Computer Aided Geometric Design 47 (2016) 130–149

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Algorithms for detecting dependencies and rigid subsystems for CAD ☆

James Farre [a,1], Helena Kleinschmidt [b], Jessica Sidman [b,2], Audrey St. John [b,*,3], Stephanie Stark [b,4], Louis Theran [c,5], Xilin Yu [b,6]

[a] Department of Mathematics, University of Utah, Salt Lake City, UT, USA
[b] Departments of Mathematics & Statistics, Computer Science, Mount Holyoke College, South Hadley, MA, USA
[c] School of Mathematics and Statistics, University of St Andrews, Scotland, United Kingdom

ARTICLE INFO

Article history:
Available online 28 June 2016

Keywords:
Sparsity matroid
Pebble game algorithm
Cad constraints

ABSTRACT

Automated approaches for detecting dependencies in structures created with Computer Aided Design software are critical for developing robust solvers and providing informative user feedback. We model a set of geometric constraints with a bi-colored multigraph and give a graph-based pebble game algorithm that allows us to determine combinatorially if there are generic dependencies. We further use the pebble game to yield a decomposition of the graph into factor graphs which may be used to give a user detailed feedback about dependent substructures in a specific realization of a system of CAD constraints with non-generic properties.

1. Introduction

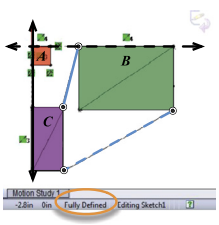
In this paper, we present graph-based algorithms for analyzing and decomposing the underlying combinatorial structure of a system of geometric constraints, such as those appearing in popular constraint-based Computer Aided Design (CAD) software. While a *generic* (essentially random) realization of a given system of constraints may be rigid, a specific realization may be in a *special position* which admits internal motions and contains (redundant) dependencies. In such a situation, our decomposition allows us determine which subsystems of constraints remain independent and which are causing the dependencies.

☆ An extended abstract of this article appeared as "Detecting dependencies in geometric constraint systems" in the 10th International Workshop on Automated Deduction in Geometry (ADG 2014), Coimbra, Portugal, 2014.

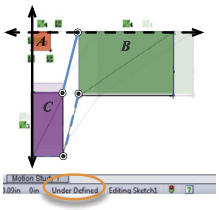
* Corresponding author.
E-mail addresses: jamesrfarre@gmail.com (J. Farre), klein23h@mtholyoke.edu (H. Kleinschmidt), jsidman@mtholyoke.edu (J. Sidman), astjohn@mtholyoke.edu (A. St. John), stark23s@mtholyoke.edu (S. Stark), louis.theran@st-andrews.ac.uk (L. Theran), yu25x@mtholyoke.edu (X. Yu).

[1] Partially supported by NSF DMS-0849637.
[2] Partially supported by NSF DMS-0849637 and the Hutchcroft fund.
[3] Partially supported by the Clare Boothe Luce Foundation, NSF DMS-0849637, NSF IIS-1253146 and the Hutchcroft fund.
[4] Partially supported by the Mount Holyoke College Lynk Program.
[5] Partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013)/ERC grant agreement No. 247029-SDModels, Academy of Finland (AKA) project COALESCE, and the Hutchcroft fund.
[6] Partially supported by NSF IIS-1253146, an Aalto Science Institute (AScI) Internship, and the Mount Holyoke College Lynk Program.

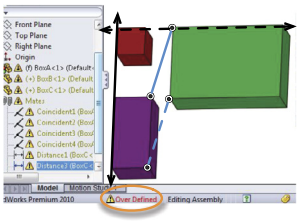
http://dx.doi.org/10.1016/j.cagd.2016.06.001
0167-8396/



(a) SolidWorks' 2D Sketch environment identifies the **rigid** *generic* embedding as "Fully Defined.'



(b) SolidWorks' 2D Sketch environment identifies the **flexible** *special position* as "Under Defined," but does not allow the user to explore the resulting motion.



(c) SolidWorks' 3D Assembly environment gives a different identification of "Over Defined" for the same **flexible** *special position* embedded in 3D.

Fig. 1. Changing a constraint in a generically rigid system to be dependent (by making the dashed distance constraint parallel to the other distance constraint) results in a flexible special position.

1.1. Motivation

CAD software allows engineers to create designs using intuitive geometric constraints. When a user adds a constraint that is *dependent*, the resulting system is *over-constrained*. To provide useful feedback, efficient approaches are required to detect the minimal sub-system containing the dependency.

Fig. 1 shows an example of a system of geometric constraints on 3 bodies constructed in SolidWorks. Bodies A and B are constrained to lie on the horizontal line; bodies A and C are constrained to lie on the vertical line; there are two distance constraints between points on bodies B and C. Together these 3 geometric constraints impose 6 independent conditions on the degrees of freedom of the system of 3 bodies, making the *body-and-cad* framework *generically* rigid.

A problem may arise if a specific realization of a system of geometric constraints has special features that imply a dependency that is not present generically. Commercial CAD software, such as SolidWorks, is unreliable when presented with such a *flexible special position*. Figs. 1(b) and 1(c) show a special realization of the system. Here, a dependency arises because the two point–point distance constraints between B and C are along parallel lines. SolidWorks' 2D and 3D environments produce different analyses. Although the 2D Sketch environment does identify the system as "Under Defined," the faded position in 1(b) was only found by suppressing the dependent constraint, investigating the motion, then unsuppressing it.

Our ultimate goal is to be able to reliably identify such special dependencies that arise in more complicated constraint systems in terms of the geometry of the constraints as above, e.g., to be able to inform the user that the non-generic behavior is due to the fact that two bars are parallel.

1.2. Related work

Rigidity theory is applicable to a wide variety of systems of geometric constraints. For example, in bar-and-joint rigidity theory, constraints are specified by fixing the distance ("bar") between pairs of points ("joints") and can be represented by quadratic equations. In body-and-bar rigidity theory, fixed-length bars are attached to rigid bodies at flexible joints. The rigidity models of 2D *bar-and-joint* and d-dimensional *body-and-bar* are well-known for having combinatorial characterizations of *generically rigid* frameworks.

Though it is possible to use algebraic methods to study systems of geometric constraints (see Chou, 1988; Gao et al., 2005; Geiß and Schreyer, 2009), working algebraically with polynomials is limited because it is computationally intensive. Hence, much work in rigidity theory instead focuses on *infinitesmal rigidity*, which is defined in terms of the rank of a matrix obtained by linearizing the constraint equations. As a matrix drops rank on a closed set, *almost every* framework with the same combinatorics is infinitesimally rigid (and therefore rigid).

Infinitesimal rigidity theory may be studied numerically or combinatorially. One may detect generic dependencies numerically, by picking random realizations. This is the approach taken by the "witness method" of Michelucci and Foufou (2007). The drawback of numerical methods is that fast, stable, algorithms, such as SVD, do not identify the support of a minimal dependency while those based on Gaussian elimination are not stable. (This can be overcome by using finite fields and the Schwartz Lemma, as discussed in Gortler et al., 2010.)

For certain structural models more robust combinatorial characterizations of generic rigidity are known, and we focus on the combinatorial approach in this paper. Particularly relevant here are results for geometric constraints arising in CAD: 2D point–line frameworks (Jackson and Owen, 2014) and body-and-cad frameworks in 2D and 3D (omitting point–point coincidences) (Lee-St.John and Sidman, 2013). Combinatorial counting conditions arise as necessary conditions for rigidity theory, usually in terms of a family of "sparse matroidal graphs" that are fundamental in generic rigidity theory (see Whiteley, 1996, Appendix, which reports work of White and Whiteley). Associated pebble game algorithms can be used to check rigidity and detect components, relying on the matroidal property of Lee and Streinu (2008) and the data structure from Lee et al. (2005). For body-and-bar (Tay, 1984) and 2D bar-and-joint (Laman, 1970) frameworks, the counting conditions are generically sufficient as well. Jackson and Owen show how to adapt Edmonds' matroid union algorithm to the

framework of pebble games for 2D point–line frameworks in Jackson and Owen (2014). A combinatorial characterization of 3D bar-and-joint rigidity remains an open problem; while the network flow approach of Sitharam and Zhou (2004) gives a polynomial time algorithm for the related concept of *module-rigidity*, the class of module-rigid graphs does not include rigid *nucleation-free graphs* (Cheng et al., 2009).

Our approach combines the traditional combinatorial counting techniques with the connections to algebra introduced by White and Whiteley. White and Whiteley define the *pure condition* of a body-and-bar framework, a polynomial obtained from taking the determinant of the rigidity matrix in White and Whiteley (1987). They showed how to interpret the irreducible factors combinatorially and how to describe some special positions using synthetic geometry via the Grassmann–Cayley algebra. In this paper we extend their philosophy to systems with additional types of constraints where the connections between the algebraic and combinatorial substructures associated to the constraint system are more intricate.

1.3. Contributions

We present algorithms for detecting dependencies in CAD systems modeled as body-and-cad frameworks. The first is a pebble game algorithm that can check for *generic dependencies* via the combinatorial property from Lee-St.John and Sidman (2013); when a constraint is determined to be dependent, we additionally detect its *fundamental circuit* (minimal set of constraints involved in the dependency). To adapt the pebble game to our setting, the algorithm needs to partition the edges in a graph and maintain (a, a)-sparsity on one part and (b, b)-sparsity on the other; this may require dynamically adjusting the partitions.

As we saw in Fig. 1, a framework that is rigid with no generic dependencies, or *generically minimally rigid*, may be in a flexible *special position* caused by the special geometry of its realization. Since a special position is indicated by the vanishing of a polynomial called a framework's *pure condition*, we develop algorithms for finding graph minors, which we refer to as *factor graphs*, corresponding to its factors. In the body-and-bar setting of White and Whiteley (1987), irreducible factor graphs correspond to irreducible minimally rigid subframeworks; their corresponding factors can be expressed as the pure condition of those subframeworks. However, in our setting, we may have irreducible factors that do not have a natural interpretation as the pure condition of any subframework. Yet, the ability to associate combinatorial structures to the factors of the pure condition could give a user additional tools to analyze a system of constraints.

1.4. Organization

We begin with an introduction to body-and-cad rigidity theory in Section 2. In Section 3, we analyze a framework and geometric conditions that lead to special positions, motivating the algorithms for finding generic dependencies (Section 4) and factor graphs for assisting with analysis of special positions (Section 5). We end with conclusions and open questions in Section 6.

2. Background

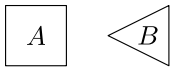
In this section, we review the fundamentals of body-and-cad rigidity theory; for full technical details, refer to Haller et al. (2012) and Lee-St.John and Sidman (2013).

2.1. The combinatorial model

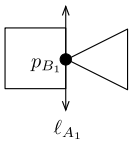
A *body-and-cad framework* consists of n full-dimensional bodies with pairwise *c*oincidence, *a*ngular, or *d*istance constraints between them; these *cad* constraints are specified between *geometric elements* which are affine linear spaces (e.g., a point, line or plane in 3D) rigidly affixed to the bodies.

The allowed motions of a body-and-cad framework are continuous motions of the bodies that preserve the given constraints. A body-and-cad framework is *rigid* when all of the allowed motions are trivial, i.e., they consist of applying the same rigid-body motion to each of the bodies; otherwise it is *flexible*. Note that our model of these systems of constraints is given solely in terms of collections of affine spaces affixed to the rigid bodies and we do not consider the geometry of the body in our analysis; only special properties of the constraints affect our analysis. (For example, any special symmetries the *bodies themselves* may have does not enter the analysis; however, special symmetries *of the constraints* do.)

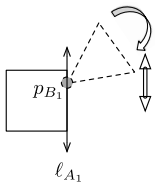
There are 9 different constraint types in 2D and 21 in 3D. Examples of constraints in 2D are: **point–point distance** (a bar), **point–line coincidence**, **point–line distance**, **line–line coincidence**, and **line–line angular**. Each geometric constraint represents *one or more equations* restricting the relative motion of the bodies involved. A constraint can then be further decomposed into *primitive constraints*, which correspond to single equations. Let d be the dimension of the ambient space. Primitive constraints come in two types, which require distinct algebraic treatment: a *blind* constraint can potentially restrict any of the $\binom{d+1}{2}$ relative degrees of freedom, while an *angular* constraint restricts only the $\binom{d}{2}$ relative *rotational* degrees of freedom. Haller et al. (2012) show how to coordinatize infinitesimal Euclidean motions by a $\binom{d+1}{2}$-dimensional vector in such a way that an angular constraint involves only $\binom{d}{2}$ of the variables.



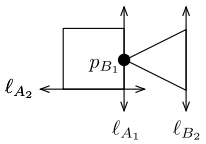
(a) The framework is composed of two rigid bodies A (square) and B (triangle).



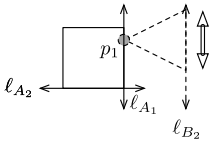
(b) Constraint 1 specifies a **point–line coincidence** that requires point p_{B_1} on the triangle to lie on line ℓ_{A_1} on the square.



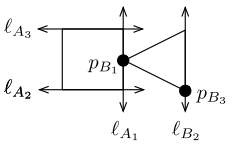
(c) After the specification of Constraint 1, the framework allows internal motion with 2 degrees of freedom.



(d) Constraint 2 specifies a **line–line perpendicular** constraint that requires line ℓ_{B_2} on the triangle to be perpendicular to line ℓ_{A_2} on the square.

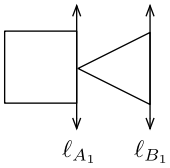


(e) After the specification of Constraints 1 and 2, the framework allows internal motion with 1 degree of freedom.

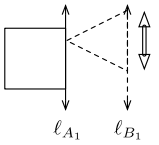


(f) Constraint 3 specifies a **point–line distance** constraint that requires point p_{B_3} on the triangle to be a fixed distance from line ℓ_{A_3} on the square. The resulting framework is minimally rigid.

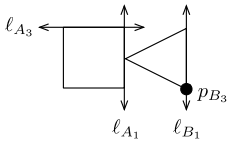
Fig. 2. A generically minimally rigid 2D body-and-cad framework.



(a) Constraint I specifies a **line–line distance** constraint that requires line ℓ_{B_1} on the triangle to be a fixed distance from line ℓ_{A_1} on the square.



(b) After the specification of Constraint I, the framework allows internal motion with 1 degree of freedom.



(c) Constraint II specifies a **point–line distance** constraint that requires point p_{B_3} on the triangle to be a fixed distance from line ℓ_{A_3} on the square. The resulting framework is minimally rigid.

Fig. 3. A 2D body-and-cad framework with different geometric constraints but the same rigidity matrix.

Example 2.1. To give some intuition about body-and-cad rigidity, consider the planar body-and-cad framework in Fig. 2. It is composed of two rigid bodies A (the square) and B (the triangle); placing three cad constraints, a **point–line coincidence**, **line–line perpendicular** and **point–line distance**, results in a *generically minimally rigid* (Definition 2.5) framework.

Now consider the framework in Fig. 3; it is composed of the same two rigid bodies as in Fig. 2, but only includes two cad constraints, a **line–line distance** and **point–line distance**. Because Constraint I (**line–line distance**) is equivalent to Constraints 1 and 2 (**point–line coincidence** and **line–line perpendicular**), this system of constraints is equivalent to the figure in Fig. 2 and is *generically minimally rigid*.

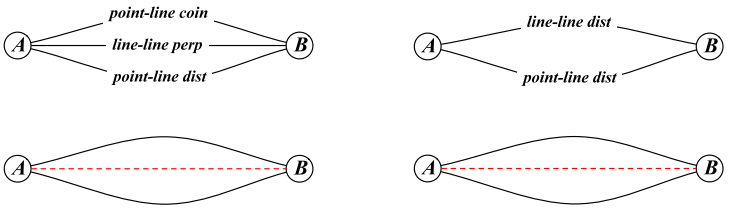
As we just saw (refer to Fig. 3), a single cad constraint may impose restrictions on multiple degrees of freedom in a framework. Each independent restriction on the degrees of freedom of the framework corresponds to a primitive condition, whose combinatorics we collect together in an associated *primitive cad graph*. Fig. 4 shows the original combinatorics (represented by a *cad graph*) and the associated primitive cad graph for the frameworks discussed in Example 2.1.

Definition 2.2. A *primitive cad graph* is a *bi-colored graph* $G = (V, E = R \sqcup B)$ on vertex set $[n] = \{1, \ldots, n\}$ with a vertex for each rigid body, a red edge (in R) for each angular constraint, and a black edge (in B) for each blind constraint.

In the rest of this paper, we will work with primitive cad graphs.

2.2. The rigidity matrix and infinitesimal rigidity

As is standard in the field, we will linearize the geometric constraint equations and consider *infinitesimal rigidity*. Here, the core object of study is a *rigidity matrix* (derived in Haller et al., 2012), whose kernel consists of the infinitesimal motions of the framework.



(a) The *cad graph* (top) for the framework in Fig. 2 has three edges between the vertices for bodies A and B. The **line–line perpendicular** constraint is associated to a primitive angular constraint, so the *primitive cad graph* (bottom) has one red edge.

(b) The *cad graph* (top) for the framework in Fig. 3 has two edges between the vertices for bodies A and B. Because the **line–line distance** constraint is associated to two primitive cad constraints (one blind, one angular), the *primitive cad graph* (bottom) has three edges.

Fig. 4. Combinatorics of body-and-cad frameworks in Figs. 2 and 3: both *cad graphs* (top) are associated to the same *primitive cad graph* (bottom).

To describe body-and-cad rigidity matrices combinatorially, we use the following concept.[7]

Definition 2.3. For integers a, b, let $k = a + b$. We define an $[a, b]$-*frame* $G(\mathbf{p})$ to be a bi-colored graph $G = (V, E = R \sqcup B)$ with $n = |V|$ and $|E| = kn - k$, along with a function $\mathbf{p} : E \to \mathbb{R}^k$. The function $\mathbf{p}$ labels each edge with a k-vector, which is zero in the last b entries if the edge is in R. The *generic* $[a, b]$-*frame* $G(\mathbf{x})$ has formal indeterminates replacing the nonzero coordinates of $\mathbf{p}$.

We define the rigidity matrix in terms of $[a, b]$-frames. We first fix some ordering on the edges of G.

Definition 2.4. The *rigidity matrix* $M(G(\mathbf{p}))$ of an $[a, b]$-frame $G(\mathbf{p})$ is a matrix that has k columns for each vertex i and one row for each edge of G. In the row corresponding to an edge e with endpoints i and j (where $i < j$), we have $\mathbf{p}(e)$ in the columns corresponding to i, $-\mathbf{p}(e)$ in the columns corresponding to j, and zeroes in all other entries. Order the rows of the rigidity matrix in the order of the edges of G.

Definition 2.5. We say that an $[a, b]$-frame $G(\mathbf{p})$ is *generically minimally rigid* if the associated generic $[a, b]$-frame $G(\mathbf{x})$ has a rigidity matrix $M(G(\mathbf{x}))$ with rank $kn - k$ for almost all specializations of $\mathbf{x}$.

The generic rigidity matrix for the example in Fig. 3 is shown below.

$$\begin{array}{l} \\ \text{line–line distance (blind part)} \\ \text{line–line distance (angular part)} \\ \text{point–line distance (non-generic)} \end{array} \begin{array}{c} \begin{array}{cccccc} A_1 & A_2 & A_3 & B_1 & B_2 & B_3 \end{array} \\ \begin{pmatrix} x_1 & x_2 & x_3 & -x_1 & -x_2 & -x_3 \\ y_1 & 0 & 0 & -y_1 & 0 & 0 \\ z_1 & z_2 & z_3 & -z_1 & -z_2 & -z_3 \end{pmatrix} \end{array}$$

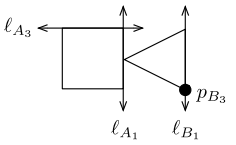
It has 3 columns for each body, corresponding to the one rotational and two translational degrees of freedom for instantaneous rigid body motion in the plane. We order the columns so that they are in groups of 3, with translational components last: column A_1 corresponds to the rotational component, and columns A_2 and A_3 to the translational components, with body B's columns ordered analogously. There is a row for each primitive constraint; notice that the row for the primitive angular constraint associated to the **line–line distance** constraint (highlighted in red) has zeroes in the columns corresponding to the translational degrees of freedom.

While the rank of a generically minimally rigid framework is $kn - k$ for almost all realizations, there are realizations for which the rank drops. These correspond to non-generic realizations, or *special positions*, of the generically minimally rigid graph.

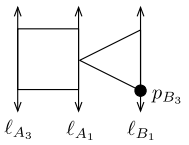
Example 2.6. In Fig. 5 we construct a special position of the framework specified by the graph from Fig. 3. By changing the placement of the line on body A in Constraint II, the resulting framework remains flexible. Its rigidity matrix is shown below[8] and contains a dependency (the third row is the sum of the first two), causing its rank to drop.

[7] The $[a, b]$-frame defined here is equivalent to the $(a + b, a)$-frame defined in Lee-St.John and Sidman (2013).

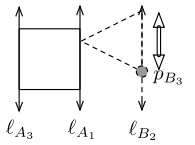
[8] The coordinates of the framework are determined by the specification from Haller et al. (2012).



(a) As in Fig. 3, we specify a **point–line distance** constraint that requires point p_{B_3} on the triangle to be a fixed distance from line ℓ_{A_3} on the square. The resulting (generic) framework is minimally rigid.



(b) An alternative choice of the line on body A involved in Constraint II.



(c) The resulting flexible framework has a dependency and admits an internal motion with 1 degree of freedom.

Fig. 5. A **flexible special position** of a generically minimally rigid 2D body-and-cad framework.

$$\begin{array}{l} \\ \text{point–line coincidence} \\ \text{line–line perpendicular} \\ \text{point–line distance (non-generic)} \end{array} \begin{array}{c} \begin{array}{cccccc} A_1 & A_2 & A_3 & B_1 & B_2 & B_3 \end{array} \\ \begin{pmatrix} 1 & 0 & -1 & -1 & 0 & 1 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

3. Motivating case study

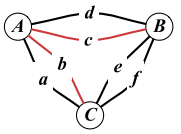
In this section we analyze in detail the 2D body-and-cad framework consisting of 3 bodies, 2 bars, and 2 line–line coincidence constraints depicted in Fig. 1. This case study will help motivate the two algorithms that we will subsequently present that detect generic dependencies (Algorithm 1 in Section 4) and find factor graphs for dependencies arising in special positions (Algorithm 4 in Section 5).

The associated primitive cad graph, in which an edge corresponds to a linear constraint, is given in Fig. 6(a). In this graph, each line–line coincidence is represented by two edges: a red edge that corresponds to a line–line parallel constraint (which restricts only angular motion) and a black edge that corresponds to a point–line coincidence constraint (which restricts one translational degree of freedom). Each bar eliminates 1 degree of freedom and is represented by a black edge.

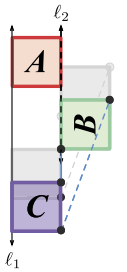
Since the framework is in 2D, we will work with the generic rigidity matrix associated to the $[1,2]$-frame using edge labels a, b, c, d, e, f:

$$\begin{array}{c} \begin{array}{ccccccccc} A_1 & A_2 & A_3 & B_1 & B_2 & B_3 & C_1 & C_2 & C_3 \end{array} \\ \begin{pmatrix} a_1 & a_2 & a_3 & 0 & 0 & 0 & -a_1 & -a_2 & -a_3 \\ b_1 & 0 & 0 & 0 & 0 & 0 & -b_1 & 0 & 0 \\ c_1 & 0 & 0 & -c_1 & 0 & 0 & 0 & 0 & 0 \\ d_1 & d_2 & d_3 & -d_1 & -d_2 & -d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & e_1 & e_2 & e_3 & -e_1 & -e_2 & -e_3 \\ 0 & 0 & 0 & f_1 & f_2 & f_3 & -f_1 & -f_2 & -f_3 \end{pmatrix} \end{array}$$

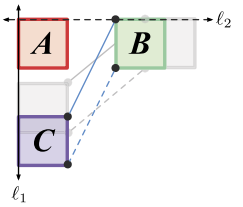
One can check that the rank of this matrix is exactly 6 generically; since $k = 3$ for 2D frameworks, this implies that the framework is generically minimally rigid. Any additional constraints would cause a *generic dependency*. In Section 4, we present Algorithm 1, which determines generic rigidity via a combinatorial characterization; if a generic dependency is present, it further identifies the minimal set of dependent constraints, or *fundamental circuit*.



(a) The primitive cad graph for the framework in Fig. 1. Red edges represent angular constraints.



(b) The special position in which $\ell_1 \| \ell_2$.



(c) The special position in which the bars are parallel.

Fig. 6. A 2D body-and-cad example on 3 bodies; we assume that body C is tied down.

As we will see in Section 5, it is convenient to work with a polynomial called the *pure condition* that captures the rank of the rigidity matrix. We imagine immobilizing body C, which has the effect of removing the last three columns of the rigidity matrix. The determinant of what remains gives the pure condition and may be expressed as the *bracket polynomial* $[abc][def]-[abd][cef]$, where the *bracket* $[abc]$ denotes the 3×3 determinant of the matrix whose rows are a, b and c. This pure condition is not identically zero, which is expected since the framework is generically minimally rigid. Since special positions arise when the rank of the rigidity matrix drops, these occur precisely when the pure condition vanishes. We are particularly interested in identifying subsystems where a dependency occurs, motivating the study of the factors of the pure condition.

A special position occurs when a factor of the pure condition vanishes. To give a geometric interpretation of the vanishing of the factors, we must first explain how the edge labels are obtained. We adopt the projective geometry setting of White and Whiteley (1987), so that the framework is realized (as a set of bodies) in the affine patch of $\mathbb{P}^2$ that has coordinates $[1 : x : y]$. This apparent complication is justified by the fact that infinitesimal Euclidean motions are then naturally identified with points in $(\mathbb{P}^2)^*$. Since the infinitesimal constraints implied by the edge labels are, essentially, blocked motions, we interpret them as points in $(\mathbb{P}^2)^*$ as well, allowing us to study their geometry as a point configuration. (See White and Whiteley (1987) and also Haller et al. (2012) for a detailed treatment and the specific construction.)

We can see from the matrix that the bracket $[abc]$ evaluates to zero, simplifying the pure condition to $[abd][cef]$. A special position exists when either factor of $[abd][cef]$ vanishes. Note that the brackets $[abd]$ and $[cef]$ represent reducible polynomials in the coordinates of $a, \ldots, f$. In fact, $[abd] = b_1(a_2d_3 - a_3d_2)$, and $[cef] = c_1(e_2f_3 - e_3f_2)$. Thus, the pure condition is composed of four irreducible factors b_1, c_1, $a_2d_3 - a_3d_2$, and $e_2f_3 - e_3f_2$, whose vanishing characterize the special positions of this framework.

The next conceptual step is to relate the vanishing of the irreducible factors of the brackets derived above to our projective geometry setup. First note that since $b = [b_1 : 0 : 0]$ is a point in $(\mathbb{P}^2)^*$, b_1 cannot be zero. Therefore, if $[abd] = b_1(a_2d_3 - a_3d_2) = 0$ then $(a_2, a_3) = \lambda(d_2, d_3)$ for some nonzero λ.

The point $a \in (\mathbb{P}^2)^*$ determines a line in $\mathbb{P}^2$; we can think of a as a vector in $\mathbb{R}^3$ that is normal to a plane that projectivizes to a line in $\mathbb{P}^2$. The intersection of these two lines in $\mathbb{P}^2$ (respectively, planes in $\mathbb{R}^3$), is given by a system of equations represented by the matrix

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ d_1 & d_2 & d_3 \end{pmatrix}$$

Assuming that $a \neq d$, if $a_2d_3 - a_3d_2 = 0$ this is equivalent to a system of equations of the form

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & d_2 & d_3 \end{pmatrix}.$$

In other words, the intersection of the lines in $\mathbb{P}^2$ corresponding to a and d is a point of the form $[0 : -d_3 : d_2]$, which is a point on the line at infinity. Two lines meet at a point on the line at infinity if and only if they are parallel. Hence, we can conclude that $[abd]$ vanishes when the lines ℓ_1 and ℓ_2 which are denoted by ℓ_1 and ℓ_2 in Fig. 6(b) are parallel. The analysis for the vanishing of the irreducible factors of $[cef]$ is analogous; there is a special position when the lines determined by the bars e and f are parallel, as in Fig. 6(c).

As this case study shows, finding the factors of the pure condition can provide information about the subframeworks of a framework in a special position that contain a dependency. The algorithm that we will present in Section 5 (Algorithm 4) finds a canonical set of graph minors, which we refer to as *factor graphs*, associated with the factors of the pure condition; for this example, it produces factor graphs associated with precisely the irreducible factors described above, as depicted in Fig. 9.

4. Detecting generic dependencies with $[a, b]$-sparsity

We review the notion of $[a, b]$-sparsity, used to characterize body-and-cad rigidity, and present the $[a, b]$-**pebble game** algorithm, which characterizes $[a, b]$-sparsity and consequently addresses *generic* body-and-cad rigidity. If the addition of an edge results in a dependency in a generic realization of the system, the pebble game will find its fundamental *circuit* (minimal set of constraints involved in the dependency).

4.1. The combinatorics of minimally rigid graphs

A result from Lee-St.John and Sidman (2013) gives a combinatorial characterization of generic minimal rigidity for 2D body-and-cad frameworks (with $[1, 2]$-frames) and, omitting point–point coincidence constraints, for 3D body-and-cad frameworks (with $[3, 3]$-frames):

Theorem 1. *An $[a, b]$-frame with bi-colored graph $G = (V, E = R \sqcup B)$ is generically minimally rigid if and only if $\exists B' \subseteq B$ such that:*

- *$(V, R \cup B')$ is the edge-disjoint union of a trees, and*

- $(V, B \setminus B')$ *is the edge-disjoint union of* b *trees*

Theorem 1 can also be stated in terms of *hereditary sparsity*, which we now recall. A multigraph $G = (V, E)$ is (k, ℓ)-*sparse* if every subset of n' vertices spans at most $kn' - \ell$ edges; if in addition, G has exactly $kn - \ell$ edges, it is called (k, ℓ)-*tight*. For brevity, (k, ℓ)-tight graphs will be called (k, ℓ)-*graphs*. A subset of vertices of G that induces a (k, ℓ)-graph is a (k, ℓ)-*block*. When $\ell \in [0, 2k)$, (k, ℓ)-graphs are the bases of the (k, ℓ)-*matroid* (Lee and Streinu, 2008).

Definition 4.1. Let $G = (V, E = B \sqcup R)$ be a bi-colored graph, a, b be positive integers, and $k = a + b$. Then G is $[a, b]$-*sparse*[9] if $\exists B' \subseteq B$ such that:

- $(V, R \cup B')$ is (a, a)-sparse, and
- $(V, B \setminus B')$ is (b, b)-sparse

Additionally, if G has exactly $kn - k$ total edges, then G is $[a, b]$-*tight* and referred to as an $[a, b]$-*graph*.

A subset of vertices of an $[a, b]$-sparse graph that induces an $[a, b]$-graph is an $[a, b]$-*block*.

That this class is matroidal follows from the Matroid Union Theorem (Brylawski, 1986, Prop. 7.6.14). Therefore, for an $[a, b]$-sparse graph $G = (V, E)$ and an edge e not in E, we will say that e is *independent* of G if $G + e$ is also $[a, b]$-sparse and *dependent* otherwise.

A straightforward application of the Nash-Williams and Tutte Theorem (Tutte, 1961; Nash-Williams, 1961) implies that generic minimal rigidity of body-and-cad frameworks is characterized by $[1, 2]$-sparsity in the plane and, omitting point–point coincidence constraints, $[3, 3]$-sparsity in 3D (Lee-St.John and Sidman, 2013).

4.2. Pebble games for $[a, b]$-sparsity

Algorithm 1 describes our $[a, b]$-*pebble game* for determining $[a, b]$-sparsity properties as well as detecting the fundamental circuit of a dependent edge. This algorithm belongs to a family of pebble game algorithms (Streinu and Theran, 2009; Lee and Streinu, 2008) that are based on a set of local moves applied to the edges of a directed graph, where the edges and vertices are covered by pebbles representing degrees of freedom. The specific preconditions for each type of move, which are related to the sparsity parameters, determine the sparsity family recognized by the game.

One way of intuitively understanding the $[a, b]$-pebble game is to imagine separate (a, a)- and (b, b)-pebble games played on sets A and T, which partition the current edge set, respectively. The aqua-colored pebbles track the edges in A as well as the (a, a)-sparsity of that partition; the tan-colored pebbles do the same for T with (b, b)-sparsity counts.[10] The $[a, b]$-pebble game relies on moves that permit black edges to move between A and T in a controlled manner, which corresponds to collecting additional pebbles of certain colors, using a subroutine described in Algorithm 2. To find a sequence of these moves, Algorithm 2 specializes Knuth's matroid union algorithm (Knuth, 1973) to the $[a, b]$-sparsity matroid using pebble games. By enqueuing unvisited edges (in $F' \setminus F$), it uses a breadth-first approach to find the shortest path (stored with *predecessor* pointers) to an edge whose pebble color can be exchanged. To help illustrate the algorithm, Fig. 7 shows some steps of the pebble game.

4.3. Correctness

We show correctness of Algorithm 1.

Theorem 2. *A bi-colored graph is* $[a, b]$*-sparse if and only if it can be constructed with the* $[a, b]$*-pebble game.*

We are going to prove that Algorithm 1 correctly characterizes $[a, b]$-sparse graphs and that it can be used to find circuits. As mentioned above, we rely on Knuth's algorithm for matroid union (Knuth, 1973) (see Schrijver, 2003, Sec. 42.3, for a modern treatment). Before giving the details, we remark that Knuth's algorithm is a meta-algorithm that relies on calls to independence oracles for the two matroids underlying the union. Here, these two matroids are the (a, a)- and (b, b)-sparsity matroids, with appropriately chosen ground sets. Thus, we could simply apply Knuth's scheme using the (a, a)- and (b, b)-pebble games as the oracles, instead of developing the $[a, b]$-pebble game and showing it simulates Knuth's search procedure. However, the $[a, b]$-pebble game presented here saves a factor of $O(n)$ over the more naïve approach, as

[9] As we will rely on both concepts of sparsity, we draw the reader's attention to the use of *parentheses* to denote the parametrized (k, ℓ)-sparsity of uncolored graphs (appearing for classical bar-and-joint and body-and-bar rigidity) and the use of *square brackets* to denote the parametrized $[a, b]$-sparsity counts of bi-colored graphs (introduced for body-and-cad rigidity).

[10] We chose the colors aqua and tan to be associated with A and T; in the rigidity setting, the aqua-covered A partition can be thought of as tracking the angular degrees of freedom of the system, while the tan-colored T partition tracks the translational degrees of freedom.

Algorithm 1 The $[a, b]$-pebble game algorithm.

Input: A bi-colored graph $G = (V, E = R \sqcup B)$, with red R and black B edges.
Output: $[a, b]$-sparsity property **tight**, **sparse**, **dependent and contains spanning tight**, or **dependent**.
Setup: *Initialize an empty directed graph H on vertex set V. On each vertex, place a aqua pebbles and b tan pebbles.*
Allowed moves:

Add red edge ij [Precondition: $\geq a+1$ aqua pebbles on i and j.]
– *Add the new edge, cover it with an aqua pebble from i (there is one by the precondition).*
– *Orient ij out of i.*
Add black edge ij [Precondition: $\geq a+1$ aqua pebbles on i and j or $\geq b+1$ tan pebbles on i and j.]
– *Add the new edge; cover it with a pebble from i using aqua (if there are $a+1$ aqua) or tan (if there are $b+1$ tan).*
– *Orient ij out of i.*
Edge reversal [Precondition: vertex j has a pebble on it and an in-edge ij covered by the same color.]
– *Reverse the edge by orienting it as ji out of j, covering with the pebble from j and returning the (same color) pebble originally covering ij to i.*
Aqua exchange edge reversal [Precondition: vertex j has an aqua pebble on it and a black in-edge ij covered by a tan pebble; i and j do not belong to the same (a, a)-component of aqua pebble covered edges.]
– *Reverse the edge by orienting it as ji out of j, covering with the aqua pebble from j and returning the tan pebble originally covering ij to i.*
Tan exchange edge reversal [Precondition: vertex j has a tan pebble on it and a black in-edge ij covered by an aqua pebble; i and j do not belong to the same (b, b)-component of tan pebble covered edges.]
– *Reverse the edge by orienting it as ji out of j, covering with the tan pebble from j and returning the aqua pebble originally covering ij to i.*

Method:

1. For each edge $e \in E$
   (a) If e is black: attempt to collect $b+1$ tan pebbles on its endpoints with Algorithm 2.
   (b) If Algorithm 2 returns **true**: insert e with an **add black edge** move.
   (c) Else, or if e is red: attempt to collect $a+1$ aqua pebbles on its endpoints with Algorithm 2.
   (d) If Algorithm 2 returns **true**: insert e with an **add black/red edge** move.
   (e) Else: *reject* it and highlight the edges returned by Algorithm 2 as the fundamental circuit of the edge (if e is black, this is the union of both calls to Algorithm 2).
2. If every edge is added: output **tight** if there are $a+b$ pebbles left and **sparse** otherwise.
3. Else, there were rejected edges: output **dependent and contains spanning tight** if there are $a+b$ pebbles left and **dependent** otherwise.

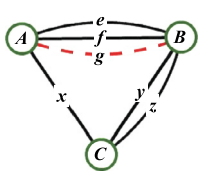
Algorithm 2 The subroutine for finding pebbles for the $[a, b]$-pebble game.

Input: An $[a, b]$-pebble game configuration (a directed bi-colored graph), an edge e, and a desired additional pebble color c_e (**aqua** or **tan**).
Output: true if $a+1$ aqua (if c_e is aqua) or $b+1$ tan (if c_e is tan) pebbles can be collected on the endpoints of e or **false** otherwise, along with the set of visited edges.
Method:

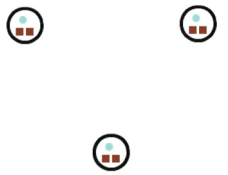
1. Initialize set $F = \emptyset$.
2. Initialize queue $Q = \emptyset$. Entries of Q will be of the form (f, c), recording an edge on which to cover with a pebble of color c.
3. Set $e.predecessor = \text{NIL}$.
4. Enqueue (e, c_e) into Q.
5. While Q is not empty
   (a) Dequeue (f, c).
   (b) If $f \neq e$ and f is red, continue to the next iteration of the loop.
   (c) Use the basic pebble game rules to try to collect $a+1$ (if c is **aqua**) or $b+1$ (if c is **tan**) pebbles on the endpoints of f; let F' be the set of edges visited by that search.
   (d) If the pebbles were collected
      i. Let $g = f$.
      ii. While $g.predecessor \neq \text{NIL}$
         A. Let d be the color of the pebble covering g, $\overline{d}$ be the opposite color, u and v the source and target of g.
         B. Collect a pebble of color $\overline{d}$ on v using the basic pebble game rules with **edge reversal** moves.
         C. Perform a $\overline{d}$ **exchange edge reversal** move to reverse the edge from v to u, covering it with the $\overline{d}$-colored pebble and releasing a d-colored pebble back onto u.
         D. Set $g = g.predecessor$.
      iii. Collect $a+1$ (if c is **aqua**) or $b+1$ (if c is **tan**) pebbles on the endpoints of $g(= e)$.
      iv. Output **true** and $F \cup F'$.
   (e) Otherwise
      i. For each edge $g \in F'$ that is not in F
         A. Set $g.predecessor = f$; let $\overline{c}$ be the opposite of color c.
         B. Enqueue $(g, \overline{c})$ into Q.
      ii. Assign $F = F \cup F'$.
6. Output **false** and F.

we will see in the complexity analysis. Nonetheless, Lemma 4.2, below, follows from Theorem 3 and the proof of Lemma 4.3, so the direct argument below is meant to be informative.

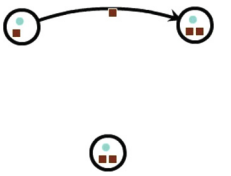
In what follows, we describe a pebble game configuration as (H, A, T), where H is a bi-colored directed graph on vertex set V, with a pebble covering every edge and some free pebbles on vertices; A is the set of edges covered by aqua pebbles and T is the set of edges covered by tan pebbles. We will also abuse notation slightly and use the same symbols H, A, and T to describe their underlying undirected graphs. Finally, we will use the notation $S + e$ to denote $S \cup \{e\}$ and $S - e$ to denote $S \setminus \{e\}$.



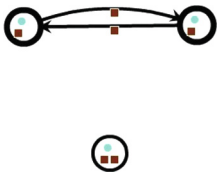
(a) The input is a primitive cad graph with 5 black (solid) edges and 1 red (dashed) edge.



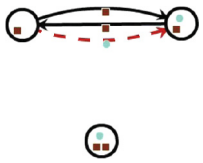
(b) The setup stage places $a = 1$ aqua (circular) pebble and $b = 2$ tan (square) pebbles on each vertex.



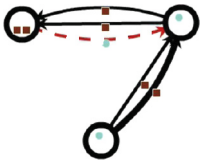
(c) Since there were at least $b + 1 = 3$ tan pebbles on its endpoints, the black edge e is inserted with an **add black edge** move.



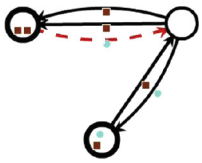
(d) Another **add black edge** move inserts the edge f. While the direction is arbitrarily chosen, a pebble from the source is used to cover the edge.



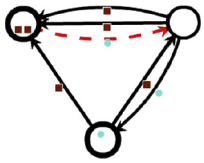
(e) Since there were $a + 1 = 2$ aqua pebbles on its endpoints, the red edge g is inserted with an **add red edge** move.



(f) A few moves later, an **aqua edge exchange reversal** move is possible: B has an aqua pebble and a tan pebble covered black in-edge z, whose endpoints are not in a $(1, 1)$-component.



(g) The edge z is subsequently reversed, covered by the aqua pebble from B, releasing a tan pebble onto C.



(h) Finally, all edges are successfully inserted with exactly $a + b = 3$ pebbles remaining; the output is **tight**.

Fig. 7. The $[a, b]$-pebble game (Algorithm 1) played on a primitive cad graph for a generically minimally rigid framework determines that it is $[1, 2]$-tight.

Lemma 4.2. *The underlying graph of any pebble game configuration is $[a, b]$-sparse with A as an (a, a)-sparse graph and T as a (b, b)-sparse graph.*

Proof. We show something slightly stronger, which is that the underlying graph H constructed by applying *any sequence* (as opposed to only the ones found by the algorithm) of the pebble game moves is always $[a, b]$-sparse.

The key invariant is that, after any sequence of moves, A and T both induce pebble game configurations for the *basic* (uncolored) pebble game from Lee and Streinu (2008). As an immediate consequence, we obtain that A remains (a, a)-sparse and B remains (b, b)-sparse. This certifies that H is $[a, b]$-sparse. The invariant clearly holds at initialization, so we proceed by induction on the number of moves.

For the inductive step, we first consider all the moves except for the **exchange edge reversal** moves. We observe that, assuming the required preconditions, these operate entirely on either A or T as a configuration, so the inductive step for them follows directly from Lee and Streinu (2008).

To complete the induction, consider the **aqua edge exchange reversal** move, since the tan one has an analogous proof. The precondition, that the edge $ij \in T$ is not in an (a, a)-component of A, implies that $A + ij$ is (a, a)-sparse. From this, the pebble game invariants of Lee and Streinu (2008) imply that there are a aqua pebbles (distinct from the one on j) reachable from i via paths using only edges in A. By induction, A could have been built by the basic (a, a)-pebble game; then ij could be added to A by basic pebble game searches. Notice that returning the tan-colored pebble to j maintains T as a basic (b, b)-pebble game configuration. □

The other direction is captured by the following lemma.

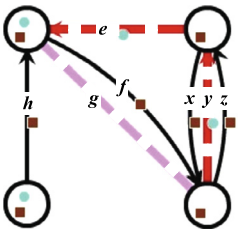
Lemma 4.3. *If an edge is independent of the underlying graph of a pebble game configuration, then the pebble game will successfully insert it.*

Before giving the proof, we briefly review Knuth's algorithm and establish some terminology, specialized to our setup. The algorithm operates on a directed, bipartite graph associated with a pebble game configuration (H, A, T) and an edge e, which is not in H. This graph, denoted Γ_{H+e}, has vertex set given by the edges of H, i.e., $A \sqcup T$, along with two *terminal* vertices α and τ, and an additional vertex for the edge e. First, we describe the edges originating and terminating at a vertex $x \notin \{e, \alpha, \tau\}$.

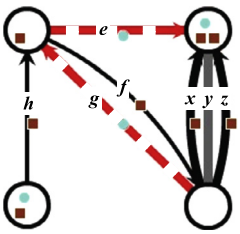
There is a directed edge from vertex x to y, written $x \to y$ if

$$\begin{cases} (A - y) + x \text{ is } (a, a)\text{-sparse} & \text{if } x \in T \ \& \ y \in A \\ (T - y) + x \text{ is } (b, b)\text{-sparse} & \text{if } y \in T \ \& \ x \in A \cap B, \text{ i.e., } x \text{ is a black edge in the aqua partition} \end{cases}$$

Additionally, there is an edge $x \to \alpha$ if $x \in T$ and $A + x$ is (a, a)-sparse, and there is an edge from $x \to \tau$ if $x \in A \cap B$ and $T + x$ is (b, b)-sparse. The edges originating at e are defined similarly. This case distinction is simply to make it clear that no edges in Γ_{H+e} have e as their target.



(a) The fundamental $[1,2]$-circuit for rejected edge g detected by the pebble game is the set of edges $\{e, g, y\}$. The set $\{e, y\}$ forms an angularly rigid block.



(b) The fundamental $[1,2]$-circuit for rejected edge y detected by the pebble game is the set of edges $\{e, g, x, y, z\}$. The set $\{x, z\}$ forms a contextually rigid block due to the presence of the angular block $\{e, g\}$.

Fig. 8. Unlike (k,ℓ)-circuits, removing any edge from a $[1,2]$-circuit may not produce a $[1,2]$-graph.

A path $x_0 \to x_1 \to \cdots \to x_\pi$ has a *shortcut* in a graph if there exists a $j > i+1$ such that $x_i \to x_j$ is an edge. In particular, if $x_0 \to x_1 \to \cdots \to x_\pi$ is a shortest path in a graph, it does not have a shortcut.

Given a path from e to a terminal vertex, *recoloring along the path* means putting x_i in the part of the partition containing x_{i+1}, with α always in A and τ always in T.

The main result of Knuth (1973), again specialized for our setup, is:

Theorem 3. *Let (H, A, T) be a pebble game configuration and e an edge not in the underlying graph. Then there is a directed path in Γ_{H+e} from e to α or τ if and only if $H+e$ is independent. Moreover, given a path $e = x_0 \to x_1 \to \ldots \to x_\pi \in \{\alpha, \tau\}$ in Γ_{H+e} that does not have a shortcut, a partition of $H+e$ certifying $[a,b]$-sparsity can be found by recoloring along this path.*

The proof of Lemma 4.3 amounts to showing that Algorithm 2 is simulating Knuth's algorithm.

Proof of Lemma 4.3. Assume that e is independent of the underlying graph of a pebble game configuration H. We need to show that Algorithm 2 will succeed in collecting enough pebbles on the endpoints of e. This is done by comparing the pebble search procedure in Algorithm 2 to Knuth's algorithm.

First consider, in the main loop of Algorithm 2, the conditional block predicated upon when $a+1$ (if c is aqua) or $b+1$ (if c is tan) pebbles can be collected on the endpoints of f. Note that a aqua or b tan pebbles can always be collected on any vertex by Lee and Streinu (2008). The additional pebble can be collected if and only if the edge f can be moved to the opposite part of the partition without violating sparsity. This is equivalent to there being an edge $f \to \{\alpha, \tau\}$ in Γ_{H+e}.

Otherwise, the pebble search fails. In this case, Lee and Streinu (2008) implies that $F' + f$ is the fundamental circuit of f in the c-colored part of the partition; i.e., $g \in F'$ if and only if there is an edge $f \to g$ in Γ_{H+e}. Therefore, F' is exactly the set of neighbors of f in Γ_{H+e}.

By enqueuing those edges in F' not already in F, Algorithm 2 is, in fact, searching Γ_{H+e} in a breadth-first fashion. By Theorem 3, the assumption that $H+e$ is $[a,b]$-sparse implies that there is a path from e to a terminal vertex in Γ_{H+e}. Therefore, Algorithm 2 will be able to collect $a+1$ aqua or $b+1$ pebbles on the endpoints of some edge f, implying that there is an edge from f to a terminal in Γ_{H+1}. Let p be the path in Γ_{H+e} defined by following predecessor pointers from f. Since Algorithm 2 implements breadth-first search on Γ_{H+e}, p is shortcut free.

Theorem 3 then implies that recoloring along p preserves the (a,a)- and (b,b)-sparsity of A and T at every step. The main results of Lee and Streinu (2008) then imply that it will always be possible to meet the preconditions of the **exchange edge reversal** moves to implement the recoloring by using only basic pebble searches on A or T. Thus, the pebble game moves implementing the recoloring along p will succeed, and the $[a,b]$-pebble game will insert e. □

4.4. Circuits

The pebble game also detects $[a,b]$-circuits, an approach that is perhaps less well-known, but appears before in Lee and Streinu (2008, Section 6), and has been used in Berardi et al. (2011). Note that the presence of red edges creates the possibility of many types of circuits. Some may be circuits as uncolored $(a+b, a+b)$ graphs, others may be (a,a)-circuits, and there are yet other types. The examples in Fig. 8 demonstrate a property of circuits that does not arise in the (k,ℓ)-sparsity matroids. While every (k,ℓ)-circuit is (k,ℓ)-spanning, or "rigid," an $[a,b]$-circuit may actually be "flexible." Dropping an edge of a (k,ℓ)-circuit always results in a tight graph, but dropping an edge of an $[a,b]$-circuit can result in a sparse (but not tight) graph.

Whenever we fail to insert an edge, Algorithm 2 finds its fundamental circuit.

Lemma 4.4. *Let F be the set of edges returned by Algorithm 2. The fundamental circuit of e in the configuration graph H is $F+e$.*

Proof. We must show that $F + e$ is dependent and that, for any $y \in F$, $F + e - y$ is independent. Observe that $F + e$ corresponds to the set of vertices reachable from e in the Knuth graph Γ_{H+e}. By the definition of F, every directed path in Γ_{H+e} that starts at e is contained in Γ_{F+e}. Therefore, there is no path from e to a terminal in Γ_{F+e}, and Theorem 3 implies that $F + e$ is not $[a, b]$-sparse.

Now, let $y \in F$. By construction of F, y is on a short-cut free directed path starting at e. Therefore, there is an $x \in F + e$ on this path with $x \to y$ an edge of Γ_{H+e}. By definition, removing y results in an edge from x to a terminal, providing a path from e to a terminal. In other words, $F + e - y$ is $[a, b]$-sparse for all $y \in F$, which completes the proof of correctness. □

4.5. Complexity analysis

The running time of Algorithm 1 for a graph with n vertices and m edges is $O(mn^2)$, which is $O(n^4)$. First we observe that collecting the initial $a + b$ pebbles in Algorithm 1 requires $O(n)$ for each edge (a total of $O(mn)$) and that the rest of the steps may be charged to $O(m)$ invocations of Algorithm 2.

The running time of Algorithm 2, which is additionally used to detect the fundamental circuit of a dependent edge, is $O(n^2)$. This is because each of the $O(n)$ edges in the configuration is enqueued at most once in the main loop, and each edge that is enqueued triggers a (k, k)-pebble game search requiring $O(n)$ steps by Lee and Streinu (2008), after first copying a configuration of size $O(n)$.

By way of comparison, a direct application of Knuth's algorithm leads to a more expensive running time. In this approach, one might build the bipartite graph explicitly and use the basic pebble game to test each possible edge. The graph Γ_{H+e} has $O(n^2)$ edges, and each check would require an $O(n^2)$-time run of the (k, k)-pebble game; this would result in a total of running time of $O(mn^4) = O(n^5)$ (any call to the pebble game would have $O(n)$ edges as the input graph is sparse).

4.6. Finding components

Within an $[a, b]$-sparse graph, an *induced* $[a, b]$*-component* is a vertex-maximal $[a, b]$-block. It is straightforward to adapt Algorithm 1 to maintain and detect induced $[a, b]$-components, as in the (k, ℓ)-pebble game algorithm with components described in Lee and Streinu (2008). The running time would remain $O(n^4)$.

Note that any edge with vertices contained in an induced $[a, b]$-component is dependent and will be rejected by this adapted pebble game in $O(1)$ time. However, an edge may be dependent without being contained in an induced $[a, b]$-component, as the circuits in Fig. 8 demonstrate. Therefore, *unlike the* (k, ℓ)*-sparsity case*, we do not save a factor in the running time by maintaining induced $[a, b]$-components.

5. Analyzing non-generic dependencies with factor graphs

As demonstrated in Section 3, a generically minimally rigid framework may be in a *special position* that admits internal motions and contains dependencies. In this section, we develop algorithms that will decompose such a framework into its *factor graphs*, permitting analysis of which subframeworks contain the dependencies. We begin by reviewing basic information about the pure condition of an $[a, b]$-graph. We then present a simple algorithm that uses the pebble game to find the factor graphs of the pure condition of an (uncolored) (k, k)-graph, which correspond to the irreducible factors of its pure condition. Finally, we provide some technical background required for analyzing the more complicated structure of the pure condition of an $[a, b]$-graph and conclude with factor graph algorithms in this setting.

5.1. Structure of the pure condition

In this section, we review the definition of the polynomial called the *pure condition* that is associated to a body-and-cad framework. If G is a minimally rigid graph and $\mathbf{p}$ is generic, the kernel of $M(G(\mathbf{p}))$ contains exactly the space of trivial infinitesimal motions of $G(\mathbf{p})$, corresponding to rigid-body motions of the entire framework as a single unit. To remove these, we choose some i with $1 \leq i \leq n$, and construct the *standard tie-down* at body i by appending to $M(G(\mathbf{p}))$ a $k \times kn$ matrix whose only nonzero entries are given by the identity matrix in the k columns associated to body i.[11] We denote the rigidity matrix of $G(\mathbf{p})$ with a tie-down by $M_T(G(\mathbf{p}))$.

Definition 5.1. The *pure condition* P_G of a tied down $[a, b]$-graph G is the determinant of $M_T(G(\mathbf{x}))$.

The pure condition depends, a priori, on the choice of tie-down of G. We will show that as in the body-bar setting of White and Whiteley (1987), this dependence can be removed.

[11] The notion of tie-downs can be substantially generalized to generic tie-downs of any $[a, b]$-sparse graph. For our purposes, this would only complicate the notation, so we restrict our attention to standard tie-downs.

Theorem 4. *The pure condition of a tied down $[a,b]$-graph G is non-zero for* any *choice*[12] *of the tie down and has the form $P_G = T_G \cdot C_G$, where T_G depends on the tie down and C_G is independent of the tie down.*

The proof is relatively standard and similar to what can be found in White and Whiteley (1987), Whiteley (1988); for completeness we provide it in Appendix A. This theorem implies that we only need to consider C_G, the "critical factor" of the pure condition; for the remainder of the paper, we will abuse notation and refer to C_G as the pure condition of G.

5.2. Factors of the pure condition and factor graphs

We start with the simpler setting arising in *body-and-bar*, where rigidity is characterized via (b,b)-sparsity counts, before moving to the more complicated setting of body-and-cad.

5.2.1. Body-and-bar factors

Note that a $[0,b]$-graph is a (b,b)-graph, and in this case White and Whiteley (1987) showed that there is a one-to-one correspondence between factors of C_G and what we will call *factor graphs* of G. A graph H is a factor graph of a (k,k)-graph G if H is a minor,[13] and $C_G = hf$, where the variables that appear in h are the edge labels of H. Denote by G/H the minor obtained by deleting all the edges in H and identifying its vertices. All the factor graphs H in White and Whiteley (1987) have the same form: H is it itself a (b,b)-graph, and the factor associated with H is the pure condition of $H(\mathbf{x})$. A corollary of White and Whiteley's result is that, in fact, every factor of C_G arises this way.

The intuition for studying these factor graphs to analyze dependencies arising from special positions is as follows. Suppose that G is a (b,b)-graph and has a subgraph H that is a (b,b)-block. Then $M(H(\mathbf{x}))$ is a submatrix of $M(G(\mathbf{x}))$; thus, a special position of $H(\mathbf{x})$ is also a special position of $G(\mathbf{x})$, no matter what edge labels we put on the edges of G that are not between two vertices of H. Similarly, there are special positions of $G(\mathbf{x})$ that remain so after changing the edge labels of H to anything at all. These come from special positions of $(G/H)(\mathbf{x})$. In general, special positions are characterized by which factors of the pure condition vanish, so they are associated with specific subgraphs.

Using this foundation, Algorithm 3 finds the factor graphs of a (k,k)-graph recursively.

Algorithm 3 The k-factor graphs algorithm.

Input: A (k,k)-graph $G=(V,E)$.
Output: The factor graphs of the irreducible factors of C_G.
Method:

1. Initialize $\mathcal{P}=\emptyset$ and $H=G$.
2. Play the $(k,k+1)$-pebble game on G to find a maximal $(k,k+1)$-sparse graph G'.
3. For every edge e that is rejected (there is at least one):
   (a) Use the $(k,k+1)$-pebble game to detect the fundamental $(k,k+1)$-circuit G_e of e in G'.
   (b) Set $\mathcal{P}=\mathcal{P}+G_e$; set $H=H/G_e$.
4. If H is not a single vertex:
   (a) Recursively use the ***k*-factor graphs algorithm** on H to obtain factor graphs $\mathcal{P}'$.
   (b) Set $\mathcal{P}=\mathcal{P}\cup\mathcal{P}'$.
5. Output $\mathcal{P}$.

5.2.2. Body-and-cad factors

The body-and-cad setting is more complicated than the body-and-bar setting of White and Whiteley (1987). An $[a,b]$-graph may have factors that correspond to $[a,b]$-blocks, factors that correspond to (a,a)- or (b,b)-blocks, and other factors that do not correspond to any induced block. This requires some additional technicalities, which we now introduce, before describing our algorithm for finding factor graphs in $[a,b]$-graphs.

Definition 5.2. Let G be an $[a,b]$-graph with pure condition $C_G = fg$. We say that the *edge support* of the factor f is the set E_f of edges e in G such that some variable of $\mathbf{x}(e)$ is in f.

We can show that the supports of distinct factors define edge-disjoint subgraphs of G, ultimately leading to a decomposition of G.

Theorem 5. *Let $C_G = fg$. Then the edge supports of f and g are disjoint and partition $E(G)$. Moreover, every monomial of a factor contains exactly one coordinate from each edge in its support.*

[12] The standard tie-down pins k coordinates of one body, but we can also choose "generic" tie-downs that pin k coordinates chosen from different bodies.
[13] Graph minors are obtained by deleting edges and vertices and contracting edges.

The proof is standard and presented in Appendix A for completeness.

Theorem 5 implies that the factor graphs mentioned above are a well-defined concept, which we repeat, since they are central in what follows.

Definition 5.3. A bi-colored graph H is a *factor graph* of G if H is a minor of G and $C_G = hf$, with the factor h supported on H. If h is an irreducible factor of C_G, then H is an *irreducible factor graph*.

If H only contains red edges, we abuse notation and interpret it as an (a,a)-graph labeled by the first a coordinates of the edge vectors. If H only contains black edges and is a (b,b)-graph whose vertices are contained in a red (a,a)-graph, then we interpret it as a (b,b)-graph labeled by the last b coordinates of the edge vectors. With this interpretation, we can make the following definition.

Definition 5.4. The factor graph H is *proper*, if $h = C_H$ (for some tie down of H). Otherwise, it is *improper*.

To see the difference between proper and improper, consider the $[1,1]$-graph graph in Fig. 10. It has three factor graphs: each diagonal of the outer square is a factor graph, and so is the outer square itself. The two diagonals are both proper; since a red edge is, trivially, a $(1,1)$-graph, their corresponding factors are pure conditions. On the other hand, the factor corresponding to the outer square is not a pure condition, which can been seen by noting that this factor graph is neither a $(1,1)$-graph nor a $[1,1]$-graph; its rigidity matrix is not square after a standard tie-down is applied. Thus, the outer square is an improper factor graph.

5.3. Detecting factor graphs

Algorithm 4 adapts the k-factor graphs algorithm to account for the different types of factor graphs that can appear in $[a,b]$-graphs. It relies on a subroutine, Algorithm 5, that detects the additional types of factors, both proper and improper.

Algorithm 4 The $[a,b]$-factor graphs algorithm.

Input: an $[a,b]$-graph $G = (V, E = R \sqcup B)$, with a set of red edges R and a set of black edges B.
Output: Proper (irreducible) and improper factor graphs of G that together provide a factorization of G.
Method:

1. Initialize $\mathcal{P} = \emptyset$; $\mathcal{I} = \emptyset$; $H = G$; set $k = a + b$.
2. Play the $(k, k+1)$-pebble game on G to find a maximal $(k, k+1)$-sparse graph G'.
3. For every edge e that is rejected (there is at least one):
   (a) Use the $(k, k+1)$-pebble game to detect the fundamental $(k, k+1)$-circuit G_e of e in G'.
   (b) Use the $[a,b]$-**red-factor graphs algorithm** on G_e to obtain sets of factor graphs $\mathcal{P}_e$ and $\mathcal{I}_e$.
   (c) Set $\mathcal{P} = \mathcal{P} \cup \mathcal{P}_e$ and $\mathcal{I} = \mathcal{I} \cup \mathcal{I}_e$; set $H = H/G_e$.
4. If H is not a single vertex:
   (a) Recursively use the $[a,b]$-**factor graphs algorithm** on H to obtain sets of factor graphs $\mathcal{P}'$ and $\mathcal{I}'$.
   (b) Set $\mathcal{P} = \mathcal{P} \cup \mathcal{P}'$ and $\mathcal{I} = \mathcal{I} \cup \mathcal{I}'$.
5. Output $\mathcal{P}$ and $\mathcal{I}$.

Algorithm 5 The $[a,b]$-red-factor graphs algorithm.

Input: An $[a,b]$-bi-colored graph $G = (V, E = R \sqcup B)$, with a set of red edges R and a set of black edges B, that contains no proper $(a+b, a+b)$-components.
Output: Proper (irreducible) and improper factor graphs of G that together provide a factorization of G.
Method:

1. Play the (a,a)-pebble game on (V, R) and detect red (a,a)-components.
2. If no (a,a)-components of (V, R) were found, output $\{G\}$.
3. Otherwise, let $H_1, \ldots, H_t$ be the red (a,a)-components.
   (a) For each graph $(V(H_i), E(V(H_i)) \cap B)$, play the (b,b)-pebble game to detect (b,b)-components.
   (b) If no (b,b)-components are found, use the a-**factor graphs algorithm** on each H_i to obtain a set of proper factor graphs $\mathcal{P}$ that are the a-factor graphs of the H_i. Then return $\mathcal{P}$ and $\mathcal{I} = \{(V, E \setminus \{E(V(H_i)) : i \in [t]\})\}$.
   (c) Otherwise, there is a (b,b)-component J in the vertex span of some component H_i. Set $H = H_i$.
      i. Use the a-**factor graphs algorithm** on H to obtain a set of factor graphs $\mathcal{P}_H$.
      ii. Use the b-**factor graphs algorithm** on J to obtain a set of factor graphs $\mathcal{P}_J$.
      iii. Obtain a pebble game configuration of G by fixing a vertex in $v \in V(J)$ and using the $[a,b]$-pebble game to collect k pebbles on v. Delete every edge of H with its tail in $V(J)$. Contract $V(J)$ into a single vertex to get a graph G'.
      iv. Use the $[a,b]$-**factor graphs algorithm** on G' to get a set of factor graphs $\mathcal{P}'$ and $\mathcal{I}'$.
      v. Find the factor graph F_H in $\mathcal{P}'$ that contains an edge of H.
      vi. Return $\mathcal{P}_H \cup \mathcal{P}_J \cup (\mathcal{P}' \setminus \{F_H\})$ and $\mathcal{I}'$.

We provide an overview of how the $[a,b]$-factor graphs algorithm performs on a $[1,2]$-graph with only proper factors in Fig. 9 and on a $[1,1]$-graph with one improper factor in Fig. 10. A more comprehensive trace of the algorithm is given in on a $[1,1]$-graph in Fig. 11.

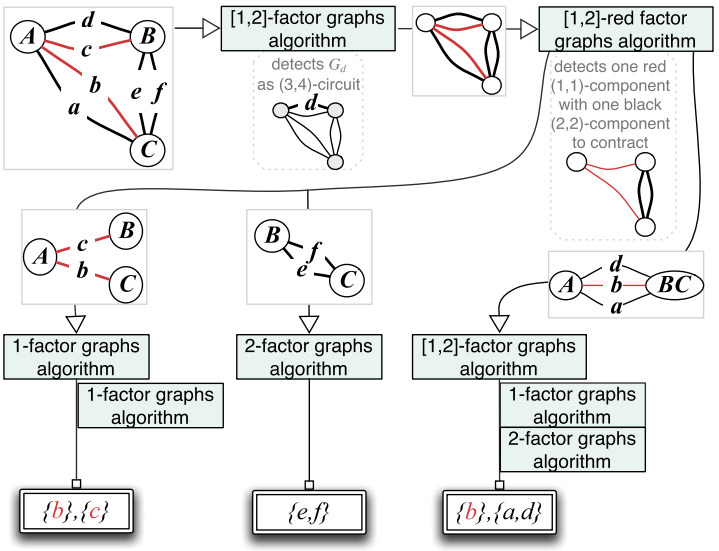


Fig. 9. Algorithm 4 finds the proper factor graphs of the $[1,2]$-graph underlying Fig. 1: $\{\{e,f\},\{a,d\},\{b\},\{c\}\}$; there are no improper factor graphs. The pure condition is $\pm[ef]_{(2,3)}[ad]_{(2,3)}[b]_{(1)}[c]_{(1)}$. A bracket subscripted by ordered tuple T denotes the determinant of the $|T|\times|T|$ matrix with coordinates specified by T.

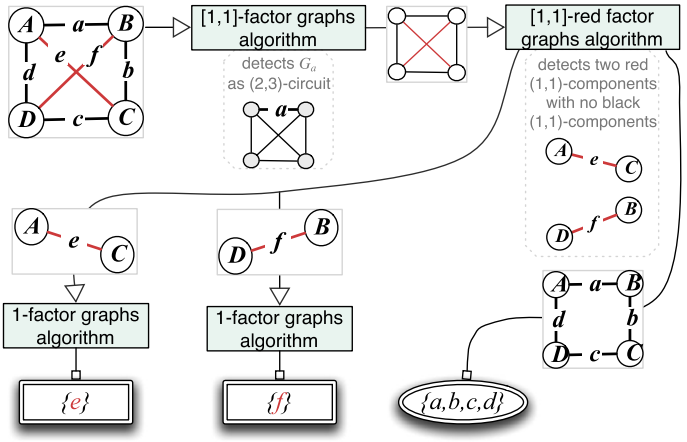


Fig. 10. Algorithm 4 on a $[1,1]$-graph finds proper factor graphs $\{\{e\},\{f\}\}$ and an improper factor graph $\{\{a,b,c,d\}\}$. The pure condition is $[e]_{(1)}[f]_{(1)}(a_2b_1c_1d_1-a_1b_2c_1d_1-a_1b_1c_2d_1+a_1b_1c_1d_2)=e_1f_1(a_2b_1c_1d_1-a_1b_2c_1d_1-a_1b_1c_2d_1+a_1b_1c_1d_2)$. A bracket subscripted by ordered tuple T denotes the determinant of the $|T|\times|T|$ matrix with coordinates specified by T.

5.3.1. Correctness

The proof of correctness relies on some structural results about the pure condition that we establish first. This result is an extension of the similar statement from White and Whiteley (1987) that underlies the ***k*-factor-graphs** algorithm discussed above to the setting of $[a,b]$-graphs, and it has a similar geometric meaning.

Theorem 6. *Let G have a proper subgraph H that is an $[a,b]$-block. Then H and G/H are both proper factor graphs and $C_G = C_H \cdot C_{G/H}$.*

For clarity of presentation, we leave the proof for Appendix A. Theorem 6 implies the correctness of Algorithm 3.

Claim 5.5. *Every graph returned by Algorithm 3 is an* irreducible *and* proper *factor graph.*

Proof. Properness comes from Theorem 6. Irreducibility of the pure condition of a (k,k)-graph is characterized by White and Whiteley (1987, p. 27): the pure condition of a graph is irreducible if and only if the graph contains no proper block. The graph contains no proper block if and only if, for every proper subgraph with n' vertices and m' edges, $m' < kn' - k$ (i.e.,

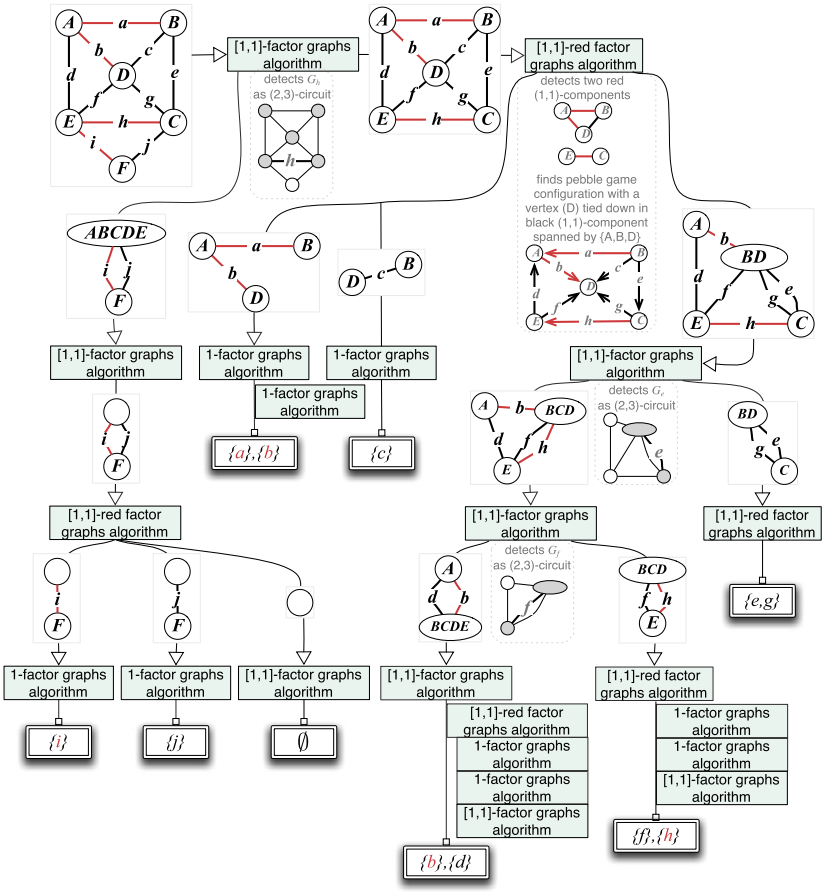


Fig. 11. Algorithm 4 on a $[1,1]$-graph detects proper factor graphs: $\{\{a\},\{b\},\{c\},\{i\},\{j\},\{f\},\{h\},\{d\}\{e,g\}\}$. There are no improper factor graphs. The pure condition is $\pm[a]_{(1)}[b]_{(1)}[c]_{(2)}[i]_{(1)}[j]_{(2)}[f]_{(2)}[h]_{1)}[d]_{(2)}[eg]_{(1,2)}$.

strict inequality holds on proper subgraphs). Since we are considering a (k,k)-graph with all proper subgraphs $(k,k+1)$ sparse, this holds precisely when the graph is a $(k,k+1)$-circuit. □

The next few theorems deal with the new cases arising in body-and-cad. Again, for clarity, we leave their proofs for Appendix A.

Theorem 7. *Let G be an $[a,b]$-graph with a subgraph H that contains only red edges and is an (a,a)-block. Then H is a proper factor graph of G.*

Figs. 9, 10 and 11 depict examples with red (a,a)-blocks appearing as proper factor graphs. The geometric interpretation of Theorem 7 is that as a sub-framework H is not rigid, but *angularly rigid*: only translational degrees of freedom are permitted between the vertices spanned by H. In light of this, we define the following.

Definition 5.6. Let G be $[a,b]$-sparse. If H is a subgraph that contains only red edges and is an (a,a)-block, we say that it is an *angular block*. If G is an $[a,b]$-graph, then an angular block is an *angular factor graph*.

A key difference between the situation in Theorem 6 and that of Theorem 7 is that we cannot simply contract H and obtain another proper factor graph. Indeed, as the example in Fig. 10 shows, $G \setminus H$ may be an *improper* factor graph.

The final type of proper factor graph arises as a subset of an angular block. Suppose that G has an angular block H, and that $J \subsetneq H$ is a (b,b)-block on t vertices (necessarily) containing only black edges. Angular rigidity of the bodies spanned by H implies that there are only $b(t-1)$ translational degrees of freedom remaining among them. Since J is a (b,b)-block containing only blind constraints, there are, in fact, no remaining internal degrees of freedom in J. We call J *contextually rigid*, since it is rigid in the context of G, but not as an induced subgraph. It is possible to continue finding factor graphs after discovering a contextually rigid block, but the construction is more delicate, in that it requires modifying H after contracting J.

Step 3 of Algorithm 5 handles detection of contextually rigid components. Theorem 8 ensures its correctness by making the discussion above precise We need an additional combinatorial definition for the statement.

Definition 5.7. Let G be an $[a, b]$-graph. An $[a, b]$-*tree decomposition*

$$\mathcal{T} = (A_1, \ldots, A_a, T_1, \ldots, T_b)$$

of G is a partition of the edges into k spanning trees such that all the red edges are in the sets A_i.

It follows from the definitions in Section 4.1 that there is an $[a, b]$-tree decomposition of a bi-colored graph if and only if it is an $[a, b]$-graph.

Theorem 8. *Let G be an $[a, b]$-graph with an all red (a, a)-block H as a subgraph and an all black (b, b)-block J with $V(J) \subset V(H)$. Tie down a vertex in J, fix an $[a, b]$-tree decomposition $\mathcal{T}$ of G, and orient all edges towards this root. Define G' by removing all of the edges of H whose tails are in J and then contracting J. Then,*

1. *H and J are proper factor graphs.*
2. *$C_G = C_H \cdot C_J \cdot f$, where f is supported on a subgraph F whose edges are the edges of G that are not in H or J.*
3. *G' is an $[a, b]$-graph with $C_{G'} = hf$, where h is supported on the red (a, a)-block $H' = H/J$.*

We now show correctness of Algorithm 5.

Claim 5.8. *Every graph returned by Algorithm 5 is a factor graph, and the proper factor graphs are irreducible.*

Proof. The calls to the a-factor graphs algorithm and b-factor graphs algorithm produce irreducible proper factor graphs by the correctness of Algorithm 3. In Step 3(c)iii, we obtain a pebble game configuration certifying (a, a)-sparsity of the aqua partition and (b, b)-sparsity of the tan partition. Via sparsity, we also know that there exists an $[a, b]$-tree decomposition consistent with this $[a, b]$-pebble game configuration if we direct all edges towards the chosen vertex v. Therefore, by Theorem 8 we know that G' is an $[a, b]$-graph.

Also by Theorem 8, $H' = H/J$ is a (red) (a, a)-block. Thus, after contracting and recursively calling Algorithm 4, there is exactly one factor graph H' containing an edge from H, so Step 3(c)v is well-defined. Finally, removing the factor graph $F_H = H'$ from the set of returned factor graphs completes the algorithm's correctness. □

This, along with Theorem 6, allows us to conclude correctness of our main Algorithm 4:

Claim 5.9. *Every graph returned by Algorithm 4 is a factor graph, and the proper factor graphs are irreducible.*

We do not know if the improper factor graphs are irreducible, since we do not have a nice representation for them.

Question 9. *Are all the factor graphs found by Algorithm 4 irreducible?*

6. Conclusions

The approach presented in this paper is part of a larger research path to provide computational tools that will give users information about dependencies present in CAD structures in terms of the original geometric constraint system. A prototype of Algorithm 1 has been implemented, with a long-term goal to see the pebble game and factor algorithms incorporated into commercial CAD software packages. By analyzing the pure condition, we can detect special positions of a generically minimally rigid body-and-cad structure. However, since C_G vanishes when $G(\mathbf{p})$ is *infinitesimally* flexible, special positions that we find may not be truly flexible. These positions may still be of interest to a CAD user, as an infinitesimally flexible framework carries an internal stress, indicative of structural weaknesses. Moreover, we may be able to combine conditions implying a special position to create degenerate embeddings with true motions. We conclude with a brief discussion of open questions that arise as we move toward further development of our approach.

Algorithm 4 returns factor graphs of the pure condition of an $[a, b]$-graph, but it remains open as to whether these factors are irreducible or not. When $b = 0$, the results of White and Whiteley (1987) show that irreducible factors of C_G correspond to circuits in G; this correspondence implies that Algorithm 3 produces irreducible factors for body-and-bar graphs. A better understanding of circuits would allow us to similarly conclude of the factors identified by Algorithm 4 are always irreducible.

We were able to carry out an analysis in the case study of Section 3 where the pure condition was just a product of brackets, and its vanishing was implied by either making two bars parallel or two lines parallel. More generally, a geometric interpretation of the vanishing of a more complicated non-monomial bracket polynomial may be possible via the process of

Cayley factorization, which takes as input a polynomial written in terms of brackets of vectors and outputs an expression in terms of meets and joins in the Grasmann–Cayley algebra of those points if such an expression exists. There is a Cayley factorization algorithm due to White (White, 1991; Sturmfels, 2008), and it would be interesting to see if it could be modified (and sped up) if the input bracket polynomial is known to be a pure condition.

Even when a Cayley factorization does exist, it may be nontrivial to extract geometric information about the original framework from it. One issue that adds complexity in general is that a single *cad* constraint may impose multiple linear constraints, so conditions may need to be expressed in terms of sets of vectors. Furthermore, in 3D, the vectors in the brackets do not live in a space dual to our realization space (as they do in 2D), complicating translation of the vanishing of the pure condition into the setting of our original constraints.

Finally, the results in this work rely on the combinatorial characterization of Lee-St.John and Sidman (2013), which apply to 3D body-and-cad structures *without point–point coincidence constraints*. While a combinatorial characterization that incorporates these constraints remains unknown, 3D body-and-cad frameworks with point–point coincidences share similar properties with presumed barriers to a combinatorial characterization of 3D bar-and-joint frameworks.

Figures and acknowledgements

Some figures created in SolidWorks 2010 and 2012. We thank Ruimin Cai for implementing some of the algorithms presented here and Josephine Yu for her work in implementing the Cayley factorization algorithm at the Macaulay2 workshop (Colorado College, Aug. 8–12, 2010; supported by NSF Grant No. 0964128 and NSA Grant No. H98230-10-1-0218). We are grateful for support from the AIM workshop on Configuration Spaces of Linkages and the BIRS workshop on Advances in Combinatorial and Geometry Rigidity Theory. This paper has benefited greatly from the thoughtful and constructive feedback from anonymous reviewers whom we also thank here.

Appendix A. Proofs for Section 5

A.1. Proof of Theorem 4

The proof follows from a combinatorial formula for the pure condition in terms of *tree decompositions* that we now derive. To make the connection between tree decompositions and C_G we define the *tree decomposition monomial* $\mathbf{x}^{\mathcal{T}}$ to be

$$\mathbf{x}^{\mathcal{T}} := \prod_{e\in A_i,\ i\in[a]} \mathbf{x}(e)_i \prod_{e\in T_j,\ j\in[b]} \mathbf{x}(e)_{a+j}$$

The tree monomials are precisely the monomials appearing in P_G.

Theorem 10. *If G is a tied down $[a,b]$-graph*

$$P_G = \sum_{\substack{\text{tree decomps.}\\ \mathcal{T}}} \pm\mathbf{x}^{\mathcal{T}} \qquad \text{in } \mathbb{R}[\mathbf{x}]. \tag{A.1}$$

The signs can be determined using the definition of the determinant of a matrix. The precise formula for the signs is technical and not needed in what follows, so we do not describe it here.

Proof. We sketch a proof that follows the proof of Theorem 2.18 in White and Whiteley (1987). First we reorder the columns of $M_T(G(\mathbf{x}))$ by the coordinates of $\mathbf{p}(e)$ so that we have the n first coordinates, followed by the n second coordinates, etc. In doing this we can see that if we ignore the last k rows corresponding to the tie down, each successive collection of n columns is just an incidence matrix for the directed multigraph G whose rows are the edges of G, and whose columns correspond to the vertices.

We expand the determinant of $M_T(G(\mathbf{x}))$ along these successive groups of n columns. To do this, in each set of n columns, we need to choose n rows. Since this set of n columns is an incidence matrix, if this subdeterminant is nonzero, then these rows must correspond to a spanning tree plus a row corresponding to the tie down. Moreover, this subdeterminant is actually a monomial as it is possible to expand it row by row, choosing the tie down as the first row, choosing an edge incident to the tied-down vertex as the next row, and choosing successive rows by taking an edge that was adjacent to an edge already chosen. Proceeding in this way, each row that we choose only contributes one nonzero entry.

The resulting $n \times n$ determinant is multiplied by k others, all chosen in the same way, to obtain a monomial. Since we cannot permit any row to appear twice in such a product, this product of k $n \times n$ determinants corresponds to a decomposition of G into k edge-disjoint spanning trees.

Moreover, we argue that such a product is nonzero if and only if the k trees form an $[a,b]$-tree decomposition. To see this, note that within the last b groups of n columns, the rows corresponding to red edges have only zeroes as entries. So, all of the red edges are in the a trees corresponding to the first a coordinates. □

Proof of Theorem 4. It follows from Theorem 10 that, since the tree decompositions are independent of the tie-down, P_G is determined up to sign. We can therefore define the *critical factor*[14] C_G to be P_G with respect to the tie-down that pins vertex 1, to establish a convention. □

A.2. Proof of Theorem 5

Proof. From Equation (A.1) we can see that every monomial of C_G contains exactly one coordinate from each edge. If the coordinates of $\mathbf{x}(e)$ were split between two distinct factors, then their product would have terms divisible by more than one coordinate of $\mathbf{x}(e)$, which would be a contradiction. □

A.3. Proof of Theorem 6

The proof requires the following standard lemma.

Lemma 6.1. *Let G be a (k,k)-graph, and let H be a proper block. Then G/H is also a (k,k)-graph.*

Proof. Fix a tree decomposition $T_1,\dots,T_k$ of G, and let n' be the number of vertices of H. Since H has $m' = kn' - k$ edges, all of which are covered by one of the trees, $|T_i \cap H| = n' - 1$ for every i. Thus, contracting H involves contracting a subtree of each of the T_i. Since contracting a connected subtree of any tree T produces a smaller tree T', contracting H produces a set of k trees that cover the edges of G/H. By the Tutte–Nash-Williams Theorem G/H is also a (k,k)-graph. □

Proof of Theorem 6. Since H is an $[a,b]$-graph, it has a pure condition C_H. Lemma 6.1 then implies that G/H is also an $[a,b]$-graph, so $C_{G/H}$ is defined as well. If both C_H and $C_{G/H}$ divide C_G, then $C_G = C_G \cdot C_{G/H}$, since the supports of H and G/H are disjoint and partition the edges of G.

To see that C_H and $C_{G/H}$ divide C_G, we will use Equation (A.1), and a small elaboration of the proof of Lemma 6.1. Pick tree decompositions $\mathcal{T}_H$ and $\mathcal{T}_{G/H}$ of H and G/H. Suppose that the union T of T_1^H and $T_1^{G/H}$ contains a cycle in G. Since T_1^H is a tree, this cycle is not contained entirely in H. This implies that $T/T_1^H = T_1^{G/H}$ is not acyclic, which is a contradiction.

Because $\mathcal{T}_H$ and $\mathcal{T}_{G/H}$ were arbitrary, a tree decompisiton of G is determined by picking one of H and G/H independently. The desired result then follows from Equation (A.1). □

A.4. Proof of Theorem 7

Proof. Because H is a completely red (a,a)-block, any tree decomposition of G induces a decomposition of H into a edge-disjoint spanning trees. The trees covering H can be chosen independently of the trees covering the remaining edges, so the theorem follows from Equation (A.1). □

A.5. Proof of Theorem 8

Proof. That H is a factor graph of G follows from Theorem 7. To see that J also is, observe that, since $V(J) \subset V(H)$, all the edges of J are covered by trees T_j (and not A_i). Thus, the decomposition of J into b trees is independent of the rest of any tree decomposition. Hence, C_J (the pure condition of a (b,b)-graph) divides C_G, using the same arguments as above.

To show that G' is an $[a,b]$-graph, we will show that $\mathcal{T}$ restricts to an $[a,b]$-tree decomposition of G'. In each tree there was a unique directed path from each vertex in G to the root in J. Since our construction only deletes edges directed out of J, there is still a unique directed path in each tree from each vertex remaining in G' to the root. If an undirected cycle were created in this process, it is easy to see that there would have also been an undirected cycle in the original tree. Therefore, the restriction of $\mathcal{T}$ to $\mathcal{T}'$ in G' is an $[a,b]$-tree decomposition.

Finally, we argue that $H' = H/J$ is a red (a,a)-block in G'. This will imply that H' is a factor graph of G', yielding the last assertion of the theorem. Above, we argued that the a red trees of $\mathcal{T}$ in H restrict to a red trees in $\mathcal{T}'$ in H'. By Nash-Williams–Tutte, H' is a (a,a)-block (of all red edges). By Theorem 7, H' is a proper factor graph of G' and $C_{G'} = hf$ with h supported on H'. □

References

Berardi, M., Heeringa, B., Malestein, J., Theran, L., 2011. Rigid components in fixed-lattice and cone frameworks. In: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG). http://arxiv.org/abs/1105.3234.

Brylawski, T., 1986. Constructions. In: Theory of Matroids. In: Encyclopedia Math. Appl., vol. 26. Cambridge Univ. Press, Cambridge, pp. 127–223.

[14] The name comes from the setting of general tie-downs, where ± 1 is replaced with the determinant of a $k \times k$ matrix.

Cheng, J., Sitharam, M., Streinu, I., 2009. Nucleation-free 3d rigidity. In: Proceedings of the 21st Canadian Conference of Computational Geometry, pp. 71–74. http://dblp.uni-trier.de/db/conf/cccg/cccg2009.html#ChengSS09.

Chou, S.-C., 1988. Mechanical Geometry Theorem Proving. Mathematics and Its Applications, vol. 41. D. Reidel Publishing Co., Dordrecht. With a foreword by Larry Wos.

Gao, X.-S., Lei, D., Liao, Q., Zhang, G.-F., 2005. Generalized Stewart–Gough platforms and their direct kinematics. IEEE Trans. Robot. 21 (2), 141. http://geometrie.uibk.ac.at/institutsangehoerige/husty/dld/Quebec.pdf.

Geiß, F., Schreyer, F.-O., 2009. A family of exceptional Stewart–Gough mechanisms of genus 7. In: Interactions of Classical and Numerical Algebraic Geometry. In: Contemp. Math., vol. 496. Amer. Math. Soc., Providence, RI, pp. 221–234.

Gortler, S.J., Healy, A.D., Thurston, D.P., 2010. Characterizing generic global rigidity. Am. J. Math. 132 (4), 897–939. http://dx.doi.org/10.1353/ajm.0.0132.

Haller, K., Lee-St. John, A., Sitharam, M., Streinu, I., White, N., 2012. Body-and-cad geometric constraint systems. Comput. Geom. 45 (8), 385–405.

Jackson, B., Owen, J.C., 2014. A characterisation of the generic rigidity of 2-dimensional point–line frameworks. Preprint, arXiv:1407.4675. http://arxiv.org/abs/1407.4675.

Knuth, D.E., 1973. Matroid partitioning. Tech. rep., Stanford, CA, USA.

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

Lee, A., Streinu, I., 2008. Pebble game algorithms and sparse graphs. Discrete Math. 308 (8), 1425–1437. http://dx.doi.org/10.1016/j.disc.2007.07.104.

Lee, A., Streinu, I., Theran, L., 2005. Finding and maintaining rigid components. In: Proceedings of the 17th Canadian Conference of Computational Geometry. Windsor, Ontario. http://cccg.cs.uwindsor.ca/papers/72.pdf.

Lee-St.John, A., Sidman, J., 2013. Combinatorics and the rigidity of cad systems. Comput. Aided Des. 45 (2), 473–482.

Michelucci, D., Foufou, S., 2007. Detecting all dependences in systems of geometric constraints using the witness method. In: Botana, F., Recio, T. (Eds.), Automated Deduction in Geometry, 6th International Workshop (ADG 2006). In: Lecture Notes in Computer Science, vol. 4869. Springer.

Nash-Williams, C.S.J.A., 1961. Edge-disjoint spanning trees of finite graphs. J. Lond. Math. Soc. 36, 445–450.

Schrijver, A., 2003. Combinatorial Optimization. Polyhedra and Efficiency, vol. B. Algorithms Comb., vol. 24. Springer-Verlag, Berlin. Matroids, trees, stable sets, Chapters 39–69.

Sitharam, M., Zhou, Y., 2004. A tractable, approximate characterization of combinatorial rigidity in 3d. In: 5th Automated Deduction in Geometry (ADG).

Streinu, I., Theran, L., 2009. Sparsity-certifying graph decompositions. Graphs Comb. 25, 219–238. http://dx.doi.org/10.1007/s00373-008-0834-4.

Sturmfels, B., 2008. Algorithms in Invariant Theory, 2nd edition. Texts and Monographs in Symbolic Computation. Springer, New York, Vienna.

Tay, T.-S., 1984. Rigidity of multigraphs. I. Linking rigid bodies in n-space. J. Comb. Theory, Ser. B 36 (1), 95–112. http://dx.doi.org/10.1016/0095-8956(84)90016-9.

Tutte, W.T., 1961. On the problem of decomposing a graph into n connected factors. J. Lond. Math. Soc. 36, 221–230.

White, N.L., 1991. Multilinear Cayley factorization. J. Symb. Comput. 11 (5–6), 421–438. http://dx.doi.org/10.1016/S0747-7171(08)80113-7. Invariant-theoretic algorithms in geometry (Minneapolis, MN, 1987).

White, N., Whiteley, W., 1987. The algebraic geometry of motions of bar-and-body frameworks. SIAM J. Algebraic Discrete Methods 8 (1), 1–32. http://dx.doi.org/10.1137/0608001.

Whiteley, W., 1988. The union of matroids and the rigidity of frameworks. SIAM J. Discrete Math. 1 (2), 237–255. http://dx.doi.org/10.1137/0401025.

Whiteley, W., 1996. Some matroids from discrete applied geometry. In: Bonin, J., Oxley, J.G., Servatius, B. (Eds.), Matroid Theory. In: Contemp. Math., vol. 197. American Mathematical Society, pp. 171–311.