Computational Geometry: Theory and Applications 118 (2024) 102064

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

journal homepage: www.elsevier.com/locate/comgeo

Enumerating combinatorial resultant trees

Goran Malić [1], Ileana Streinu *,[1]

Department of Computer Science, Smith College, Northampton, MA, USA

ARTICLE INFO

Article history:
Received 1 May 2023
Received in revised form 7 September 2023
Accepted 26 October 2023
Available online 31 October 2023

Keywords:
Rigidity matroid
Circuit polynomial
Combinatorial resultant
Inductive construction
Resultant tree

ABSTRACT

A 2D rigidity circuit is a minimal graph $G = (V, E)$ supporting a non-trivial stress in any generic placement of its vertices in the Euclidean plane. All 2D rigidity circuits can be constructed from K_4 graphs using *combinatorial resultant (CR)* operations. A *combinatorial resultant tree (CR-tree)* is a rooted binary tree capturing the structure of such a construction. The CR operation has a specific algebraic interpretation, where an essentially unique *circuit polynomial* is associated to each circuit graph. Performing Sylvester resultant operations on these polynomials is in one-to-one correspondence with CR operations on circuit graphs. This mixed combinatorial/algebraic approach led recently to an effective algorithm for computing circuit polynomials. Its complexity analysis remains an open problem, but it is known to be influenced by the depth and shape of CR-trees in ways that have only partially been investigated.

In this paper, we present an effective algorithm for enumerating all the CR-trees of a given circuit with n vertices. Our algorithm has been fully implemented in Mathematica and allows for computational experimentation with various optimality criteria in the resulting, potentially exponentially large collections of CR-trees.

1. Introduction

Combinatorial resultants. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with non-empty edge set intersection and $e \in E_1 \cap E_2$, their *combinatorial resultant* (shortly, *c-resultant*) with respect to edge e is defined as $\mathrm{CRes}(G_1, G_2, e) = (V_1 \cup V_2, (E_1 \cup E_2) - \{e\})$. In rigidity theory, this graph operation has a natural algebraic counterpart, the classical *Sylvester resultant*, and is used by [12,14] in an algorithm for calculating the unique *circuit polynomial* associated to a rigidity circuit[2] (shortly, a *circuit*).

C-resultants applied to 2D rigidity circuits have interesting combinatorial properties. Any circuit G on $n \geq 5$ vertices can always be obtained ("decomposed") [12,14] as the c-resultant of two smaller circuits: $G = \mathrm{CRes}(G_1, G_2, e)$, where G_1 and G_2 are circuits with $|V(G_1)|, |V(G_2)| < |V(G)| = n$. We can continue decomposing G_1 and G_2 as long as they have at least 5 vertices to obtain a *combinatorial resultant tree (CR-tree)* rooted at the circuit G: its internal nodes are marked with circuits of decreasing sizes along any path down from the root to each leaf. The leaves of a CR-tree are marked with isomorphic versions of K_4, the complete graph on 4 vertices. In general, a circuit may allow for more than one CR-tree decomposition: an example appears in Fig. 1.

* Corresponding author.
E-mail address: istreinu@smith.edu (I. Streinu).

[1] Funding: Both authors acknowledge funding from the NSF CCF:1703765 and CCF:2212309 grants to Ileana Streinu.

[2] Complete definitions will be given in Section 2.

https://doi.org/10.1016/j.comgeo.2023.102064

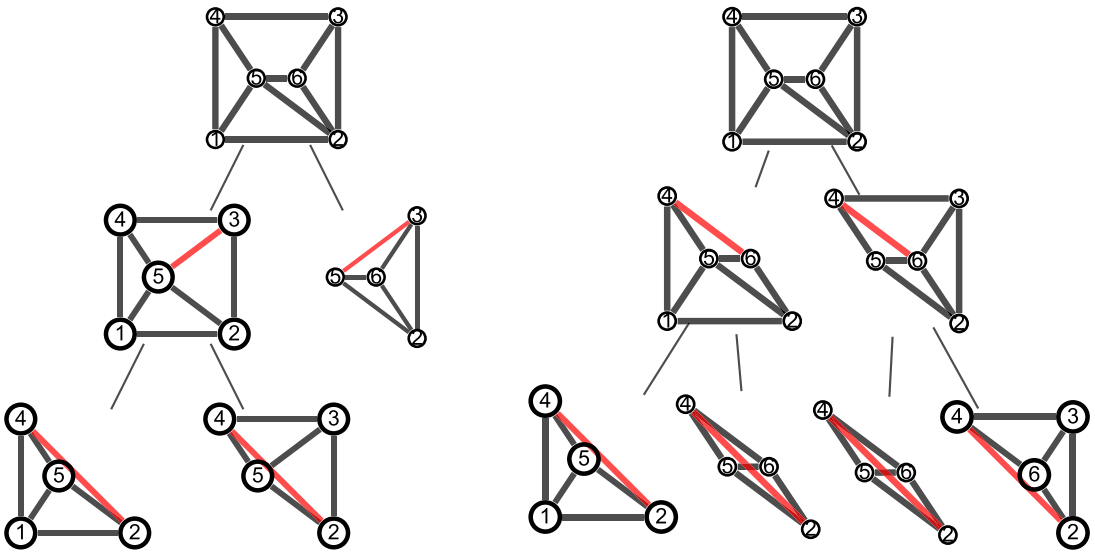

Fig. 1. Two combinatorial resultant trees for the Desargues-plus-one circuit.

This paper addresses the following:

Main Problem. Given a rigidity circuit, enumerate all of its possible combinatorial resultant trees, up to isomorphism.

Circuit polynomials. Let $\mathbb{F}$ be a field and $\mathbb{K} \supset \mathbb{F}$ a field extension with finite transcendence degree over $\mathbb{F}$. It is well-known that the collection of elements in a finite subset E of $\mathbb{K}$ that are algebraically independent over $\mathbb{F}$ satisfy matroid axioms [17,21]. Any matroid arising in such way is called *algebraic* with E as its ground set. If $\mathcal{M}$ is an algebraic matroid over $\mathbb{F}$ with ground set $E \subset \mathbb{K}$, then there exists a map $\rho : E \to \mathbb{K}$ such that for every $C \subseteq E$ we have $\text{rank}(C) = \text{trdeg}(\rho(C))$, where trdeg denotes the transcendence degree over $\mathbb{F}$. For $E = \{e_1, \ldots, e_n\}$ the map ρ induces a homomorphism $\mathbb{F}[x_1, \ldots, x_n] \to \mathbb{K}$. If we denote by $X_I = \{x_i \mid e_i \in I\}$ for any $I \subseteq E$, then the kernel $\ker \rho$ is a prime ideal which satisfies $\ker \rho \cap \mathbb{F}[X_I] = \{0\}$ if and only if $I \subseteq E$ is independent in $\mathcal{M}$. That is, the collection $\{X_I \subset X_E \mid \ker \rho \cap \mathbb{F}[X_I] = \{0\}\}$ defines a matroid isomorphic to $\mathcal{M}$. If C is a circuit in $\mathcal{M}$, then there exists a non-constant polynomial $p_C \in \mathbb{F}[X_C]$ irreducible over $\mathbb{F}$, unique up to multiplication with an element in $\mathbb{F}$, such that all elements of C simultaneously vanish on p_C [7]. The polynomial p_C is called the *circuit polynomial* for C.

To the best of our knowledge, the computation of circuit polynomials in arbitrary polynomial ideals was first studied in Zvi Rosen's thesis [18] and popularized in the American Mathematical Monthly paper [19], where two examples of circuit polynomials corresponding to the rigidity circuits given by the complete graph K_4 and the wheel W_4 of four vertices appeared for the first time. The circuit polynomial corresponding to K_4 is obtained "for free" by taking the determinant of the Cayley-Menger matrix on 4 vertices, whereas Gröbner basis methods were used to find the circuit polynomial corresponding to W_4. The out-of-the-box Gröbner basis methods fail for all other rigidity circuits, however in [12,14] we succeeded in computing the circuit polynomials for all rigidity circuits on 6 vertices, as well as some circuits on 7 and 8 vertices by making use of CR-trees.

Previous work: enumeration, matroids, rigidity. There is a rich literature on enumeration algorithms for combinatorial structures, with particular success in avoiding repetitions and backtracking when a matroidal or polytopal structure underlies the objects to be enumerated [1,2]. Interest from the engineering community led to methods for enumerating minimally rigid frameworks with particular properties (such as non-crossing ones) [3]. The objects enumerated by the algorithm of this paper are quite new and, although related both to rigidity and matroids, do not fall into the patterns where previous techniques could have been applied. Several algorithms for recognizing 2D minimally rigid graphs are available in the literature, but the 2D pebble game of [10], generalized by [11] for sparsity matroids, is the most relevant for the purpose of identifying rigidity circuits.

Combinatorial resultant trees were introduced in [12], where a CR-tree for a rigidity circuit G guides the computation of the circuit polynomial for G by eliminating variables one by one via the classical Sylvester resultant. Each elimination step corresponds to a non-leaf node in a CR-tree for G, hence the cost of computing the circuit polynomial for G varies depending on which CR-tree is used to guide the elimination process. For example, the Desargues-plus-one circuit, which appears as the root in Fig. 1, could be obtained as the resultant of the circuit polynomials of a single 4-wheel polynomial and a K_4 polynomial (left), or as the resultant of two isomorphic 4-wheels (right).

Table 1
Comparison of the cost of the elimination steps for the two CR-tree's from Fig. 1. The entries ij in the first columns indicate the edges e_{ij} and indeterminates x_{ij} that are eliminated in $\mathrm{CRes}(G_1, G_2, e_{ij})$ and $\mathrm{Res}(p_1, p_2, x_{ij})$.

	Elimination guided by the left CR-tree in Fig. 1
Elim. 24	Syl Det Dimension: 4×4 Syl Det Result: hom. poly. with hom. deg. 8
Elim. 35	Syl Det Dimension: 6×6 Syl Det Result: hom. poly. with hom. deg. 20

	Elimination guided by the right CR-tree in Fig. 1
Elim. 24	Syl Det Dimension: 4×4 Syl Det Result: hom. poly. with hom. deg. 8
Elim. 24	Syl Det Dimension: 4×4 Syl Det Result: hom. poly. with hom. deg. 8
Elim. 46	Syl Det Dimension: 8×8 Syl Det Result: hom. poly. with hom. deg. 48

Motivation: open problems underlying the computation of circuit polynomials. The CR-tree on the left of Fig. 1 has two non-leaf nodes, whereas the CR-tree on the right has three: if the elimination process is guided by the tree on the left, then we have to compute one fewer Sylvester resultant than if we were guided by the CR-tree on the right. With the CR-tree on the right we also have to do computations with determinants of larger dimension and work with larger polynomials (in the sense of homogeneous degree and number of monomial terms) than necessary. Table 1 provides a comparison of the effort (in terms of elimination steps) between the two CR-tree trees in Fig. 1.

The complexity analysis of the mixed combinatorial/algebraic algorithm of [12] is left as an explicit *open problem*. As can be inferred from the previous example, its time and space complexity is influenced by the number of nodes, and the depth and the shape of the guiding CR-trees in ways that have not been investigated until now. In [12], we explicitly calculated the Desargues-plus-one circuit polynomial, accomplished with Mathematica in just 15 seconds compared to several days of calculations required by a classical GroebnerBasis method, by being guided by the CR-tree on the left. In this particular case, a mere visual inspection helped decide which of the two CR-trees would give the best performance. But in general the number of CR-trees grows fast with the number n of vertices and depends on the circuit graph, not just on n.

Implementation. The enumeration algorithm presented in this paper has been fully implemented as a Mathematica prototype and can generate efficiently large collections of CR-trees for rigidity circuits with up to 13 vertices. The circuits with $n = 13$ vertices took approx. 6 hrs and produced 6×10^{33} trees. We were able to push the implementation further for the class of wheel graphs of 15 or less vertices. This provides experimental data for addressing some of the conjectures and open questions posed in [12]. By selectively searching through this data one would hope to find the right CR-trees amenable to shorter algebraic calculations to be contributed to their database of circuit polynomials [13]. Ultimately, the goal is to shed light on how the computational complexity of the algebraic part of our algorithm is influenced by the choice of a CR-tree from the very large collection of possibilities. In this paper we make the first step in this direction by demonstrating how to efficiently enumerate and keep track of all the possibilities, without repetitions. We accomplish this goal by using *truncated* CR-trees, briefly introduced now.

Truncated CR-trees. Rather than producing complete CR-trees, we use instead a space-saving structure called a *truncated CR-tree* for G. This is essentially a CR-tree where one or more of its subtrees are truncated if, during the execution of the algorithm, it is found that their roots are isomorphic to some previously encountered circuits for which subtrees have already been generated (Fig. 2). We keep track of all the isomorphisms and therefore can recover, on demand, an entire CR-tree from a truncated one.

Connectivity. There is a substantial difference in the complexity of decomposing a circuit that is 3-connected vs. one that is not. In the 3-connected case our decomposition algorithm from [12] finds all decompositions in polynomial time, but cannot be applied to circuits that are not 3-connected. In that case an exponential time brute-force search among the subgraphs of the circuit has to be performed, and it is an open problem to find a better algorithm. This is hardly ideal, so we apply the following heuristic: circuits that are not 3-connected are decomposed only as *2-splits* (cf. Section 2), which is accomplished in linear time. To not detract from the main goal of this paper, the justification for this heuristic we give in the preprint [15].

Generalization. Our algorithm is not confined to rigidity theory, but generalizes to matroids in which an operation similar to the c-resultant can be defined. More precisely, given a circuit C in the matroid on ground set E, we require the existence of two distinct circuits C_1 and C_2 and $e \in C_1 \cap C_2$ such that $C_3 = (C_1 \cup C_2) - e$.

Overview of the paper. All relevant definitions from 2D rigidity theory are presented in Section 2. In Section 3, we give the definitions of combinatorial resultants and related concepts. In Section 4, we recall and analyze our CR-decomposition algorithm from [12,14]. In Section 5, we discuss enumeration of all truncated CR-decompositions, followed by Section 6, where we describe the data structures that we maintain in the enumerative algorithm. In Section 7, we describe the algorithms for enumerating circuits and their decompositions that appear in a CR-tree. In Section 8, we describe a recursive algorithm for enumerating truncated CR-trees, followed by an algorithm for reconstructing CR-trees from truncated CR-trees in Section 9.

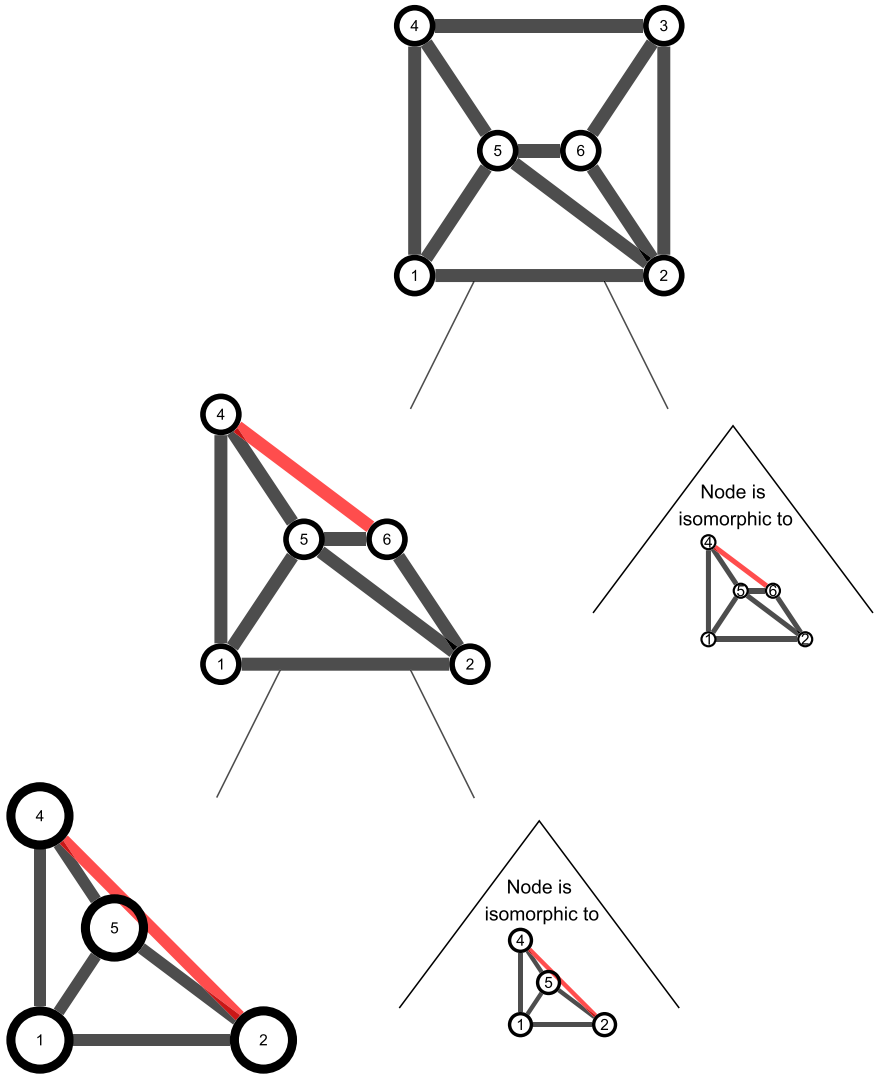

Fig. 2. A truncated CR-tree for the Desargues-plus-one circuit. Compare with the CR-tree on the right of Fig. 1.

In Section 10, we present the results of our implementation of the enumeration algorithm. We conclude with Section 11 where we outline future directions of research.

2. Preliminaries: 2D rigidity circuits

We start by reviewing, in an informal way, those concepts and results from combinatorial rigidity theory in 2D that are relevant for our paper.

Notation. Unless explicitly stated otherwise, a graph G has its vertex set $V(G)$ labeled with a subset of $[n] := \{1, \ldots, n\}$. Its support is its set of edges $E(G)$. The *vertex span* $V(E)$ of edges E is the set of all edge-endpoint vertices. A subgraph G is *spanning* if its edge set $E(G)$ spans $[n]$. The *neighbors* $N(v)$ of v are the vertices adjacent to v in G. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are connected graphs, we denote by $V_\cup = V_1 \cup V_2$ and $E_\cup = E_1 \cup E_2$.

Rigid graphs. A graph $G = (V, E)$ is *rigid* if, up to rigid motions, it has only a finite number of possible placements compatible with any collection of generic lengths associated to its edges, and *flexible* otherwise. A *minimally rigid* or *Laman* graph is a rigid graph which becomes flexible when any of its edges is removed. Laman graphs are characterized by a relationship between their number of vertices and edges: (a) has a total of $2n - 3$ edges and (b) no subset of $n' \leq n = |V|$ vertices spans more than $2n' - 3$ edges. A *Laman-plus-one graph* is obtained by adding one edge to a Laman graph: the graph has a total of $2n - 2$ edges and thus it violates the $(2, 3)$-sparsity condition on V and possibly on some other proper subsets of V.

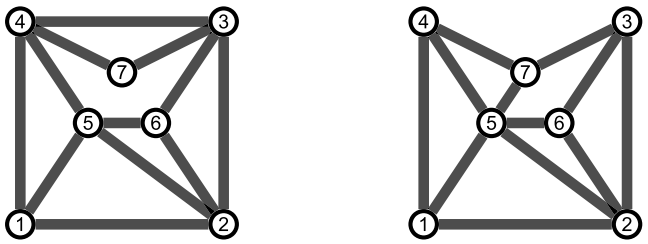

Fig. 3. Left: a Laman-plus-one graph on 7 vertices whose unique circuit is the Desargues-plus-one graph from Fig. 1. Right: a circuit on 7 vertices.

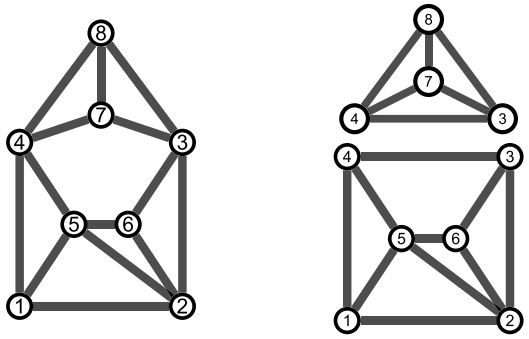

Fig. 4. Splitting a 2-connected circuit (left) to get two 3-connected circuits (right).

Rigidity Matroids. A matroid on a *ground set* E is a pair $(E, \mathcal{I})$ where $\mathcal{I}$ is a collection of subsets of E called *independent sets*, which satisfy certain axioms found in [17] which we skip (they are not relevant for our presentation). A maximal independent set is called a *base* and a set which is not independent is said to be *dependent*. A *minimal dependent set* is called a *circuit*. Relevant for our purposes are the following general aspects: (a) (hereditary property) a subset of an independent set is also independent; (b) all bases have the same cardinality, called the *rank* of the matroid. The *rigidity matroid* is defined on a ground set given by all the edges $E_n := \{ij : 1 \leq i < j \leq n\}$ of the complete graph K_n. Its bases are Laman graphs on n vertices. A *Laman-plus-one* graph is a dependent graph obtained by adding an edge to a Laman graph, thus it has exactly $2n - 2$ edges. A *(rigidity) circuit* is a dependent graph with minimum edge support: removing any of its edges leads to a Laman graph on its spanned vertex set. A circuit whose vertex set is $[n]$ is said to be a *spanning rigidity circuit* (Fig. 3). In particular, a circuit G in the rigidity matroid is a subgraph of K_n with $k \leq n$ vertices and (a) exactly $2k - 2$ edges and such that (b) any subgraph of G with $k' < k$ vertices has at most $2k' - 3$ edges. A spanning rigidity circuit $G = (V, E)$ is a special case of a Laman-plus-one graph. Furthermore, a Laman-plus-one graph contains a unique circuit, which in general is not spanning.

Connectivity. A circuit is always a 2-connected graph, but it may not be 3-connected. If G_1 and G_2 are two graphs with exactly one common edge uv, we define their 2-*sum* as the graph $G = (V_\cup, E_\cup \setminus \{uv\})$. It is well known that the 2-sum of two circuits is a circuit [4]. If G_1 and G_2 are 3-connected, then their 2-sum is a 2-connected graph that decomposes into two 3-connected components G_1 and G_2. The inverse operation of decomposing G into G_1 and G_2 along the non-edge uv is called a 2-*split*. It is well-known that the 2-split of a circuit gives a pair of circuits [4], see Fig. 4. A classical theorem due to Tutte [20] implies that any circuit which is not 3-connected can be decomposed in a unique way into smaller 3-connected circuits via 2-splits.

3. Combinatorial resultants

This section reviews basic definitions on combinatorial resultant trees from [14] and clarifies the isomorphisms taken into account by our enumeration algorithm.

Notation. We adopt the following notation. Recall that for connected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ we denote by $V_\cup = V_1 \cup V_2$ and $E_\cup = E_1 \cup E_2$. If $e \in E_1 \cap E_2$ exists, we denote by $\mathrm{CRes}(G_1, G_2, e)$ the graph $(V_\cup, E_\cup - e)$, which is, by definition, the *combinatorial resultant* (c-resultant) of G_1 and G_2 on edge e.

C-resultants of circuits. Given two circuits $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with non-empty edge set intersection and $e \in E_1 \cap E_2$, their c-resultant with respect to edge e is not always a circuit. When the common part $(V_\cap, E_\cap)$ is a Laman graph on its vertex set $V_\cap$, then the c-resultant is necessarily a Laman-plus-one graph but will not always a circuit. We will work exclusively with the case when the c-resultant of two circuits is a circuit.

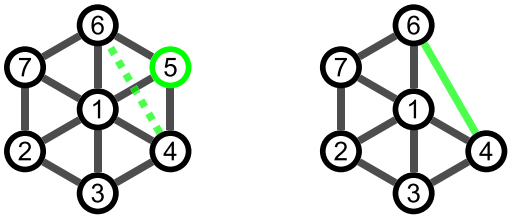

Fig. 5. Left: vertex 5 and the non-edge e_{46} in the wheel W_6 are an admissible pair. Right: the graph $W_6 - 5 + e_{46}$ is a rigidity circuit.

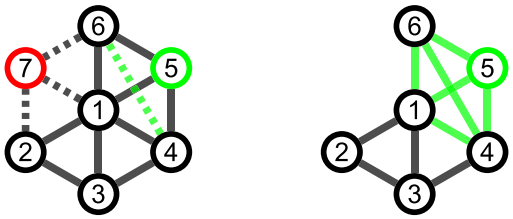

Fig. 6. Left: by removing vertex 7 a Laman-plus-one graph is obtained. Right: the graph $W_5 - 7 + e_{46}$ contains a unique circuit, a K_4 on the vertices 1, 4, 5, 6 (in green). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

CR decompositions. If a circuit G is obtained as a c-resultant $G = \mathrm{CRes}(G_1, G_2, e)$ of two other circuits G_1 and G_2, then we say that (G_1, G_2, e) is a *combinatorial resultant decomposition* (CR-decomposition) of G. In the case when G is not 3-connected and $e = uv$ a non-edge whose endpoints form a separating pair uv of vertices, we can do a 2-split on uv. Indeed, this is a special case of a CR-decomposition. When G is 3-connected, it was shown in [12] that it has a c-resultant decomposition into two circuits G_1 and G_2 (which may not be 3-connected) that both have fewer vertices than G. By further decomposing G_1 and G_2 we obtain a tree whose nodes are circuits and whose leaves are isomorphic to K_4.

C-resultant trees. For the remainder of the paper we assume that G is a rigidity circuit with $n \geq 5$ vertices.

Definition 1. Let G be a rigidity circuit with $n \geq 5$ vertices. A *combinatorial resultant tree (CR-tree)* for G is a rooted binary tree with G as its root, such that:

(i) each node is labeled with a rigidity circuit;
(ii) the children L and R of a node G' satisfy $G' = \mathrm{CRes}(L, R, e)$ for some $e \in L \cap R$ and both have fewer vertices than G';
(iii) leaves are labeled with graphs isomorphic to K_4.

We remark that in [14], this is referred to as a *combinatorial circuit resultant tree*. A rigidity circuit G can have more than one CR-decomposition, and therefore more than one CR-tree. In [12] an algorithm for constructing CR-decompositions of 3-connected circuits is given; before recalling it, we introduce the following notation.

Admissible pairs. Let v be a vertex of a rigidity circuit G and $N(v)$ the set of its neighbors in G. If G is not the smallest circuit K_4, then any vertex of degree 3 has a non-edge $e = \{u, w\} \subset N(v)$ among its neighbors. We denote by $G - v + e$ the graph obtained by deleting v from G and adding e to its edges. If $G - v + e$ is a rigidity circuit we say that (v, e) is an *admissible vertex-non-edge pair* (or just admissible pair for short). An example is shown in Fig. 5.

Lemma 1 *([4]). Let G be a 3-connected rigidity circuit and e, e' non-edges of G that are not necessarily distinct. Then there exist at least two admissible pairs (v, e) and (w, e') such that v and w are non-adjacent.*

If (v, e) is an admissible pair in a circuit with more than 5 vertices, then by removing a vertex v of degree 3 we obtain a Laman graph with one fewer vertex. By adding to it the new edge e we obtain a Laman-plus-one graph, and this contains a unique circuit, see Fig. 6. This operation is the essential ingredient in computing a CR-decomposition of G. In the next section we review the algorithm for computing the unique circuit induced by an admissible pair.

Decomposition algorithm for 3-connected circuits [12].
The correctness of Algorithm 1 follows from Lemma 1 (lines 1-3) and sparsity considerations (line 4). Note that L is 3-connected [4] but R does not have to be. If G has n vertices then L has $n - 1$ vertices but R could have fewer than $n - 1$ vertices.

The details and complexity analysis of this algorithm, which have only been briefly sketched in [14], appear in Section 4.

Admissible triple. Motivated by the above algorithm we define an *admissible vertex-non-edge triple* (or just admissible triple for short) (v, w, e) where (v, e) is an admissible pair in G and w a vertex of G of degree 3 not adjacent to v.

Algorithm 1: CR-DECOMPOSITION(G).

Input: a 3-connected rigidity circuit G with at least 5 vertices
Output: rigidity circuits L and R and a non-edge e of G such that $G = \text{CRes}(L, R, e)$

```
1 for vertex v of degree 3 do
2     if there is admissible pair (v, e) and a non-adjacent degree 3 vertex w then
3         L = G − v + e;
4         R = unique circuit in G − w + e containing e;
5         return (L, R, e)
```

4. Analysis of the CR-decomposition algorithm

In this section we analyze the CR-decomposition Algorithm 1 that decomposes a 3-connected rigidity circuit G into rigidity circuits L and R such that $G = \text{CRes}(L, R, e)$ with $|V(L)|, |V(R)| < |V(G)|$. We split the algorithm in several subproblems that can be treated independently. In Section 4.1, for a minimally rigid graph G and a non-edge e of G we give a polynomial-time algorithm FIND-UNIQUE(G, e) which returns the unique circuit in $G + e$ containing e. The FIND-UNIQUE algorithm is based on the Pebble Game algorithm of [11]. In Section 4.2, for a 3-connected circuit G we give a polynomial-time algorithm FIND-ADMISSIBLE(G) that outputs all the admissible pairs (v, e) and all admissible triples (v, w, e) for G. In Section 4.3 we describe the algorithm CRD(G, v, w, e) that for a 3-connected circuit G and an admissible triple (v, w, e), outputs a single CR-decomposition $G = \text{CRes}(L, R, e)$, where $L = G - v + e$ and R is the unique circuit in $G - w + e$ containing e. In Section 4.4 we briefly discuss the case when G is a 2-connected rigidity circuit.

4.1. Finding the unique circuit in a Laman-plus-one graph $G + e$

In this section we describe the polynomial-time algorithm FIND-UNIQUE-CIRCUIT(G, e) that finds the unique rigidity circuit C containing e in a Laman-plus-one graph $G + e$. It uses the Decision Pebble Game of [11] to test the independence of subsets of edges of $G + e$.

Algorithm 2: FIND-UNIQUE-CIRCUIT(G, e).

Input: a Laman graph G and a non-edge e of G
Helper function: `PEBBLE-GAME`(H) returns True if the graph H is independent, and False otherwise
Output: Unique circuit in $G + e$ containing e

```
1 D = G + e;
2 forall e' ∈ G do
3     if PEBBLE-GAME(D − e') is False then
4         D ← D − e'
5 return D
```

This algorithm is based on the following properties: (a) $G + e$ is a dependent graph; (b) being a Laman-plus-one graph, $G + e$ contains a unique circuit; (c) the unique circuit contains the edge e. We construct the circuit by eliminating from the initial dependent graph $D = G + e$ (line 2) those edges $e' \in G$ which are guaranteed not to be in the circuit; namely, when $D - e'$ is still dependent (line 3).

Proof of correctness of Algorithm 2. Let C be the unique circuit in $G + e$ that contains e. If $D - e'$ is dependent for some $e' \in E(G)$ then $D - e'$ contains the circuit C and we can remove e' from D. On the other hand, if for some $e' \in E(G)$ we have an independent $D - e'$, then necessarily $e' \in C$ as otherwise C would be independent. Therefore D has to remain unchanged.

Deleting or retaining edges of D in this fashion ensures that once all the edges of G have been considered, what remains of D is the unique circuit in $G + e$ containing e.

Complexity of Algorithm 2. The Decision Pebble Game runs in $O(n^2)$ time, and it is invoked for each of the $2n - 2$ edges in $G + e$, resulting in a complexity of $O(n^3)$.

4.2. Finding admissible vertex-non-edge pairs and triples in a 3-connected circuit G

In this section we will describe an algorithm FIND-ADMISSIBLE(G) that finds all admissible vertex-non-edge pairs (v, e) in a 3-connected rigidity circuit G in polynomial time. The sparsity condition for G implies that G has at least four vertices of degree 3, and by Lemma 1 there is at least one admissible vertex-non-edge pair (v, e).

To find the admissible vertex-non-edge pairs in a 3-connected rigidity circuit G we iterate over all vertices v of degree 3 and all non-edges e with endpoints in $N(v)$. The vertex-non-edge pair (v, e) is admissible if and only if the output of FIND-UNIQUE-CIRCUIT($G - v + e$,e) is $G - v + e$. In the worst case, when all but two vertices of G are of degree 3 we iterate over all $3(n - 2)$ possibilities. Once an admissible pair (v, e) has been identified, we can immediately determine

Algorithm 3: FIND-ADMISSIBLE(G).

Input: a 3-connected rigidity circuit G
Output: Lists of all admissible pairs (v, e) and admissible triples (v, w, e) of G

```
admissiblePairs = Empty;
admissibleTriples = Empty;
forall vertices v of degree 3 in G do
    forall non-edges e of G with endpoints in N(v) do
        if FIND-UNIQUE-CIRCUIT(G − v + e) == G − v + e then
            append (v, e) to admissiblePairs;
            for every w ∉ N(v) of degree 3 append (v, w, e) to admissibleTriples
return {admissiblePairs,admissibleTriples}
```

all admissible triples (v, w, e) by expanding (v, e) with every degree 3 vertex w not incident to v. The complexity for FIND-ADMISSIBLE(G) is $O(n^4)$.

4.3. A single CR-decomposition for a 3-connected circuit

Algorithm 4 described below computes a single CR-decomposition for a 3-connected rigidity circuit G, assuming that an admissible triple (v, w, e) in G is given. This algorithm runs in $O(n^3)$ time due to FIND-UNIQUE-CIRCUIT.

Algorithm 4: CRD(G, v, w, e).

Input: 3-connected rigidity circuit G, one admissible triple (v, w, e)
Output: A pair of rigidity circuits (L, R) such that $G = \text{CRes}(L, R, e)$

```
L = G − v + e ;
R = FIND-UNIQUE-CIRCUIT(G − w + e, e);
return (L,R)
```

This concludes the detailed analysis of Algorithm 1. It remains to check what can be done for circuits that are not 3-connected.

4.4. The CR-decomposition of a 2-connected rigidity circuit

In the rest of this section we discuss the case when G is a 2-connected but not 3-connected rigidity circuit. In this case, it is not always possible to find an admissible pair in G. For example, the "*double banana*" shown in the top row of Fig. 7 has no admissible pairs and therefore Algorithms 3 and 4 can not be applied.

However, it is well known that any 2-split of G results in two rigidity circuits [4, Lemmas 4.1, 4.2] and by a theorem of Tutte [20] G can be decomposed into unique 3-connected components via 2-splits. A 2-split is a CR-decomposition (G_1, G_2, e) where e is a non-edge between a separating pair of G, hence a CR-tree for G can be obtained by first splitting G into the 3-connected components given by Tutte's theorem, and then by decomposing each 3-connected component with Algorithm 4 until we reach K_4 graphs.

Moreover, G can have CR-decompositions into 3-connected circuits that are not 2-splits. For example, the double banana has a CR-decomposition into two wheels on 4 vertices, as shown in Fig. 7. The graphs (G_1, G_2, e) in a CR-decomposition of G necessarily have to intersect on a Laman graph by following a sparsity count, however we are not aware of any sufficient criteria.

Open Problem 1. *Find necessary and sufficient conditions for a 2-connected rigidity circuit on $n \geq 6$ vertices to have a CR-decomposition other than a 2-split.*

Experimental observations regarding 2-connected circuits. We conjecture that a decomposition of G by 2-splits is the *best possible* in the sense that the corresponding circuit polynomials are the least complicated with respect to the number of indeterminates, number of monomial terms, degrees in individual indeterminates, or the homogeneous degree. Our conjecture is based on experimental observation: Table 2 compares the cost of computing the circuit polynomial for the double banana, guided by the decomposition trees shown in Fig. 7. The computation guided by the 2-split was completed within 0.02 seconds,[3] producing an irreducible polynomial of homogeneous degree 8 with 1752 monomial terms.

[3] Computational time refers to wall-clock time measured by the RepeatedTiming function in Mathematica V13.

Table 2
Comparison of the cost of performing a resultant computation guided by the decompositions shown in Fig. 7.

Elimination guided by the 2-split in Fig. 7 Left	
Syl Det Dimension	4×4
Syl Det Result	Hom. poly. of hom. deg. 8
# of mon. terms	1752
Computational time	0.017 seconds

Elimination guided by the CR-decomposition in Fig. 7 Right	
Syl Det Dimension	8×8
Syl Det Result	Hom. poly. of hom. deg. 48
# of mon. terms	Computation exceeded memory capacity
Computational time	Computation exceeded memory capacity

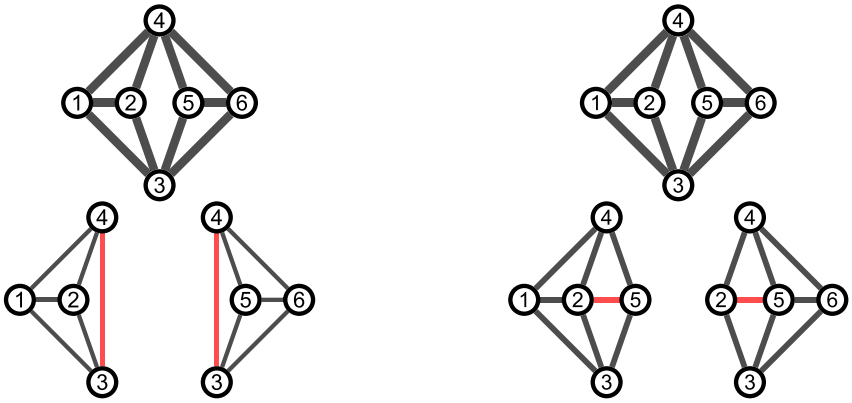

Fig. 7. Left: a CR-decomposition of the double banana into two K_4 graphs which is a 2-split. Right: a CR-decomposition of the double banana into two wheels on 4 vertices with labels 13542 and 23645. The elimination edge for both cases is shown in red.

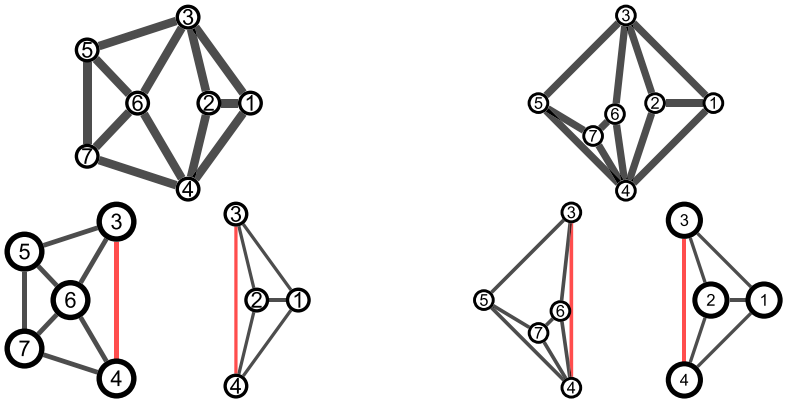

Fig. 8. Two 2-connected rigidity circuits on 7 vertices with CR-decompositions given by Tutte's theorem.

However, the computation guided by the combinatorial resultant of the two wheels of 4 vertices crashed after memory capacity[4] was exhausted. The homogeneous degree could be computed because we can apply the formula for the homogeneous degree of the Sylvester resultant determinant $\mathrm{Syl}(f, g, x)$ given by $h_f d_g + h_g d_f - d_f d_g$ where h_f (resp. h_g) is the homogeneous degree of f (resp. g) and d_f (resp. d_g) is the degree in x of f (resp. g) [14].

There are entire families of 2-connected circuits with non-2-split CR-decompositions. Two examples on 7 vertices are shown in Fig. 8 (decomposition into 3-connected components given by 2-splits, i.e. via Tutte's theorem) and Fig. 9 (CR-decomposition via a non-2-split, where one of the circuits is 2-connected and the other is 3-connected). These examples can be easily generalized to larger families by, for example, applying Henneberg-2 or 2-sum operations to the circuits in

[4] The computation was performed on a machine with 64GB of RAM.

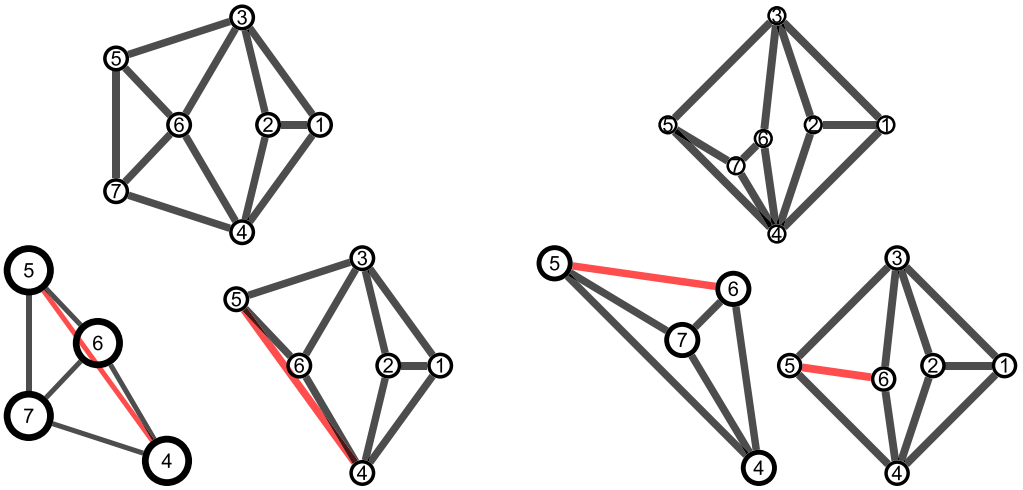

Fig. 9. The same two 2-connected circuits from Fig. 8, this time decomposed into a double banana and a K_4.

Table 3
Comparison of the cost of performing a resultant computation guided by the decompositions given by Tutte's theorem and the decompositions given in Fig. 9.

Elimination guided by Fig. 8 (Left) and Fig. 9 (Left)	
Syl Det Dimension	6×6
Syl Det Result	Hom. poly. of hom. deg. 20
# of mon. terms	1 053 933
Computational time via Tutte's decomposition	16.94 seconds
Computational time via CR-decomposition in Fig. 9 Left	23.50 seconds

Elimination guided by Fig. 8 (Right) and Fig. 9 (Right)	
Syl Det Dimension	6×6
Syl Det Result	Hom. poly. of hom. deg. 20
# of mon. terms	2 579 050
Computational time via Tutte's decomposition	42.19 seconds
Computational time via CR-decomposition in Fig. 9 Right	59.88 seconds

the decomposition [15]. They are relevant to our discussion because we could effectively perform the algebraic calculations (Sylvester resultants) guided by these different decompositions and compare the algebraic complexity of the polynomials implicated in the process. Specifically, in support of our conjecture, Table 3 shows that the 2-split (Tutte) decompositions lead to a faster computation time of the desired circuit polynomial.

Decomposing 2-connected rigidity circuits. In spite of lacking a definitive answer to Open Problem 1, the experimental observations from the previous section suggest that the enumeration of *all* possible CR-decompositions of 2-connected circuits may not be necessary. Instead, we focus on enumerating only those given by Tutte's theorem. The 3-connected components can be found in linear time using the algorithm of Hopcroft and Tarjan [9]. The process of assembling G from the 3-connected components via 2-sums can also be given by the *SPQR-tree* [5,6] which can be constructed in linear time [8]. For the detailed analysis of the 2-connected case we refer the reader to our preprint [15].

5. Enumerating all CR-decompositions for a 3-connected circuit

In this section we discuss the enumeration of all possible CR-decompositions of a 3-connected rigidity circuit G into 3-connected components. We achieve this by computing CRD(G, v, w, e) via Algorithm 4, for every triple (v, w, e) in the output of FIND-ADMISSIBLE(G) via Algorithm 3. If one of the children returned by CRD(G, v, w, e) is 2-connected, we truncate the tree and leave it as partially constructed. For the rest of the paper, unless explicitly noted, we assume that all circuits are 3-connected.

Isomorphic CR-decompositions. It is possible to have two distinct admissible triples (v_1, w_1, e_1) and (v_2, w_2, e_2) that give isomorphic CR-decompositions (L_1, R_1, e_1) and (L_2, R_2, e_2), where by isomorphic we mean that there either exist graph isomorphisms $L_1 \to L_2$ and $R_1 \to R_2$ or $L_1 \to R_2$ and $R_1 \to L_2$. For example, in Fig. 10 two distinct, but isomorphic decompositions of the wheel W_4 on four vertices are given. In this particular case it is clear that isomorphic CR-decompositions

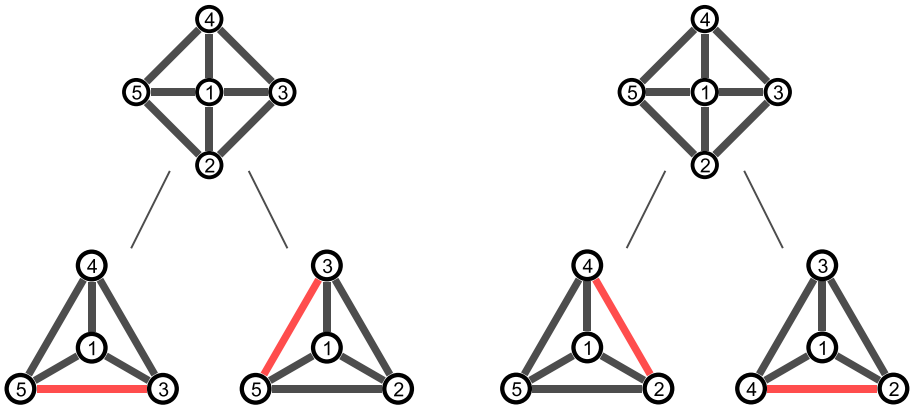

Fig. 10. Two distinct but isomorphic CR-decompositions of the wheel W_4 on 4 vertices. The edge that is eliminated in shown in red (edge 35 in the left tree and edge 24 in the right tree).

appeared because of the symmetry of W_4, however in this paper we are not considering the action of the automorphism group of G.

When enumerating the CR-decompositions, we want to retain only one representative from each isomorphism class of decompositions.

Left-Right symmetry of a decomposition and its truncation. It is possible to have a decomposition (L, R, e) with L isomorphic to R, as it is the case for both CR-decompositions in Fig. 10. With the goal of enumerating CR-trees in mind, when this case occurs we do not recompute the decomposition subtree for the right circuit, but truncate the tree at the corresponding right node. How truncation is implemented in our algorithm is described further down.

Observed decomposition. In the case of an asymmetric decomposition where one or both of the children have already appeared as roots of subtrees in previously enumerated CR-decompositions, we replace such a child with a truncation described below.

Truncation. To handle the cases above, our method for enumerating CR-trees will maintain a list of circuit representatives from each isomorphism class encountered up to the current step of the enumeration algorithm. If a need for truncation is identified, the truncated node will be replaced with a pointer to the corresponding representative circuit, together with the isomorphism operation, see Fig. 11. The enumeration algorithm will not be computing CR-decompositions of truncated nodes.

Isomorphic CR-trees. We consider two CR-trees T_1 and T_2 for G isomorphic if they are isomorphic as rooted binary trees, i.e. if there exists a graph isomorphism $f : T_1 \to T_2$ such that **(i)** r is the root of T_1 if and only if $f(r)$ is the root of T_2 and such that **(ii)** w is a left (resp. right) child of v iff $f(w)$ is a left (resp. right) child of $f(v)$. Moreover, we require that sibling nodes v and w are isomorphic as graphs to either $f(v)$ and $f(w)$, or to $f(w)$ and $f(v)$. Fig. 10 is an example of isomorphic CR-trees.

Due to the numerous isomorphism checks that are required to decide whether two CR-trees are isomorphic, we decided for time-efficiency reasons to instead enumerate all truncated CR-trees from which all CR-trees can be recovered. We leave as an open problem the incorporation of CR-tree isomorphisms into the enumeration algorithm, as well as its complexity analysis.

Conventions for partially eliminating duplicate data. While in principle the CR-decompositions (L, R, e) and (R, L, e) of a circuit are distinct and lead to distinct CR-trees, they imply the same elimination strategy, i.e. the elimination of e from circuits L and R. To avoid the duplication of this data, when a 3-connected circuit G on n vertices is decomposed into circuits L and R according to an admissible triple (v, w, e) we always take (L, R, e) to be as in Algorithm 1: L is the circuit on $n-1$ vertices obtained as $G - v + e$ and R is the unique circuit in $G - w + e$. Moreover, when G is not 3-connected, we take L to be the circuit with most vertices. If both circuits have the same number of vertices, (i) in the case when one is 3-connected but the other is not we take L to be the 3-connected circuit, (ii) if both or none are 3-connected, we take L to be the circuit that is computed first in an algorithm that is computing a 2-split.

6. Data structures

We now describe the data structures used by the enumeration algorithm. Their content at the termination of the algorithm is the output. The main elements are a list of non-isomorphic circuits and a list of truncated trees. Each truncated tree is made out of two types of nodes called C-nodes (C for "Circuit") and B-nodes (B for "Branching") described below.

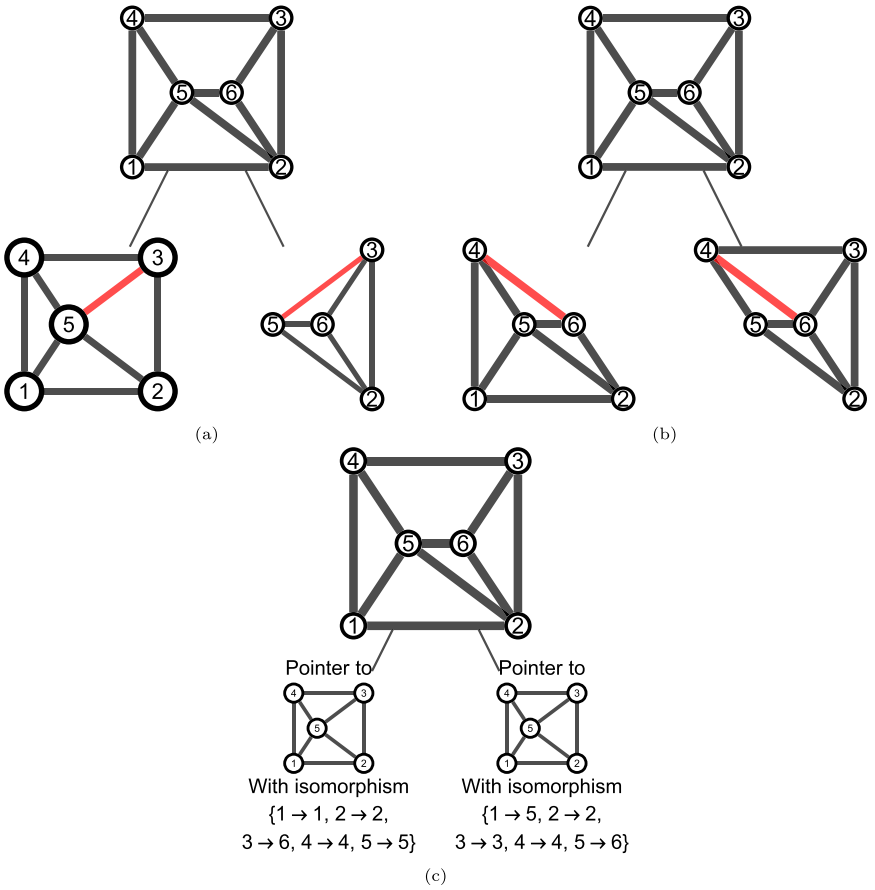

Fig. 11. CR-decompositions for the Desargues-plus-one circuit. (a) The first CR-decomposition that is computed. The left child (a wheel W_4) and the right child (a K_4) are encountered for the first time so no truncation occurs. (b) The second CR-decomposition that is computed. Both children are isomorphic to W_4 which was encountered in a previous step; truncation occurs. (c) Truncation: instead of keeping the graphs in the second decomposition, we keep a pointer to the isomorphic representative circuit together with the corresponding isomorphism.

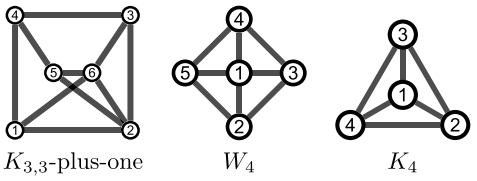

Fig. 12. The list of circuit representatives that appear during the enumeration of truncated CR-trees for the $K_{3,3}$-plus-one circuit. The order of the circuits in this list corresponds to the order in which they were encountered during the execution of the enumeration algorithm.

6.1. List of circuit representatives

Each circuit representative is given a name (e.g. K_4, W_4, $K_{3,3}$-plus-one etc.) and has an identifier given by its position in the list. For example, the list of circuit representatives that appears in the enumeration of truncated CR-trees for $K_{3,3}$-plus-one is in Fig. 12. The order of the circuits in this list corresponds to the order in which they were encountered during the execution of the enumeration algorithm.

6.2. List of truncated trees

The list of truncated trees stores all the truncated trees in the order in which they are generated by the algorithm. When a new tree is generated, it is truncated as necessary according to the current list of circuit representatives.

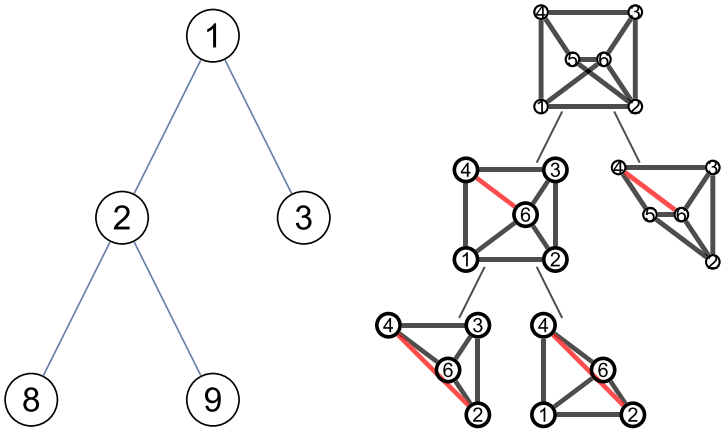

Fig. 13. Left: a truncated CR-tree for the $K_{3,3}$-plus-one circuit represented with C-nodes 1, 2, 3, 8 and 9. Right: the same truncated CR-tree in which the C-nodes are replaced with the circuits that they represent.

6.3. Representation of truncated trees

A truncated tree is represented by a collection of C-nodes and B-nodes. A C-node is an ordered triple of the following data:

- a pointer to a representative circuit
- an isomorphism
- a list of pointers to B-nodes.

Each C-node corresponds to a circuit G that has appeared in the execution of the enumeration algorithm, and keeps track whether G was found to be isomorphic to some previously encountered graph or not. A B-node corresponds to a single CR-decomposition of a circuit from the list of representative circuits. Only the C-nodes with the identity isomorphism $v \mapsto v$ for all $v \in V(G)$ have a non-empty list of B-nodes. The nodes of truncated trees at which truncation occurs will be exactly those C-nodes which have an empty list of B-nodes.

6.4. C-nodes

When G is encountered, it is compared against the graphs in the list of circuit representatives. Two cases can occur:

- G is isomorphic to some representative circuit H. In this case the C-node for G is initialized as $(\mathrm{pt}_H, f, \{\})$, where pt_H is a pointer to H, $f : H \to G$ is a graph isomorphism, and the list of pointers to B-nodes is empty.
- G is not isomorphic to any representative circuit. In this case G is appended to the list of representatives and the C-node for G is initialized as $(\mathrm{pt}_G, f, \{\})$, where pt_G is a pointer to G, $f : G \to G$ is the identity isomorphism, and the list of pointers to B-nodes is empty. The list of B-nodes will be filled in a subsequent step.

6.5. B-nodes

A B-node is an ordered triple (p, l, r) of pointers to C-nodes. Here p, l and r stand for (pointer to) parent, left child, and right child, respectively. The B-node (p, l, r) captures the information about a single CR-decomposition of the graph corresponding to the C-node pointed to by p.

6.6. An example of a representation of a CR-tree

Let G be the $K_{3,3}$-plus one circuit shown in Fig. 12. This circuit has 3 admissible vertices $\{1, 3, 5\}$ and all admissible triples are of the form (v, w, e_{24}) and (v, w, e_{46}) where $v, w \in \{1, 3, 5\}$ with $v \neq w$. Therefore, there are 12 CR-decompositions for G and all are given by two wheels W_4 of 4 vertices. A truncated CR-tree for G is shown in Fig. 13.
The nodes in Fig. 13 correspond to the following C-nodes:

1: $(G, \text{identity}, \{B_1, \ldots, B_{12}\})$

2: $(G_1, \text{identity}, \{B_{13}, \ldots, B_{16}\})$

3: $(G_1, f_3, \{\})$

8: $(G_2, \text{identity}, \{\})$

9: $(G_2, f_9, \{\})$

The graph G_1 is a wheel W_4 with vertices labeled $\{1, 2, 3, 4, 6\}$, where 6 is the central vertex. The graph G_2 is a K_4 on the vertices $\{2, 3, 4, 6\}$. These graphs form the unique representative circuits. Any other circuit that appears in a CR-tree for G is isomorphic to G_1 or G_2. The isomorphisms f_3 and f_9 are given as follows:

f_3: $(1, 2, 3, 4, 6) \mapsto (5, 2, 3, 4, 6)$

f_9: $(2, 3, 4, 6) \mapsto (4, 2, 6, 1)$.

The C-node 3 is truncated as it represents an isomorphic copy of G_1. To be consistent, the node 9 we also consider as truncated, even though it is a leaf by definition. The B-nodes relevant for this truncated CR-tree are $B_1 = (1, 2, 3)$ and $B_{13} = (2, 8, 9)$.

This CR-tree is completely represented by the set of C-nodes $\{1, 2, 3, 8, 9\}$ and the B-nodes $\{B_1, B_{13}\}$. To reconstruct the tree from this representation we simply follow the pointers in B-nodes starting with B_1.

The B-nodes $B_2, \ldots, B_{12}$ correspond to the other 11 CR-decompositions of G, and the B-nodes B_{14}, B_{15}, B_{16} correspond to the remaining 3 CR-decompositions of G_2.

The enumeration algorithm described in the following section will in the same step compute not just the C-nodes for this particular CR-tree, i.e. but all C-nodes in the CR-decompositions of the root node. This is why the labels for C-nodes jump from 3 to 8; the C-nodes 4 to 7 correspond to the isomorphic copies of G_1 that appear in other truncated CR-trees for G. The enumeration algorithm will produce two more C-nodes, both representing circuits isomorphic to G_3. Therefore all truncated CR-trees for G are represented by the following data:

Representative circuits: G, G_1, G_2;
C-nodes 1, 2, …, 11.
B-nodes (1,2,3), (1,4,5), (1,2,6), (1,4,7), (1,3,2), (1,5,4), (1,3,6), (1,5,7), (1, 6, 2), (1, 7, 4), (1, 6, 3), (1, 7, 5), (2, 8, 9), (2, 10, 11), (2, 11, 10), (2, 9, 8),

and all can be reconstructed by following the pointers in the B-nodes.

Note that in this example certain B-nodes carry symmetric data. For example there is no reason to consider both $(1, 2, 3)$ and $(1, 3, 2)$. While the CR-trees will be distinct, the information that they carry, i.e. the elimination strategy for computing the circuit polynomial of G, will be the same. In this presentation we keep this symmetric data, however in our implementation we have optimized this step so that such symmetries are excluded.

7. Enumerating C-nodes and B-nodes

We describe now Algorithm 5, ENUMERATE-NODES(G). It takes as input a 3-connected circuit G and produces representatives of isomorphism classes of circuits, all the C- and all the B-nodes that can occur in any truncated CR-tree for G. The algorithm maintains a queue of pointers to C-nodes and uses two methods: UPDATE-C-NODE, Algorithm 7, and CREATE-B-NODE method, Algorithm 6. These algorithms create new C- and B-nodes and apply isomorphism tests. The details of the CREATE-B-NODE and UPDATE-C-NODE methods are explained below.

Algorithm 5: ENUMERATE-NODES(G).

Input: Rigidity circuit G
Output: Lists of representative circuits, C- and B-nodes for G

```
1 listOfCircReps = {G};
2 C_G = (pt_G, identity isomorphism, Null);
3 listOfAllC-Nodes = {pt_{C_G}};
4 listOfAllB-Nodes = Empty;
5 processingQueue = {pt_{C_G}};
6 while processingQueue ≠ Empty do
7 |   currentC-Node = processingQueue.dequeue();
8 |   UPDATE-C-NODE(currentC-Node)
9 return {listOfCircReps, listOfAllC-Nodes, listOfAllB-Nodes}
```

7.1. Creating a single B-node

Algorithm 6, CREATE-B-NODE is invoked within the UPDATE-C-NODE method (Section 7.2), which is invoked for a C-node C_G corresponding to a representative circuit G. The input that the algorithm receives distinguishes whether G is 3-connected or not: it is either an admissible triple (v, w, e), or a triple (v, w, Null) where $\{v, w\}$ is a separation pair. The necessary checks are performed in Algorithm 7. CREATE-B-NODE algorithm computes a single CR-decomposition (L, R) of a circuit G, tests L and R for isomorphism against the circuits that are in the list of circuit representatives, creates new C-nodes C_L and C_R, and outputs a triple of pointers $(\text{pt}_{C_G}, \text{pt}_{C_L}, \text{pt}_{C_R})$ to C_G, C_L and C_R.

Algorithm 6: CREATE-B-NODE(pt_{C_G}, v, w, e).

Input: pointer pt_{C_G} to a C-node C_G and an admissible triple (v, w, e) or a triple (v, w, Null) where $\{v, w\}$ is a separation pair
Output: New B-node $(\text{pt}_{C_G}, \text{pt}_{C_L}, \text{pt}_{C_R})$
Global variables: `listOfCircReps`, `listOfAllC-Nodes` initialized in Algorithm 5

```
1  if e ≠ Null then
2      (L, R) = CRD(G, v, w, e)
3  else
4      (L, R) = a 2-split of G along the separation pair {v, w}
5  for J in {L, R} do
6      if J is not isomorphic to any representative circuit then
7          append circuit to listOfCircReps;
8          create a new C-node C_J = (pt_J, identity, Null);
9          append pt_{C_J} to listOfAllC-Nodes;
10         enqueue pt_{C_J} in processingQueue
11     if J is isomorphic to a representative circuit H with isomorphism f : H → J then
12         create a new C-node (pt_H, f, Null);
13         append pt_{C_J} to listOfAllC-Nodes
14 return (pt_{C_G}, pt_{C_L}, pt_{C_R})
```

Analysis of Algorithm 6. During its execution, this algorithm will append new elements to the global lists of circuit representatives and all (pointers to) C-nodes.

In lines 1-5 we find a CR-decomposition either by invoking CRD(G, v, w, e), Algorithm 4 or by computing a 2-split along $\{v, w\}$. Starting with line 5, first the circuit L, and then R, is tested for isomorphism against currently known circuit representatives.

Lines 6-10 account for the case when L (resp. R) is not isomorphic to any currently known circuit representative. In this case it becomes a new representative, and a new C-node C_L (resp. C_R) is created for it. Because this circuit is the first instance of its isomorphism class that appeared, C_L (resp. C_R) does not exist in the list of all C-nodes so we are free to append it to this list. For the same reason, we enqueue a pointer to C_L (resp. C_R) to the processing queue.

Lines 11-13 account for the case when L (resp. R) is isomorphic to some known circuit representative H with isomorphism $f : H \to L$. A new C-node C_L (resp. C_R) is created and we append a pointer pt_{C_L} (resp. pt_{C_R}) to it. In line 14 the newly created B-node is returned.

Except for the isomorphism check, the rest of the algorithm runs in polynomial time. We are assuming that the isomorphism check always runs in exponential time and do not account for the special cases when it is log space (e.g. for planar graphs) or polynomial time (e.g. for graphs of bounded genus).

7.2. Updating a C-node

UPDATE-C-NODE, Algorithm 7 takes as input a pointer pt_{C_G} to the C-node C_G that is enqueued in the processing queue in Algorithm 5. The output is the C-node C_G updated with a list of pointers to B-nodes. Recall that only those C-nodes that correspond to circuit representatives are in the processing queue.

During its execution, this algorithm will append new elements to the global lists of pointers to all B-nodes. Line 1 initializes an empty list that is going to be filled in by pointers to B-nodes. In line 2 we check if G is not 3-connected. If so, we find all separation pairs, create a B-node for each 2-split with CREATE-B-NODE, Algorithm 6, and append it to the local and global lists of B-nodes. Separation pairs can be found in linear time [8]; for full details see our preprint on the enumeration of the 2-connected case [15]. Similarly, in lines 9-13 we obtain all triples (v, w, e) with FIND-ADMISSIBLE(G), Algorithm 3, create a B-node for each triple with CREATE-B-NODE and append it to the local and global lists of B-nodes. In line 14 the C-node C_G is modified so that it now contains a non-empty list of pointers to B-nodes.

Remark. The complexity of this algorithm depends on the isomorphism checks performed within the CREATE-B-NODE method. While the complexity of the general Graph Isomorphism Problem has not been settled theoretically, color-refinement heuristics perform very well in practice [16]. Mathematica's GraphIsomorphism function, which we have used in the prototype implementation of the algorithms described in this paper, ran very fast on our specific class of graphs,

Algorithm 7: UPDATE-C-NODE(pt_{C_G}).

Input: pointer pt_{C_G} to C-node C_G
Output: C-node C_G updated with a list of pointers to B-nodes
Global variables: `listOfAllB-Nodes` initialized in ENUMERATE-NODES, Algorithm 5

```
1  pointersToBNodes = Empty;
2  if VERTEX-CONNECTIVITY(G) ≠ 3 then
3      SP = all separation pairs for G;
4      forall (v, w) in SP do
5          B = CREATE-B-NODE(pt_{C_G}, v, w, Null);
6          append pt_B to listOfAllB-Nodes;
7          append pt_B to pointersToBNodes
8  else
9      admissibleTriples = FIND-ADMISSIBLE(G);
10     forall (v, w, e) in admissibleTriples do
11         B = CREATE-B-NODE(pt_{C_G}, v, w, e);
12         append pt_B to listOfAllB-Nodes;
13         append pt_B to pointersToBNodes
14 C_G ← (pt_G, identity, pointersToBNodes);
15 return
```

rigidity circuits. It remains an open problem to find an algorithm for circuit-isomorphism with a theoretically-proven time complexity.

8. Enumerating truncated CR-trees of a circuit G

The input to Algorithm 8 ENUM-TRUNCATED-TREES is a tree T with C-nodes as its nodes. The initial call ENUM-TRUNCATED-TREES(G) is understood as the tree G in which the only node is the C-node C_G that corresponds to G. The algorithm follows the pointers of B-nodes computed by ENUMERATE-NODES(G), Algorithm 5, recursively in a depth-first manner. We describe only the recursive enumeration and assume that ENUMERATE-NODES(G) has been computed and an empty list `listOfAllTruncatedTrees` has been initialized before the initial call of Algorithm 8.

Algorithm 8: ENUM-TRUNCATED-TREES(T).

Input: tree T with C-nodes as its nodes
Output: the list `listOfAllTruncatedTrees` of all truncated trees for G
Global variables: empty list `listOfAllTruncatedTrees`, initialized before the first call of this algorithm

```
1  if all leaves of T have empty list of B-nodes then
2      append T to listOfAllTruncatedTrees;
3      return
4  for leaf C_J of T with a non-empty list of B-nodes do
5      for (pt_{C_J}, pt_{C_L}, pt_{C_R}) in list of B-nodes of C_J do
6          T ← T + (C_J C_L, C_J C_R);
7          return ENUM-TRUNCATED-TREES(T)
```

Let C_G be the C-node corresponding to a circuit G on n vertices. The initial call of the algorithm is for the tree T in which C_G is the only node. We assume that the leaves are found with a depth-first search; in particular, the leaves are considered from left to right. The tree T is extended whenever it has a leaf C_J that has a non-empty list of pointers to B-nodes. We create two children (C_L, C_R) for C_J that are pointed to by the pointers in the B-node $(\text{pt}_{C_J}, \text{pt}_{C_L}, \text{pt}_{C_R})$ and extend T with the edges $(C_J C_L, C_J C_R)$. These nodes are the new leaves, and we recursively call the algorithm on the new enlarged tree.

The complexity of enumerating truncated trees depends greatly on the input circuit G. Those circuits that exhibit a lot of symmetry will have many isomorphic CR-decompositions thereby reducing the number of possible truncated trees.

9. Reconstructing CR-trees from a single truncated CR-tree

Let $\mathcal{T}_G$ denote the list of all truncated trees for G enumerated by Algorithm 8, and let $T \in \mathcal{T}_G$. In this section we describe how to recover all CR-trees whose truncation is T.

For clarity of presentation, we enlarge $\mathcal{T}_G$ by adding to it all truncated trees for all representative circuits that appear in $\mathcal{T}_G$. In other words, if H is a representative circuit in a truncated tree for G, we add $\mathcal{T}_H$ to $\mathcal{T}_G$. Note that for implementation purposes we do not have to do this since any truncated tree in $\mathcal{T}_H$ can be reconstructed from the data generated by $\mathcal{T}_G$

by following the pointers in the appropriate B-nodes and applying isomorphisms, starting from the B-nodes of the form $(\mathrm{pt}_H, \mathrm{pt}_A, \mathrm{pt}_B)$.

Algorithm 9: RECONSTRUCT-CR-TREES($\mathcal{T}_G$,T).

Input: All truncated trees $\mathcal{T}_G$ for a circuit G and $T \in \mathcal{T}_G$
Output: List completeResultantTrees of CR-trees for G whose truncation is T

```
1  completeResultantTrees = Empty;
2  processingQueue = {T};
3  while processingQueue is not empty do
4      currentTree = processingQueue.dequeue();
5      nonTerminatingLeaves = set of C-node leaves in currentTree that point to a circuit not isomorphic to K4;
6      if nonTerminatingLeaves is empty then
7          append currentTree to completeResultantTrees
8      else
9          (pt_H, f, Null) = first element of nonTerminatingLeaves that points to a circuit H;
10         forall T' ∈ T_H do
11             enqueue T + f(T') to processingQueue
12 return completeResultantTrees
```

The algorithm uses a processing queue of partial CR-trees. The while loop starting at line 3 dequeues a current tree. At each iteration, the leaves of the current tree are checked (line 5). If there are no leaves isomorphic to a K_4, then the current tree is a complete resultant tree and we append it to the list of complete resultant trees (lines 5-7). Otherwise, in lines 9-11, we take the first leaf $(\mathrm{pt}_H, f, \mathrm{Null})$ found (e.g. the first detected by DFS), that points to a 3-connected circuit H that is not isomorphic to a K_4. We then extend T with all possible branchings of H. These branchings are given by the truncated trees $T' \in \mathcal{T}_H$ rooted at H. However, before we can attach T' to T, we have to relabel all the nodes in T' with the isomorphism f stored in $(\mathrm{pt}_H, f, \mathrm{Null})$. By $f(T')$ we denote the tree obtained from T' such that a C-node $(\mathrm{pt}_J, g, *)$ of T' is mapped to the C-node $(\mathrm{pt}_J, f \circ g, \mathrm{Null})$ of $f(T')$, where f is restricted to the appropriate domain.

10. Implementation and results

We have implemented the algorithm for enumerating truncated CR-trees in Mathematica V13.3.1 and have successfully run it on circuits with as many as 15 vertices. In our implementation we have performed optimizations in which duplicate B-nodes are ignored, and for symmetric branchings given by B-nodes of the form (pt_G, pt_A, pt_B) and (pt_G, pt_B, pt_A) only one is counted.

The wheels W_n of n vertices. The wheel W_n of n vertices is obtained by connecting each vertex of the cycle graph on n vertices to a new vertex. These graphs exhibit dihedral symmetry and can be realized as regular n-gons in which every vertex on the boundary is connected to the barycenter. The number of truncated CR-trees and the number of unique circuits appearing in those trees is given in Table 4.

For $n \geq 4$ the wheel W_n has up to isomorphism only one CR-decomposition given by (W_{n-1}, K_4), with $W_3 = K_4$. As a consequence, W_n has up to isomorphism only one CR-tree obtained by first decomposing the root W_n into $L = W_{n-1}$ and $R = K_4$ and then inductively decomposing W_i for all $4 \leq i \leq n-1$ into (W_{i-1}, K_4).

The number of truncated trees for W_n is given by the following lemma.

Lemma 2. *Let $n \geq 4$ and let $\tau(n)$ denote the number of truncated trees for W_n. Then $\tau(n) = \frac{1}{2}(n^2 - n - 8)$.*

Proof. Let v be a vertex of degree 3 in W_n and u, w its degree 3 neighbors. Let z be any of the $n-4$ vertices of degree 3 not adjacent to v. Then (v, z, e_{uw}) is an admissible triple such that the corresponding CR-decomposition is given by W_{n-1} with vertex set $V(W_n) - \{v\}$ and the K_4 on vertices $\{u, v, w, x\}$, where x is the unique vertex of degree n in W_n. Therefore, all such admissible triples give the same B-node, of which we keep only one. This means that the C-node for W_n has n B-nodes, one for each vertex of degree 3.

Only the first of these n B-nodes points to C-nodes that are not truncated. Therefore, if $\tau(n)$ denotes the number of truncated trees for W_n, then $\tau(n) = \tau(n-1) + n - 1$ with $\tau(4) = 2$. The solution to this recurrence relation is $\tau(n) = \frac{1}{2}(n^2 - n - 8)$. □

The role of symmetry is visible when we compare Table 4 with the results for circuits that have less symmetry.

Desargues-plus-1 with Henneberg-2 moves. The Desargues-plus-1 is the circuit on 6 vertices given in Fig. 1. A Henneberg-2 move on a graph G with respect to an edge e and vertex $v \notin e$ is defined as the operation in which we subdivide e with a new vertex w and add a new edge vw. It is well known that any Henneberg-2 move on a 3-connected circuit on n vertices results with a 3-connected circuit on $n+1$ vertices.

Table 4
Statistics for the wheel W_k of k vertices given by the edge-set $\{\{1, j\} \mid j = 2, \ldots, k\}$. Computational time is given in seconds.

n	circuit	# truncated trees	# unique circuits appearing	comp. time
6	W_5	6	3	0.503314
7	W_6	11	4	1.386
8	W_7	17	5	3.15491
9	W_8	24	6	6.23663
10	W_9	32	7	11.1559
11	W_{10}	41	8	18.5865
12	W_{11}	51	9	29.6419
13	W_{12}	62	10	45.216
14	W_{13}	74	11	66.3113
15	W_{14}	87	12	94.6188

Table 5
Statistics for the rigidity circuits obtained from the Desargues-plus-one circuit by applying Henneberg-2 moves randomly. The notation Dp1-rand-i stands for the Desargues-plus-1 on which i random Henneberg-2 moves have been applied. The edge-sets of the circuits are given in Table 6. Computational time is given in seconds.

n	circuit	# truncated trees	# unique circuits appearing	comp. time
7	Dp1-rand-1	23	6	1.87185
8	Dp1-rand-2	145	11	6.27536
9	Dp1-rand-3	5 648	21	14.3079
10	Dp1-rand-4	654 711	70	99.9289
11	Dp1-rand-5	94 182 020 875	289	685.383
12	Dp1-rand-6	243 674 119 275	752	2182.71
13	Dp1-rand-7	6 693 672 561 546 338 585 509 181 772 986 135	3730	22115.7

Table 6
The edge-sets of circuits from Table 5. The notation Dp1-rand-i stands for Desargues-plus-1 with i random Henneberg-2 moves.

n	circuit	edge-set
7	Dp1-rand-1	{{1, 4}, {1, 5}, {1, 6}, {1, 7}, {2, 4}, {2, 5}, {2, 6}, {2, 7}, {3, 5}, {3, 6}, {3, 7}, {4, 6}}
8	Dp1-rand-2	{{1, 3}, {1, 4}, {1, 6}, {1, 8}, {2, 4}, {2, 6}, {2, 7}, {3, 5}, {3, 6}, {4, 6}, {4, 7}, {4, 8}, {5, 7}, {5, 8}}
9	Dp1-rand-3	{{1, 3}, {1, 4}, {1, 5}, {1, 7}, {2, 4}, {2, 5}, {2, 6}, {2, 9}, {3, 5}, {3, 9}, {4, 6}, {4, 8}, {5, 7}, {6, 8}, {6, 9}, {7, 8}}
10	Dp1-rand-4	{{1, 3}, {1, 4}, {1, 5}, {1, 6}, {2, 4}, {2, 6}, {2, 10}, {3, 6}, {3, 8}, {4, 6}, {4, 8}, {4, 9}, {5, 7}, {5, 9}, {6, 7}, {7, 8}, {7, 10}, {9, 10}}
11	Dp1-rand-5	{{1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 9}, {2, 5}, {2, 6}, {2, 7}, {3, 5}, {3, 7}, {3, 9}, {3, 11}, {4, 6}, {4, 7}, {6, 8}, {7, 8}, {7, 10}, {8, 11}, {9, 10}, {10, 11}}
12	Dp1-rand-6	{{1, 3}, {1, 4}, {1, 6}, {1, 8}, {1, 10}, {2, 5}, {2, 7}, {2, 9}, {2, 11}, {3, 6}, {3, 8}, {4, 6}, {4, 7}, {5, 9}, {5, 10}, {6, 7}, {6, 12}, {7, 10}, {8, 9}, {9, 12}, {10, 11}, {11, 12}}
13	Dp1-rand-7	{{1, 4}, {1, 7}, {1, 8}, {1, 12}, {2, 4}, {2, 5}, {2, 9}, {3, 5}, {3, 10}, {3, 11}, {3, 12}, {3, 13}, {4, 6}, {4, 7}, {5, 11}, {6, 7}, {6, 9}, {6, 13}, {7, 10}, {8, 9}, {8, 10}, {8, 11}, {9, 12}, {10, 13}}

In this example we have generated 7 circuits with an increasing number of vertices by successively applying a Henneberg-2 move to the Desargues-plus-one circuit. The results of the enumeration are given in Table 5. The choice of an edge e and vertex v for the Henneberg-2 move was random. The edge-sets of the resulting circuits are given in Table 6.

$K_{3,3}$-plus-1 with Henneberg-2 moves. The $K_{3,3}$-plus-1 is the circuit on 6 vertices given in Fig. 12. As in the previous example, we have generated 7 circuits with an increasing number of vertices by successively applying a Henneberg-2 move to the $K_{3,3}$-plus-1 circuit. The results of the enumeration are given in Table 7. The choice of an edge e and vertex v for the Henneberg-2 move was random. The edge-sets of the resulting circuits are given in Table 8.

Table 7
Statistics for the rigidity circuits obtained from the $K_{3,3}$-plus-one circuit by applying Henneberg-2 moves randomly. The notation $K_{3,3}$-rand-i stands for $K_{3,3}$-plus-1 with i random Henneberg-2 moves. The edge-sets of the circuits are given in Table 8. Computational time is given in seconds.

n	circuit	# truncated trees	# unique circuits appearing	comp. time
7	$K_{3,3}$-rand-1	57	6	3.74199
8	$K_{3,3}$-rand-2	126	12	8.87238
9	$K_{3,3}$-rand-3	9 646	27	41.4054
10	$K_{3,3}$-rand-4	2 474 671	100	269.798
11	$K_{3,3}$-rand-5	42 354 053 225	97	225.857
12	$K_{3,3}$-rand-6	4 267 627 299 735	293	981.639
13	$K_{3,3}$-rand-7	383 529 900 996 175 910 365 699 391 713	1961	11653.9

Table 8
The edge-sets of circuits from Table 7. The notation $K_{3,3}$-rand-i stands for $K_{3,3}$-plus-1 with i random Henneberg-2 moves.

n	circuit	edge-set
7	$K_{3,3}$-rand-1	{{1, 5}, {1, 6}, {1, 7}, {2, 4}, {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3, 6}, {4, 6}, {4, 7}, {6, 7}}
8	$K_{3,3}$-rand-2	{{1, 4}, {1, 6}, {1, 7}, {2, 4}, {2, 5}, {2, 6}, {2, 7}, {3, 4}, {3, 6}, {3, 8}, {4, 6}, {5, 7}, {5, 8}, {7, 8}}
9	$K_{3,3}$-rand-3	{{1, 5}, {1, 7}, {1, 9}, {2, 4}, {2, 5}, {2, 6}, {2, 8}, {3, 4}, {3, 5}, {3, 6}, {4, 6}, {4, 7}, {4, 8}, {4, 9}, {6, 9}, {7, 8}}
10	$K_{3,3}$-rand-4	{{1, 4}, {1, 6}, {1, 7}, {1, 8}, {1, 10}, {2, 4}, {2, 5}, {2, 6}, {3, 4}, {3, 7}, {3, 9}, {4, 6}, {4, 10}, {5, 7}, {5, 9}, {5, 10}, {6, 8}, {8, 9}}
11	$K_{3,3}$-rand-5	{{1, 4}, {1, 7}, {1, 8}, {2, 4}, {2, 6}, {2, 7}, {2, 11}, {3, 4}, {3, 5}, {3, 6}, {3, 10}, {4, 6}, {4, 9}, {4, 11}, {5, 8}, {5, 10}, {6, 7}, {7, 9}, {8, 9}, {10, 11}}
12	$K_{3,3}$-rand-6	{{1, 4}, {1, 5}, {1, 6}, {2, 4}, {2, 6}, {2, 7}, {2, 8}, {3, 5}, {3, 6}, {3, 7}, {3, 10}, {3, 11}, {4, 6}, {4, 7}, {4, 9}, {5, 8}, {7, 9}, {7, 12}, {8, 11}, {9, 12}, {10, 11}, {10, 12}}
13	$K_{3,3}$-rand-7	{{1, 6}, {1, 7}, {1, 8}, {1, 9}, {2, 6}, {2, 9}, {2, 11}, {2, 12}, {3, 4}, {3, 5}, {3, 7}, {3, 10}, {3, 12}, {4, 6}, {4, 10}, {4, 11}, {5, 9}, {5, 13}, {6, 8}, {6, 12}, {7, 10}, {8, 11}, {8, 13}, {11, 13}}

Additional experiments. We have generated data for the number of all truncated CR-trees of all 3-connected circuits with $n = 6, 7, 8$. Due to the large number of circuits considered, we omit the presentation here, however the data is available on the GitHub repository [13] in the Mathematica notebook format.

11. Conclusion and future work

In this paper we described the combinatorial aspects of algorithms necessary to compute circuit polynomials of rigidity circuits. The results of the presented experiments suggest that while the number of truncated CR-trees grows rapidly, the growth might be controlled in circuits that exhibit lots of symmetry, as was the case with the wheel graphs. We leave this direction of research as a future project. It remains to be seen how much of a performance improvement do truncated CR-trees provide over considering isomorphism classes of CR-trees where numerous isomorphism checks have to be performed.

We do not include the discussion on what kind of insight do CR-trees provide for the rigidity matroid for the reason that we believe that this question is better suited in a more general context. Namely, C-resultants and CR-trees naturally appear in the class of matroids for which a sort-of converse of the circuit elimination axiom holds, i.e. the property that for any circuit C of cardinality not minimal among all circuits there exists a pair of distinct circuits C_1, C_2 with less elements than C and such that $C_3 = (C_1 \cup C_2) - e$ for some $e \in C_1 \cap C_2$. To our knowledge this class of matroids has not been studied, hence any insight given by CR-trees in this general context will translate immediately to the rigidity matroid.

Furthermore, we do not touch upon the motivational problem of performing algebraic calculations necessary to obtain the circuit polynomials. These polynomials, while incomprehensibly large, are sparse in the sense that most coefficients in the dense representation $\sum a_{i_1,\ldots,i_n} x_{i_1}^{k_{i_1}} \cdots x_{i_n}^{k_{i_n}}$ are 0, and therefore the data structures used for representing polynomials are not always optimized for our case. Furthermore, the algorithms for computing resultants implemented in software like Mathematica, Maple, Singular etc. are general purpose algorithms that work on polynomials arising in any context, and

not optimized for the very large circuit polynomials. The implementation of new data structures and new computational algorithms that would be necessary to compute larger circuit polynomials we see as a comprehensive future project for which this paper is the starting point.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ileana Streinu reports financial support was provided by National Science Foundation.

Data availability

No data was used for the research described in the article.

References

[1] David Avis, Komei Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete Comput. Geom. 8 (1992) 295–313.
[2] David Avis, Komei Fukuda, Reverse search for enumeration, Discrete Appl. Math. 65 (1996) 21–46.
[3] David Avis, Naoki Katoh, Makoto Ohsaki, Ileana Streinu, Shin-ichi Tanigawa, Enumerating constrained non-crossing minimally rigid frameworks, Discrete Comput. Geom. 40 (2008) 31–46, https://doi.org/10.1007/s00454-007-9026-x.
[4] Alex R. Berg, Tibor Jordán, A proof of Connelly's conjecture on 3-connected circuits of the rigidity matroid, J. Comb. Theory, Ser. B 88 (1) (2003) 77–97, https://doi.org/10.1016/S0095-8956(02)00037-0.
[5] G. Di Battista, R. Tamassia, Incremental planarity testing, in: 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 436–441.
[6] Giuseppe Di Battista, Roberto Tamassia, On-line graph algorithms with spqr-trees, in: Michael S. Paterson (Ed.), Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1990, pp. 598–611.
[7] A. Dress, L. Lovász, On some combinatorial properties of algebraic matroids, Combinatorica 7 (1) (1987) 39–48, https://doi.org/10.1007/BF02579199.
[8] Gutwenger Carsten, Petra Mutzel, A linear time implementation of SPQR-trees, in: Joe Marks (Ed.), Graph Drawing, Springer Berlin Heidelberg, 2001, pp. 77–90.
[9] John E. Hopcroft, Robert Endre Tarjan, Dividing a graph into triconnected components, SIAM J. Comput. 2 (3) (1973) 135–158, https://doi.org/10.1137/0202012.
[10] Donald J. Jacobs, Bruce Hendrickson, An algorithm for two-dimensional rigidity percolation: the Pebble game, J. Comput. Phys. 137 (2) (1997) 346–365, https://doi.org/10.1006/jcph.1997.5809.
[11] Audrey Lee-St John, Ileana Streinu, Pebble game algorithms and sparse graphs, Discrete Math. 308 (8) (April 2008) 1425–1437, https://doi.org/10.1016/j.disc.2007.07.104.
[12] Goran Malić, Ileana Streinu, Combinatorial resultants in the algebraic rigidity matroid, in: Kevin Buchin, Éric Colin de Verdière (Eds.), 37th International Symposium on Computational Geometry (SoCG 2021), Dagstuhl, Germany, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 189, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 52:1–52:16.
[13] Goran Malić, Ileana Streinu, CayleyMenger - Circuit Polynomials in the Cayley Menger ideal, a GitHub repository, https://github.com/circuitPolys/CayleyMenger, 2021–2023.
[14] Goran Malić, Ileana Streinu, Computing circuit polynomials in the algebraic rigidity matroid, SIAM J. Appl. Algebra Geom. 7 (2) (2023) 345–385, https://doi.org/10.1137/21M1437986.
[15] Goran Malić, Ileana Streinu, Enumerating Combinatorial Resultant Decompositions of 2-Connected Rigidity Circuits, 2023, arXiv:2309.12248v1 [math.CO].
[16] Brendan D. McKay, Adolfo Piperno, Practical graph isomorphism, ii, J. Symb. Comput. 60 (2014) 94–112, https://doi.org/10.1016/j.jsc.2013.09.003.
[17] James Oxley, Matroid Theory, second edition, Oxford Graduate Texts in Mathematics, vol. 21, Oxford University Press, Oxford, 2011.
[18] Zvi Rosen, Algebraic Matroids in Applications, PhD thesis, University of California, Berkeley, 2015, https://digitalassets.lib.berkeley.edu/etd/ucb/text/Rosen_berkeley_0028E_15261.pdf.
[19] Zvi Rosen, Jessica Sidman, Louis Theran, Algebraic matroids in action, Am. Math. Mon. 127 (3) (February 2020) 199–216, https://doi.org/10.1080/00029890.2020.1689781.
[20] William T. Tutte, Connectivity in Graphs, Toronto University Press, Toronto, 1966.
[21] D.J.A. Welsh, Matroid Theory, L. M. S. Monographs, vol. 8, Academic Press [Harcourt Brace Jovanovich Publishers], London, 1976.