2–MANIFOLD RECOGNITION IS IN LOGSPACE *

Benjamin A. Burton,† *Murray Elder,*‡ *Arkadius Kalka,*§ *and Stephan Tillmann*¶

Abstract. We prove that the homeomorphism problem for 2–manifolds can be decided in logspace. The proof relies on Reingold's logspace solution to the undirected s,t-connectivity problem in graphs.

1 Introduction

Low space complexity algorithms have recently become a focus of some research in combinatorial group theory, motivated in part by the emergence of non-commutative group-based cryptosystems. Recent work includes [7, 8, 9, 18, 19, 27], with one of the earliest results in this area being Lipton and Zalcstein's logspace algorithm to solve the word problem for linear groups [17]. Logspace is of interest in complexity theory, as it seems to be the smallest amount of space that supports interesting computational phenomena, and suffices for solving many natural computational problems.

In computational low dimensional topology the focus has been on the time complexity of problems, such as deciding if a knot is the unknot, 3–sphere recognition, and the classification of 3–manifolds. In this article we consider a far easier and classical problem, recognition of 2–manifolds, from the point-of-view of space complexity. We prove that given two finite 2–dimensional triangulations, one can decide whether they represent homeomorphic surfaces using space logarithmic in the size of the triangulations. Our result relies on Reingold's remarkable logspace solution to the undirected s,t-connectivity problem in graphs. Letting L be the class of problems that can be decided in deterministic logspace, our result is formally stated as: [1]

Theorem 1. 2–manifold Recognition *is in* L.

Two compact, connected surfaces with (possibly empty) boundary are homeomorphic precisely when they have the same Euler characteristic, the same number of boundary components and they are either both orientable or both non-orientable. This triple of invariants leads to the classification theorem for compact surfaces, which has roots in work of Jordan and Möbius in the 1860s, and Dehn and Heegaard in 1907, with the first rigorous

*Supported by the Australian Research Council (project DP110101104).

† *The University of Queensland*, bab@maths.uq.edu.au

‡ *The University of Newcastle*, murray.elder@newcastle.edu.au (Corresponding author)

§ *Bar Ilan University*, Arkadius.Kalka@rub.de

¶ *The University of Sydney*, tillmann@maths.usyd.edu.au

[1] The definitions of "2–manifold Recognition" and "*deterministic logspace*" are given in §2.

proof due to Brahana [5] in 1921 under the hypothesis that the compact surfaces are triangulated; a modern proof with this hypothesis is due to Conway and presented by Francis and Weeks [10]. The fact that every compact surface has a triangulation, thus completing the classification theorem, was established by Radó [24] in 1925, and a modern proof using the "Kirby torus trick" was recently given by Hatcher [13].

There is a remarkable gap between surfaces and higher dimensional manifolds. Manifolds of dimension three were shown to be triangulated by Moise [22] in 1952, and a discussion of the homeomorphism problem can be found in the recent survey by Aschenbrenner, Friedl and Wilton [3]. Kuperberg [16] recently announced a proof that the homeomorphism problem for closed, oriented 3–manifolds is elementary recursive. Many important algorithms for 3–manifolds have been implemented [6], and many important decision problems, such as unknot recognition [12] and 3–sphere recognition [26], have been shown to be in the complexity class NP.

The next qualitative gap arises between dimensions three and higher. There are compact 4–dimensional manifolds, such as the E_8 manifold discovered by Freedman in 1982, that are not homeomorphic to any simplicial complex. Manolescu [20] has recently announced that there are also such examples in dimensions five and higher. Even if one restricts to compact, simplicial manifolds of dimension four and higher, the homeomorphism problem was shown to be undecidable by Markov [21] in 1958 as a consequence of the unsolvability of the isomorphism problem for finitely presented groups, which is due to Adyan [1, 2] and Rabin[23]. In particular, for the development of algorithms in higher dimensions one needs to restrict to special classes of manifolds, or else be content with heuristic methods.

We remark that while *a priori* an algorithm that uses only logarithmic space has no time bound, it is easy to show that logspace algorithms run in polynomial time (see for example Lemma 4 in [9]). An important open problem is whether the class L of problems that can be decided by a deterministic logspace algorithm is a proper subset of those that can be decided in polynomial time.

We now give an informal description of our logspace algorithms. The starting point is a logspace algorithm which, given a single triangulation as input:

1. checks that the triangulation is a 2–manifold;
2. counts the number of connected components, c;
3. if the input is a connected 2–manifold:
   (a) decides if it is orientable or non-orientable;
   (b) computes the Euler characteristic, χ;
   (c) counts the number of boundary components, b;
4. if there is more than one connected component, the algorithm outputs the following data: $(o_1, \chi_1, b_1), \ldots, (o_c, \chi_c, b_c)$ where $o_i = 0$ if the i-th connected component is orientable and 1 otherwise, χ_i is its Euler characteristic, and b_i is its number of

boundary components. Moreover, the algorithm outputs this data in the following (lexicographic) order:

(a) $o_i < o_{i+1}$, or
(b) $o_i = o_{i+1}$ and $\chi_i < \chi_{i+1}$, or
(c) $o_i = o_{i+1}, \chi_i = \chi_{i+1}$, and $b_i \leq b_{i+1}$.

This output is a complete invariant of the homeomorphism type of the 2–manifold, and so the solution to the homeomorphism problem then follows by running this algorithm simultaneously on two triangulations.

This paper is organised as follows. We give precise definitions of the complexity class and data structures we use in §2. The algorithms to verify that an input triangulation represents a surface and count the number of components are described in §3. The algorithm to compute the complete invariants of a connected, triangulated surface is given in §4, and this algorithm is then applied in §5 to compute the invariants of each connected component of a disconnected surface.

2 Preliminaries

A *deterministic logspace transducer* consists of a finite state control, a read-head, and three tapes:

1. the *input tape* is read-only, and stores the input string;
2. the *work tape* is read-write, but is restricted to using at most $c \log n$ squares, where n is the length of the string on the input tape and c is a fixed constant; and
3. the *output tape* is write-only, and is restricted to writing left to right only. The space used on the output tape is not added to the space bounds.

A transition of the transducer takes as input a letter of the input tape at the position of the read-head, a state of the finite state control, and a letter on the work-tape. On each transition the transducer can modify the work tape, change states, and write at most a fixed constant number of letters to the output tape, moving to the right along the output tape for each letter printed.

Since the position of the read-head of the input tape is an integer between 1 and n (the length of the input), we can store it in binary on the work tape. In addition we can store a finite number of *pointers* to positions on the input tape.

A problem is in deterministic logspace if it can be decided using a deterministic logspace transducer. Since all transducers in this article will be deterministic, we will say logspace for deterministic logspace throughout.

A key property of logspace transducers is that they can be composed together to give new logspace transducers. Formally, let X, Y be finite alphabets, and let X^* denote the set of all finite length strings in the letters of X. We call $f : X^* \to Y^*$ a *logspace computable function* if there is a logspace transducer that on input $w \in X^*$ computes $f(w)$.

Lemma 2 (Lemma 2 in [9]). *If $f, g : X^* \to X^*$ can both be computed in logspace, then their composition $f \circ g : X^* \to X^*$ can also be computed in logspace.*

Proof. Let M_f, M_g be logspace transducers that compute f and g respectively. On input $w \in X^*$, run M_f. Each time M_f calls for the jth input letter, run M_g on w; however, instead of writing the output of M_g to a tape, we add 1 to a counter (in binary) each time M_g would normally write a letter. Continue running M_g until the counter has value $j-1$, at which point we return the next letter that M_g would output back to M_f. □

Finally, a *logspace algorithm* is an algorithm that runs on a logspace transducer. Lemma 2 implies that a logspace algorithm may assume that its input is the output of some other logspace algorithm.

In this article we show that the following decision problem is in logspace:

Problem: 2–MANIFOLD RECOGNITION
Instance: Two 2–dimensional triangulations $\mathcal{T}_1, \mathcal{T}_2$
Question: Do $\mathcal{T}_1$ and $\mathcal{T}_2$ represent homeomorphic 2–manifolds?

For a positive integer n let $[n]$ denote the set $\{1, \ldots, n\}$. A triangulation $\mathcal{T}$ is specified by a list of n triangles, where each triangle $t \in [n]$ has vertices labeled $1, 2, 3$ (which induces an orientation on the triangle), and edges glued according to a table as follows:

	(12)	(23)	(31)
1	a_1	b_1	c_1
2	a_2	b_2	c_2
⋮			
n	a_n	b_n	c_n

with

$$a_t, b_t, c_t \in \{\emptyset\} \cup \{(s, e) \mid s \in [n], e \in \{(12), (21), (23), (32), (31), (13)\}\}.$$

The entry $a_t = (s, e)$ in row t column (12) means that the edge (12) in t is glued to the edge e in triangle s, whereas $a_t = \emptyset$ means that the edge (12) in t is not glued to anything (i.e., it is a boundary edge); likewise for columns (23) and (31).

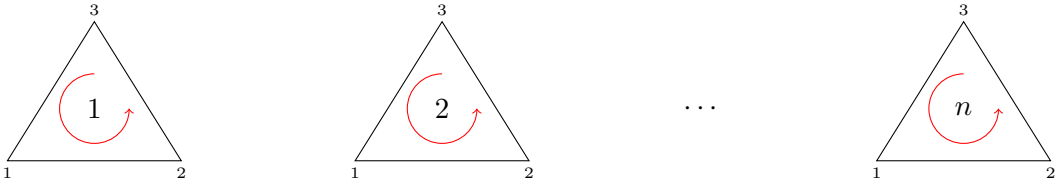

Figure 1: Input triangles.

A triangulation is given to a logspace transducer by writing the string

$$\#\ a_1\ b_1\ c_1\ \#\ a_2\ b_2\ c_2\ \#\ \ldots\ \#\ a_n\ b_n\ c_n$$

on the input tape using the alphabet $\{\#, \emptyset, 0, 1, (12), (23), (31), (21), (32), (13)\}$, where a_i, b_i, c_i are written as either $\emptyset$ or a binary number followed by $(12), (23)$ or (31).

Example 3. Figure 2 illustrates a triangulation of a punctured Klein bottle, and Table 1 shows the corresponding table of edge gluings. For this triangulation, the input tape of our logspace transducer would read as follows:

$$\#\ 10\ (13)\ 11\ (12)\ 11\ (32)\ \#\ 11\ (13)\ \emptyset\ 1\ (21)\ \#\ 1\ (23)\ 1\ (13)\ 10\ (21)$$

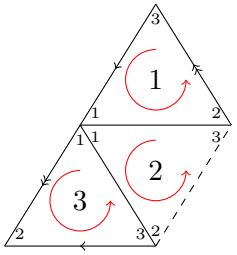

Figure 2: Triangles for Example 3 with edges identified. Dashed line is the boundary.

Table 1: Input table for Example 3.

	(12)	(23)	(31)
1	2, (13)	3, (12)	3, (32)
2	3, (13)	$\emptyset$	1, (21)
3	1, (23)	1, (13)	2, (21)

A triangulation of n triangles has input size $N \in O(n \log n)$. We will prove that the data required to identify the homeomorphism type of the input can be output by a transducer using $O(\log N)$ squares of the work tape. It follows that for a triangulation with n triangles, the homeomorphism type can be computed using $O(\log N) = O(\log n + \log \log n) = O(\log n)$ space.

It can easily be checked in logspace that the input is written in the required form – the number n of # symbols can be computed and written in binary on the work tape by scanning # symbols, then one can check that each binary number on the tape has value between 1 and n. So we may assume the input is correctly specified. However, we do not assume that the gluing instructions are consistent or give a manifold. For example, we may have $\emptyset$ in row t column (12) but $(t, (12))$ may appear as a different entry in the table, which would be inconsistent. The algorithm we describe will check this.

Note that the set $\epsilon = \{(12), (21), (23), (32), (31), (13)\}$ comes natually equipped with an involution $\bar{\ } : \epsilon \to \epsilon$ given by $\overline{(ij)} = (ji)$.

When describing our algorithms we will refer to *row* t *column* e of the input tape, which means the entry in row t column e of the gluing table. This can be located in logspace by scaning the input tape from left to right counting the number of # symbols.

Throughout this paper we make use of a deterministic logspace algorithm due to Reingold [25] which takes input (V, E, s, t), where (V, E) is an undirected graph, $s, t \in V$, and returns *Yes* if there is an edge path from s to t, and *No* otherwise. We call this algorithm REIN.

We present the algorithms in this paper using pseudocode. Note that for-loops in the pseudocode are straightforward to implement a logspace transducer, using a binary number on the work tape for each loop. All of our algorithms make implicit use of the fact that logspace functions are closed under composition (Lemma 2).

3 Initial tests

3.1 Counting components of a graph

We begin with a simple tool that we call upon repeatedly in this paper: a logspace algorithm to compute the number of connected components of an undirected graph. This algorithm follows immediately from REIN. It operates as follows; see Algorithm 1 for the pseudocode.

Assume the graph is written on the input tape with vertices $[n]$ and edges given as a list $E \subseteq [n] \times [n]$. Initialise a counter $c = 1$ for the component containing vertex 1. The key idea is to iterate through the remaining vertices, and to increment c each time we encounter the lowest-numbered vertex of some connected component.

More precisely: Algorithm 1 runs through each vertex $t > 1$, calling REIN to test whether t is connected to any vertex $s < t$. If it is, we leave c unchanged and move to the next vertex. If it is not, we increment c and move to the next vertex.

Algorithm 1: Count connected components.

Input: Undirected graph $([n], E)$.
Output: Number of connected components, c.
Write a counter $c = 1$ (in binary) to the work tape;
for $t = 2$ `to` n **do**
  Set $b =$ `false`;
  for $s = 1$ `to` $t - 1$ **do**
    Run REIN on $([n], E, s, t)$. If REIN returns `true`, set $b =$ `true`;
  end
  If $b =$ `false`, increment c by 1;
end
return c;

3.2 Checking the input is a surface

Our first task is to test the validity of the input. Algorithm 2 decides whether the input represents a surface by iterating through (t, e) for each $t \in [n], e \in \{(12), (23), (31)\}$ and checking:

1. that the entry (t, e) or $(t, \overline{e})$ appears at most once in the table;
2. if row t column e of the table is $\emptyset$, that neither (t, e) nor $(t, \overline{e})$ appear in the table;
3. if row t column e of the table is (s, f) for $f \in \{(12), (23), (31)\}$, that row s column f is (t, e);
4. if row t column e of the table is (s, f) for $f \in \{(21), (32), (13)\}$, that row s column $\overline{f}$ is $(t, \overline{e})$;
5. that row t column e of the table is not (t, e) or $(t, \overline{e})$.

Algorithm 2: Check surface.

Input: Triangulation data on input tape.
Output: *Yes* if input is a surface, *No* otherwise.
for $t \in [n]$ **do**
  for $e \in \{(12), (23), (31)\}$ **do**
    Write a counter $c = 0$ to the work tape;
    Scan the tape from left to right reading each entry (s, f);
      If $(s, f) \in \{(t, e), (t, \overline{e})\}$, increment c by 1;
      If $c > 1$, output *No* and stop;
    Read the entry $y = (s, f)$ in row t column e;
    If $y = \emptyset$ and $c \neq 0$, output *No* and stop;
    If $f \in \{(12), (23), (31)\}$, read the entry z in row s column f. If $z \neq (t, e)$, output *No* and stop;
    If $f \in \{(21), (32), (13)\}$, read the entry z in row s column $\overline{f}$. If $z \neq (t, \overline{e})$, output *No* and stop;
    If $y \in \{(t, e), (t, \overline{e})\}$, output *No* and stop;
  end
end
If *No* not printed, return *Yes*;

3.3 Counting the number of connected components

Next, we count the number of connected components of the input surface. To do this we construct the *face-dual graph* of the surface, which is an undirected graph whose vertices correspond to the triangles of the surface, and whose edges correspond to triangle gluings. More precisely, the vertices of the face-dual graph are $[n]$, and the edges of the face-dual graph are pairs (s, t) for which (t, e) is in row s of the table for some $e \in \{(12), (23), (31), (21), (32), (13)\}$ (and therefore (s, e) appears in row t for some e also).

Note that the face-dual graph as defined here is a simple graph: it does not include loops or parallel edges (which do not affect connectivity). As an example, the face-dual graph for the punctured Klein bottle of Example 3 has vertices $\{1, 2, 3\}$, and edges $\{1, 2\}$, $\{2, 3\}$, $\{1, 3\}$.

Algorithm 3 takes triangulation data as input and outputs the face-dual graph as an undirected graph $([n], E')$, using a simple scan through the table.

Algorithm 3: Construct the face-dual graph.

Input: Triangulation data on input tape.
Output: Face-dual graph $([n], E')$.
Scan the tape counting # symbols in binary on the work tape, then store this number n and write $[n]$ to the output tape;
for $t \in [n]$ **do**
  for $s = t+1, \ldots, n$ **do**
    Check whether (t, e) is in row s of the table for some $e \in \{(12), (21), (23), (32), (31), (13)\}$. If true, write (s, t) to the output tape;
  end
end

To count components of the input triangulation in logspace, we use Algorithm 3 to construct the face-dual graph, and we count components using Algorithm 1 with this face-dual graph as input. Call the composition of these algorithms *Algorithm A*.

4 Algorithm for one connected component

In this section we assume the input surface is connected and compute its homeomorphism type. In the next section we extend this to surfaces with more than one connected component.

4.1 Orientability

We can determine whether or not a manifold is orientable by taking its double cover, which is connected if the manifold is non-orientable, and which has two components if the manifold is orientable.

Recall that each triangle has a fixed orientation determined by the corner labels $1, 2, 3$. The double cover is given by a set of $2n$ triangles $\{t, t' \mid t \in [n]\}$ with a gluing table constructed from the original table as follows:

1. the rows of the table are $\{t, t' \mid t \in [n]\}$ and the columns are $\{(12), (23), (31)\}$;
2. if row t column e of the original table contains $\emptyset$, then write $\emptyset$ in rows t, t' column e of the new table;
3. if row t column e of the original table contains (s, f) with $f \in \{(12), (23), (31)\}$, then write (s', f) in row t column e and (s, f) in row t' column e of the new table;
4. if row t column e of the original table contains (s, f) with $f \in \{(21), (32), (13)\}$, then write (s, f) in row t column e and (s', f) in row t' column e of the new table.

Table 2 illustrates the double cover of the punctured Klein bottle from Example 3.

Table 2: Double cover data for Example 3.

	(12)	(23)	(31)
1	$2, (13)$	$3', (12)$	$3, (32)$
2	$3, (13)$	$\emptyset$	$1, (21)$
3	$1', (23)$	$1, (13)$	$2, (21)$
$1'$	$2', (13)$	$3, (12)$	$3', (32)$
$2'$	$3', (13)$	$\emptyset$	$1', (21)$
$3'$	$1, (23)$	$1', (13)$	$2', (21)$

We can easily describe a logspace algorithm to produce this double cover gluing table from the original input; see Algorithm 4 for the details. To test the orientability of the original input triangulation, we now compose this with Algorithm A from Section 3.3: Algorithm 4 constructs the double cover, and Algorithm A tests whether the double cover has one or two components.

4.2 Euler characteristic

For a triangulation of a surface S we have $\chi(S) = |V| - |E| + n$, where V and E are the vertex set and edge set of S respectively, and where n is the number of triangles.

Let x be the number of edges of triangles that are not glued to any other edge, i.e., the number of $\emptyset$ symbols on the input tape. Then the number of edges is $|E| = (3n - x)/2 + x = (3n + x)/2$, since the remaining $3n - x$ triangle edges are identified in pairs. We can compute n and x, and hence $|E|$, in logspace by counting the number of # and $\emptyset$ symbols on the input tape.[2]

It remains to compute $|V|$. We do this by tracking the identifications of individual vertices of triangles. For this we construct an undirected graph K, which we call the *vertex identification graph*, as follows. The graph K has vertex set

$$W = \{w_{t,1}, w_{t,2}, w_{t,3} \mid t \in [n]\},$$

where $w_{t,i}$ represents vertex i of triangle t. Note that the graph K has $|W| = 3n$ vertices overall. In the punctured Klein bottle from Example 3, these vertices are

$$W = \{w_{1,1},\ w_{1,2},\ w_{1,3},\ w_{2,1},\ w_{2,2},\ w_{2,3},\ w_{3,1},\ w_{3,2},\ w_{3,3}\}.$$

The edge set of the graph K is $F = \{(w_{t,i}, w_{s,j}) \mid w_{t,i}, w_{s,j}$ are identified directly$\}$, where by "identified directly" we mean that some edge triangle t is glued to some edge of triangle s in a way that maps vertex i of triangle t to vertex j of triangle s. For the

[2]Note that addition and division by two are both logspace computable.

Algorithm 4: Construct the double cover of a triangulation.

Input: Triangulation data on input tape.
Output: Triangulation data for the double cover, with triangles ordered as $1, \ldots, n, 1', \ldots, n'$.

```
for t ∈ [n] do
    Write # to the output tape;
    for e ∈ {(12), (23), (31)} do
        If row t column e of the input table is ∅, write ∅ to the output tape;
        If row t column e of the input table is (s, f) with f ∈ {(12), (23), (31)},
        write s′ f to the output tape;
        If row t column e of the input table is (s, f) with f ∈ {(21), (32), (13)},
        write s f to the output tape;
    end
end
for t ∈ [n] do
    Write # to the output tape;
    for e ∈ {(12), (23), (31)} do
        If row t column e of the input table is ∅, write ∅ to the output tape;
        If row t column e of the input table is (s, f) with f ∈ {(12), (23), (31)},
        write s f to the output tape;
        If row t column e of the input table is (s, f) with f ∈ {(21), (32), (13)},
        write s′ f to the output tape;
    end
end
```

punctured Klein bottle example, this edge set is

$$\begin{aligned} F = \{&\{w_{1,1}, w_{2,1}\},\ \{w_{1,2}, w_{2,3}\},\\ &\{w_{2,1}, w_{3,1}\},\ \{w_{2,2}, w_{3,3}\},\\ &\{w_{1,1}, w_{3,2}\},\ \{w_{1,3}, w_{3,3}\},\\ &\{w_{1,2}, w_{3,1}\},\ \{w_{1,3}, w_{3,2}\}\}. \end{aligned}$$

Algorithm 5 constructs this graph in logspace, simply by walking through the gluing table for the input triangulation. Note that, as it is presented here, Algorithm 5 writes each edge to the output tape twice; if desired this can easily be avoided using a lexicographical test.

Two vertices of K are in the same connected component of K if and only if the corresponding triangle vertices are identified in the input triangulation, and so $|V|$ is the number of connected components of the graph K. Algorithm 1 with input $K = (W, F)$ computes this number, and from this we can now compute the Euler characteristic $\chi(S)$ of the input surface.

Algorithm 5: Construct the vertex identification graph $K = (W, F)$.

Input: Triangulation data on input tape.
Output: The graph $K = (W, F)$.
for $t \in [n]$ **do**
  for $i \in \{1, 2, 3\}$ **do**
    Write $w_{t,i}$ to the output tape;
  end
end
for $t \in [n]$ **do**
  for $e = (ij) \in \{(12), (23), (31)\}$ **do**
    Read the entry $y = (s, (pq))$ in row t column e;
    If $y \neq \emptyset$, write $(w_{t,i}, w_{s,p})$ and $(w_{t,j}, w_{s,q})$ to the output tape;
  end
end

4.3 Number of boundary components

To count the number of boundary components in our surface, we build another auxiliary graph K', which we call the *boundary identification graph*. This begins with the vertex identification graph K, and introduces additional edges that join together different paths in K that correspond to vertices on the same boundary component of the surface.

More precisely, this graph K' has vertex set $W' = W$ as described above. The edge set of K' is

$$F' = F \cup \{(w_{t,i}, w_{t,j}) \mid \text{edge } (ij) \text{ of triangle } t \text{ is not glued to anything}\}.$$

For the punctured Klein bottle example, this edge set is

$$\begin{aligned} F' = \{&\{w_{1,1}, w_{2,1}\},\ \{w_{1,2}, w_{2,3}\},\\ &\{w_{2,1}, w_{3,1}\},\ \{w_{2,2}, w_{3,3}\},\\ &\{w_{1,1}, w_{3,2}\},\ \{w_{1,3}, w_{3,3}\},\\ &\{w_{1,2}, w_{3,1}\},\ \{w_{1,3}, w_{3,2}\},\\ &\{w_{2,2}, w_{2,3}\}\}. \end{aligned}$$

Algorithm 6 shows how the graph K' is constructed.

We can analyse the structure of the vertex identification graph K and the boundary identification graph K':

- K is a disjoint union of cycles and paths, with one cycle for each internal vertex of the surface, and one path for each boundary vertex of the surface.
- K' is a disjoint union of cycles, with one cycle for each internal vertex of the surface, and one cycle for each boundary component of the surface.

Algorithm 6: Construct the boundary identification graph $K' = (W', F')$.

Input: Triangulation data on input tape.
Output: The graph $K' = (W', F')$.
for $t \in [n]$ **do**
  for $i \in \{1, 2, 3\}$ **do**
    Write $w_{t,i}$ to the output tape;
  end
end
for $t \in [n]$ **do**
  for $e = (ij) \in \{(12), (23), (31)\}$ **do**
    Read the entry $y = (s, (pq))$ in row t column e;
    If $y \neq \emptyset$, write $(w_{t,i}, w_{s,p})$ and $(w_{t,j}, w_{s,q})$ to the output tape;
    If $y = \emptyset$, write $(w_{t,i}, w_{t,j})$ to the output tape;
  end
end

Moreover, the number of boundary vertices of the surface is equal to the number of boundary edges; that is, the number of $\emptyset$ symbols in the gluing table for the input triangulation. Therefore counting boundary components becomes a simple matter of counting boundary edges and counting components of K and K'. Algorithm 7 gives the details.

Algorithm 7: Count the number of boundary components.

Input: Triangulation data on input tape.
Output: Number of boundary components, b.
Compose Algorithms 5 and 1 to find k, the number of connected components of K;
Compose Algorithms 6 and 1 to find k', the number of connected components of K';
Count the number of $\emptyset$ symbols on the input tape, and store this as the integer x;
Write $b = k' - k + x$ to the output tape;

We summarise this section in the following statement.

Proposition 4. *There is a logspace algorithm, Algorithm B, which given a triangulation of a connected surface S as input, computes (o, χ, b) where $o = 0$ if S is orientable and 1 if nonorientable, $\chi = \chi(S)$ is the Euler characteristic of S, and b is the number of boundary components of S.*

5 More than one connected component

We now assume the output to Algorithm 1 is $c > 1$. We will compute the following data:

$$(o_1, b_1, \chi_1), \ldots, (o_c, b_c, \chi_c),$$

where $o_i = 0$ if the i-th connected component is orientable and 1 otherwise, b_i is its number of boundary components, and χ_i is its Euler characteristic. Moreover, we output this data in lexicographic order:

- $o_i < o_{i+1}$, or
- $o_i = o_{i+1}$ and $\chi_i < \chi_{i+1}$, or
- $o_i = o_{i+1}, \chi_i = \chi_{i+1}$, and $b_i \leq b_{i+1}$.

The pseudocode is shown below, and followed by a discussion of the meta-algorithm.

Algorithm 8: Outputs the triangulation data for the i-th connected component only.

Input: Triangulation data on input tape with n triangles; integer $i \leq n$
Output: Triangulation data for the connected surface which is the i-th connected component of the input surface.

```
Initialise counters t = 1 and c = 1 (in binary) on the work tape;
while c < i do
    Increment t by 1;
    Set b = false;
    for s = 1 to t − 1 do
        If REIN([n], E, s, t) returns true, set b = true;
    end
    If b = false, increment c by 1;
end
for s = t to n do
    if REIN([n], E, s, t) returns true then
        Write # to the output tape;
        for e ∈ {(12), (23), (31)} do
            Read the entry y = (u, f) in row s column e;
            if y = ∅ then
                Write ∅ to the output tape;
            end
            else
                Initialise counter u′ = 0;
                for x = t to s do
                    If REIN([n], E, x, t) returns true then increment u′ by 1;
                end
                Write (u′, f) to the output tape;
            end
        end
    end
end
```

Here is the meta-algorithm. Assume we have checked that the input is a surface, computed the number of triangles n and counted the number of connected components $c > 1$. Then using the algorithms described above, we compute the number of boundary components b and Euler characteristic $\chi(S)$ for the entire (disconnected) surface S. Note that the number of boundary components (and connected components) is at most n.

The Euler characteristic for a connected surface is at most 2. We compute a lower bound on χ for each connected component as follows. If $S = \bigcup_{i=1}^{c} S_i$ are the connected components, we have $\chi(S) = \sum_{i=1}^{c} \chi(S_i)$ so for one component we have $\chi(S_{i_0}) = \chi(S) - \sum_{i \neq i_0} \chi(S_i)$. This is minimised when the negative term on the left is maximised, and since the maximum Euler characteristic for any connected surface is 2, we have $\chi(S_{i_0}) \geq \chi(S) - 2(c-1)$.

We then iterate through all possible triples (o, x, χ) for $o \in \{0, 1\}, 0 \leq x \leq b, \chi(S) - 2(c-1) \leq \chi \leq 2$ (which is a finite list), in the order given above. For each such triple (o, χ, x), we run through each connected component of S and compute (o_i, χ_i, b_i), and if $(o_i, \chi_i, b_i) = (o, \chi, x)$ then we write this triple to the output.

This meta-algorithm repeatedly uses Algorithms A and B above. It also requires a logspace algorithm for extracting the i-th connected components of the input surface. We present such a procedure in Algorithm 8 which takes as input an integer i and triangulation data for a surface with possibly many connected components, and outputs the triangulation data for only the i-th connected component.

Algorithm 8 is a straightforward extension of Algorithm 1 (which just counts connected components). We draw attention to the final loop over the counter x, which is used to reindex the triangles on the output tape so that they are numbered consecutively as $1, 2, \ldots, k$, where k is the number of triangles in the i-th component.

6 Concluding remarks

The main result of this paper is in fact stronger than presented in the statement of Theorem 1 (that 2–manifold recognition is in L): the proof gives a logspace algorithm for the *function problem* to compute the homeomorphism type (essentially a "normal form") of a given 2–manifold. This is in contrast to problems on some groups, such as braid groups with at least four strands, where there is a logspace solution to the word problem [4, 14, 17] but no known logspace algorithm for computing a normal form.

It was stated in the introduction that unknot recognition and 3–sphere recognition are both in NP. At the time of writing it is known that these two problems are in co-NP modulo the Generalised Riemann Hypothesis due to work of Kuperberg [15] and Hass and Kuperberg [11], but it is unknown whether or not they are in P. The current algorithmic frameworks seem unlikely to support logspace algorithms for these problems, and a significant breakthrough in this direction would likely require a completely new approach.

References

[1] Sergeĭ I. Adyan. Finitely presented groups and algorithms. *Dokl. Akad. Nauk SSSR (N.S.)*, 117:9–12, 1957.

[2] Sergeĭ I. Adyan. Unsolvability of some algorithmic problems in the theory of groups. *Trudy Moskov. Mat. Obšč.*, 6:231–298, 1957.

[3] Matthias Aschenbrenner, Stefan Friedl, and Henry Wilton. Decision problems for 3-manifolds and their fundamental groups, 2014. http://arxiv.org/abs/1405.6274.

[4] Stephen J. Bigelow. Braid groups are linear. *J. Amer. Math. Soc.*, 14(2):471–486 (electronic), 2001.

[5] Henry R. Brahana. Systems of circuits on two-dimensional manifolds. *Ann. of Math. (2)*, 23(2):144–168, 1921.

[6] Benjamin A. Burton, Ryan Budney, William Pettersson, et al. Regina: Software for 3-manifold topology and normal surface theory. http://regina.sourceforge.net/, 1999–2015.

[7] Volker Diekert and Jonathan Kausch. Logspace computations in graph products. In *ISSAC 2014—Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 138–145. ACM, New York, 2014.

[8] Volker Diekert, Jonathan Kausch, and Markus Lohrey. Logspace computations in Coxeter groups and graph groups. In *Computational and combinatorial group theory and cryptography*, volume 582 of *Contemp. Math.*, pages 77–94. Amer. Math. Soc., Providence, RI, 2012.

[9] Murray Elder, Gillian Elston, and Gretchen Ostheimer. On groups that have normal forms computable in logspace. *J. Algebra*, 381:260–281, 2013.

[10] George K. Francis and Jeffrey R. Weeks. Conway's ZIP proof. *Amer. Math. Monthly*, 106(5):393–399, 1999.

[11] Joel Hass and Greg Kuperberg. The complexity of recognizing the 3-sphere (non-refereed extended abstract). In Gert-Martin Greuel, editor, *Triangulations*, volume 9 of *Oberwolfach Reports*, pages 1425–1426. EMS Publishing House, Zurich, Switzerland, 2012.

[12] Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *J. ACM*, 46(2):185–211, 1999.

[13] Allen Hatcher. The Kirby torus trick for surfaces, 2013. http://arxiv.org/abs/1312.3518.

[14] Daan Krammer. Braid groups are linear. *Ann. of Math. (2)*, 155(1):131–156, 2002.

[15] Greg Kuperberg. Knottedness is in NP, modulo GRH. *Adv. Math.*, 256:493–506, 2014.

[16] Greg Kuperberg. Algorithmic homeomorphism of 3-manifolds as a corollary of geometrization, 2015. http://arxiv.org/abs/1508.06720.

[17] Richard J. Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *J. Assoc. Comput. Mach.*, 24(3):522–526, 1977.

[18] Markus Lohrey and Saul Schleimer. Efficient computation in groups via compression. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*, volume 4649 of *Lecture Notes in Computer Science*, pages 249–258. Springer, 2007.

[19] Jeremy Macdonald, Alexei Miasnikov, Andrey Nikolaev, and Svetla Vassileva. Logspace and compressed-word computations in nilpotent groups, 2015. http://arxiv.org/abs/1503.03888.

[20] Ciprian Manolescu. Pin(2)-Equivariant Seiberg-Witten Floer homology and the Triangulation Conjecture. *J. Amer. Math. Soc.*, 29(1):147–176, 2016.

[21] A. Markov. The insolubility of the problem of homeomorphy. *Dokl. Akad. Nauk SSSR*, 121:218–220, 1958.

[22] Edwin E. Moise. Affine structures in 3-manifolds. V. The triangulation theorem and Hauptvermutung. *Ann. of Math. (2)*, 56:96–114, 1952.

[23] Michael O. Rabin. Recursive unsolvability of group theoretic problems. *Ann. of Math. (2)*, 67:172–194, 1958.

[24] Tibor Radó. Über den Begriff der Riemannschen Fläche. *Acta Scientarum Mathematicarum Universitatis Szegediensis*, 2:101–121, (1924–26).

[25] Omer Reingold. Undirected ST-connectivity in log-space. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 376–385. ACM, New York, 2005.

[26] Saul Schleimer. Sphere recognition lies in NP. In *Low-dimensional and symplectic topology*, volume 82 of *Proc. Sympos. Pure Math.*, pages 183–213. Amer. Math. Soc., Providence, RI, 2011.

[27] Svetla Vassileva. *Space and time complexity of algorithmic problems in groups*. PhD thesis, McGill University, 2013.