RAIRO-Theor. Inf. Appl. 58 (2024) 9
https://doi.org/10.1051/ita/2024004

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

HUMAN-VERIFIABLE PROOFS IN THE THEORY OF WORD-REPRESENTABLE GRAPHS

Sergey Kitaev[1,*] and Haoran Sun[2]

Abstract. A graph is word-representable if it can be represented in a certain way using alternation of letters in words. Word-representable graphs generalise several important and well-studied classes of graphs, and they can be characterised by semi-transitive orientations. Recognising word-representability is an NP-complete problem, and the bottleneck of the theory of word-representable graphs is convincing someone that a graph is non-word-representable, keeping in mind that references to (even publicly available and user-friendly) software are not always welcome. (Word-representability can be justified by providing a semi-transitive orientation as a certificate that can be checked in polynomial time). In the literature, a variety of (usually ad hoc) proofs of non-word-representability for particular graphs, or families of graphs, appear, but for a randomly selected graph, one should expect looking at $O(2^{\#\text{edges}})$ orientations and justifying that none of them is semi-transitive. Even if computer would print out all these orientations and would point out what is wrong with each of the orientations, such a proof would be essentially non-checkable by a human. In this paper, we develop methods for an automatic search of human-verifiable proofs of graph non-word-representability. As a proof-of-concept, we provide "short" proofs of non-word-representability, generated automatically by our publicly available user-friendly software, of the Shrikhande graph on 16 vertices and 48 edges (6 "lines" of proof) and the Clebsch graph on 16 vertices and 40 edges (10 "lines" of proof). Producing such short proofs for relatively large graphs would not be possible without the instrumental tool we introduce (allowing to assume orientations of several edges in a graph, not just one edge as it was previously used) that is a game changer in the area. As a bi-product of our studies, we correct two mistakes published multiple times (two graphs out of the 25 non-word-representable graphs on 7 vertices were actually word-representable, while two non-word-representable graphs on 7 vertices were missing).

Mathematics Subject Classification. 05C62.

Received December 1, 2022. Accepted February 23, 2024.

1. Introduction

There is a long line of research papers in the literature dedicated to the theory of word-representable graphs (*e.g.* see [1, 2] and references therein). The motivation to study these graphs is their relevance to algebra, graph

Keywords and phrases: Word-representable graph, semi-transitive orientation, automated proof, non-word-representability, Clebsch graph, Shrikhande graph.

[1] Department of Mathematics and Statistics, University of Strathclyde, 26 Richmond Street, Glasgow G1 1XH, UK.
[2] College of Science, Donghua University, Shanghai 201620, PR China.

* Corresponding author: sergey.kitaev@strath.ac.uk

theory, computer science, combinatorics on words, and scheduling [2]. In particular, word-representable graphs generalize several fundamental classes of graphs (*e.g. circle graphs*, 3-*colorable graphs* and *comparability graphs*).

Two letters x and y alternate in a word w if after deleting in w all letters but the copies of x and y we either obtain a word $xyxy\cdots$ (of even or odd length) or a word $yxyx\cdots$ (of even or odd length). A graph $G=(V,E)$ is *word-representable* if and only if there exists a word w over the alphabet V such that letters x and y, $x\neq y$, alternate in w if and only if $xy\in E$. We say that w *represents* G. A graph is word-representable if and only if each of its connected components is word-representable [2]. Hence, WLOG we can assume that graphs under our consideration are always connected.

The minimum (by the number of vertices) non-word-representable graph is on 6 vertices, and the only such graph is the wheel graph W_5 (that is obtained from Graph 1 in Figure 3 after removing vertex 7), while there are 25 non-word-representable graphs on 7 vertices [2]. A word is *uniform* if each letter occurs in it the same number of times. We will need the following two results.

Theorem 1.1 ([3]). *If a graph G is word-representable then there exists a uniform word representing it.*

Theorem 1.2 ([3]). *Let $w=w_1w_2$ be a uniform word representing a graph G, where w_1 and w_2 are words. Then the word w_2w_1 also represents G.*

An orientation of a graph is *semi-transitive* if it is acyclic, and for any directed path $v_0\to v_1\to\cdots\to v_k$ either there is no edge between v_0 and v_k, or $v_i\to v_j$ is an edge for all $0\leq i<j\leq k$. An induced subgraph on vertices $\{v_0,v_1,\ldots,v_k\}$ of an oriented graph is a *shortcut* if its orientation is acyclic (contains no directed cycles) and non-transitive, and there is the directed path $v_0\to v_1\to\cdots\to v_k$ and the edge $v_0\to v_k$ called the *shortcutting edge*. A semi-transitive orientation can then be alternatively defined as an acyclic shortcut-free orientation. A fundamental result in the area is the following theorem.

Theorem 1.3 ([4]). *A graph is word-representable if and only if it admits a semi-transitive orientation.*

Remark 1.4. An efficient way to obtain a semi-transitive orientation of a graph G from a word w representing it, which is used in the proof of Theorem 1.3, is as follows: orient an edge xy in G as $x\to y$ if the leftmost occurrence of x in w is to the left of the leftmost occurrence of y in w. Note that the leftmost letter in w will be a *source* in G, that is a vertex having edges incident with it oriented away from it.

Recognizing word-representability of a graph is an NP-complete problem [2], and the bottleneck of the theory of word-representable graphs is in ways to convince someone that a graph is non-word-representable keeping in mind that references to (even publicly available and user-friendly) software, such as [5], are not always welcome. (Word-representability can be justified by providing a semi-transitive orientation as a certificate that can be checked in polynomial time [2]).

1.1. Approaches to deal with non-word-representability

It is known [3] that the neighbourhood of each node in a word-representable graph is a comparability graph, and recognition of a comparability graph is a polynomially solvable problem [6]. Hence, we have a polynomial test for non-word-representability of a graph G: for each vertex, check whether its neighbourhood is a comparability graph; if a "non-comparability neighbourhood" is found, G is not word-representable. However, if all neighbourhoods are comparability graphs, then the test gives no information, namely, the graph may or may not be word-representable [2].

Thus, if the test above does not work and a known non-word-representable induced subgraph is not found, basically we are left with three options to recognise and then justify non-word-representability: either

(a) to come up with some sort of an ad hoc smart argument, usually using properties and/or symmetries of the graph in question, or its induced subgraph, or

(b) to go through $O(2^{\#\text{edges}})$ orientations justifying that none of them is semi-transitive (symmetries can be used here sometimes to reduce the search space, in particular, any given edge can be assumed to be oriented in any way), or
(c) to go through all $O(\#\text{vertices}^2)$ words containing each of the vertex labels equal number of times and to justify that none of them has the right alternation properties (if a graph with n vertices is word-representable then there is a word of length at most n^2 representing it [4]).

Approach (a) above is preferable, but usually is hard to implement. Approach (c) requires going through $O(n^{2n})$ words, however, constraint programming can be used here to speed up the process [7]. In either case, how do we convince someone that the graph is non-word-representable without a reference to software? A variation of approach (b) is used in some existing pieces of software [5, 8]. It works as follows: orient an edge e_1 in a given graph G, then consider a still undirected edge e_2 in G and branch on it, namely, create two copies of the partially oriented graph by orienting e_2 differently; then branch on e_3, *etc.* At each step, make sure that no directed cycles or shortcuts are created (if they are, the respective branch is not to be considered). In any case, even if computer would print out all these orientations (or the entire branching process) and would point at a directed cycle or a shortcut in each of the orientations, such a proof would be essentially non-checkable by a human, as it would typically be a way too long.

1.2. A game changer approach

In this paper we consider producing "short" proofs of non-word-representability dropping the number of cases to be considered from exponential to polynomial, and thus enabling human to verify such proofs. The basic idea is in modifying the branching process by avoiding unnecessary branching *via* certain pre-processing. The following lemma is the key to our approach.

Lemma 1.5 ([9])**.** *Suppose that an undirected graph G has a cycle $C = x_1x_2\cdots x_mx_1$, where $m \geq 4$ and the vertices in $\{x_1, x_2, \ldots, x_m\}$ do not induce a clique in G. If G is oriented semi-transitively, and $m-2$ edges of C are oriented in the same direction (i.e. from x_i to x_{i+1} or vice versa, where the index $m+1 := 1$) then the remaining two edges of C are oriented in the opposite direction.*

Hence, if we try to find a semi-transitive orientation by exhaustively going through all possibilities to orient one edge at a time, and we see a cycle, that does not induce a clique, with all but two edges oriented in the same direction, we do not need to branch on the remaining two edges as they must be oriented in the opposite direction by Lemma 1.5. Similarly, if we see a non-clique cycle with all but two edges oriented in the same direction and an edge e in the cycle oriented in the opposite direction, then we know that the remaining edge is oriented in the same direction as e.

The following theorem allows us to reduce further dramatically the length of a proof of non-word-representability for a graph.

Theorem 1.6. *Suppose that a graph G is word-representable, and v is a vertex in G. Then, there exists a semi-transitive orientation of G where v is a source (or a sink so that all edges incident with v are oriented towards it).*

Proof. Let w be a word representing G. By Theorem 1.1, we can assume that w is uniform, and by Theorem 1.2 that w begins with v. Then we can use Remark 1.4 to obtain a semi-transitive orientation of G in which v is a source. Reversing the orientations of all edges in G, one obtains a semi-transitive orientation in which v is a sink. □

To demonstrate the power of Theorem 1.6, just applying Lemma 1.5 in the branching process for the wheel graph W_5, one needs to branch 6 times. However, assuming WLOG by Theorem 1.6 that the all-adjacent vertex in W_5 is a source, one needs to branch just once. In fact, using the symmetry of W_5, this only branching is not necessary, so non-word-representability of W_5 can be proved by Lemma 1.5 and Theorem 1.6 and observing the symmetry without branching. Usually, picking a vertex of the highest degree in a given non-word-representable

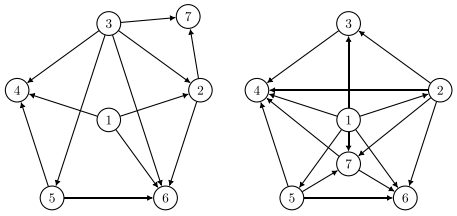

FIGURE 1. The undirected versions of Graphs 12′ and 17′ were assumed to be non-word-representable several times in the literature, for example in [2], although they are actually word-representable as is witnessed by the semi-transitive orientations given in the figure. Graph 12′ misses the edge (1,3). Graph 17′ should not have the edge (1,7).

graph and assuming it to be a source would result in a shortest proof. However, there are cases when assuming a vertex of a smaller degree to be a source results in a shorter proof. Also, note that assuming a particular vertex to be a source may result in a longer proof than making no assumption at all.

In Section 2, we introduce three algorithms getting use of Lemma 1.5 to generate shorter proofs for non-word-representable graphs. The primarily criteria of the efficiency of an algorithm is the number of "lines" (in the sense specified below) in the proof it produces; the secondary criteria is the running time of the algorithm, whose discussion is omitted from this paper as less relevant (the problem is NP-complete, so the running time is exponential, and further details are not so important for us). We test and rank our algorithms on the 25 non-word-representable graphs on 7 vertices correcting, as a by-product, two mistakes published multiple times, *e.g.* in [2]. Indeed, two graphs out of the 25 graphs were produced incorrectly. These incorrect graphs are the undirected versions of the semi-transitively oriented graphs in Figure 1. We leave it to the Reader as a straightforward exercise to prove that the orientations in Figure 1 are indeed semi-transitive. A correct list of the 25 non-word-representable graphs can be found in Figure 3. We note that in our test we choose not to use Theorem 1.6; using this theorem and going through all possible assumptions about a source may give a different outcome for comparison of the algorithms.

We would like to emphasise that the approach involving Lemma 1.5 is not novel: several papers, including [9], use it or its simpler version (considering cycles of length 3 and 4 only) to justify non-word-representability. However, our paper is the first one to discuss an automated search of human-verifiable proofs of graph non-word-representability that allowed us to create publicly available user-friendly software [8]. Moreover, our Theorem 1.6 enlarges significantly the class of graphs for which "reasonably short" proofs of non-word-representability can be produced. As a proof-of-concept, we use the software [8] to find "short" proofs of non-word-representability, generated automatically, of the *Shrikhande graph* on 16 vertices and 48 edges (6 "lines" of proof; see Section 3) and the *Clebsch graph* on 16 vertices and 40 edges (10 "lines" of proof; see Section 4). Proving in a verifiable way (without referring to computer software) that the Clebsch graph and the Shrikhande graph are non-word-representable would be extremely challenging without the tools we introduce.

2. Three algorithms to search for short proofs of non-word-representability

In this section, we consider three algorithms to find shorter proofs of non-word-representability of graphs. All three algorithms use the observation that the branching process should not involve any edges that do not belong to a cycle, as such edges can be oriented arbitrarily (they will never be involved in a directed cycle of a shortcut). Further, all three algorithms use the assumption that to produce a shorter proof, branching should be made on edges belonging to many cycles (which should increase the number of applications of Lem. 1.5).

1. 12→15 B14→15 (Copy 2) B12→14 (Copy 3) O7→15 O12→7 (C7-15-14-12) [**other instructions**] S:7-4-8-16
2. MC4 16→7 O16→15 (C7-16-15) [**other instructions**] S:4-11-3-7

FIGURE 2. Parts of the first two lines in the proof of non-word-representability of the Shrikhande graph in Figure 4.

2.1. The format of a proof

By a "line" of a proof we mean a sequence of instructions that directs us in orienting a partially oriented graph and necessarily ends with detecting a shortcut or another contradiction showing that this particular orientation branch will not produce a semi-transitive orientation. The idea is that if no branch produces a semi-transitive orientation then the graph is non-semi-transitively orientable (and hence non-word-representable by Theorem 1.3).

Each proof begins with $A \to B$ showing the orientation of an edge AB, the first edge we orient, that is selected by an algorithm in a certain way. Because reversing all orientations in a semi-transitively oriented graph produces a semi-transitively oriented graph, WLOG we can omit considering (partially) oriented graphs having $B \to A$, which significantly reduces the number of cases to consider. Further, there are four types of instructions:

- "B" followed by "$X \to Y$ (Copy Z)" means "Branch on edge XY, orient the edge as $X \to Y$, create a copy of the current version of the graph except orient the edge XY there as $Y \to X$, and call the new copy Z; leave Copy Z aside and continue to follow the instructions". The instruction B occurs when the software detects that no application of Lemma 1.5 is possible in the partially oriented graph.
- "MC" followed by a number X means "Move to Copy X", where Copy X of the graph in question is a partially oriented version of the graph that was created at some point in the branching process. This instruction is always followed by an oriented edge $A \to B$ reminding on the directed edge obtained after application of the branching process.
- One "O" followed by "$X \to Y$", in turn followed by, in brackets, "C" followed by a cycle "X-Y-Z". This instruction tells us to orient the edge XY as $X \to Y$ because otherwise, in the triangle XYZ, we would get a directed cycle. If instead of a triangle we see a longer cycle, then we deal with an application of Lemma 1.5 to a cycle where all but two edges are oriented in one direction, and one of the remaining two edges is oriented in the opposite direction.
- Two "O"s followed by "$X \to Y$", in turn followed by, in brackets, "C" followed by a cycle "X-Y-Z-$\cdots$". This instruction tells us to which cycle Lemma 1.5 can be applied and which edges will become oriented.

Each line ends with either "$S : X - Y - \cdots - Z$" or with "$E : X - Y - \cdots - Z$". In the former case, a shortcut with the shortcutting edge $X \to Z$ is obtained, while in the latter case, all but one edges in the non-clique cycle $X - Y - \cdots - Z$ are oriented in the same direction, while the remaining edge e is not oriented, which is a contradiction since there is no way to orient e without creating a shortcut or a directed cycle ("E" stands for "Error"). In the proofs in this paper, the symbol "E" does not appear, but it appears occasionally when the software [8] is used.

Next, we will explain parts of the first two lines, given in Figure 2, in the proof of non-word-representability of the Shrikhande graph in Figure 4 that does not used Theorem 1.6 (and hence is different from the proof in Sect. 3).

To begin checking the proof, one should arrange 9 undirected copies of the Shrikhande graph, possibly printed on a single page. Begin with orienting edge 12→15 in the first copy of the graph. Branching is necessary at this stage, we orient edge 14→15 in Copy 1 and create partially oriented Copy 2 currently having edges 12→15 and 15→14. We continue with considering Copy 1. Another branching is required, and we orient the edge 12→14 and create partially oriented Copy 3 currently having edges 12→15, 14→15 and 14→12. Looking at the cycle 7-15-14-12 in Copy 1, we can see that Lemma 1.5 can be applied and we can orient edges 7→15 and 12→7.

Continuing following the instructions, we see that the shortcut 7-4-8-16 will eventually be obtained in Copy 1 showing that Copy 1 can now be disregarded as any way to complete its orientation will result in a shortcut being present (so that the orientation would be non-semi-transitive).

We can now consider any of the three partially oriented copies of the graph (Copies 2, 3, 4). Our algorithm suggests considering the latest created copy (Copy 4) that has the most number of oriented edges. MC4 instructs us to do so, and 16→7 reminds us on the correct orientation of the edge (7,16) obtained as the result of the branching process (when Copy 4 was created). Next, we look at the triangle 7-16-15 where we must orient edge 16→15 or else we obtain a directed cycle of length 3. Continuing following the instructions, we see that the shortcut 4-11-3-7 will eventually be obtained in Copy 4 showing that Copy 4 can now be disregarded, and another copy should be considered.

2.2. Algorithm 1

Algorithm 1 sorts edges according to the number of cycles they are in, then branches on an edge belonging to the most number of cycles (whenever branching is required). If there are two or more such edges, the choice on branching is done lexicographically.

2.3. Algorithm 2

Algorithm 2 selects a cycle C with the smallest number of non-oriented edges. The non-oriented edges in C are sorted, similarly to Algorithm 1, based on the number of cycles they are in and branching is done on an edge belonging to the most number of cycles. If there are two or more such edges, the choice on branching is done lexicographically.

2.4. Algorithm 3

Algorithm 3 is similar to Algorithm 2, but it selects a cycle that has the biggest number N of edges oriented in the same direction. Among the cycles with the same N, Algorithm 3 selects a cycle C that has smallest number of non-oriented edges. Then, similarly to Algorithm 2, the non-oriented edges in C are sorted based on the number of cycles they are in and branching is done on an edge belonging to the most number of cycles. If there are two or more such edges, the choice on branching is done lexicographically.

2.5. Ranking of algorithms

Note that Algorithm 1 is static while Algorithms 2 and 3 are dynamic meaning that they require resorting edges whenever an orientation is added to an edge.

To make general statements on the efficiency of algorithms in the sense of the number of lines they produce, or about the time complexity, does not seem to be feasible. However, an indication of the efficiency of the algorithms can be obtained by looking at their performance on small non-word-representable graphs. For example, on the wheel graph W_5 (on 6 vertices), Algorithm 1 produces 10 lines of proof, while Algorithms 2 and 3 produce 7 lines of proof. As the next step, we test the algorithms on all 25 non-word-representable graphs in Figure 3, and the results of the test are presented in Table 1. Recall that in our tests we do not use Theorem 1.6. It turns out that Algorithm 2 is (much) better/not worse in 24 out of 25 cases, and what is somewhat surprising, Algorithm 1 being clearly the worst one, has actually the best performance on Graph 11. On average, Algorithms 2 and 3 are essentially the same. In any case, Algorithm 2 is used in the software [8].

3. The shrikhande graph

The Shrikhande graph is the graph on 16 vertices and 48 edges in Figure 4. Among numerous properties of this graph [10], it is known for being the smallest *distance-regular* graph that is not *distance-transitive* [11], p. 136. In Figure 4, we also present Shrikhande graph's minimal non-word-representable subgraph S labelled in the original way, and in the way used in the proof below (removing any vertex in S results in a word-representable

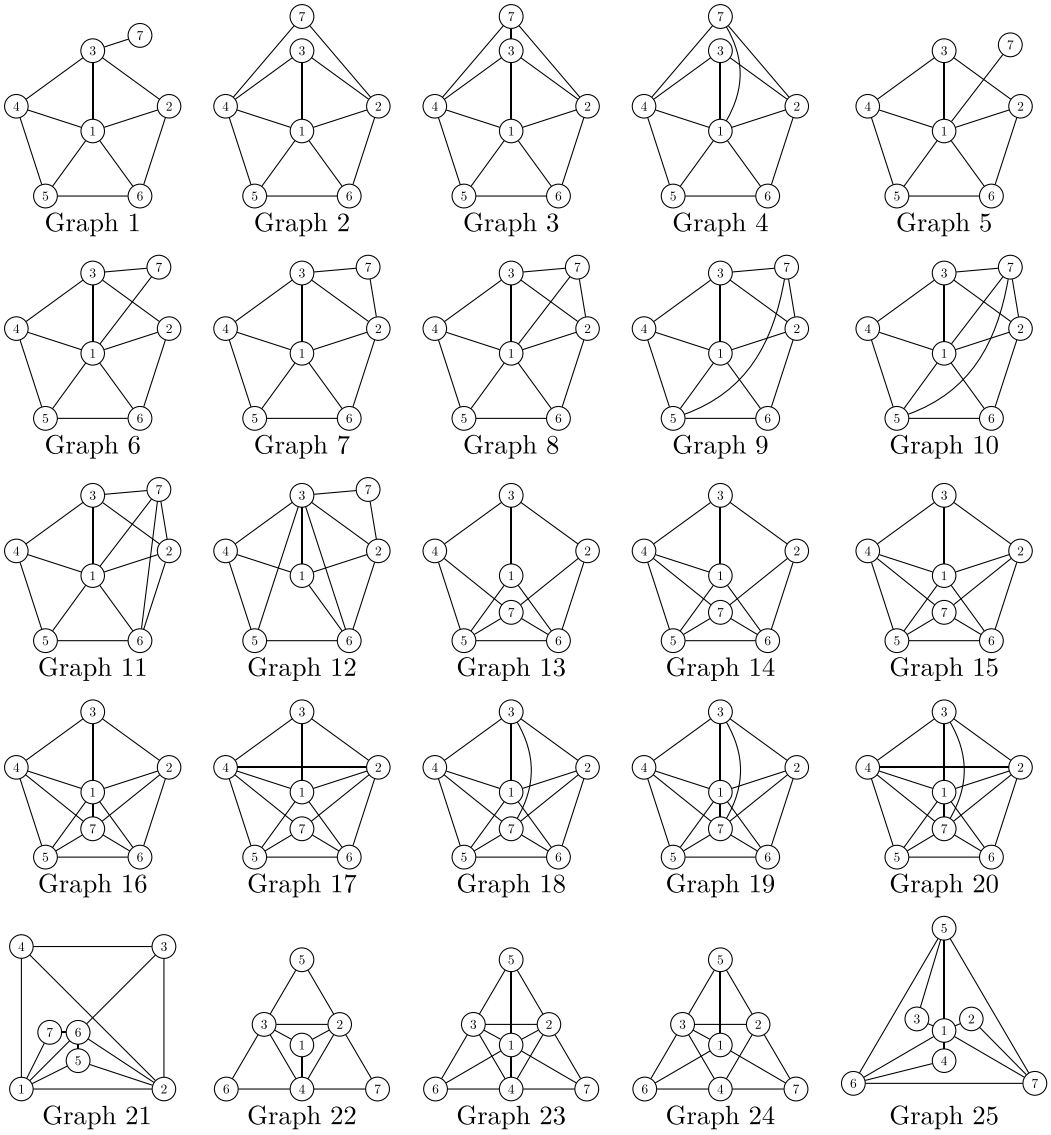

FIGURE 3. All non-word-representable graphs on 7 vertices.

graph). We next give a proof for non-word-representability of S that will give non-word-representability of the Shrikhande graph taking into account the hereditary nature of word-representability. Referring to the rightmost graph in Figure 4, our proof goes as follows. By Theorem 1.6 we can assume that vertex 1 is a source and the rest is given by the following 6-lines proof:

1. B5→6 (Copy 2) O5→4 (C1-6-5-4) O3→4 (C1-5-4-3) O3→2 (C1-4-3-2) B5→7 (Copy 3) O2→7 (C1-5-7-2) B3→8 (Copy 4) O4→8 (C1-4-8-3) O9→8 O5→9 (C4-8-9-5) O6→9 (C1-6-9-5) O7→9 (C5-7-9-6) S:3-2-7-9-8

2. MC4 8→3 O8→4 (C3-8-4) O9→7 O8→9 (C2-7-9-8-3) O5→9 (C4-8-9-5) O6→9 (C1-6-9-5) S:5-6-9-7

3. MC3 7→5 O7→2 (C1-5-7-2) O7→9 O9→6 (C5-7-9-6) O9→5 (C1-6-9-5) O8→4 O9→8 (C4-8-9-5) O8→3 (C1-4-8-3) S:7-9-8-3-2

4. MC2 6→5 O4→5 (C1-6-5-4) O4→3 (C1-5-4-3) O2→3 (C1-4-3-2) B5→7 (Copy 5) O2→7 (C1-5-7-2) O9→7 O6→9 (C5-7-9-6) O5→9 (C1-6-9-5) O4→8 O8→9 (C4-8-9-5) O3→8 (C1-4-8-3) S:2-3-8-9-7

TABLE 1. Ranking of the algorithms.

Graph	Algorithm 2	Algorithm 3	Algorithm 1
1	7 lines	7 lines	10 lines
2	7 lines	7 lines	13 lines
3	10 lines	10 lines	17 lines
4	7 lines	7 lines	13 lines
5	7 lines	7 lines	10 lines
6	7 lines	7 lines	10 lines
7	11 lines	11 lines	11 lines
8	16 lines	20 lines	18 lines
9	9 lines	11 lines	15 lines
10	9 lines	11 lines	15 lines
11	21 lines	21 lines	15 lines
12	8 lines	8 lines	12 lines
13	9 lines	9 lines	17 lines
14	9 lines	9 lines	14 lines
15	9 lines	9 lines	13 lines
16	11 lines	12 lines	14 lines
17	9 lines	9 lines	12 lines
18	7 lines	7 lines	13 lines
19	7 lines	7 lines	16 lines
20	9 lines	11 lines	11 lines
21	10 lines	10 lines	10 lines
22	6 lines	6 lines	19 lines
23	9 lines	11 lines	14 lines
24	7 lines	7 lines	15 lines
25	9 lines	12 lines	11 lines
Average	9.2 lines	9.8 lines	13.5 lines

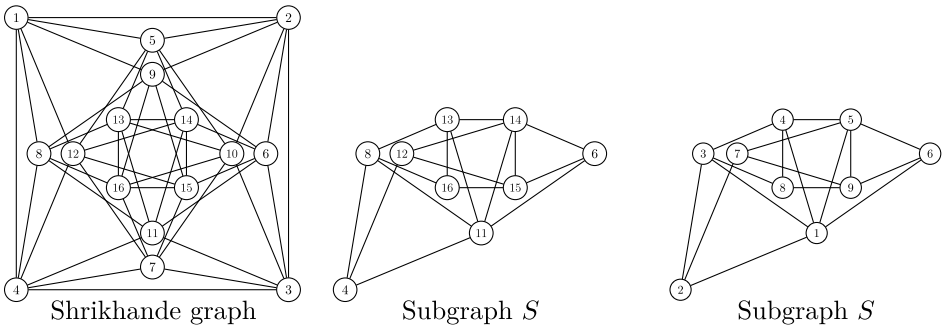

FIGURE 4. The Shrikhande graph and its minimal non-word-representable subgraph S (with the original labelling and the labelling used in our proof).

5. MC5 7→5 O7→2 (C1-5-7-2) B3→8 (Copy 6) O4→8 (C3-8-4) O7→9 O9→8 (C2-7-9-8-3) O9→5 (C4-8-9-5) O9→6 (C1-6-9-5) S:7-9-6-5
6. MC6 8→3 O8→4 (C1-4-8-3) O8→9 O9→5 (C4-8-9-5) O9→6 (C1-6-9-5) O9→7 (C5-7-9-6) S:8-9-7-2-3

4. The clebsch graph

The Clebsch graph, also known as the *Greenwood-Gleason graph* [12], p. 284 and shown in Figure 5, is a strongly regular quintic graph on 16 vertices and 40 edges that enjoys many interesting properties [13]. Figure 5 also gives the subgraph C of the Clebsch graph that is confirmed by software [5, 8] to be minimal

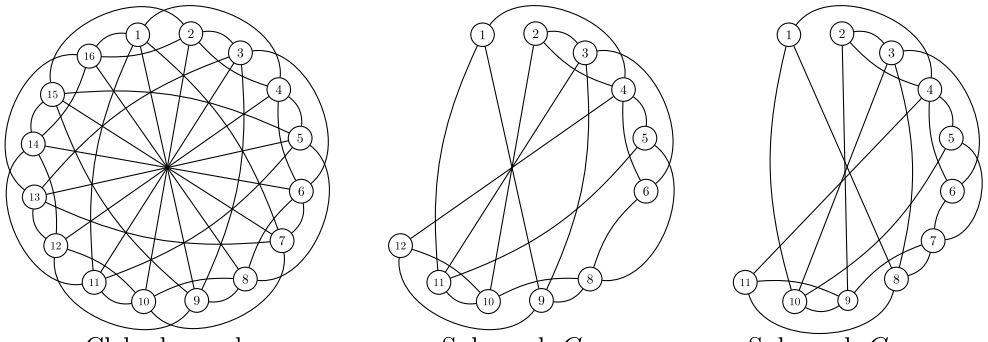

FIGURE 5. The Clebsch graph and its minimal non-word-representable subgraph C (with the original labelling and labelling used in the proof).

non-word-representable. We next give a 10-line proof of non-word-representability of C that confirms non-word-representability of the Clebsch graph. Referring to the rightmost graph in Figure 5, our proof goes as follows. By Theorem 1.6 we can assume that vertex 4 is a source and the rest is given by the following proof:
1. B6→7 (Copy 2) O5→7 (C4-6-7-5) B3→6 (Copy 3) O3→2 (C2-4-6-3) O3→8 O8→7 (C3-8-7-6) B8→11 (Copy 4) O8→1 (C1-8-11-4) O10→1 O3→10 (C1-10-3-8) O10→5 (C1-10-5-4) O10→9 O9→7 (C5-10-9-7) O2→9 (C2-9-10-3) O11→9 (C2-9-11-4) S:8-11-9-7
2. MC4 11→8 O1→8 (C1-8-11-4) O9→7 O11→9 (C7-9-11-8) O2→9 (C2-9-11-4) O10→9 O3→10 (C2-9-10-3) O1→10 (C1-10-3-8) O5→10 (C1-10-5-4) S:5-10-9-7
3. MC3 6→3 O2→3 (C2-4-6-3) B2→9 (Copy 5) O11→9 (C2-9-11-4) B7→9 (Copy 6) O5→10 O10→9 (C5-10-9-7) O1→10 (C1-10-5-4) O10→3 (C2-9-10-3) O8→3 O1→8 (C1-10-3-8) O11→8 (C1-8-11-4) O8→7 (C3-8-7-6) S:11-8-7-9
4. MC6 9→7 O11→8 O8→7 (C7-9-11-8) O1→8 (C1-8-11-4) O8→3 (C3-8-7-6) O1→10 O10→3 (C1-10-3-8) O5→10 (C1-10-5-4) O10→9 (C2-9-10-3) S:5-10-9-7
5. MC5 9→2 O9→10 O10→3 (C2-9-10-3) O9→11 (C2-9-11-4) O5→10 O9→7 (C5-10-9-7) O1→10 (C1-10-5-4) O8→3 O1→8 (C1-10-3-8) O11→8 (C1-8-11-4) O8→7 (C3-8-7-6) S:9-11-8-7
6. MC2 7→6 O7→5 (C4-6-7-5) B3→6 (Copy 7) O3→2 (C2-4-6-3) B2→9 (Copy 8) O10→9 O3→10 (C2-9-10-3) O11→9 (C2-9-11-4) O10→5 O7→9 (C5-10-9-7) O10→1 (C1-10-5-4) O3→8 O8→1 (C1-10-3-8) O8→11 (C1-8-11-4) O7→8 (C3-8-7-6) S:7-8-11-9
7. MC8 9→2 O9→11 (C2-9-11-4) B7→9 (Copy 9) O8→11 O7→8 (C7-9-11-8) O8→1 (C1-8-11-4) O3→8 (C3-8-7-6) O10→1 O3→10 (C1-10-3-8) O10→5 (C1-10-5-4) O9→10 (C2-9-10-3) S:7-9-10-5
8. MC9 9→7 O10→5 O9→10 (C5-10-9-7) O10→1 (C1-10-5-4) O3→10 (C2-9-10-3) O3→8 O8→1 (C1-10-3-8) O8→11 (C1-8-11-4) O7→8 (C3-8-7-6) S:9-7-8-11
9. MC7 6→3 O2→3 (C2-4-6-3) O8→3 O7→8 (C3-8-7-6) B8→11 (Copy 10) O8→1 (C1-8-11-4) O7→9 O9→11 (C7-9-11-8) O9→2 (C2-9-11-4) O9→10 O10→3 (C2-9-10-3) O10→1 (C1-10-3-8) O10→5 (C1-10-5-4) S:7-9-10-5
10. MC10 11→8 O1→8 (C1-8-11-4) O1→10 O10→3 (C1-10-3-8) O5→10 (C1-10-5-4) O9→10 O7→9 (C5-10-9-7) O9→2 (C2-9-10-3) O9→11 (C2-9-11-4) S:7-9-11-8.

5. Concluding remarks

In this paper, we introduce methods to generate automatically proofs of non-word-representability of a graph that can be verified, in a robust way, by a human. We do believe that our work and software [8] will have a dramatic impact to the further development of the theory of word-representable graphs. Indeed, now we can argue non-word-representability for many more (larger) graphs without referring to software, which is a very welcoming news.

As for open problems, we see improving Algorithms 2 and 3 by modifying our approach of selecting edges to branch: for example, we can look for branching edges that increase the number/length of directed paths in the graph, which should increase usability of Lemma 1.5.

Finally, understanding how to estimate, say, the average efficiency of our algorithms, or relevant algorithms yet to be introduced, in terms of certain parameters (number of cycles or alike) is a good theoretical question that seems to be very challenging. The time complexity of our algorithms is also a very interesting and challenging direction of research that was completely ignored by us because our focus was in producing short proofs.

Acknowledgements. The first author is grateful to the London Mathematical Society for supporting work on this project under "Computer Science Small Grants – Scheme 7".

References

[1] S. Kitaev. A comprehensive introduction to the theory of word-representable graphs. Lect. Notes in Comp. Sci. **10396** (2017) 36–67.
[2] S. Kitaev and V. Lozin, Words and Graphs. Springer (2015).
[3] S. Kitaev and A. Pyatkin, On representable graphs. *J. Autom., Lang. Combin.* **13** (2008) 45–54.
[4] M.M. Halldórsson, S. Kitaev, and A. Pyatkin, Semi-transitive orientations and word-representable graphs. *Discr. Appl. Math.* **201** (2016) 164–171.
[5] M. Glen, Software available at personal.strath.ac.uk sergey.kitaev/word-representable-graphs.html
[6] M.C. Golumbic, The complexity of Comparability Graph Recognition and Coloring. *Computing* **18** (1977) 199–208.
[7] H. Zantema, Software REPRNR to compute the representation number of a graph, 2018. Available at https://www.win.tue.nl/~hzantema/reprnr.html
[8] H. Sun, Software available at personal.strath.ac.uk/sergey.kitaev/human-verifiable-proofs.html
[9] S. Kitaev and A. Pyatkin, On semi-transitive orientability of triangle-free graphs. *Discuss. Math. Graph Theory* **43** (2023) 533.
[10] E.W. Weisstein, Shrikhande Graph. From MathWorld – A Wolfram Web Resource. https://mathworld.wolfram.com/ShrikhandeGraph.html
[11] A.E. Brouwer, A.M. Cohen and A. Neumaier, Distance-Regular Graphs. Springer-Verlag, New York (1989).
[12] R.C. Read and R.J. Wilson. An Atlas of Graphs. Oxford University Press, Oxford, England (1998).
[13] E.W. Weisstein, Clebsch Graph. From MathWorld – A Wolfram Web Resource. https://mathworld.wolfram.com/ClebschGraph.html