EASAL User Manual: User Interface and Functionalities

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

August 9, 2016

This user guide accompanies the EASAL software described in [2] which implements a suite of algorithms that elucidate the structure and geometric properties of configuration spaces in molecular assembly. The algorithms presented use new theoretical results by some of the authors for: stratification of configuration spaces, efficient sampling by convex parametrization, and intuitive visualization. Technical concepts and definitions required to use this software can be found in the main paper.

1 Introduction

EASAL elucidates the structure and geometric properties of configuration spaces in molecular assembly [5, 3, 6]. EASAL implements algorithms that use new theoretical results, some of which are presented in [2, 3, 4, 5]. EASAL is open source and can be downloaded from `https://www.bitbucket.org/geoplexity/easal`.This user guide describes in depth the features and options available 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 demonstrating EASAL in action is available at `http://www.cise.ufl.edu/~sitharam/EASALvideo.mpg`. Though this version of the video is slightly dated in that it uses an older version of the software (only the GUI in the Atlas View has changed since), it should serve as a good resource to gain familiarity with EASAL. A newer version of the video is being created and will be uploaded soon to `http://cise.ufl.edu/~rprabhu/EASALvideo`.

2 Software Functionalities

This section introduces the main functionalities of EASAL and gives details of how it has been implemented in the software.

2.1 Input

The input to EASAL consists of the following:

- Two rigid molecular units.
- The pairwise component of the potential energy or enthalpy function.
- The global, non-pairwise component of the potential energy is modeled as the angle constraint between the principal axes of the two rigid molecular units.

These can be specified in the input window which the user is presented with when EASAL is first run (see Fig. 5).

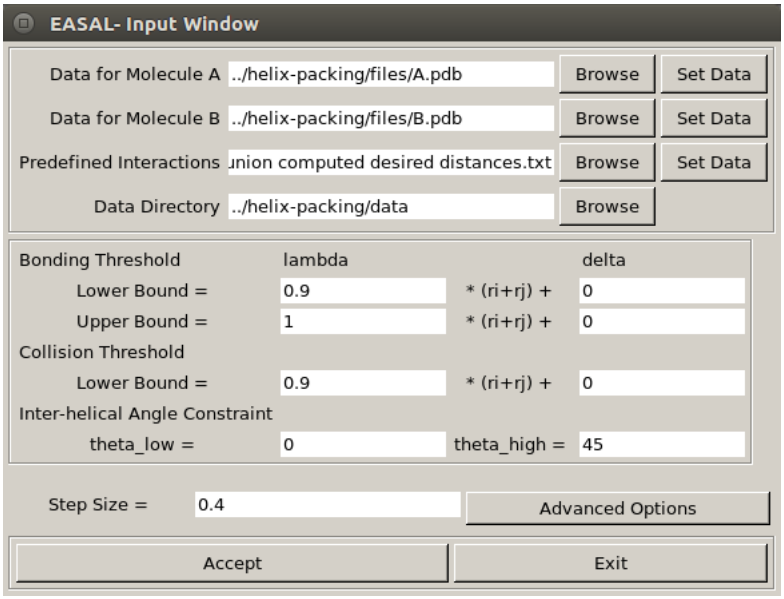

Figure 1: Input Window

- **Data for Molecule (A/B)** This accepts the molecular data describing the molecular composites and their 3D structure pdb format. The user can either type in the location of the input molecular 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 molecular data before starting sampling.
- **Predefined Interactions:** Predefined Interactions is information about the minimum and maximum distances between atom pairs participating in the bonding. The input file for this can either specify pair-wise distance for each atom pair participating in the reaction from either molecules or only specify distances of atoms the user is interested in.
- **Inter-helical Angle Constraint:** This specifies the upper and lower bound for angle constraints between the two atoms. This forms the global component of the energy function.

2.2 Geomtrization

The Lennard-Jones potential function is typically discretized to take constant values on three intervals where the repulsive forces dominate, the at- tractive forces dominate and neither dominates. All these bounds are specified as part of the input to the assembly model for each pair of atoms. Using the Input window, the user can specify these distances.

- **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 $\lambda * (r_i + r_j) \pm \delta$. Where 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.
- **Collision Threshold:** This is the minimum distance between atoms below which atoms will collide. This too is given as $\lambda * (r_i + r_j) \pm \delta$.

2.3 Stratification

The Atlas View visualizes the stratification of an assembly configuration space. Strata of each dimension for the assembly constraint system are visually represented as nodes of one color (see Fig. 2). The stratification

is a directed acyclic graph where each node represents an active constraint region represented by an active constraint graph. Edges indicate containment in a parent region one dimension higher.

The Atlas View shows the atlas as it is being built, as a dynamic forest of trees. It provides controls that allow the user to do real-time intervention at any stage of the sampling process; to halt, redirect, and resume the sampling process and to access the atlas for different queries or views.

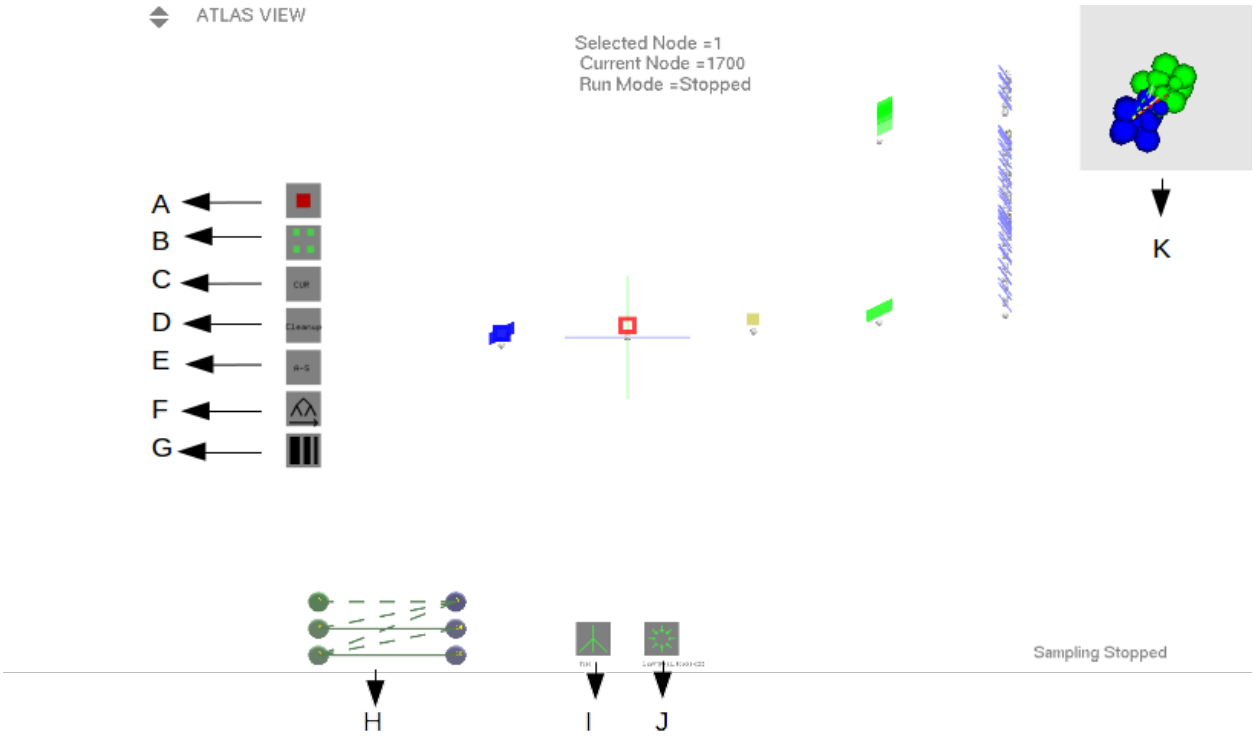

Figure 2: The Atlas View in `EASAL` . (A) Button to stop the sampling. (B) Button to start the constraint selection dialogue box. (C) Button to continue sampling the current tree. (D) Button to Clean Up Sampling. (E) Button to run EASAL in the Auto-Solve mode. (F) Button to start exploring the Atlas 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.

2.3.1 Sampling

`EASAL` generates an *atlas* [5] of the configuration space. This atlas is a forest of trees where each d-dimensional node represents an active constraint region of dimension d. Each tree in the atlas has has either a four or five dimensional root node and each successive level of the tree has nodes of one dimension lesser than the previous level. The atlas is generated by sampling the configuration space in any of the following sampling modes.

- **Auto-Solve:** This is the default sampling mode. When starting afresh in this mode, `EASAL` generates all possible root nodes of either four or five dimensions depending on the user input. Then it proceeds to recursively sample all these nodes till the 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.

- **Current Tree:** In this mode of sampling, `EASAL` starts from the node currently selected and samples its tree to completion.
- **Clean Up:** In this mode, `EASAL` samples all incomplete trees to completion.
- **Ad-hoc:** In this mode, the user can select any node of any dimension using the "constraint selection dialogue box" and sample the sub-tree rooted at that node.
- **Breadth First:** While all other sampling modes sample the tree depth first, this mode samples the tree breadth first. Starting from a selected node, this mode samples the sub-tree rooted at that node in a breadth first fashion.
- **Refine Sampling:** The user can use this mode for a more refined level of sampling. In this mode, we re-sample all the completed nodes with a step size equal to half the original step size.

2.4 Convex Cayley Parametrization

The space view shows the active constraint region in the Cayley parametrized chart representation for a particular node. Clicking a node loads its active constraint region to RAM and the space view is brought when user press enter key. The parametrized chart view (Fig. 3) shows green cubes (such that the cube size is proportional to the step size chosen) where parameters do not result in collision. Clicking a cube displays, in the upper right corner, a Cartesian realization associated with the parameter values (respec- tive Cayley point). (EASAL can also display parameters resulting in irreconcilable inter- molecular collisions or ones that are not realizable.) For more than three dimensions, arrow-controlled sliders select and display 3D slices. Since interior points are easily occluded, the third dimension can also be switched to a slider. The left side of the view enumerates newly formed lower dimensional boundaries of the active constraint region and enumerates color-coded boundaries.

2.5 Cartesian Realization

The realization view shows the active constraint region in the Cartesian realizations representation for a particular node. The realization view (Fig. 4) shows realizations with constraints and parameters displayed as lines. Valid realization flips are shown on the lower right side of the view. One option to view the valid parameters in the order of detection is via video controls (bottom right) with reverse, pause, play and stop options.

3 User Interface

3.1 Input Window

This section describes all the input options not previously described.

- **Data directory:** This is the directory where the atlas is stored. If `EASAL` finds a partially generated atlas at this location, it presents the user with 3 options. The user can choose to (i) Continue sampling the old atlas, in which case the old atlas is loaded. (ii) Overwrite the existing data and start fresh sampling. (iii) Go back and choose a different location for the atlas.
- **Step size:** This is the step size used for sampling the Cayley Space.
- **Advanced Options:** This gives a set of advanced options to the user.
  - **Reverse Witness:** Some nodes in the atlas do not have convex parameterizations and hence cannot be directly populated. However these can be populated indirectly from a child node encountered through another path. When this option is enabled, `EASAL` adds a witness point to all parent nodes found this way.

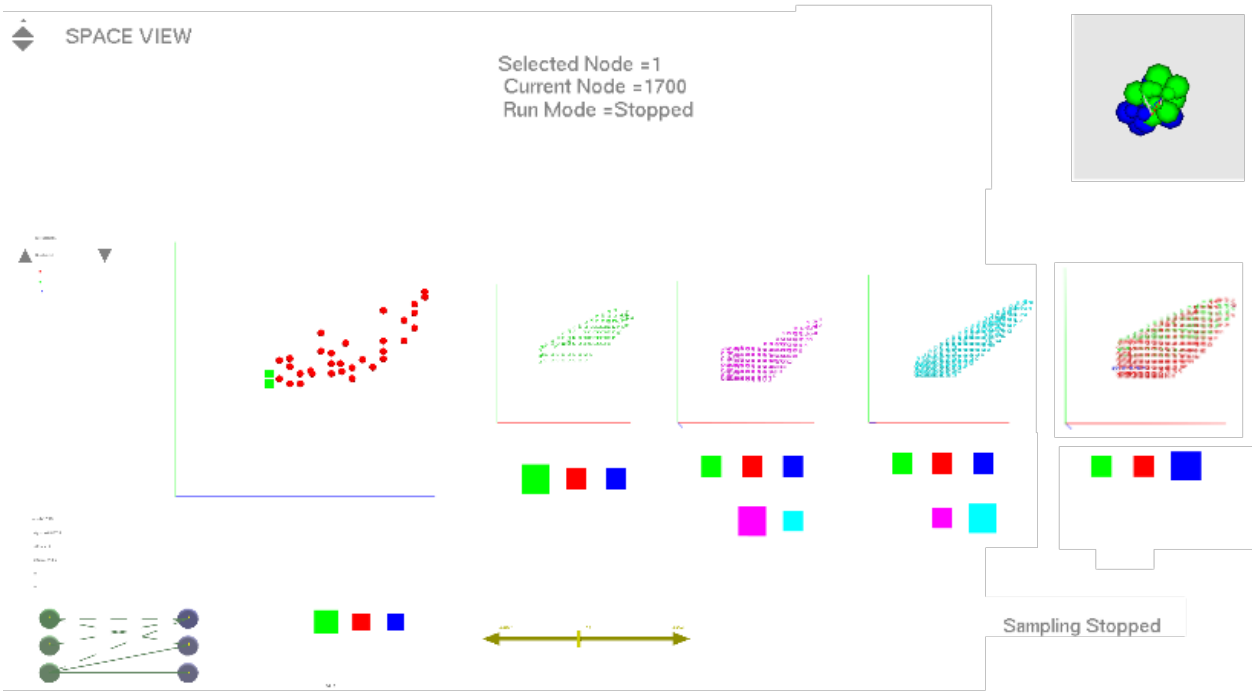

Figure 3: Cayley Space View. The points outside the box show the boundary points. The points in the box show the various cayley points. From left to right Good points, Collision points, points which violate the steric constraints and all the points sampled.

- **Whole Collisions:** By default, `EASAL` checks for collisions between every pair of atoms. When the input atoms are large, this may not be feasible. Whole Collisions provides a way of checking for collisions between a group of atoms.
- **4D Root Node:** Choosing this option makes the root nodes in the atlas correspond to 4 dimensional active constraint regions instead of the default 5 dimensional regions.
- **Dynamic Step Size Among/Within:** When chosen, uses a step size dynamically determined based on the volume of the region to be sampled instead of the user supplied step size. Among chooses dynamic step size for every parameter. Within choose a dynamic step size just for one parameter.
- **Reverse Pair Dumbbells:** For the $4D$ case, by default we choose the initial contacts so that the atom with the lower number on one molecule pairs up with the atom with the lower number on the other molecule. Using Reverse pair dumbbells removes this restriction.
- **Use Participating Atom Z Distance:** This option limits the z-axis distance between atoms participating in the bonding.
- **Participating Atom Z Distance:** This option specifies the threshold for participating atom z-distance.
- **Short Range Sampling:** This allows for flexible sampling.
- **Participating Atom Index Low/High:** These force the constraints to be picked from the middle of the molecules. Low and High are the atom numbers between which the user wants the constraints to be picked.
- **Initial 4D Contact Separation Low/High:** In the $4D$ case, this is the maximum separation between atoms in the same molecule that form a dumbbell.

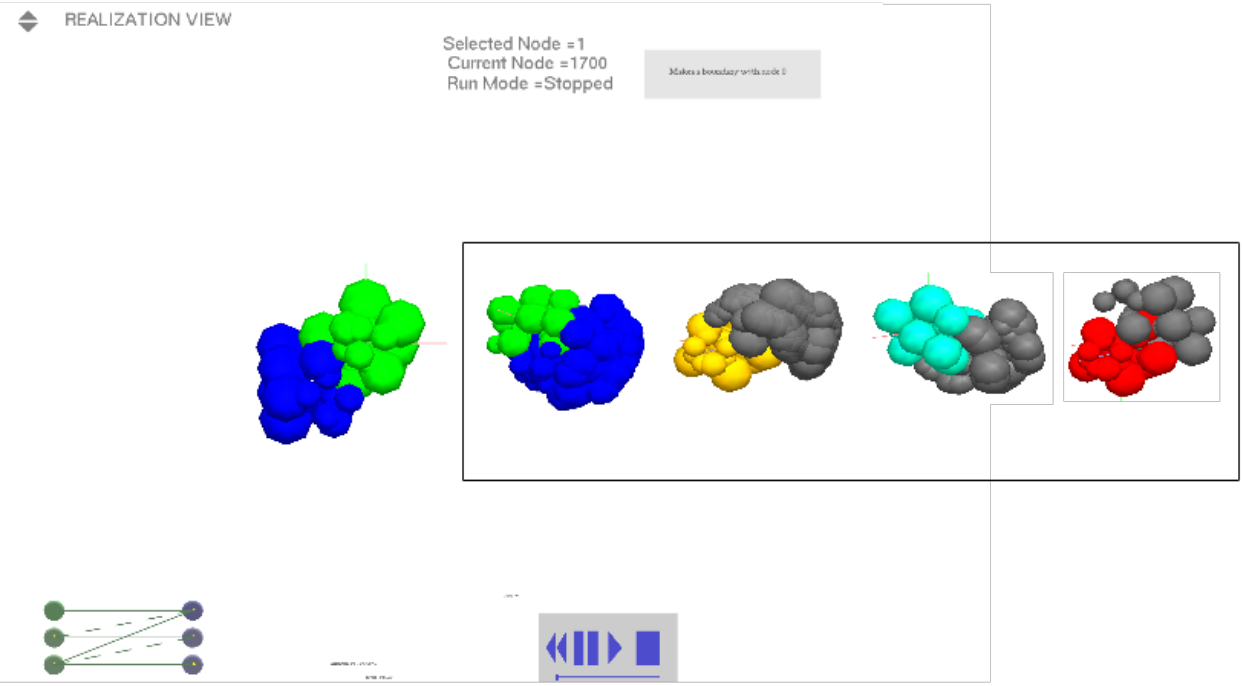

Figure 4: Realization View. The images in the box show the different realizations along different boundaries.

– **Save Points Frequency:** The frequency with which sampled points are saved to a file in the data directory.

Clicking Accept on the Input Window takes the user to the Atlas View of the software where sampling starts in the default Auto-Solve mode.

3.2 Atlas View

- **Stop Sampling:** Clicking this button stops the sampling of the Atlas and presents options to redirect the sampling.
- **The Constraint Selection Dialogue box** (see figure 6) 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 is more relevant to them. The user can select a node by either specifying a node number or by specifying active constraints between the molecules. The spheres with the indices denote the atoms. A thick line between the atoms indicates a bond participating in the reaction. 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 the partially sampled trees.
- **A-S:** A-S stands for Auto-Solve. This button starts sampling in the Auto-Solve mode.
- **BFS:** Clicking this button forces `EASAL` to explore the atlas in a breadth first fashion which it otherwise does in a depth first fashion.

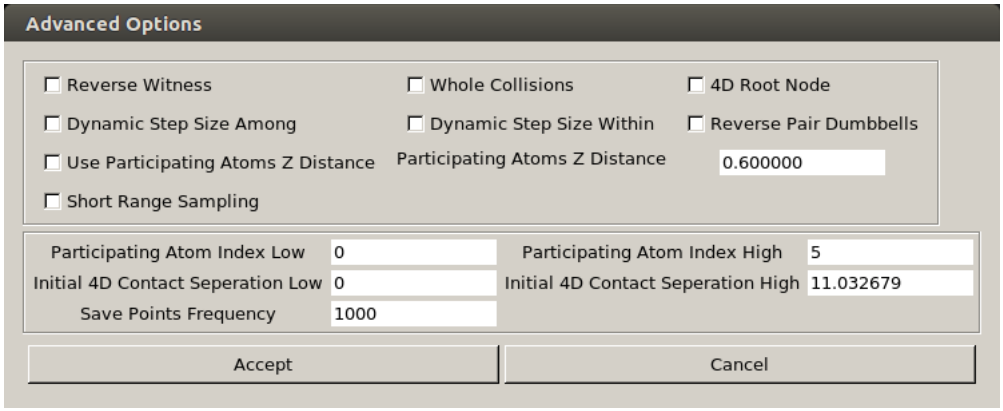

Figure 5: Advanced Options Window

- **Refine Sampling:** On clicking this button `EASAL` starts sampling with half the current step size.
- **Tree:** Clicking 'Tree' after selecting a particular node from the atlas shows us only those nodes that are either the ancestors or the descendants of the node selected.
- **Gravity:** `EASAL` implements a spring-repulsion algorithm that yields a spacious layout of the nodes in atlas. This button enables/disables the spring-repulsion algorithm.
- **Keyboard Shortcuts:**
  - **'+'** - Zoom into the atlas.
  - **'-'** - Zoom out of the atlas.
  - **'f'** - Toggle forces. `EASAL` implements a spring repulsion algorithm to display the atlas. This toggles the spring repulsion on and off.
  - **'l'** - Enables level alignment. Enabling this arranges the nodes of the atlas according to their dimensions.
  - **'i'** - Align view to x axis.
  - **'j'** - Align view to y axis.
  - **'k'** - Align view to z axis.
  - **'t'** - Toggles the tree view. When the tree view is on, it shows only the tree the the currently selected node is part of. When it is off, it shows the entire atlas. This can also be achieved by clicking the Tree control at the bottom.
  - **'1'** - Toggles the display of node numbers.
  - **'w'** - Toggle hide/show atlas edges.
  - **'a'** - Toggles between showing and not showing nodes with no realizations. By default, nodes which do not have any realizations are not shown in the atlas.
  - **'s'** - `EASAL` saves the atlas periodically depending on the 'save points frequency' set by the user. Pressing 's' forces it to save the atlas to the disk even if it is in between two save cycles.

3.3 Space View

This section discusses all the user options available to explore the Cayley Space.

- **View dim:** Clicking on this allows the user to step through points in the Cayley space in each dimension. A slider is provided to visualize points in higher dimensions.

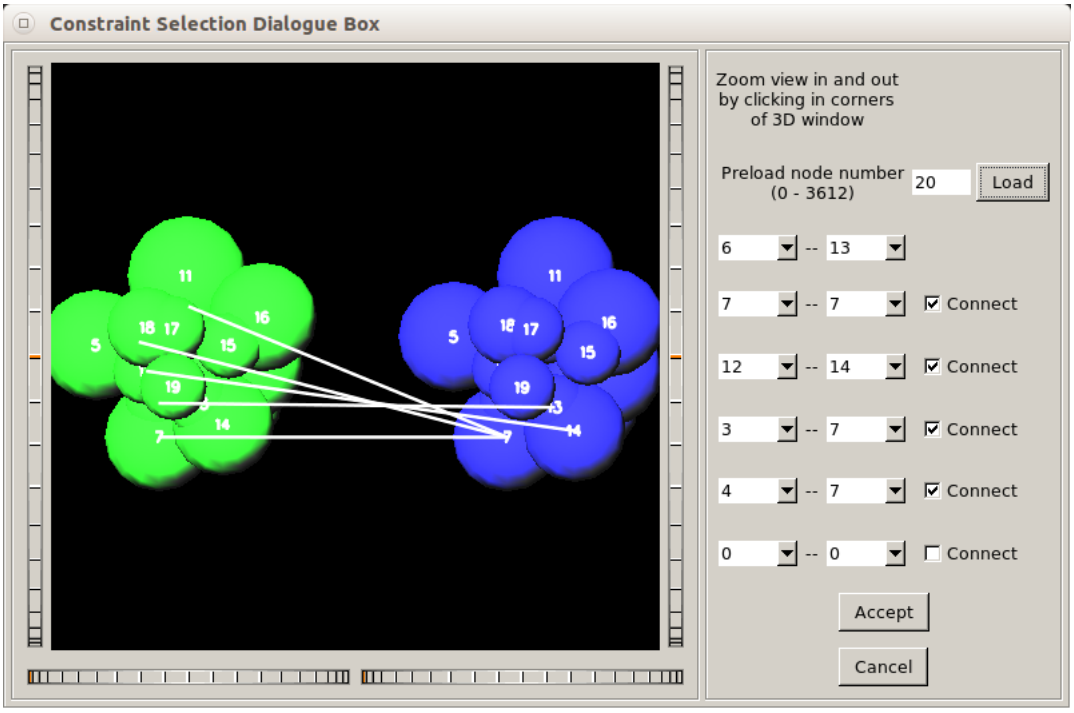

Figure 6: Constraint Selection Dialogue

- **Boundaries:** allows the user to inspect the boundaries of the Cayley space. Different colors are used to represent different boundaries in the Cayley Space. The user can step through each of the boundaries using arrows provided.
- **Categorization of Cayley points:** In the Calyey 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 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.
- **Keyboard Shortcuts:**
  - **'6'** - Switches the sliders to 6-d view.
  - **'3'** - Switches the sliders to 3-d view.
  - **'2'** - Switches the sliders to 2-d view.
  - **'b'** - Toggles showing the boundary points.
  - ',' / '<' - Cycles through the different boundary points downwards.
  - '.' / '>' - Cycles through the different boundary points upwards.

3.4 Realization View

This section discusses all the user options available to explore the realization.

- Pressing 'v' generates a sweep of all possible realizations for the node selected. Pressing 'v' again generates an instance of realization for the Cayley Point selected in Space View. The user can step through all possible flips by pressing the up and down arrow keys.

- Clicking on boundaries generates the sweep of all realizations along different boundaries of the Cayley space.
- Video controls at the bottom allow for animation of all the realizations and their flips.
- Keyboard Shortcuts
  - **'v'** - Generates a sweep of all possible realizations.
  - **numbers 0-7** - Jump to flip number pressed.
  - **'s'** - Toggle show and hide the whole s-tree.
  - **'l'** - Show the bonds between the atoms.

4 EASAL Software Architecture and Pseudocode

4.1 Major Classes and Architecture of EASAL

Fig. 7 gives an overview of the structure of the major classes of EASAL. The core classes are AtlasBuilder, ActiveConstraintGraph, CayleyParameterization, ConvexChart, and CartesianRealizer.

4.1.1 Atlas

The 'Atlas' class is the directed acyclic graph that represent the relation of active constraint regions
Major Attributes:

- **atlas**: The set of all AtlasNodes
- **atlasRoot**: The set of AtlasNodes that are at the root of a tree.

Major Methods:

- **search(node)**: Depth first search the atlas to check whether the node exists in atlas or not. It is used to avoid repeated sampling of the same region. The time complexity of the search is $O(log(\text{depth of the tree})) = O(log(6))$ considering we stop creating children nodes with overconstrained ActiveConstraintGraphs.

4.1.2 AtlasNode

AtlasNodes make up the Atlas. Each AtlasNode represents an active constraint region labeled by 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.

4.1.3 ActiveConstraintGraph

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

- **activeConstraints**: The set of atom marker index pairs that represent contacts.
- **verticesA**: Participating atom marker indices from first molecular unit.

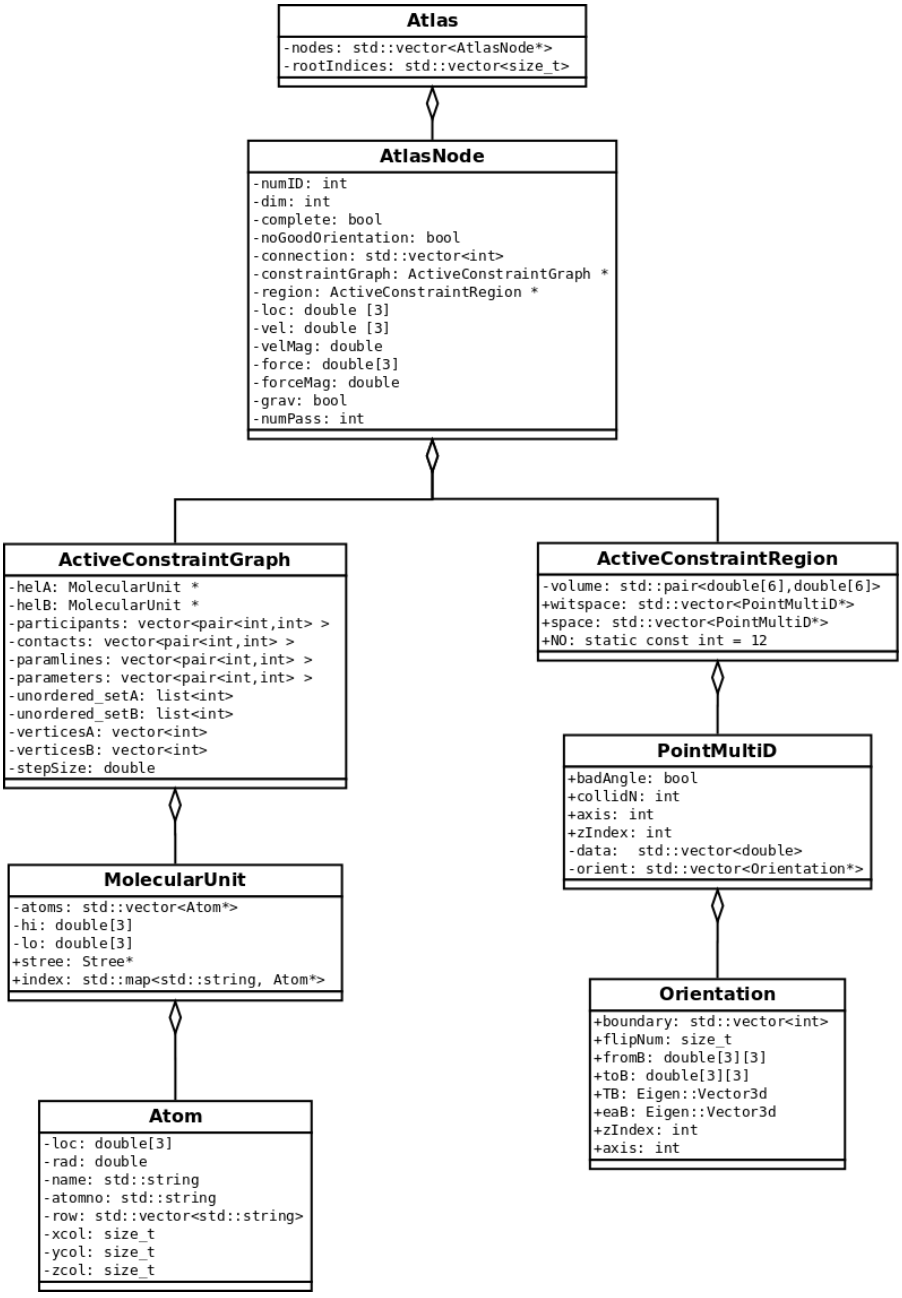

Figure 7: EASAL UML Diagram

- **verticesB**: Participating atom marker indices from second molecular unit.
- **parameters**: The set of atom marker index pairs that represent parameters

Major Methods:

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

4.1.4 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 got from an ancestor node.

Major Methods:

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

4.1.5 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.

4.1.6 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 to Cartesian realization for the entire molecular unit.
Major Attributes:

- **FromB**: Cartesian coordinate of three atom markers from the first molecular unit before the transformation.
- **ToB**: Cartesian coordinate 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.

4.1.7 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 to track whether an ActiveConstraintGraph is partial 3-tree or not.
- **parameters**: 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. [**?**]. It sets parameters by checking if the contacts of current active constraint graph is a subset of any of the isomorphisms of the graphs in the list of complete 3 trees. If found then the edges in that graph (except the contacts) will be used as the parameter set. For this purpose a look up table is created, that holds all complete 3 trees.
- **parameterMinDeviation()**: An alternate method to pick the parameters for 5 dimensional regions such that the range of each parameter has similar size. It aims at sampling more uniformly in the Cartesian space by setting Cayley parameter space as the order of non-edges can guarantee efficient sampling where finding bounds for each non-edge is either solving a linear inequality or solving a single quadratic of one variable [1]. Here we use a simplified routine to order parameters with the following heuristics: Fixing a non-edge requires the range computation for the non-fixed parameter that takes place in same tetrahedron. Lets D be a graph with vertices as parameters and edges representing if the two parameters takes place in the same tetrahedron. Then which parameter should we choose as a root so that the depth of the graph D is minimum. In order to achieve this goal: Order the parameters in the same order you built the tetrahedra. If there is one parameter in a tetrahedron, it is trivial, just add it to paramter list. Otherwise, prioritize the parameter which is shared by more tetrahedrons, i.e. the edge with max degrees.
- **built3tree()**: 3-tree formed by starting with a 4-vertex complete graph and then 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 attribute **tetrahedra**.

4.1.8 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. When there is one parameter in a tetrahedron, the range of parameter can be computed through tetrahedral inequalities. If there are 2 unfixed parameters at the moment in a tetrahedron, then the range of one of the parameters will be computed through triangular inequalities and fixed. And the range of other parameter will be computed through tetrahedral inequalities. The range computation way depends on the order of parameters.
- **setRangeByTriangleInequality(v1, v2)**: Computes the range of the non-edge $v1 - v2$ through triangular inequalities. uses all triangles that this edge is shared.
- **setRangeByTetrahedralInequality(v1, v2, tetrahedron)**: Computes the range of the non-edge $v1 - v2$ through tetrahedral inequality using input tetrahedron
- **stepGrid**: Sets parameter point to the next grid point within the computed range.
- **stepNeighbour()**: Sets the parameter point to the neighbour grid point in all dimensions consecutively.
- **stepGridBinary()**: Sets the parameter point to somewhere between current point and neighbour grid point according to binary search procedure in boundaryDetection.

4.1.9 CartesianRealizer

The CartesianRealizer class CartesianOrientatior) contains routines that computes Orientation that represents transformations of the rigid molecular composite s relative to each other. It computes Cartesian realization of an ACG with the parameter lengths taken from cayleyPoint and active constraint lengths for a specific flip. It intentionally ignores the remainder of the assembly constraint system, namely atom markers not in G and their constraints.
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.

4.1.10 ConstraintCheck

The ConstraintCheck class is designed to check whether any non-active constraints become active or whether they are violated. User have the option to define set of constraints of interest. In that case, new constraint activation check will be done only for this set. For an input Orientation, ConstraintCheck first computes the Cartesian realization for the entire molecular composite then passes it to the subroutines to do user specified constraint check such as steric constraints, tethering constraints or angle constraints.

4.1.11 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, witness point. If activeConstraintGraph is not partial 3-tree, then the region is populated by ray tracing, i.e. witness points coming from all ancestors.
Major Attributes:

- **rootGraphs**: Set of ActiveConstraintGraphs of the root nodes.
- **atlas**: An atlas object that will be poulated by AtlasBuilder. This object is shared between front-end and back-end of the algorithm.

Major Methods:

- **startAtlas()**: For each (bi)tether creates root atlasNode labeled with a contact graph G_F where F is the (bi)Tether i.e. 1 (or 2) specific contacts. Then calls sampleExplore for that atlasNode.
- **sampleExplore(atlasNode)**: Elaborated in the sample explore section.
- **determineStepSizeDynamically()**: Finds out s by given T, total number of samples. Each 5d-atlas node has its own s computed by volume of Cayley parameter space of the node over total number of samples per node. The volume of 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 ignorable since the volume of regions in low dimensional nodes are ignorable compared to the regions of high dimensional nodes.
- **boundaryDetection()**: Elaborated in the boundary detection section.
- **rayTracing()**: A recursive routine to explore regions that do not have an active constraint graph that is a partial 3-tree, by inheriting samples from the parent node(s).

5 Dependencies and Installation

This section discusses all the dependencies EASAL requires to run.

- Ubuntu Linux 12.04 or higher. Even though technically it should run on any Unix variant, it has been thoroughly tested only on Ubuntu.
- 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, the open-source 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++ to compile the source code. We use some features from c++11 and hence require g++ version 4.8 or higher.
- `EASAL` uses the GNU Make utility to compile the source files. Make version 4.1 is required.

5.1 Installation

You will need to download and install some third party libraries and update the Makefile provided before you can successfully compile and run `EASAL`.

- Install GLUT
  - sudo apt-get install freeglut3 freeglut3-dev
  - sudo apt-get install binutils-gold
- Install FOX-Toolkit (version 1.6)
  - sudo apt-get install libfox-1.6-0 libfox-1.6-dev
  - Download the fox library headers from `http://fox-toolkit.org/` and extract it.
- Install GNU Make
  - sudo apt-get 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' from the root/build directory.

References

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

[2] Aysegul Ozkan, Rahul Prabhu, Ruijin Wu, Troy Baker, James Pence, Jorg Peters, and Meera Sitharam. Easal: Software architecture and functionalities. (on arxiv), 2014.

[3] 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.

[4] Meera Sitharam and Heping Gao. Characterizing graphs with convex and connected configuration spaces. *CoRR*, abs/0809.3935, 2008.

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

[6] 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.