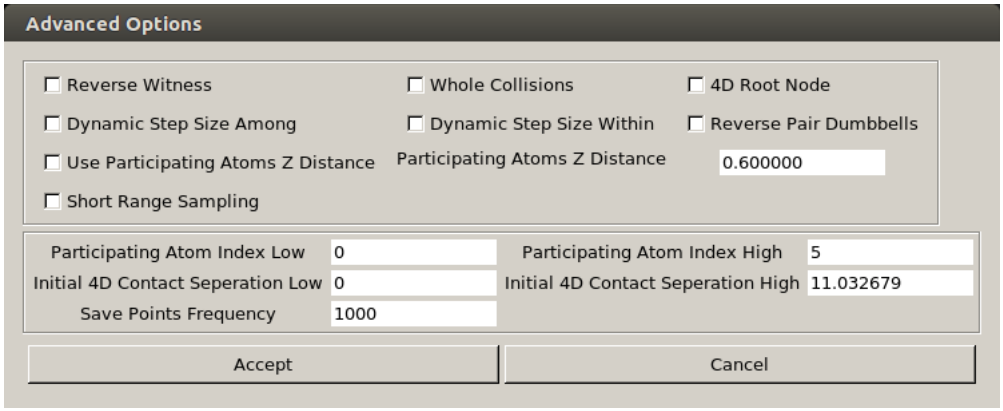

Figure 4: Advanced Options Window

Constraint Selection Dialogue Box. - Sample the Current Tree. - Sample all Incomplete Trees. - Auto-Solve. - BFS Sampling. - Refine Sampling.

- Clicking on a particular node does the following - Loads the Active Constraint Graph at the bottom left corner. - Loads the Cartesian realization for that node in the top right corner if the sampling for the node is complete.
- Pressing the space bar after selecting a node takes the user to the Cayley space view of that node.
- In the Cayley Space View, initially green points are shown which correspond to all the realizable points in the Cayley space.
- Clicking on the Red square at the bottom, shows the points that have collision. Clicking the red also shows two more options viz, cyan and pink. The Cyan points represent the points which have angle collision and the pink points represent points which have distance collision.
- Clicking on the Blue square shows all the points sampled including, the good, the collision and the unrealizable.
- Clicking on the Boundaries shows the boundary points and the user can step through them along each dimension.
- Pressing the space bar here takes the user to the *realization view*.
- Pressing v on the keyboard generates the sweep view of the point set.
- Once the sweep view is generated, the user can use the up and down arrow keys to view all the flips of the point set.
- Clicking on boundaries shows the different realization along different boundaries.
- Clicking on the video controls at the bottom and clicking the play button on it animates and shows all the possible realizations of the point set.

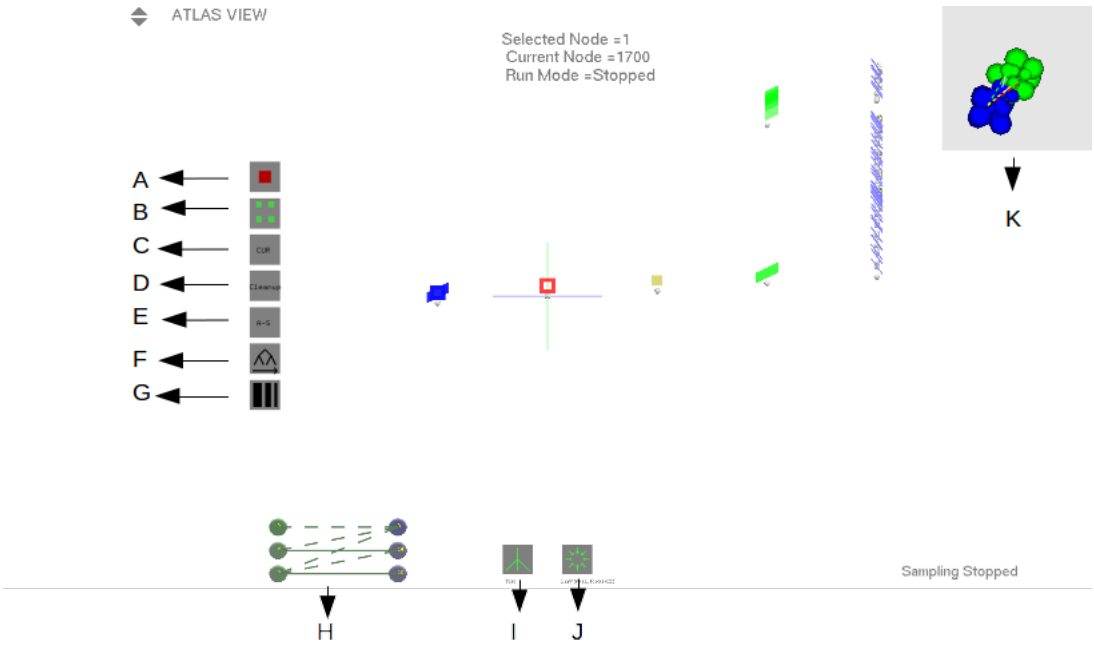

Figure 5: The Atlas View in EASAL . (A) Button to stop the sampling. (B) Button to start the constraint selection dialogue box. (C) Button to continue sampling the current tree. (D) Button to Clean Up Sampling. (E) Button to run EASAL in the Auto-Solve mode. (F) Button to start exploring the Atlas in a breadth first fashion. (G) Button to refine the sampling using a smaller step size. (H) Active Constraint graph of the node selected. The spheres with the indices denote the atoms. A thick line between the atoms indicates a bond and a dotted line indicates a parameter. (I) Button to toggle Tree View. (J) Button to toggle Gravity. (K) Cartesian realization of the current node.

5 Developer Guide for EASAL Software: Major Classes and Methods

5.1 Major Classes of `EASAL`

Fig. 8 gives an overview of the structure of the major classes of EASAL. Each of these classes are explained in the subsections below.

5.1.1 AtlasBuilder

The AtlasBuilder class populates the ActiveConstraintRegion for each activeConstraintGraph by sampling inside the boundaries of its ConvexChart. It creates and explores only regions that contain at least one Cartesian realization.

Major Attributes:

- **rootGraphs**: The set of all possible 4D or 5D ActiveConstraintGraphs of the root nodes generated before sampling.
- **atlas**: An atlas object that is populated by the AtlasBuilder by sampling. This object is shared between front-end and back-end of the algorithm.

Major Methods:

```
sampleAtlasNode
input  : atlasNode: node
output: Complete sampling of the atlasNode and all its children
H = node.activeConstraints
G_H = node.activeConstraintGraph
if G_H is minimally rigid then
| stop;
end
F = complete3Tree(G_H)
C = computeConvexChart(G_H, F)
for each cayleyPoint p within convexChart C do
| R = computeRealizations(p)
| for each realization r in R do
| | if !aPosterioriConstraintViolated(r) then
| | | if isBoundaryPoint(r) && hasNewActiveConstraint(r, G_H) then
| | | | e = newActiveConstraint(r, G_H);
| | | | G' := G_H ∪ {e} ;
| | | | if G' is not already present in the current atlas then
| | | | | childNode = new atlasNode(G')
| | | | | sampleAtlasNode(childNode);
| | | | end
| | | | else
| | | | | childNode = findNode(G');
| | | | end
| | | | node.setChildNode(childNode);
| | | end
| | end
| end
end
```

Algorithm 1: High level EASAL pseudocode

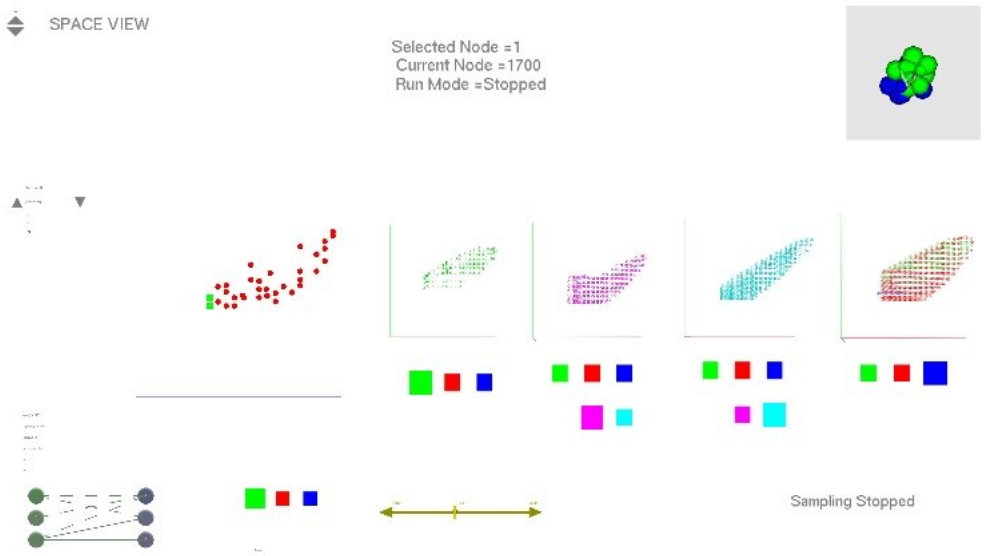

Figure 6: Cayley Space View. The points outside the box show the boundary points. The points in the box show the various Cayley points. From left to right Good points, Collision points, points which violate the steric constraints and all the points sampled.

- **startAtlasBuilding()**: For each of the generated root graph, creates an atlasNode labeled with a contact graph G_F where F is the set of contacts. Then calls the recursive sampleTheNode method for each of the root atlas nodes.

- **sampleTheNode(atlasNode)**: The exploration of the atlas is done by the recursive **sampleAtlasNode** algorithm (see Algorithm 1) using one of the generated atlas root nodes as input. This algorithm is implemented by the sampleTheNode method. Using depth first search this algorithm samples the atlas node and all its descendants.

  Base case of recursion: If active constraint graph G_H of the node is minimally rigid i.e., the active constraint region is 0-dimensional, we have no more sampling to do, return.

  The recursion step: If G_H is not minimally rigid, we use the **complete3Tree** algorithm to find find a set of parameters F so as to form a maximal 3-tree to leverage the convex parametrization theory [3]. This also ensures that $H \cup F$ is minimally rigid and easily realizable.

  The method computeConvexChart shown in the pseudocode finds the convex chart for the parameters F is done by the ConvexChart class explained later. Method ComputeRealizations computes the realization for a Cayley point and is done by the findRealizations method in the software. The aPosterioriConstraintViolated method which checks for angle and steric violations is implemented in the ConstraintCheck class explained later. Next we make a call to the findBoundary to detect boundaries and newly active constraints.

- **determineStepSizeDynamically()**: Finds out the step size s given T, the total number of samples. Each 5D atlas node has its own s computed by using the volume of the Cayley parameter space of the node over total number of samples per node. The volume of the Cayley parameter space of the node is approximately computed by exhaustive sampling within the exact chart without considering any constraints. The number of samples per node roughly can be computed by T over total number of root(starter) atlas nodes, m. The number of samples in child nodes are negligible since the volume of regions in low dimensional nodes are negligible compared to the regions of high dimensional nodes.

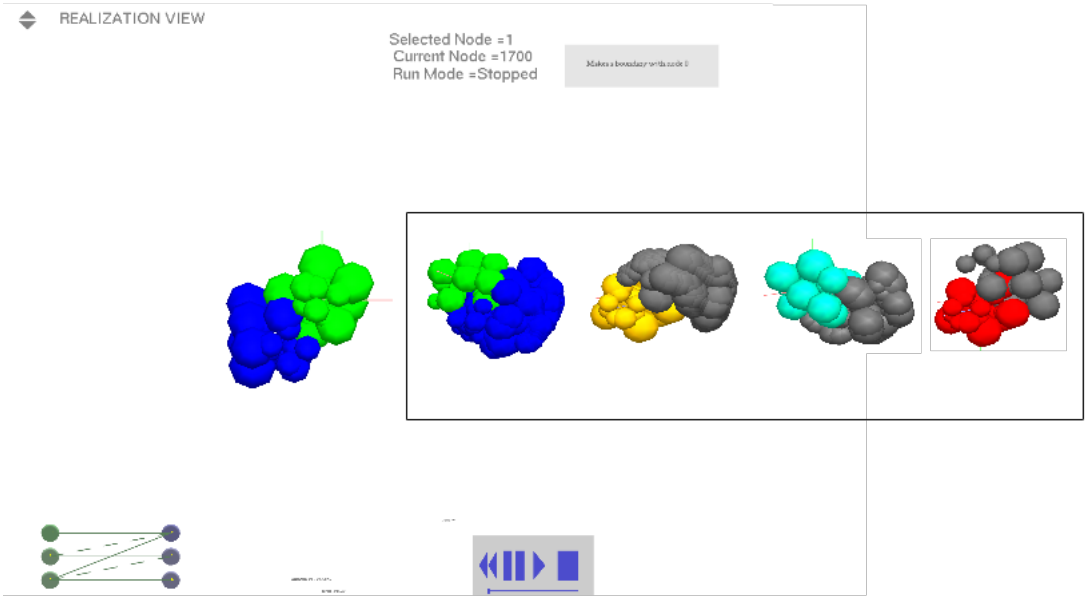

Figure 7: Realization View. The images in the box show the different realizations along different boundaries.

- **findBoundary()**: Boundary detection ensures that sampling stays in the feasible region and minimizes discarded samples. The findBoundary method which detects boundary points, checks for newly formed active constraints and makes function calls which in turn call sampleTheNode for a child region.

5.1.2 Atlas

The 'Atlas' class stores the directed acyclic graph that represents the relationship between active constraint regions.
Major Attributes:

- **nodes**: A vector of all AtlasNodes.
- **rootIndices**: The indices of all the root nodes in the atlas.

Major Methods:

- **search(node)**: Uses depth first search on the atlas to check whether the node exists in the atlas or not. It is used to avoid repeated sampling of the same region. The time complexity of the search is $O((\text{depth of the tree})) = O(6(k-1))$ which in our case is $O(1)$ since we fix k to be 2.

5.1.3 AtlasNode

AtlasNodes make up the Atlas. Each AtlasNode represents an active constraint region reperesented by an ActiveConstraintGraph.
Major Attributes:

- **acg**: The active constraint graph corresponding to the node.
- **region**: The set of Cayley points in the active region.
- **connection**: The id of the nodes in the atlas that represent the boundary of this node's region.

5.1.4 ActiveConstraintGraph

The ActiveConstraintGraph class is used to store the set of active constraints.
Major Attributes:

- **activeConstraints**: The set of atom index pairs that represent contacts.
- **verticesA**: Participating atom indices from first molecular unit.
- **verticesB**: Participating atom indices from second molecular unit.
- **parameters**: A vector of atom index pairs that represent parameters.

Major Methods:

- **completeTo3by3Graph()**: Adds atoms to make sure there are at least 3 atoms from each molecular unit so that the graph is realizable. While choosing additional atoms, it has 2 options, choosing the atoms closest to each other or the atoms that lead to a user specified angle.

5.1.5 ActiveConstraintRegion

The ActiveConstraintRegion class contains the set of feasible Cayley points generated by sampling.
Major Attributes:

- **space**: The set of feasible Cayley points.
- **witspace**: The set of feasible witness Cayley points obtained from an ancestor node.

Major Methods:

- **convertSpace(activeConstraintRegion)**: Re-parametrizes a region using an input regions parameters. This method converts each Cayley point in the input activeConstraintRegion to the Cayley point parametrized by the input region's parametrization.

5.1.6 CayleyPoint

The CayleyPoint class represents a multi-dimensional point in the Cayley parameter space and stores the corresponding Cartesian space orientations of the molecular unit.
Major Attributes:

- **data**: Values of the Cayley parameters (non-edge lengths).
- **orients**: The set of Cartesian space Orientations of the molecular unit that were computed by realizing the active constraint graph with the given length of the edges and non-edges.

5.1.7 Orientation

The Orientation class is the Euclidean transformation of molecular units. The Orientation class stores only the information necessary to compute the transformation matrix that will yield a Cartesian realization for the entire molecular unit.
Major Attributes:

- **FromB**: Cartesian coordinates of three atom markers from the first molecular unit before the transformation.
- **ToB**: Cartesian coordinates of three atom markers from the second molecular unit after the transformation.
- **connections**: The set of node indices that this orientation belongs to. An orientation be on the boundary of multiple regions.

5.1.8 CayleyParameterization

The CayleyParameterization class chooses non-edges in an ActiveConstraintGraph that convert the graph into complete 3-tree. Those non-edges are called the parameters. The complexity of the sampling algorithm varies based on the choice of non-edges and the order in which they are fixed.
Major Attributes:

- **partial3tree**: A boolean variable indicating whether an ActiveConstraintGraph is partial 3-tree or not.
- **parameters**: The set of atom marker pairs that represent non-edges.
- **tetrahedra**: The ordered tetrahedron set that helps in defining the order of parameters that is required during the sampling procedure. This data is later passed to ConvexChart and CartesianRealizer to help in their computations.
- **updateList**: Adjacency map containing the dependency of parameters. It provides the set of parameters whose range will be updated when one of the parameters is fixed.
- **boundaryComputationWay**: Inequalities that express the range of a parameter can be classified into either a linear or non-linear class. This variable is the characterization of the parameter that tells what inequality is needed to compute the parameter range i.e., triangular or tetrahedral inequality.
- **complete3trees**: The set of complete 3 trees.

Major Methods:

- **defineParameters()**: The parameters of an active constraint graph are selected as maximal 3-realizable (3-tree) extensions by leveraging the convex parametrization theory. [3]. It creates a look-up table containing all possible complete 3-trees. We find a graph in the look-up table so that active constraint graph is a proper subset of either the graph or one of its isomorphisms.
- **parameterMinDeviation()**: An alternate way to pick the parameters for 5D regions is by ensuring that the range of each parameter is similar. The aim here is to sample more uniformly in the Cartesian space.
- **built3tree()**: The 3-tree formed by starting with a 4-vertex complete graph and repeatedly adding vertices in such a way that each added vertex is edge-connected to the face of a tetrahedron. Store the tetrahedrons in the order they are created in the attribute **tetrahedra**.

5.1.9 ConvexChart

The ConvexChart class is used to determine the chart that parameterize the regions i.e., it computes the range of parameters of ActiveConstraintGraph. An exact convex chart yields feasible Cayley points for the current active constraint region. The resulting Cayley configuration space is convex, before collisions or other (e.g. angle) constraints are introduced. The range of parameters are computed by triangle and tetrahedral inequalities.
Major Attributes:

- **param_lengthUpper**: The upper bound of the parameters' range
- **param_lengthLower**: The lower bound of the parameters' range
- **param_length**: current value of parameters

Major Methods:

- **initializeChart()**: Initalizes the boundaries of convex chart. Tighter bounds are given in [1].

- **computeRange(v1, v2)**: Computes the range of the parameter $v1-v2$ in order to eliminate sampling infeasible grid points. Range computation is required in every iteration for dependent parameters.
- **setRangeByTriangleInequality(v1, v2)**: Computes the range of the non-edge $v1 - v2$ through triangular inequalities.
- **setRangeByTetrahedralInequality(v1, v2, tetrahedron)**: Computes the range of the non-edge $v1 - v2$ through tetrahedral inequality.
- **stepGrid**: Sets parameter point to the next grid point within the computed range.
- **stepNeighbour()**: Sets the parameter point to the neighbor grid point in all dimensions consecutively.
- **stepGridBinary()**: Sets the parameter point to somewhere between current point and neighbor grid point according to binary search procedure in findBoundary.

5.1.10 CartesianRealizer

The CartesianRealizer class contains routines that compute orientations that represent transformations of rigid helices relative to each other. It computes Cartesian realization of an active constraint graph with the parameter lengths taken from cayleyPoint and active constraint lengths for a specific flip. **Major Attributes:**

- **positions**: Cartesian coordinates of vertices in ActiveConstraintGraph.
- **edge_length**: Contains all fixed distances plus current distance values of non-edges of ActiveConstraintGraph.

Major Methods:

- **computeRealization(activeConstraintGraph, convexChart, flipno)**: Computes the Orientation by leveraging partial 3-tree techniques. activeConstraintGraph which is a complete 3-tree is built up from a base tethedra by adding, at each step, a new vertex edge-connected to the face of a tetrahedron.
- **setBaseTetra(tetrahedron)**: Finds Cartesian coordinates of the vertices of tetrahedron by known edge lengths.
- **locateVertex(vertex, face)**: Finds Cartesian coordinates of the vertex that is connected to the face of a tetrahedron.

5.1.11 ConstraintCheck

The ConstraintCheck class is designed to check whether any non-active constraints become active. Users have the option to define a set of constraints of interest. In which case, the new constraint activation check is done only for these. For an input Orientation, ConstraintCheck first computes the Cartesian realization for the entire molecular composite then passes it to other subroutines to perform user specified constraint check such as steric constraints or angle constraints.

References

[1] Ugandhar Reddy Chittamuru. Efficient bounds for 3d cayley configuration space of partial 2-trees, 2010.

[2] Aysegul Ozkan, Ruijin Wu, Jorg Peters, and Meera Sitharam. Efficient atlasing and sampling of assembly free energy landscapes using easal: Stratification and convexification via customized cayley parametrization. (on arxiv), 2014.

[3] Meera Sitharam and Heping Gao. Characterizing graphs with convex and connected cayley configuration spaces. *Discrete & Computational Geometry*, 43(3):594–625, 2010.

[4] Meera Sitharam, Aysegul Ozkan, James Pence, and Jörg Peters. Easal: Efficient atlasing, analysis and search of molecular assembly landscapes. *CoRR*, abs/1203.3811, 2012.

[5] Ruijin Wu, Aysegul Ozkan, Antonette Bennett, Mavis Agbandje-McKenna, and Meera Sitharam. Prediction of crucial interactions for icosahedral capsid self-assembly by configuration space atlasing using easal. (on arxiv), 2014.

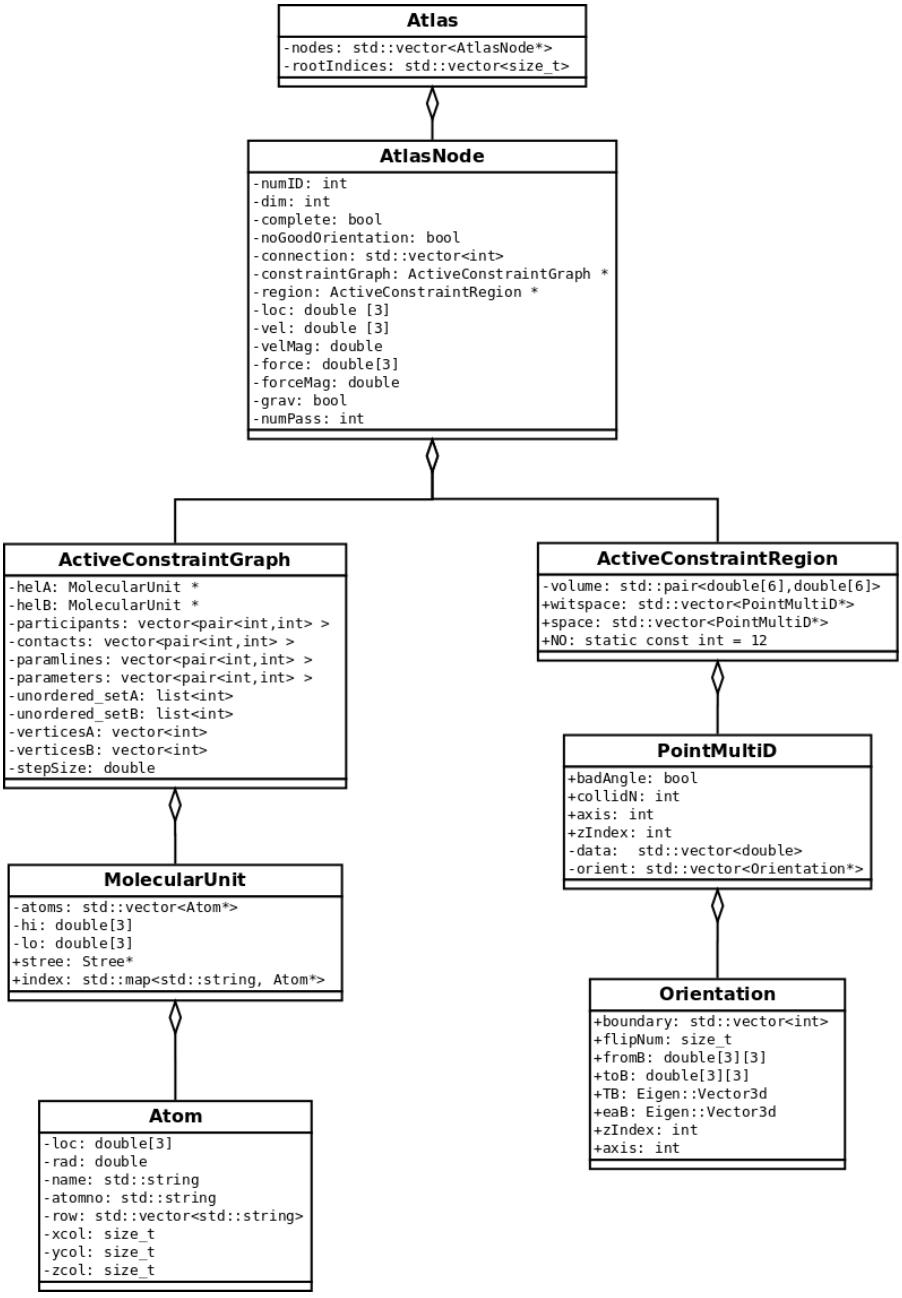

Figure 8: EASAL UML Diagram