CayMos User Manual: User Interface, Functionalities and Pseudocode

Menghan Wang and Meera Sitharam
University of Florida

October 24, 2014

`CayMos` [4] is a software system for analyzing the configuration spaces of a common class of 2D mechanisms called 1-dof tree decomposable linkages with low Cayley complexity [2]. `CayMos` implements efficient algorithmic solutions for: (a) meaningfully representing and visualizing the connected components of the Euclidean realization space of the linkage; (b) finding a path of continuous motion between two realizations in the same connected component, with or without restricting the realization type (sometimes called orientation type); (c) finding two "closest" realizations in different connected components.

1 `CayMos` Functionalities and User Interface

`CayMos` accepts user input of a linkage by allowing the user to draw the bar and joints of the linkage.

1.1 Generating Cayley configuration spaces

The user can generate the Cayley configuration spaces for 1-dof tree-decomposable linkages with low Cayley complexity. The following functionalities are provided by `CayMos`:

1. `CayMos` determines whether the given linkage is 1-dof tree-decomposable. For a 1-dof tree-decomposable linkage, it determines whether the linkage has low Cayley complexity, implementing the Four-cycle algorithm from [3].

2. For a linkage with low Cayley complexity, `CayMos` generates both the oriented and non-oriented Cayley configuration space(s), implementing the ELR algorithm from [2]. See Figure 1 and 2 for the user interface.

3. The user can change the length of the chosen base non-edge to see corresponding realizations. The generated Cayley configuration space is shown as a set of intervals, and a dot representing the current Cayley configuration is shown on the axis, as shown in Figure 1 (C). The user can use a spinner to change the Cayley configuration and show a corresponding realization, as shown in Figure 1 (B).

4. The user can specify the realization type by changing the local orientations at specified construction steps. The realizations shown, as well as the intervals of the oriented Cayley configuration spaces, are color-coded corresponding to different realization types. See Figure 2.

1.2 Visualizing a connected component of the realization space

After generating the Cayley configuration space, the user can generate and visualize the connected component of the realization space containing a given realization. The following functionalities are provided by `CayMos`:

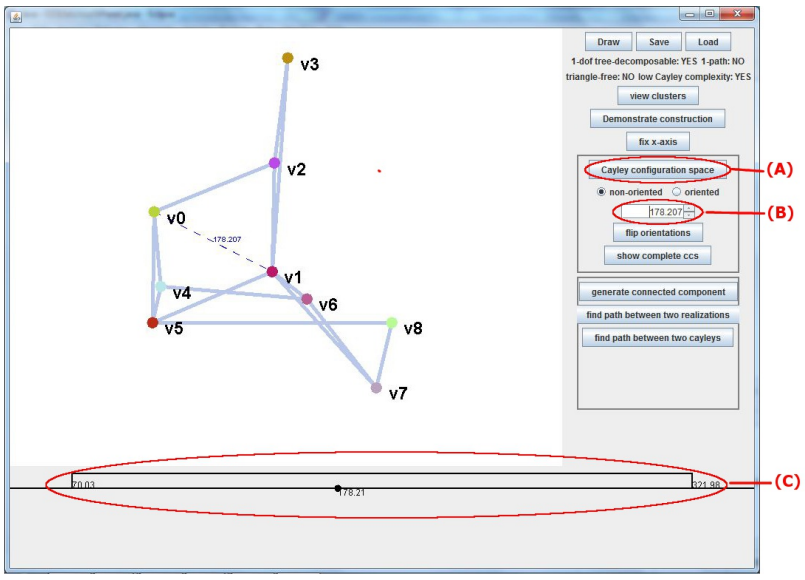


Figure 1: Generating the non-oriented Cayley configuration space of the Limacon linkage using `CayMos`. (A) Button for generating the Cayley configuration space. (B) Spinner for specifying the length of the base non-edge and navigating the Cayley configuration space. (C) Intervals of the Cayley configuration space. The black dot denotes the Cayley configuration for the current realization.

1. Showing the non-edges in the complete Cayley vector of the linkage, as well as displaying the complete Cayley distance vector for the current realization. See Figure 3 (B) and (C).

   Note: The current version of the software implements the complete Cayley vector defined in [1], which has higher dimension than the *minimal complete Cayley vector* defined in [2].

2. Generating the connected component containing the current realization, implementing the Continuous motion path algorithm from [2]. See Figure 3 (A).

3. Visualizing the connected component: the user can pick three non-edges from the complete Cayley vector, and `CayMos` shows the 3D projection of the canonical Cayley curve of the current connected component on those three non-edges. `CayMos` permits the user to use a spinner to trace the connected component, and a moving dot is shown on the curve to denote the current realization. The canonical Cayley curve is color-coded corresponding to realization types encountered in the connected component. See Figure 3 (E), (G) and (D).

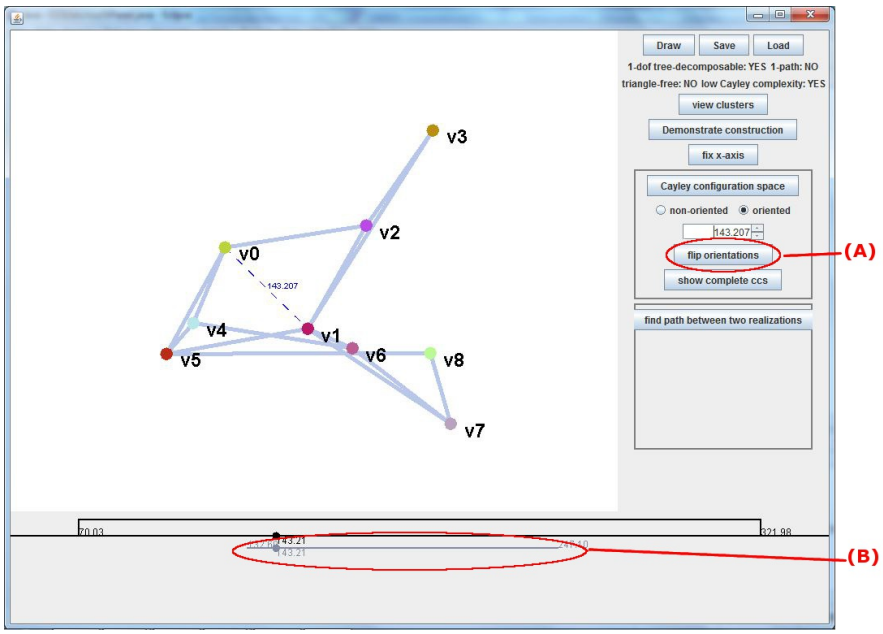
4. Showing the intervals of oriented Cayley configuration spaces contained in the current connected component. The intervals are color-coded corresponding to realization types. See Figure 3 (H).

5. Showing the curves traced out by vertices of the linkage in the current connected component. See Figure 4.

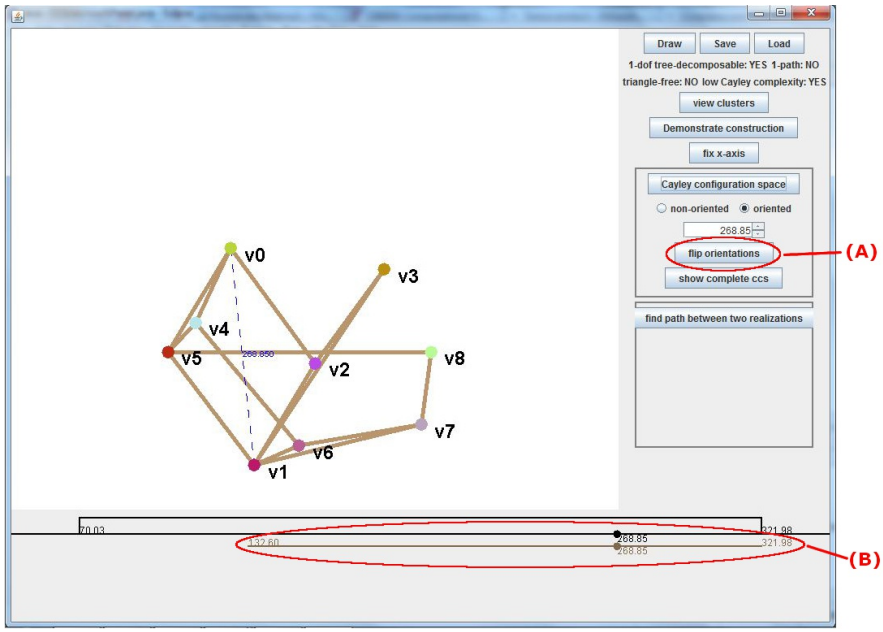
1.3 All connected components and continuous paths in the realization space

The user can generate all connected components of the realization space, and find a continuous motion path between two realizations. The following functionalities are provided by `CayMos`:

1. Generating all connected components of the realization space, implementing the adapted Continuous motion path algorithm from [1]. See Figure 3 (F).

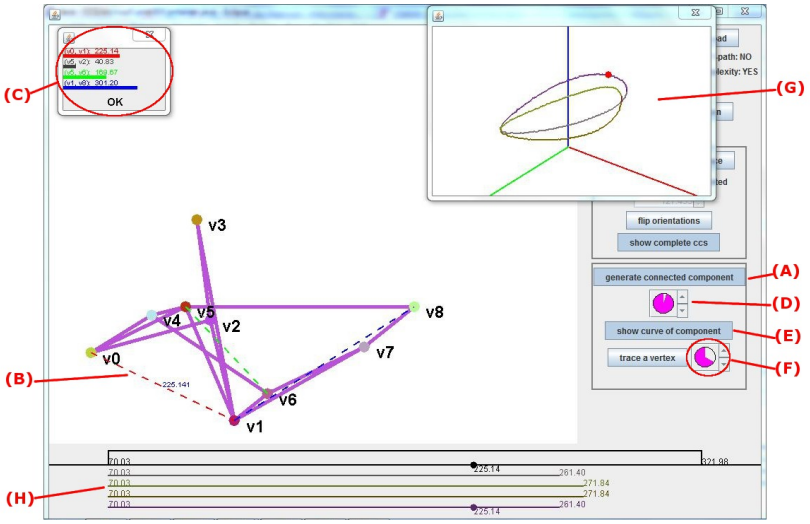


(Fig.a)

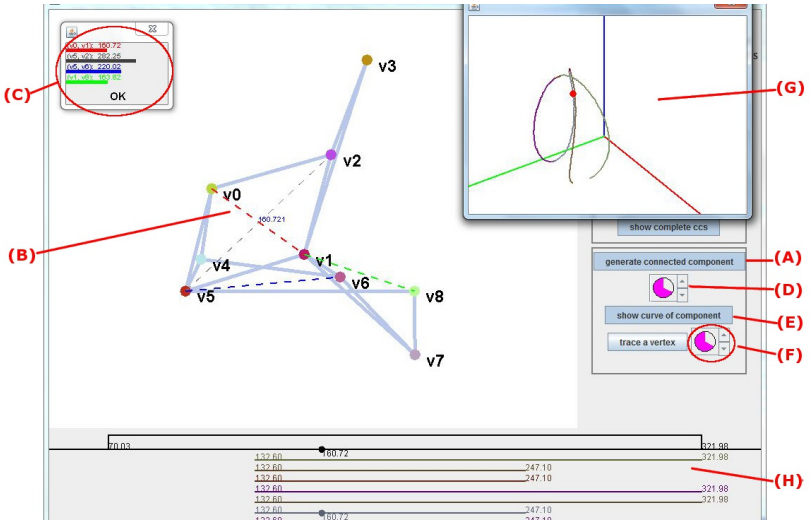


(Fig.b)

Figure 2: Two different oriented Cayley configuration spaces of the Limacon linkage. (A) Button for changing the realization type. (B) Intervals of the corresponding oriented Cayley configuration space.



(Fig.a)



(Fig.b)

Figure 3: Finding the connected components and showing corresponding canonical Cayley curves of the Limacon linkage using `CayMos`, where (Fig.a) and (Fig.b) show two different connected components. (A) Button for generating the connected components. (B) The current realization, moving as the user traces the connected component. Dashed non-edges: non-edges in the complete Cayley vector. (C) Panel showing the complete Cayley distance vector for current realization. `CayMos` allows the user to pick three non-edges for 3D projection of the canonical Cayley curve, where the picked non-edges are shown in red, green and blue in (B) and (C). (D) Spinner for tracing the current connected component. (E) Button for showing the 3D projection of the corresponding canonical Cayley curve. (F) Spinner for navigating all connected components in the realization space. (G) The 3D projection of the canonical Cayley curve of the current component. The dot denotes the current realization and moves as the user traces the connected component. The curve is color-coded according to realization types. (H) The intervals of the oriented Cayley configuration spaces contained in the current connected component.

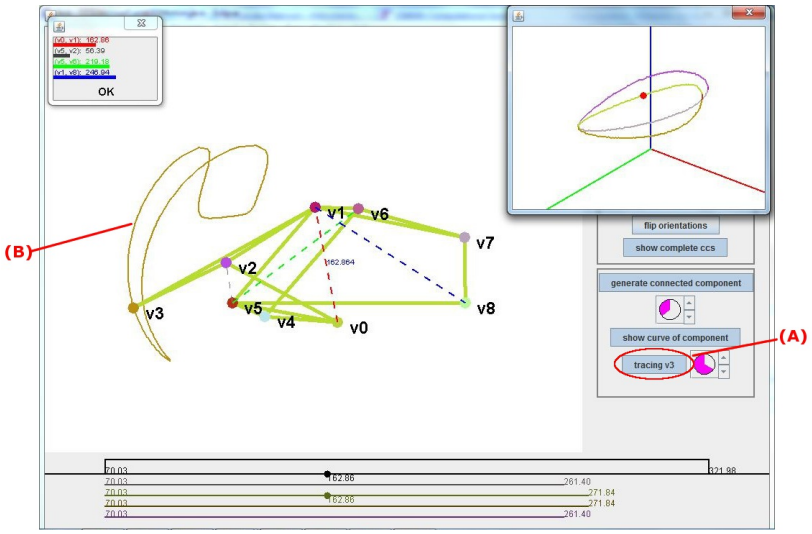


Figure 4: Showing the curve traced out by a vertex in a non-standard component of the Limacon linkage. (A) Button for tracing a vertex. (B) The curve traced by the selected vertex.

2. Finding a continuous path between two realizations specified by the user (if one exists), implementing the Continuous motion path algorithm from [2]. See Figure 5 (Fig.a).

3. Finding a continuous path between two given realizations, maintaining the same realization type. Note that such a path exists if and only if those two realizations lie in the same interval in the oriented Cayley configuration space [2]. See Figure 5 (Fig.b).

1.4 Cayley distance between connected components

When the user specifies two realizations in different connected components and tries to find a continuous motion path between them, `CayMos` will find the two nearest realizations of these two components using the Closest pair algorithm from [1]. Both connected components are shown as projected curves, and the two nearest realizations are marked on the two curves. See Figure 6.

2 `CayMos` software architecture and pseudocode

2.1 Major classes and architecture of `CayMos`

Overall the backend of `CayMos` consists of two parts, with the following major classes.

2.1.1 1-dof tree-decomposable linkages and Cayley configuration spaces

1. The TDLinkage class: represents a 1-dof tree-decomposable linkage.

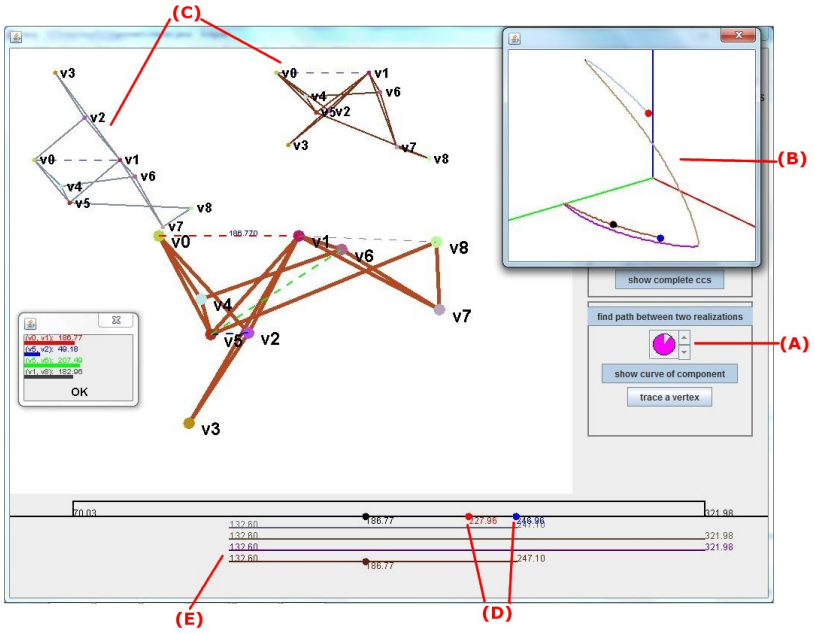
   Major Attributes:

   – graph: the underlying graph of the linkage.

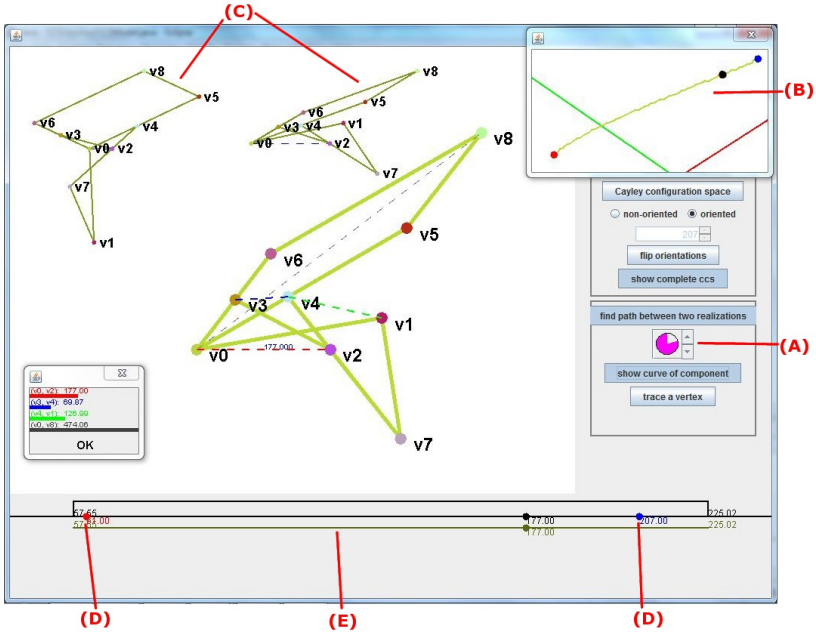
   – barLengths: the length of the bars of the linkage.

   – baseNonedge: current base non-edge of the linkage.

   – completeCayleyVector: the complete Cayley vector for the current base non-edge.



(Fig.a)



(Fig.b)

Figure 5: (Fig.a) Finding a continuous motion path between two realizations of the Limacon linkage. (Fig.b) Finding a continuous motion path maintaining the same realization type between two realizations of the Cadioid linkage. (A) Spinner for tracing the generated continuous motion path. (B) The 3D projection of the segment of the canonical Cayley curve corresponding to the continuous motion path. The red and blue dots denote the start and end realizations. The black dot denotes the current realization and moves as the user traces the path. (C) The start and end realizations. (D) The Cayley configuration for the start and end realizations. (E) The intervals of the oriented Cayley configuration spaces encountered along the path.

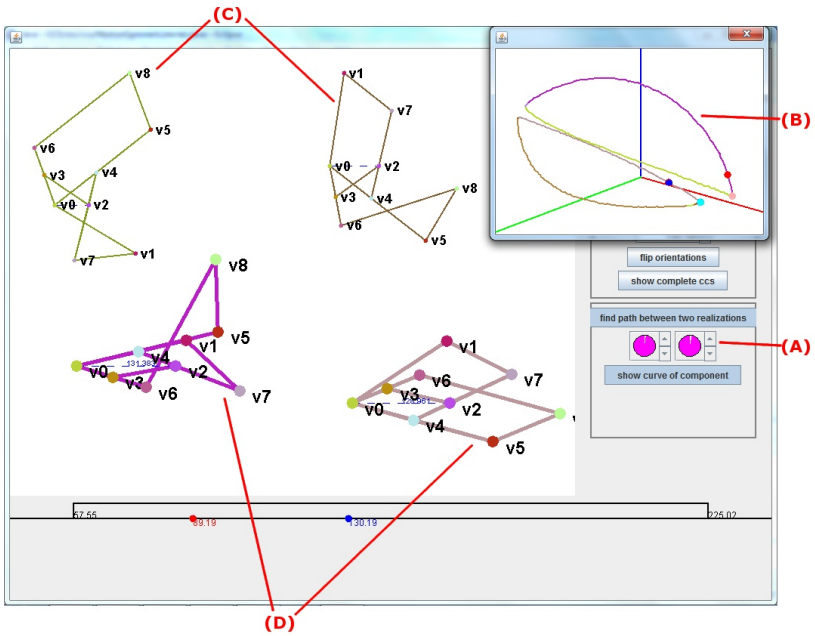


Figure 6: Finding two nearest realizations between two connected components using `CayMos`. (A) Spinners for tracing the two connected components containing the two given realizations. (B) The 3D projection of the corresponding Cayley curves of both connected components. The pink and cyan dots denote the two nearest realizations between the two components. The red and blue dots denote the current realizations and moves when tracing two components. (C) The start and end realizations. (D) The current realizations represented and traced by moving dots on the two components.

– cayleyConfigurationSpace: the Cayley configuration space on the current base non-edge.

Major Methods:

– isLow(): elaborated in Section 2.2.1.

– getCayleyConfigurationSpace(): elaborated in Section 2.2.2.

2. The Realization class: represents a realization of a TDLinkage instance.

   Major Attributes:

   – tdLinkage: the corresponding 1-dof tree-decomposable linkage of the realization.

   – points: the 2D points in the realization for the vertices of the linkage.

   Major Methods:

   – length(e:Edge): Returns the length of e in the realization, where e can be an edge or a non-edge.

   – getCompleteCayleyDistanceVector(): Returns the complete Cayley distance vector of the realization. Calls length() for each non-edge in the completeCayleyVector field of tdLinkage, with time complexity $O(|V|)$.

   – cayleyDistance(that:Realization): Returns the Cayley distance [1, Section 3.2] between this realization and that with time complexity $O(|V|)$.

3. The CayleyConfigurationSpace and OrientedCayleyConfigurationSpace classes: represent the Cayley configuration space of a TDLinkage instance. Each CayleyConfigurationSpace contains multiple OrientedCayleyConfigurationSpace instances.

4. The OrientedInterval class: represents an interval in an oriented Cayley configuration space.

   Major Attributes:

   – upper and lower: the upper and lower endpoints of the interval.

   – realizationType: the realization type of the OrientedCayleyConfigurationSpace containing the interval.

   – nextIntervalUpper and nextIntervalLower: the two OrientedInterval instances sharing a common endpoint with this interval. They are immediately reachable from this interval in a continuous motion.

2.1.2 Continuous motion generation and representation

1. The ContinuousMotion class: represents a continuous motion path between two Realization instances, or a connected component of a TDLinkage instance.

   Major Attributes:

   – tdLinkage: the corresponding 1-dof tree-decomposable linkage of the continuous motion.

   – orientedIntervals: the list of OrientedInterval instances encountered along the continuous motion.

   Major methods:

   – findComponent(), findPath() and findAllComponents(): elaborated in Section 2.2.3.

   – findNearestRealizations(): elaborated in Section 2.2.4.

   – getRealizations(): Returns a list of realization samples along the continuous motion by sampling the orientedIntervals field. Uses a sampler object so that different ways of sampling can be chosen at runtime.

   – get3DCurve(f1,f2,f3): Returns a Curve3D instance representing the 3D projection of the continuous motion's corresponding Cayley curve on f1, f2 and f3, three non-edges from the complete Cayley vector of tdLinkage. The time complexity is linear in the size of the list returned by getRealizations().

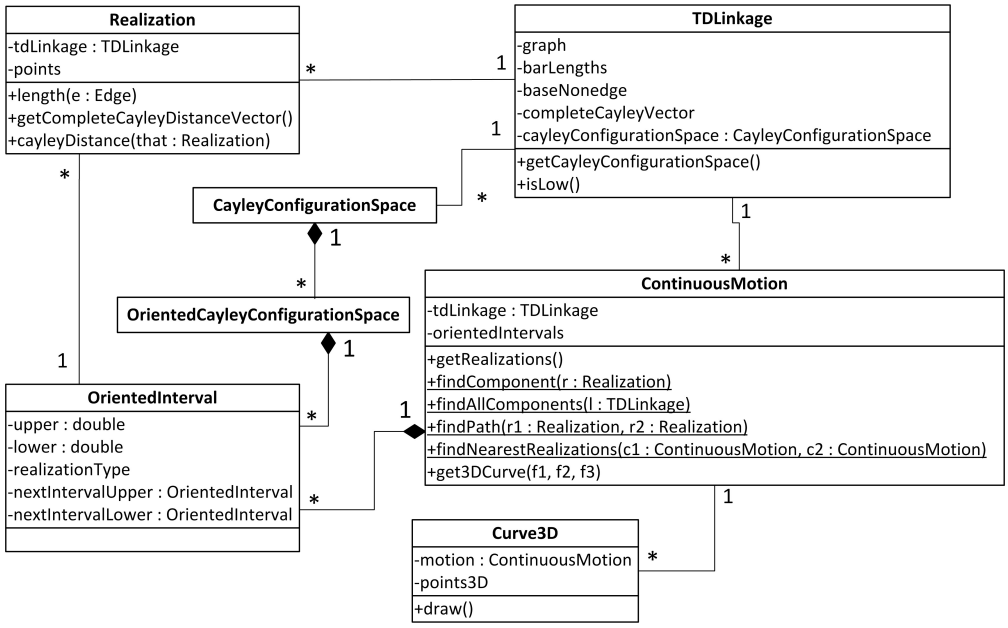


Figure 7: UML diagram for major classes.

2. The Curve3D class: supports visualization of a ContinuousMotion instance as a Cayley curve projected in 3D.

Figure 7 shows the relationships between the above classes.

2.2 Major methods

2.2.1 Method isLow()

The method implements the Four-cycle algorithm [3, Theorem 2] returning true if the calling TDLinkage instance has low Cayley complexity, or false otherwise. It also implements [1, Theorem 3] to find the complete Cayley vector and store it in the completeCayleyVector field. Since both algorithms follow the construction of the linkage, we combine the implementation into one method. The time complexity is $O(|V|^2)$.

Note: in the current version of the software, the complete Cayley vector is implemented as in [1, Theorem 3], which is not minimal. The *minimal complete Cayley vector* introduced in [2, Theorem 4] will have dimension two for a large number of 1-dof tree-decomposable linkages.

Pseudocode for isLow() :

```
boolean isLow() {
  completeCayleyVector.add(baseNonedge);

  for (each construction step s) {
  // Construction Step s: adding two clusters s.c1 and s.c2, s.v ◁ (s.v1, s.v2), s.v1 ∈ s.c1
      , s.v2 ∈ s.c2
    if (s is not directly based on the base non-edge) {
      // find a valid pair of base clusters
      for (each Vertex w in the previously constructed graph sharing a cluster with both s.v1
           and s.v2){
        Cluster c1 = the cluster containing s.v1 and w;
        Cluster c2 = the cluster containing s.v2 and w;
        if (validBasePairs.contains((c1,c2)){
          isLowStep = true;
```

```
          completeCayleyVector.add(nonedge (w, s.v));
          currentBasePairs.add((c1,c2));
        }
      }

      if (!isLowStep)  // does not have low Cayley complexity
        return false;

      for (Pair(c1,c2) in currentBasePairs){
        validBasePairs.add((s.c1, c1));
        validBasePairs.add((s.c2, c2));
      }
    }
    validBasePairs.add((s.c1,s.c2));
  }
  return true;
}
```

2.2.2 Method getCayleyConfigurationSpace()

The method implements the ELR algorithm [2] to generate and return the Cayley configuration space of the calling TDLinkage instance on the current base non-edge. The generated Cayley configuration space is also stored in the cayleyConfigurationSpace field of the calling TDLinkage instance. As pointed out in [2], computation of the Cayley configuration space is NP-hard, and this method can take time exponential in $|V|$.

Pseudocode for getCayleyConfigurationSpace() :

```
CayleyConfigurationSpace getCayleyConfigurationSpace() {
  for (each construction step s) {
    for (each extreme linkage Realization r at step s) {
      distance = r.length(baseNonedge);
      for (each complete solution type t compatible with the partial solution type of r)
        candidateEndpointLists[t].add(distance);
    }
  }

  for ( (SolutionType:t,List:l) : candidateEndpointLists) {
    sort(l);
    occs = new OrientedCaylyConfigurationSpace with SolutionType t;
    for (double cur : l) {
      double prev = the point before cur in l;
      double next = the point after cur in l;
      boolean P = realizable((prev + cur) / 2, t);
      boolean N = realizable((cur + next) / 2, t);
      if (!P && !N) {
        // cur is an isolated point
        occs.appendInterval(cur, cur);
      } else if (P && !N) {
        // cur is end of interval
        occs.appendInterval(lastEndpoint, cur);
        intervalStart = null;
      } else if (!P && N){
        // cur is start of interval
        intervalStart = cur;
      } // else: cur is in middle of interval
    }
    this.cayleyConfigurationSpace.addOrientedCayleyConfigurationSpace(occs);
  }
  return this.cayleyConfigurationSpace;
}
```

2.2.3 Methods findPath(r1:Realiztion, r2:Realization), findComponent(r:Realization), findAllComponents(l:TDLinkage)

These methods implement the Continuous motion algorithm [2, Theorem 3]. Method findPath(r1,r2) returns an instance of ContinuousMotion representing the continuous motion path between realizations r1 and r2, which are realizations of the same linkage. Method findComponent(r) returns an instance of ContinuousMotion, which is the connected component in the realization space containing the realization r. Method findAllComponents(l) returns a list of ContinuousMotion instances representing all connected components in the realization space of the linkage l.

Both findPath() and findComponent() have time complexity linear in the number of oriented Cayley configuration space endpoints contained in the ContinuousMotion instance returned. Method findAllComponents() has time complexity linear in the total number of endpoints of all oriented Cayley configuration spaces [1, Theorem 3].

Pseudocode for findComponent() (findPath() is implemented similarly):

```
static ContinuousMotion findComponent(Realization r) {
  startInt = the OrientedInterval containing the Cayley configuration of r;
  component.orientedIntervals.add(startInt);

  OrientedInterval curInt = startInt;
  double endIf = curInt.lower;
  while (true) {
    curInt = (endIf == curInt.lower? curInt.nextIntervalLower:curInt.nextIntervalUpper)
    if (curInt == startInt)
      return component;
    endIf = (endIf == curInt.lower? curInt.upper:curInt.lower);
    component.orientedIntervals.add(curInt);
  }
}
```

The method findAllComponents() is implemented by iterating over all intervals in every OrientedCayeyConfigurationSpace and calling findComponent:

```
List findAllComponents(TDLinkage t) {
  List list = new List();

  // 1. list every intervals in all oriented Cayley configuration spaces.
  Set intervalSet = new Set();
  for (OrientedCayleyConfigurationSpace oc : t.getCayleyConfigurationSpace())
    intervalSet.add(oc.orientedIntervals);

  // 2. for each interval:
  while (!intervalSet.isEmpty()) {
    // (1) generate connected component.
    OrientedInterval inv = an oriented interval from intervalSet;
    ContinuousMotion component = find the component containing inv;
    list.add(component);

    // (2) remove the intervals passed by the component from the list of intervals
    for (OrientedInterval n : component.orientedIntervals)
      intervalSet.remove(n);
  }
  return list;
}
```

2.2.4 Method findNearestRealizations(c1: ContinuousMotion, c2: ContinuousMotion)

The method implements the Closest pair algorithm [1, Theorem 5(ii)]. The method returns the pair of two nearest realizations between the two connected components c1 and c2 of the realization space of a linkage, using the Cayley distance measure. The time complexity $O(k^2|V|)$, where k is the size of the list returned by getRealizations(), which is a list of sample realizations in the connected component.

Pseudocode for findNearestRealizations():

```
static Pair findNearestRealizations(ContinuousMotion c1, ContinuousMotion c2) {
  double nearestDistance = Double.POSITIVE_INFINITY;
  Realization result1 = null, result2 = null;
  for (Realization r1 : c1.getRealizations()) {
    for (Realization r2 : c2.getRealizations()) {
      double dis = r1.cayleyDistance(r2);
      if (dis < nearestDistance) {
        result1 = r1; result2 = r2;
        nearestDistance = dis;
      }
    }
  }
  return (result1, result2);
}
```

References

[1] Meera Sitharam and Menghan Wang. How the beast really moves: Cayley analysis of mechanism realization spaces using caymos. *Computer-Aided Design*, 46(0):205 – 210, 2014. 2013 SIAM Conference on Geometric and Physical Modeling.

[2] Meera Sitharam, Menghan Wang, and Heping Gao. Cayley configuration spaces of 1-dof tree-decomposable linkages, part i: Extreme points, continuous motion paths and minimal representations. *Under review. arXiv:1112.6008*, 2011.

[3] Meera Sitharam, Menghan Wang, and Heping Gao. Cayley configuration spaces of 1-dof tree-decomposable linkages, part ii: Combinatorial characterization of complexity. *Under review. arXiv:1112.6009*, 2011.

[4] Menghan Wang and Meera Sitharam. Caymos software, 2014. Web accessible at: http://www.cise.ufl.edu/~menghan/caymos/, software available at: http://calgo.acm.org/ and http://code.google.com/p/caymos/.