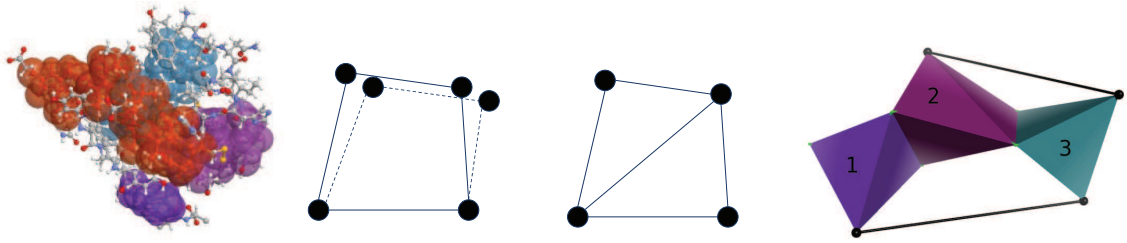
KINARI-Lib: A C++ library for mechanical modeling and pebble game rigidity analysis

Naomi Fox* Filip Jagodzinski† Ileana Streinu ‡

1 Introduction

KINARI (KINematics And RIgidity, `http://kinari.cs.umass.edu`) is a software project focused on data structures and algorithms for rigidity analysis, as applied to mechanical structures, abstract sparse graphs and molecules. Designed and developed in the last author's research group at Smith College and the University of Massachusetts Amherst (LinkageLab `http://linkage.cs.umass.edu`), KINARI's first application, KINARI-Web [1], was released in 2011 as a web server for protein rigidity and flexibility analysis. An example of KINARI-Web's output, shown in Fig. 1(a), is the rigid cluster decomposition of a protein retrieved from the Protein Data Bank (PDB).

In this abstract we describe KINARI-Lib V1.0, the first public release of the underlying core library. Written in C++, KINARI-Lib implements the pebble game algorithm and provides support for body-bar-hinge and bar-joint mechanical models (illustrated in Figs. 1(b),1(c)). Links to documentation, compiled library downloads, and to the KINARI-Web server, are available from the KINARI project website.



(a) Protein. (b) 2D bar-and-joint frameworks. (c) 3D body-bar-hinge.

Figure 1: (a) Rigid cluster decomposition of the human insulin protein (PDB 1TRZ), as calculated by the KINARI-Web application. Only the 4 largest clusters are shown. (b) Flexible and rigid 2D bar-and-joint frameworks. (c) A 3D body-bar-hinge mechanical model.

Rigidity theory is a well-established mathematical area with applications in engineering, sensor networks, chemistry, molecular biology and CAD. It aims at finding combinatorial properties of mechanical models, or frameworks, in order to efficiently describe and compute their rigidity and flexibility properties. The two best-studied classes of frameworks, bar-and-joint and body-bar-hinge, have well-understood combinatorial properties. They are studied using associated graphs, where vertices represent rigid parts and the edges correspond to fixed-length bars whose presence removes degrees of freedom. The pebble game algorithm efficiently examines such graphs, and determines, in $O(n^2)$-time, the total number of degrees of freedom, maximal rigid components, and over-constrained regions in a generic framework.

KINARI-Lib contains classes for the pebble game algorithm, the body-bar-hinge framework mechanical model, and special graphs on which the pebble game operates. Classes for reading from and writing to XML-formatted files are also provided for the body-bar-hinge framework and for graphs.

After some preliminaries from rigidity theory, we give an overview of the software architecture. To demonstrate its functionality, we also include two code examples for building executables: the first, for running the pebble game on graphs and the second for analyzing rigidity of body-bar-hinge frameworks.

*Department of Computer Science, University of Massachusetts, Amherst, MA, USA, `fox@cs.umass.edu`
†Department of Computer Science, University of Massachusetts, Amherst, MA, USA, `filip@cs.umass.edu`
‡Department of Computer Science, Smith College, Northampton, MA and Department of Computer Science, University of Massachusetts, Amherst, MA, USA, `streinu@cs.umass.edu istreinu@smith.edu`

This is an abstract related to a tutorial or demo given at GTS 2012. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear in a conference with formal proceedings and/or in a journal.

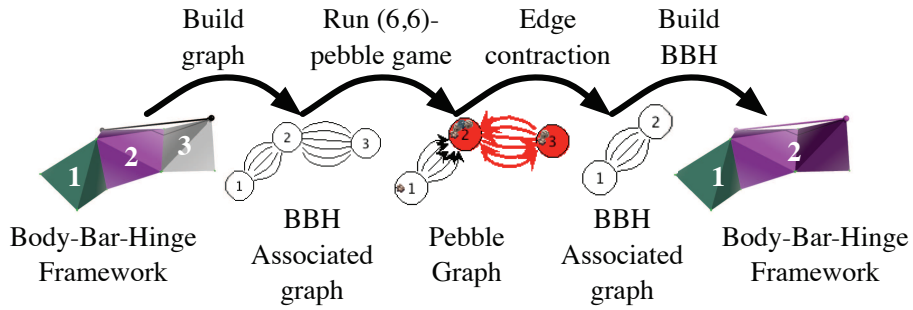


Figure 2: Rigidity analysis applied to a generic body-bar-hinge framework.

2 Background

A mechanical model, or framework, is composed of rigid pieces held together by hard constraints. The number of degrees of freedom (dofs) is the number of independent variables required to describe the position and configuration of the framework in 3D. Two well understood classes of frameworks are bar-and-joint and body-bar-hinge. A bar-and-joint framework is composed of points with distance constraints between them. Fig. 1(b) shows two bar-and-joint frameworks in the plane: one is flexible and one is rigid. A rigid framework in 2D has zero internal dofs and 3 trivial ones, which correspond to rigid motions in the plane. In 2D, the rigidity of generic bar-and-joint frameworks is completely characterized by Lamans's theorem [2].

Theorem 1 ***Laman's Theorem 2D bar-and-joint****:*
A graph G with n vertices and m edges is a graph of a generically minimally rigid bar-and-joint framework iff each subgraph G' with n' vertices spans at most $m' \leq 2n' - 3$ edges, and $m = 2n - 3$.

Unfortunately, no such characterization is known in higher dimensions for bar-joint frameworks. A somewhat less general model that still lends itself well to the modeling of molecules, and which has a Laman-type characterization in 3D and higher dimensions, is the body-bar-hinge model. Such a framework is made from rigid bodies connected along hinge axes allowing only rotation around the hinge, or by fixed length bars connected at universal joints. Body-bar-hinge frameworks have associated multigraphs with one vertex for each body, 5 edges for each hinge, and 1 edge for each bar. Tay's theorem characterizes the rigidity of such generic frameworks through similar sparsity conditions on this graph [5].

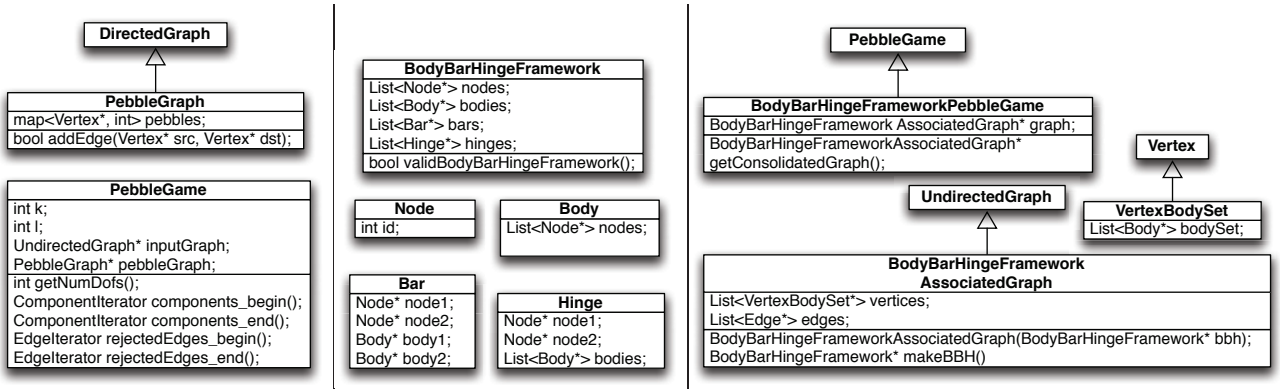
Theorem 2 ***Tay's Theorem for 3D body-bar-hinge****:*
A multigraph G with n vertices and m edges is a graph of a generically minimally rigid body-bar-hinge framework iff each subgraph G' with n' vertices spans at most $m' \leq 6n' - 6$ edges, and $m = 6n - 6$.

These theorems lead to exponential-time recognition algorithms, but better methods, based on matroid partitioning and network flow, are known. Particularly convenient is the pebble game algorithm, valid for a larger class of (k, ℓ) graph sparsity conditions [3]. The algorithm starts with k pebbles placed on each vertex. Edges are considered one at a time, and accepted if they have a total of at least ℓ pebbles on their two endpoints, or when this many pebbles can be gathered through a graph search procedure. When pebbles can't be gathered, the edge is rejected. The pebbles remaining at the end represent the degrees of freedom remaining in the system.

KINARI-Lib contains an implementation of the "component pebble game algorithm", which does more than just calculating degrees of freedom: it determines the over-constraints, dofs, and rigid components (maximal subsets of vertices satisfying the sparsity condition with equality). The correctness of the pebble game calculations is guaranteed for (k, ℓ) values with $k > 0$ and $l \in [0, 2k)$ [3]. The values needed for applying Laman's and Tay's theorems, $(2, 3)$ and $(6, 6)$, fall within this range.

Tay's theorem has guarantees only for generic body-bar-hinge frameworks. The software does not check for genericity of the coordinate set, but we have provided some checks that certain combinatorial degeneracies not occur. For example, every bar endpoint must lie in one and exactly one body.

For further background on rigidity theory and the pebble game algorithm, the LinkageLab hosts an educational website (`http://linkage.cs.umass.edu/pg/`) with interactive java applets and a tutorial video [4].



(a) (k, ℓ)-Pebble Game (b) Body Bar Hinge Framework (c) Body Bar Hinge Framework Pebble Game

Figure 3: Overview of selected classes from KINARI-Lib.

3 KINARI-Lib software architecture

KINARI-Lib has classes for pebble games and for 3D body-bar-hinge framework mechanical modeling. To support this functionality, KINARI-Lib also provides classes for representing graphs, calculating some statistics on the body-bar-hinge frameworks, and for reading and writing each data structure in XML. A brief description of the key classes is included below and in Fig. 3. Complete documentation is distributed with the software.

PebbleGame. This class contains an implementation of the component pebble game, described in [3]. The constructor runs it by default, and requires the (k, ℓ) sparsity parameters and a (multi)graph as input. Functions are provided to retrieve the number of degrees of freedom, the maximal (rigid) components, and to identify over-constrained edges. The class contains one **PebbleGraph** object as a member variable. A PebbleGraph is a special extension of a directed multigraph, on which the pebble game is played.

BodyBarHingeFramework. This class contains lists of nodes, bodies, bars, and hinges. The nodes are a discrete set of points used to define the bodies, bars, and hinges. The **Node** class represents a point, which can (optionally) have coordinates associated with it. The **Body** class contains a set of nodes. The **Bar** represents the bar connection between two Body objects at two Node endpoints. A **Hinge** connects at least two bodies, at two nodes defining the hinge axis. The **BodyBarHingeFramework::validBodyBarHingeFramework()** function can be called to check a number of conditions that are described in the API documentation.

BodyBarHingeFrameworkPebbleGame. This class provides special functionality applicable to the pebble game on body-bar-hinge frameworks. The input to this is an object of the **BBHFwkAssociatedGraph** class. The **BBHFwkAssociatedGraph** a special undirected graph associated with a body-bar-hinge framework. Each vertex points to one (or more) bodies, and the edges have bars or hinges associated with them. In this way, the post-pebble-game body-bar-hinge framework can be built. The **BodyBarHingeFrameworkPebbleGame::getConsolidatedGraph()** function is provided to output an edge-contracted graph, where each component has been consolidated into a single vertex. Because this graph is of type BBHFwkAssociatedGraph, it contains all the information necessary to build the simplified BodyBarHingeFramework, where each rigid body is maximal.

4 Code examples

We include two short examples illustrating how the KINARI-Lib functionality can be incorporated into a C++ program. The first example runs the pebble game on graphs. The second one analyzes the rigidity of 3D body-bar-hinge frameworks. The library distribution includes extended versions of the two examples.

General graph pebble game. It performs the following steps:
Step 1: Read a graph from file.
Step 2: Run the pebble game on the graph, with user-specified (k, ℓ) values
Step 3: Write an XML file containing the components, dofs, and over-constraints calculated by the pebble game.

This example also demonstrates the error handling utilities inside the KINARI code. Here, the PebbleGame constructor will throw a KinariException if the (k, ℓ) values are not in the acceptable range.

```
#include "PebbleGame.h"
#include "GraphXMLFileIO.h"
#include "ComponentsXMLWriter.h"
using namespace Kinari;
int main( int argc, char **argv ) {
  // Command line parameters:
  std::string graphfilename(argv[1]); // 1. The name of the XML file containing the input graph
  int k = atoi(argv[2]); // 2. k in the (k,l) values required to configure the pebble game
  int l = atoi(argv[3]); // 3. l in the (k,l) values required to configure the pebble game
  std::string componentfilename(argv[4]); // 4. The output file to write the components to
  try {
    GraphXMLFileIO graphReader(graphfilename);
    UndirectedGraph* ugraph = graphReader.extractGraph();
    PebbleGame pg(k,l, ugraph);
    ComponentsXMLWriter::writePebGameResultsFile(pg, componentfilename);
  } catch (KinariException e) {
      std::cerr << e.toString() << std::endl; }}
```

Rigidity analysis of Body-Bar-Hinge Frameworks. The following commented code example shows the KINARI classes and syntax for analyzing a body-bar-hinge framework using the BodyBarHingeFrameworkPebbleGame class. The steps performed mirror those shown in Fig. 2.

```
#include "BBHFwkXMLFileIO.h"
#include "GraphXMLFileIO.h"
#include "BodyBarHingeFrameworkPebbleGame.h"
using namespace Kinari;
int main( int argc, char **argv ) {
  // Read a body-bar-hinge framework from an XML file
  BBHFwkXMLFileIO bbhXMLReader("bbh.xml");
  BodyBarHingeFramework* bbh = bbhXMLReader.extractBBH();
  // Create an associated graph
  BBHFrameworkAssociatedGraph bodyGraph(bbh);
  //Play pebble game and get the output graph with edge contractions
  BodyBarHingeFrameworkPebbleGame pg(&bodyGraph);
  BBHFrameworkAssociatedGraph* edgeContractedBBHGraph = pg.getConsolidatedGraph();
  GraphXMLFileIO::writeOutGraph(*edgeContractedBBHGraph, "edgeContractedBBHGraph.xml");
  // Retrieve minimized body-bar-hinge and write to file
  BodyBarHingeFramework* consolidatedBBH = edgeContractedBBHGraph->makeBBH();
  BBHFwkXMLFileIO::writeXMLToFile(*consolidatedBBH,"postPG_BBH.xml");}
```

5 Conclusion

Because it is computationally inexpensive compared with molecular simulations, rigidity analysis is a powerful tool for the study of protein flexibility. Indeed, the main application of KINARI-Lib, deployed in our publicly available server KINARI-Web, has been in this area. This release is anticipated to facilitate our library's integration into new applications.

Acknowledgment. The development of the software described in this abstract was funded by grant DMS-0714934 as part of the Joint program in mathematical biology supported by the Directorate for Mathematical and Physical Sciences of the National Science Foundation and the National Institute of General Medical Sciences of the National Institutes of Health, and by the Defense Advanced Research Projects Agency (DARPA "23 Mathematical Challenges", under "Algorithmic Origami and Biology").

References

[1] N. Fox, F. Jagodzinski, Y. Li, and I. Streinu. KINARI-Web: a server for protein rigidity analysis. *Nucleic Acids Res.*, 39, Web Server Issue, 2011.

[2] G. Laman. On graphs and rigidity of plane skeletal structures. *J. Eng. Math.*, 4:331–340, 1970.

[3] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Math.*, 308(8):1425–1437, 2008.

[4] A. Lee, I. Streinu, and L. Theran. Analyzing rigidity with pebble games. In *Proc. SOCG*, pages 226–227. ACM, 2008.

[5] T.-S. Tay. Rigidity of multigraphs I: linking rigid bodies in n-space. *J. Comb. Theory B*, 36:95–112, 1984.