A tree traversal algorithm for decision problems in knot theory and 3-manifold topology*

Benjamin A. Burton and Melih Ozlen

Abstract

In low-dimensional topology, many important decision algorithms are based on normal surface enumeration, which is a form of vertex enumeration over a high-dimensional and highly degenerate polytope. Because this enumeration is subject to extra combinatorial constraints, the only practical algorithms to date have been variants of the classical double description method. In this paper we present the first practical normal surface enumeration algorithm that breaks out of the double description paradigm. This new algorithm is based on a tree traversal with feasibility and domination tests, and it enjoys a number of advantages over the double description method: incremental output, significantly lower time and space complexity, and a natural suitability for parallelisation.

ACM classes F.2.2; G.1.6; G.4

Keywords Normal surfaces, vertex enumeration, tree traversal, backtracking, linear programming

1 Introduction

Much research in low-dimensional topology is driven by decision problems. Examples from knot theory include the *unknot recognition* and *knot equivalence* problems, and analogous examples from 3-manifold topology include *3-sphere recognition* and the *homeomorphism problem*.

Significant progress has been made on these problems over the past 50 years. For instance, Haken introduced an algorithm for recognising the unknot in the 1960s [17], and Rubinstein presented the first 3-sphere recognition algorithm in the 1990s [26]. Since Perelman's proof of the geometrisation conjecture [22], we now have a complete (but extremely complex) algorithm for the homeomorphism problem, pieced together though a diverse array of techniques [20, 24].

A key problem remains with many 3-dimensional decision algorithms (including all of those mentioned above): the best known algorithms run in exponential time (and for some problems, worse), where the input size is measured by the number of crossings in a knot diagram, or the number of tetrahedra in a 3-manifold triangulation. This severely limits the practicality of such algorithms. For example, it took 30 years to resolve Thurston's well-known conjecture regarding the Weber-Seifert space [5]—an algorithmic solution was

*This is the shorter conference version, as presented at the 27th Annual Symposium on Computational Geometry. See `arXiv:1010.6200` for the full version of this paper.

found in 1984, but it took 25 more years of development before the algorithm became usable [12, 21]. The input size was $n = 23$.

The most successful machinery for 3-dimensional decision problems has been *normal surface theory*. Normal surfaces were introduced by Kneser [23] and later developed by Haken for algorithmic use [17, 18], and they play a key role in all of the decision algorithms mentioned above. In essence, they allow us to convert difficult "topological searches" into a "combinatorial search" in which we enumerate vertices of a high-dimensional polytope called the *projective solution space*.[1] In Section 2 we outline the essential properties of the projective solution space; see [19] for a broader introduction to normal surface theory in a topological setting.

For many topological decision problems, the computational bottleneck is precisely this vertex enumeration. Vertex enumeration over a polytope is a well-studied problem, and several families of algorithms appear in the literature. These include *backtracking algorithms* [4, 16], pivot-based *reverse search algorithms* [1, 3, 15], and the inductive *double description method* [25]. All have worst-case running times that are super-polynomial in both the input *and* the output size, and it is not yet known whether a polynomial time algorithm (with respect to the combined input and output size) exists.[2] For topological problems, the corresponding running times are exponential in the "topological input size"; that is, the number of crossings or tetrahedra.

Normal surface theory poses particular challenges for vertex enumeration. The projective solution space is a highly degenerate (or non-simple) polytope, and exact rational arithmetic is required for topological reasons. Moreover, we are not interested in *all* of the vertices of the projective solution space, but just those that satisfy a powerful set of conditions called the *quadrilateral constraints*. These are extra combinatorial constraints that eliminate all but a small fraction of the polytope, and an effective enumeration algorithm should be able to enforce them as it goes, not simply use them as a post-processing output filter.

All well-known implementations of normal surface enumeration [6, 14] are based on the double description method: degeneracy and exact arithmetic do not pose any difficulties, and it is simple to embed the quadrilateral constraints directly into the algorithm [10]. Section 2 includes a discussion of why backtracking and reverse search algorithms have not been used to date. Nevertheless, the double description method does suffer from significant drawbacks:

- It can suffer from a *combinatorial explosion*: even if both the input and output sizes are small, it can build extremely complex intermediate polytopes where the number of vertices is super-polynomial in both the input and output sizes. This leads to an unwelcome explosion in both time and memory usage, which is frustrating for normal surface enumeration where the output size is often small (though still exponential in general) [8].

- The double description method is difficult to parallelise, due to its inductive nature. There have been recent efforts [27], though they rely on a shared memory model, and for some polytopes the speed-up factor plateaus when the number of processors grows large.

- Because of its inductive nature, the double description method typically does not output *any* vertices until the entire algorithm is complete. This effectively removes the possibility of early termination, which is desirable for many topological algorithms.

[1] For some problems, instead of enumerating vertices we must enumerate a Hilbert basis for a polyhedral cone.

[2] Efficient algorithms do exist for certain classes of polytopes, in particular for non-degenerate polytopes [3].

In this paper we present a new algorithm for normal surface enumeration. This algorithm belongs to the backtracking family, and is described in detail in Section 3. Globally, the algorithm is structured as a tree traversal, where the underlying tree is a decision tree indicating which coordinates are zero and which are non-zero. Locally, we move through the tree by performing incremental feasibility tests using the dual simplex method (though interior point methods could equally well be used).

Fukuda et al. [16] discuss an impediment to backtracking algorithms, which is the need to solve the NP-complete *restricted vertex problem*. We circumvent this by ordering the tree traversal in a careful way and introducing *domination tests* over the set of previously-found vertices. This introduces super-polynomial time and space costs, but for normal surface enumeration this trade-off is typically not severe (we quantify this within Sections 4 and 5).

Our algorithm is well-suited to normal surface enumeration, since it integrates the quadrilateral constraints seamlessly into the tree structure. More significantly, it is the first normal surface enumeration algorithm to break out of the double description paradigm. Furthermore, it enjoys a number of advantages over previous algorithms:

- The theoretical worst-case bounds on both time and space complexity are significantly better for the new algorithm. In particular, the space complexity is a small polynomial in the combined input and output size. See Section 4 for details.
- The new algorithm lends itself well to parallelisation, including both shared memory models and distributed processing, with minimal inter-communication and synchronisation.
- The new algorithm produces incremental output, which makes it well-suited for early termination and further parallelisation.
- The tree traversal supports a natural sense of "progress tracking", so that users can gain a rough sense for how far they are through the enumeration procedure.

In Section 5 we measure the performance of the new algorithm against the prior state of the art, using a rich test suite with hundreds of problems that span a wide range of difficulties. For simple problems we find the new algorithm slower, but as the problems become more difficult the new algorithm quickly outperforms the old, often running orders of magnitude faster. We conclude that for difficult problems this new tree traversal algorithm is superior, owing to both its stronger practical performance and its desirable theoretical attributes as outlined above.

2 Preliminaries

For decision problems based on normal surface theory, the input is a *3-manifold triangulation*.[3] This is a collection of n tetrahedra where some or all of the $4n$ faces are affinely identified (or "glued together") in pairs, so that the resulting topological space is a 3-manifold. The *input size* is measured by the number of tetrahedra, which we denote by n throughout this paper.

We allow two faces of the same tetrahedron to be identified together; likewise, we allow different edges or vertices of the same tetrahedron to be identified. Some authors refer to such triangulations as *semi-simplicial triangulations* or *pseudo-triangulations*. In essence, we allow the tetrahedra to be "bent" or "twisted", instead of insisting that they be rigidly embedded in some larger space. This more flexible definition allows us to keep the input size n small, which is important when dealing with exponential algorithms.

[3] In knot theory we triangulate the *knot complement*—that is, the 3-dimensional space surrounding the knot.

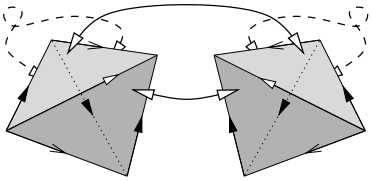

Figure 1: A 3-manifold triangulation of size $n = 2$

Figure 1 shows a triangulation with $n = 2$ tetrahedra: the back two faces of each tetrahedron are identified with a twist, and the front two faces of the first tetrahedron are identified with the front two faces of the second tetrahedron. The underlying 3-manifold is $S^2 \times S^1$.

A *closed* triangulation is one in which all $4n$ tetrahedron faces are identified in $2n$ pairs (as in Figure 1). A *bounded* triangulation is one in which some of the $4n$ tetrahedron faces are left unidentified (these faces form the *boundary* of the triangulation). The underlying topological spaces for closed and bounded triangulations are closed and bounded 3-manifolds respectively.

In normal surface theory, interesting surfaces within a 3-manifold triangulation $\mathcal{T}$ can be encoded as integer points in $\mathbb{R}^{3n}$ (here we work in *quadrilateral coordinates* as described by Tollefson [28]).[4] Each such point $\mathbf{x} \in \mathbb{R}^{3n}$ satisfies the following conditions:

- $\mathbf{x} \geq \mathbf{0}$ and $M\mathbf{x} = \mathbf{0}$, where M is a matrix of *matching equations* that depends on $\mathcal{T}$;
- $\mathbf{x}$ satisfies the *quadrilateral constraints*: for each triple of coordinates (x_1, x_2, x_3), (x_4, x_5, x_6), …, $(x_{3n-2}, x_{3n-1}, x_{3n})$, at most one of the three entries in the triple can be non-zero. There are n such triples, giving n such restrictions in total.

Any point $\mathbf{x} \in \mathbb{R}^{3n}$ that satisfies all of these conditions (that is, $\mathbf{x} \geq \mathbf{0}$, $M\mathbf{x} = \mathbf{0}$ and the quadrilateral constraints) is called *admissible*.

We define the *solution cone* $\mathcal{Q}^\vee$ to be the set $\mathcal{Q}^\vee = \left\{ \mathbf{x} \in \mathbb{R}^{3n} \;\middle|\; \mathbf{x} \geq \mathbf{0} \text{ and } M\mathbf{x} = \mathbf{0} \right\}$. This is a polyhedral cone with apex at the origin. From above, it follows that every interesting surface within our 3-manifold is encoded as an integer point within the solution cone. The *projective solution space* $\mathcal{Q}$ is defined to be the cross-section $\mathcal{Q} = \{\mathbf{x} \in \mathcal{Q}^\vee \mid \sum x_i = 1\}$.

It follows that the projective solution space is a (bounded) polytope with rational vertices. In general, both the solution cone and the projective solution space are highly degenerate (that is, extreme rays of $\mathcal{Q}^\vee$ and vertices of $\mathcal{Q}$ often belong to many more facets than required by the dimension). Non-zero points in the solution cone can be projected onto the cross-section $\mathcal{Q}$ using the map $\mathbf{x} \mapsto \mathbf{x}/\sum x_i$.

For polytopes and polyhedra, we follow the terminology of Ziegler [29]: polytopes are always bounded, polyhedra may be either bounded or unbounded, and both are always convex.

It is important to note that the quadrilateral constraints do not feature in the definitions of $\mathcal{Q}^\vee$ or $\mathcal{Q}$. These constraints are combinatorial in nature, and they cause significant theoretical problems: they are neither linear nor convex, and the region of $\mathcal{Q}$ that obeys them can be disconnected or can have non-trivial topology (such as "holes"). Nevertheless, they are extremely powerful: although the vertices of $\mathcal{Q}$ can be numerous, typically just a tiny fraction of these vertices satisfy the quadrilateral constraints [8, 11].

For topological problems that use normal surface theory, a typical algorithm runs as follows:

[4]The $3n$ coordinates essentially count the number of quadrilaterals of different types in a triangle-quadrilateral decomposition of the surface. We do not count triangles because, for most surfaces of interest, the triangles can be reconstructed from the quadrilateral counts alone.

1. Enumerate all admissible vertices of the projective solution space $\mathscr{Q}$ (that is, all vertices of $\mathscr{Q}$ that satisfy the quadrilateral constraints);

2. Scale each vertex to its smallest integer multiple, "decode" it to obtain a surface within the underlying 3-manifold triangulation, and test whether this surface is "interesting";

3. Terminate the algorithm once an interesting surface is found or all vertices are exhausted.

The meaning of "interesting" varies for different decision problems. For more of the topological context see [19]; for more on quadrilateral coordinates and the projective solution space see [7].

As indicated in the introduction, step 1 (the enumeration of vertices) is typically the computational bottleneck. Because so few vertices are admissible, it is crucial for normal surface enumeration algorithms to enforce the quadrilateral constraints as they go—one cannot afford to enumerate the entire vertex set of $\mathscr{Q}$ (which is generally orders of magnitude larger) and then enforce the quadrilateral constraints afterwards.

Traditionally, normal surface enumeration algorithms are based on the double description method of Motzkin et al. [25], with additional algorithmic improvements specific to normal surface theory [10]. In brief, we construct a sequence of polytopes $P_0, P_1, \ldots, P_e$, where P_0 is the unit simplex in $\mathbb{R}^{3n}$, P_e is the final solution space $\mathscr{Q}$, and each intermediate polytope P_i is the intersection of the unit simplex with the first i matching equations. The algorithm works inductively, constructing P_{i+1} from P_i at each stage. For each intermediate polytope P_i we throw away any vertices that do not satisfy the quadrilateral constraints, and it can be shown inductively that this yields the correct final solution set.

Other methods of vertex enumeration include backtracking and reverse search algorithms. It is worth noting why these have not been used for normal surface enumeration to date:

- Reverse search algorithms map out vertices by pivoting along edges of the polytope. This makes it difficult to incorporate the quadrilateral constraints: since the region that satisfies these constraints can be disconnected, we may need to pivot through vertices that *break* these constraints in order to map out all solutions. Degeneracy can also cause significant performance problems for reverse search algorithms [1, 2].

- Backtracking algorithms receive comparatively little attention in the literature. Fukuda et al. [16] show that backtracking can solve the *face* enumeration problem in polynomial time. However, for vertex enumeration their framework requires a solution to the NP-complete *restricted vertex problem*, and they conclude that straightforward backtracking is unlikely to work efficiently for vertex enumeration in general.

Our new algorithm belongs to the backtracking family. Despite the concerns of Fukuda et al., we find that for large problems our algorithm is a significant improvement over the prior state of the art, often running orders of magnitude faster. We achieve this by introducing *domination tests*, a trade-off that increases time and space costs but allows us to avoid solving the restricted vertex problem directly. The full algorithm is given in Section 3 below.

3 The tree traversal algorithm

The basic idea behind the algorithm is to construct admissible vertices $\mathbf{x} \in \mathscr{Q}$ by iterating through all possible combinations of which coordinates x_i are zero and which coordinates x_i are non-zero. We arrange these combinations into a search tree of height n, which we traverse in a depth-first manner. Using a combination of feasibility tests and domination

tests, we are able to prune this search tree so that the traversal is more efficient, and so that the leaves at depth n correspond precisely to the admissible vertices of $\mathscr{Q}$.

Because the quadrilateral constraints are expressed purely in terms of zero versus non-zero coordinates, we can easily build the quadrilateral constraints directly into the structure of the search tree. We do this with the help of *type vectors*, which we introduce in Section 3.1. In Section 3.2 we describe the search tree and present the overall structure of the algorithm. Proofs of all results in this section can be found in the full version of this paper.

3.1 Type vectors

Recall the quadrilateral constraints, which state that for each $i = 1, \ldots, n$, at most one of the three coordinates $x_{3i-2}, x_{3i-1}, x_{3i}$ can be non-zero. A *type vector* is a sequence of n symbols that indicates how these constraints are resolved for each i. In detail:

Definition (Type vector)**.** Let $\mathbf{x} = (x_1, \ldots, x_{3n}) \in \mathbb{R}^{3n}$ be any vector that satisfies the quadrilateral constraints. We define the *type vector* of $\mathbf{x}$ as $\tau(\mathbf{x}) = (\tau_1, \ldots, \tau_n) \in \{0, 1, 2, 3\}^n$, where

$$\tau_i = \begin{cases} 0 & \text{if } x_{3i-2} = x_{3i-1} = x_{3i} = 0; \\ 1 & \text{if } x_{3i-2} \neq 0 \text{ and } x_{3i-1} = x_{3i} = 0; \\ 2 & \text{if } x_{3i-1} \neq 0 \text{ and } x_{3i-2} = x_{3i} = 0; \\ 3 & \text{if } x_{3i} \neq 0 \text{ and } x_{3i-2} = x_{3i-1} = 0. \end{cases} \tag{1}$$

For example, if $n = 4$ and $\mathbf{x} = (0, 1, 0,\ 0, 0, 4,\ 0, 0, 0,\ 0, 0, 2) \in \mathbb{R}^{12}$, then $\tau(\mathbf{x}) = (2, 3, 0, 3)$. An important feature of type vectors is that they carry enough information to completely reconstruct any vertex of the projective solution space, as shown by the following result.

Lemma 1. *Let $\mathbf{x}$ be any admissible vertex of $\mathscr{Q}$. Then the precise coordinates $x_1, \ldots, x_{3n}$ can be recovered from the type vector $\tau(\mathbf{x})$ by simultaneously solving (i) the matching equations $M\mathbf{x} = \mathbf{0}$; (ii) the projective equation $\sum x_i = 1$; and (iii) the equations of the form $x_j = 0$ as dictated by (1) above, according to the particular value (0, 1, 2 or 3) of each entry in the type vector $\tau(\mathbf{x})$.*

Note that this full recovery of $\mathbf{x}$ from $\tau(\mathbf{x})$ is only possible for *vertices* of $\mathscr{Q}$, not arbitrary points of $\mathscr{Q}$. In general, the type vector $\tau(\mathbf{x})$ carries only enough information for us to determine the smallest face of $\mathscr{Q}$ to which $\mathbf{x}$ belongs.

However, type vectors do carry enough information for us to identify *which* admissible points $\mathbf{x} \in \mathscr{Q}$ are vertices of $\mathscr{Q}$. For this we introduce the concept of domination[5].

Definition (Domination)**.** Let $\tau, \sigma \in \{0, 1, 2, 3\}^n$ be type vectors. We say that τ *dominates* σ if, for each $i = 1, \ldots, n$, either $\sigma_i = \tau_i$ or $\sigma_i = 0$. We write this as $\tau \geq \sigma$.

For example, if $n = 4$ then $(1, 0, 2, 3) \geq (1, 0, 2, 0) \geq (1, 0, 2, 0) \geq (1, 0, 0, 0)$. On the other hand, neither of $(1, 0, 2, 0)$ or $(1, 0, 3, 0)$ dominates the other. It is clear that in general, domination imposes a partial order (but not a total order) on type vectors.

We can characterise admissible vertices of the projective solution space entirely in terms of type vectors and domination, as seen in the following result.

Lemma 2. *Let $\mathbf{x} \in \mathscr{Q}$ be admissible. Then the following statements are equivalent:*

- *$\mathbf{x}$ is a vertex of $\mathscr{Q}$;*
- *there is no admissible point $\mathbf{y} \in \mathscr{Q}$ for which $\tau(\mathbf{x}) \geq \tau(\mathbf{y})$ and $\mathbf{x} \neq \mathbf{y}$;*

[5]We use the word "domination" because domination of type vectors corresponds precisely to domination in the face lattice of the polytope $\mathscr{Q}$. See the full version of this paper for details.

- *there is no admissible vertex* $\mathbf{y}$ *of* $\mathscr{Q}$ *for which* $\tau(\mathbf{x}) \geq \tau(\mathbf{y})$ *and* $\mathbf{x} \neq \mathbf{y}$.

Because our algorithm involves backtracking, it spends much of its time working with partially-constructed type vectors. It is therefore useful to formalise this notion.

Definition (Partial type vector)**.** A *partial type vector* is any vector $\tau = (\tau_1, \ldots, \tau_n)$, where each $\tau_i \in \{0, 1, 2, 3, -\}$. We call the special symbol "$-$" the *unknown symbol*. A partial type vector that does not contain any unknown symbols is also referred to as a *complete type vector*.

We say that two partial type vectors τ and σ *match* if, for each $i = 1, \ldots, n$, either $\tau_i = \sigma_i$ or at least one of τ_i, σ_i is the unknown symbol.

For example, $(1, -, 2)$ and $(1, 3, 2)$ match, and $(1, -, 2)$ and $(1, 3, -)$ also match. However, $(1, -, 2)$ and $(0, -, 2)$ do not match because their leading entries conflict. If τ and σ are both complete type vectors, then τ matches σ if and only if $\tau = \sigma$.

Definition (Type constraints)**.** Let $\tau = (\tau_1, \ldots, \tau_n) \in \{0, 1, 2, 3, -\}^n$ be a partial type vector. The *type constraints* for τ are a collection of equality constraints and non-strict inequality constraints on an arbitrary point $\mathbf{x} = (x_1, \ldots, x_{3n}) \in \mathbb{R}^{3n}$. For each $i = 1, \ldots, n$, the type symbol τ_i contributes the following constraints to this collection:

$x_{3i-2} = x_{3i-1} = x_{3i} = 0$	if $\tau_i = 0$;
$x_{3i-2} \geq 1$ and $x_{3i-1} = x_{3i} = 0$	if $\tau_i = 1$;
$x_{3i-1} \geq 1$ and $x_{3i-2} = x_{3i} = 0$	if $\tau_i = 2$;
$x_{3i} \geq 1$ and $x_{3i-2} = x_{3i-1} = 0$	if $\tau_i = 3$;
no constraints	if $\tau_i = -$.

Note that, unlike all of the other constraints seen so far, the type constraints are not invariant under scaling. The type constraints are most useful in the solution cone $\mathscr{Q}^\vee$, where any admissible point $\mathbf{x} \in \mathscr{Q}^\vee$ with $x_j > 0$ can be rescaled to some admissible point $\lambda\mathbf{x} \in \mathscr{Q}^\vee$ with $\lambda x_j \geq 1$. We see this scaling behaviour in Lemma 3 below.

The type constraints are similar but not identical to the conditions described by equation (1) for the type vector $\tau(\mathbf{x})$, and their precise relationship is described by the following lemma. The key difference is that (1) uses strict inequalities of the form $x_j > 0$, whereas the type constraints use non-strict inequalities of the form $x_j \geq 1$. This is because we use the type constraints with techniques from linear programming, where non-strict inequalities are simpler to deal with.

Lemma 3. *Let* $\sigma \in \{0, 1, 2, 3, -\}^n$ *be any partial type vector. If* $\mathbf{x} \in \mathbb{R}^{3n}$ *satisfies the type constraints for* σ*, then* $\tau(\mathbf{x})$ *matches* σ*. Conversely, if* $\mathbf{x} \in \mathbb{R}^{3n}$ *is an admissible vector for which* $\tau(\mathbf{x})$ *matches* σ*, then* $\lambda\mathbf{x}$ *satisfies the type constraints for* σ *for some scalar* $\lambda > 0$.

We finish this section on type vectors with a simple but important note.

Lemma 4. *There is no* $\mathbf{x} \in \mathscr{Q}$ *for which* $\tau(\mathbf{x})$ *is the zero vector.*

3.2 The structure of the algorithm

Recall that our plan is to iterate through all possible combinations of zero and non-zero coordinates. We do this by iterating through all possible type vectors, which implicitly enforces the quadrilateral constraints as we go. The framework for this iteration is the *type tree*.

Definition (Type tree)**.** The *type tree* is a rooted tree of height n, where all leaf nodes are at depth n, and where each non-leaf node has precisely four children. The four edges descending from each non-leaf node are labelled 0, 1, 2 and 3 respectively.

Each node is labelled with a partial type vector. The root node is labelled $(-, \ldots, -)$. Each non-leaf node at depth i has a label of the form $(\tau_1, \ldots, \tau_i, -, -, \ldots, -)$, and its children along edges 0, 1, 2 and 3 have labels $(\tau_1, \ldots, \tau_i, 0, -, \ldots, -)$, $(\tau_1, \ldots, \tau_i, 1, -, \ldots, -)$, $(\tau_1, \ldots, \tau_i, 2, -, \ldots, -)$ and $(\tau_1, \ldots, \tau_i, 3, -, \ldots, -)$ respectively. Figure 2 shows this for $n = 2$.

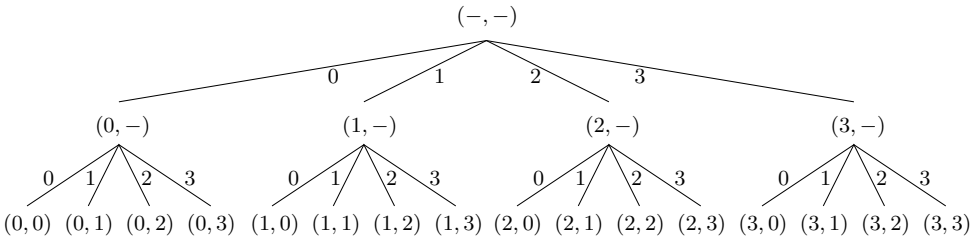

Figure 2: The type tree of height $n = 2$

The algorithm walks through the type tree in a depth-first manner, collecting admissible vertices of $\mathscr{Q}$ as it goes. The tree is large, with $O(4^n)$ nodes, and so we do not traverse it in full. Instead we prune the tree so that we only follow edges that might lead to admissible vertices. The main tools for pruning are feasibility tests (based on the type constraints and matching equations) and domination tests (based on the previous vertices found so far). In detail:

Algorithm 5 (Tree traversal algorithm)**.** *The following algorithm takes a 3-manifold triangulation as input, and outputs all admissible vertices of the projective solution space $\mathscr{Q}$.*

Construct the matching equations M (see [7] for details). Initialise an empty set V that will store type vectors for admissible vertices of $\mathscr{Q}$. Beginning at the root of the type tree, process nodes recursively according to the following instructions. In general, to process the node N:

- *If N is a non-leaf node then examine the four children of N in turn, beginning with the child along edge 0 (the order of the remaining children is unimportant). For a child node labelled with the partial type vector τ, recursively process this child if and only if all of the following conditions are satisfied:*
  - *(a)* Zero test: *τ is not the zero vector.*
  - *(b)* Domination test: *If we replace every unknown symbol $-$ with 0, then τ does not dominate any type vector in V.*
  - *(c)* Feasibility test: *There is some point $\mathbf{x} \in \mathbb{R}^{3n}$ that satisfies $\mathbf{x} \geq \mathbf{0}$, $M\mathbf{x} = \mathbf{0}$, and the type constraints for τ.*
- *If N is a leaf node then its label is a complete type vector τ, and we claim that τ must in fact be the type vector of an admissible vertex of $\mathscr{Q}$. Insert τ into the set V, reconstruct the full vertex $\mathbf{x} \in \mathbb{R}^{3n}$ from τ as described by Lemma 1, and output this vertex.*

A full proof of correctness appears in the full version of this paper.

For topological decision problems, this algorithm must be implemented using *exact arithmetic*. This is problematic because both the numerators and the denominators of the rationals that we encounter can grow exponentially large in n. A naïve implementation could simply use arbitrary-precision rational arithmetic. However, in the full version of this paper

we prove bounds that allow us to use native machine integer types (such as 32-bit, 64-bit or 128-bit integers) for many reasonable-sized applications.

Both the domination test and the feasibility test are non-trivial. The domination test uses a trie (i.e., a prefix tree) for storing V, and the feasibility test is based on incremental dual simplex iterations; for details see the full version of this paper. On the other hand, the zero test is simple; moreover, it is easy to see that it only needs to be tested once (for the very first leaf node that we process).

All three tests (zero, domination and feasibility) can be performed in polynomial time in the input and/or output size. What prevents the entire algorithm from running in polynomial time is *dead ends*: nodes that we process but which do not lead to any admissible vertices.

It is worth noting that Fukuda et al. [16] describe a backtracking framework that does not suffer from dead ends. However, this comes at a significant cost: before processing each node they must solve the *restricted vertex problem*, which essentially asks whether a vertex exists beneath a given node in the search tree. They prove this restricted vertex problem to be NP-complete, and they do not give an explicit algorithm to solve it.

We discuss time and space complexity further in Section 4, where we find that—despite the presence of dead ends—this tree traversal algorithm enjoys significantly lower worst-case complexity bounds than the prior state of the art. This is supported with experimental evidence in Section 5. In the meantime, we note some additional advantages of the tree traversal algorithm:

- *The tree traversal algorithm lends itself well to parallelisation.*

  For each non-leaf node N, the children along edges 1, 2 and 3 can be processed simultaneously (but edge 0 must still be processed first). There is no need for these three processes to communicate, since they cannot affect each other's domination tests. This three-way branching can be carried out repeatedly as we move deeper through the tree, allowing us to effectively use a large number of processors if they are available. Distributed processing is also feasible, since at each branch the data transfer has small polynomial size.

- *The tree traversal algorithm supports incremental output.*

  If the type vectors of admissible vertices are reasonably well distributed across the type tree (not tightly clustered in a few sections of the tree), then we can expect a regular stream of output as the algorithm runs. This incremental output is extremely useful:

  - It allows *early termination* of the algorithm. For many topological decision algorithms, our task is to find *any* admissible vertex with some "interesting" property P (or else conclude that no such vertex exists). As the algorithm outputs vertices we can test them immediately for property P, and terminate as soon as such a vertex is found.
  - It further assists with *parallelisation* if testing for property P is expensive (this is above and beyond the parallelisation techniques described above). An example is the Hakenness testing problem, where testing for property P is far more difficult than the initial normal surface enumeration [12]. For such problems, we can have a main tree traversal process that outputs vertices into a queue for processing, and separate parallel processes that together work through this queue and test vertices for property P.

  This is a significant improvement over the double description method (the prior state of the art for normal surface enumeration). The double description method works

inductively through a series of stages (as outlined in Section 2), and it typically does not output any solutions at all until it reaches the final stage.

- *The tree traversal algorithm supports progress tracking.*

  For any node in the type tree, it is simple to estimate when it appears (as a percentage of running time) in a depth-first traversal of the full tree. We can present these estimates to the user as the algorithm runs, in order to give them some indication of the running time remaining. Of course such estimates are inaccurate: they ignore pruning as well as variability in running times for the feasibility and domination tests. Nevertheless, they provide a rough indication of how far through the algorithm we are at any given time.

  Again this is a significant improvement over the double description method. Some of the stages in the double description method can run many orders of magnitude slower than others [2, 7], and it is difficult to estimate in advance how slow each will be. In this sense, the obvious progress indicators (which stage we are up to and how far through it we are) are extremely poor indicators of global progress through the double description algorithm.

4 Time and space complexity

Here we give theoretical bounds on the time and space complexity of normal surface enumeration algorithms. We compare these bounds for the double description method (the prior state of the art, as described in [10]) and the tree traversal algorithm that we introduce in this paper.

All of these bounds are driven by exponential factors, and it is these exponential factors that we focus on here. We replace any polynomial factors with a generic polynomial $\varphi(n)$; for both algorithms it is easy to show that these polynomial factors are of small degree.

The theoretical bounds are as follows. For proofs, the reader is again referred to the full version of this paper.

Theorem 6. *The double description method for normal surface enumeration can be implemented in worst-case $O(16^n\varphi(n))$ time and $O(4^n\varphi(n))$ space.*

Theorem 7. *The tree traversal algorithm for normal surface enumeration can be implemented in worst-case $O(4^n|V|\varphi(n))$ time and $O(|V|\varphi(n))$ space, where $|V|$ is the number of admissible vertices of $\mathscr{Q}$. In terms of n alone, the time complexity is $O(7^n\varphi(n))$, and the space complexity is $O\left(\left[\frac{3+\sqrt{13}}{2}\right]^n\varphi(n)\right) \simeq O(3.303^n\varphi(n))$ for closed triangulations or $O(4^n\varphi(n))$ for bounded triangulations.*

The bounds in terms of $|V|$ are important, because for normal surface enumeration the output size $|V|$ is found to be extremely small in practice.[6] Essentially, Theorem 7 shows that the tree traversal algorithm is able to effectively exploit a small output size. In contrast, the double description method cannot because it suffers from a well-known *combinatorial explosion*: even when the output size is small, the intermediate structures that appear can be significantly (and exponentially) more complex. See [2] for examples of this in theory, or [7] for normal surface experiments that show this in practice.

[6] Early indications of this appear in [8] (which works in the larger coordinate space $\mathbb{R}^{7n}$), and a detailed study will appear in [9]. To illustrate how extreme these results are in $\mathbb{R}^{3n}$: across all 139 103 032 closed 3-manifold triangulations of size $n = 9$, the maximum output size is just 37, and the *mean* output size is a mere 9.7.

5 Experimental performance

We must bear in mind that for both the tree traversal and double description algorithms, our theoretical bounds may still be far above the "real" time and memory usage that we observe in practice. For this reason it is important to carry out practical experiments. Here we run both algorithms through a test suite of 275 triangulations, based on the first 275 entries in the census of knot complements tabulated by Christy and shipped with version 1.9 of *Snap* [13].

We begin with a direct comparison of running times for the tree traversal algorithm and the double description algorithm. Both algorithms are implemented using the topological software package *Regina* [6]. In particular, the double description implementation that we use is already built into *Regina*; this represents the current state of the art for normal surface enumeration, and the algorithm has been finely tuned over many years [10]. Both algorithms are implemented in C++, and all experiments were run on one core of a 64-bit 2.3 GHz AMD Opteron 2356 processor.

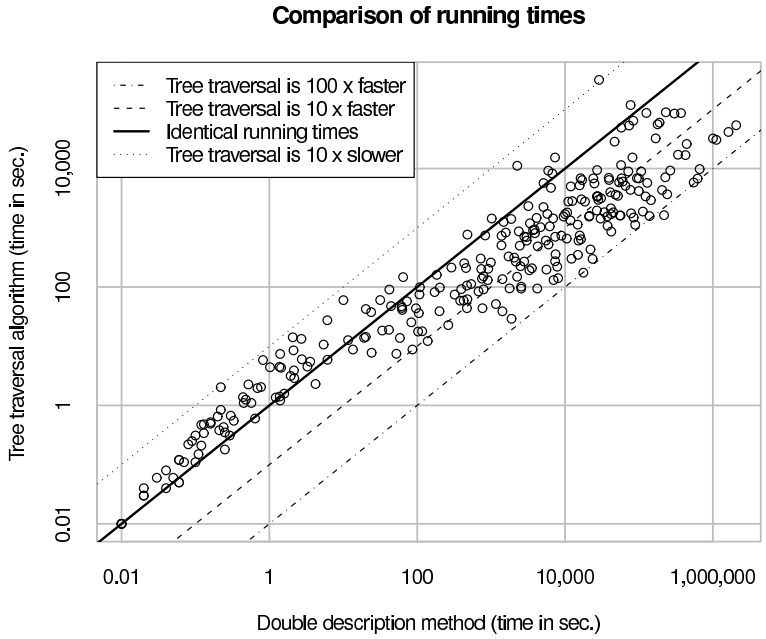

Figure 3: Comparing running times for the old and new algorithms

Figure 3 directly compares the running times for both algorithms: each point on the graph represents a single triangulation from the test suite. Both axes use a logarithmic scale. The solid diagonal line indicates where both algorithms have identical running times, and each dotted line represents an order of magnitude of difference between them.

This graph shows that for smaller problems, the tree traversal algorithm runs slower (in the worst case, around 10 times slower). However, as the problems grow more difficult the tree traversal begins to dominate, and for running times over 100 seconds the tree traversal algorithm is almost always faster (in the best case, around 100 times faster).

In the full version of this paper we also study the growth of the number of admissible vertices and the number of "dead ends" that we walk through in the type tree. We discover experimentally that the number of solutions appears to grow at a rate of roughly 1.47^n, and the number of nodes that we visit (including dead ends) grows at a rate of roughly 1.95^n. Both figures are far below their theoretical bounds of $O(4^n)$.

These small growth rates undoubtedly contribute to the strong performance of the algorithm, as seen above. However, our experiments also raise another intriguing possibility, which is that the number of nodes that we visit might be *polynomial in the output size*. If this were true in general, both the time and space complexity of the tree traversal algorithm would become worst-case polynomial in the combined input and output size. This would be a significant breakthrough for normal surface enumeration. We return to this speculation in Section 6.

6 Discussion

For topological decision problems that require lengthy normal surface enumerations, we conclude that the tree traversal algorithm is the most promising algorithm amongst those known to date. Not only does it have small time and space complexities in both theory and practice (though these complexities remain exponential), but it also supports parallelisation, progress tracking, incremental output and early termination. Parallelisation and incremental output in particular offer great potential for normal surface theory.

One aspect of the tree traversal algorithm that has not been discussed in this paper is the *ordering of tetrahedra*. By reordering the tetrahedra we can alter which nodes of the type tree pass the feasibility test, which could potentially have a significant effect on the running time. Further exploration and experimentation in this area could prove beneficial.

We finish by recalling the suggestion from Section 5 that the total number of nodes visited—and therefore the overall running time—could be a small polynomial in the combined input and output size. Our experimental data (as detailed in the full version of this paper) suggest a quadratic relationship, and similar experimentation on closed triangulations suggests a cubic relationship. Even if we examine triangulations with extremely small output sizes (such as layered lens spaces, where the number of admissible vertices is linear in n [10]), this small polynomial relationship appears to be preserved (for layered lens spaces we visit $O(n^3)$ nodes in total).

Although our algorithm cannot solve the vertex enumeration problem for *general* polytopes in polynomial time, there is much contextual information to be gained from normal surface theory and the quadrilateral constraints. Further research into this polynomial time conjecture could prove fruitful: a counterexample would be informative, and a proof would be a significant breakthrough for the complexity of decision algorithms in low-dimensional topology.

Acknowledgements

The authors are grateful to RMIT University for the use of their high-performance computing facilities. The first author is supported by the Australian Research Council under the Discovery Projects funding scheme (project DP1094516).

References

[1] David Avis, *A revised implementation of the reverse search vertex enumeration algorithm*, Polytopes—Combinatorics and Computation (Oberwolfach, 1997), DMV Sem., vol. 29, Birkhäuser, Basel, 2000, pp. 177–198.

[2] David Avis, David Bremner, and Raimund Seidel, *How good are convex hull algorithms?*, Comput. Geom. **7** (1997), no. 5–6, 265–301.

[3] David Avis and Komei Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, Discrete Comput. Geom. **8** (1992), no. 3, 295–313.

[4] M. L. Balinski, *An algorithm for finding all vertices of convex polyhedral sets*, SIAM J. Appl. Math. **9** (1961), no. 1, 72–88.

[5] Joan S. Birman, *Problem list: Nonsufficiently large 3-manifolds*, Notices Amer. Math. Soc. **27** (1980), no. 4, 349.

[6] Benjamin A. Burton, *Introducing Regina, the 3-manifold topology software*, Experiment. Math. **13** (2004), no. 3, 267–272.

[7] ———, *Converting between quadrilateral and standard solution sets in normal surface theory*, Algebr. Geom. Topol. **9** (2009), no. 4, 2121–2174.

[8] ———, *The complexity of the normal surface solution space*, SCG '10: Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry, ACM, 2010, pp. 201–209.

[9] ———, *Extreme cases in normal surface enumeration*, In preparation, 2010.

[10] ———, *Optimizing the double description method for normal surface enumeration*, Math. Comp. **79** (2010), no. 269, 453–484.

[11] ———, *Maximal admissible faces and asymptotic bounds for the normal surface solution space*, J. Combin. Theory Ser. A **118** (2011), no. 4, 1410–1435.

[12] Benjamin A. Burton, J. Hyam Rubinstein, and Stephan Tillmann, *The Weber-Seifert dodecahedral space is non-Haken*, To appear in Trans. Amer. Math. Soc., `arXiv:0909.4625`, September 2009.

[13] David Coulson, Oliver A. Goodman, Craig D. Hodgson, and Walter D. Neumann, *Computing arithmetic invariants of 3-manifolds*, Experiment. Math. **9** (2000), no. 1, 127–152.

[14] Marc Culler and Nathan Dunfield, *FXrays: Extremal ray enumeration software*, `http://www.math.uic.edu/~t3m/`, 2002–2003.

[15] M. E. Dyer, *The complexity of vertex enumeration methods*, Math. Oper. Res. **8** (1983), no. 3, 381–402.

[16] Komei Fukuda, Thomas M. Liebling, and François Margot, *Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron*, Comput. Geom. **8** (1997), no. 1, 1–12.

[17] Wolfgang Haken, *Theorie der Normalflächen*, Acta Math. **105** (1961), 245–375.

[18] ———, *Über das Homöomorphieproblem der 3-Mannigfaltigkeiten. I*, Math. Z. **80** (1962), 89–120.

[19] Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger, *The computational complexity of knot and link problems*, J. Assoc. Comput. Mach. **46** (1999), no. 2, 185–211.

[20] William Jaco, *The homeomorphism problem: Classification of 3-manifolds*, Lecture notes, Available from `http://www.math.okstate.edu/~jaco/pekinglectures.htm`, 2005.

[21] William Jaco and Ulrich Oertel, *An algorithm to decide if a* 3*-manifold is a Haken manifold*, Topology **23** (1984), no. 2, 195–209.

[22] Bruce Kleiner and John Lott, *Notes on Perelman's papers*, Geom. Topol. **12** (2008), no. 5, 2587–2855.

[23] Hellmuth Kneser, *Geschlossene Flächen in dreidimensionalen Mannigfaltigkeiten*, Jahresbericht der Deut. Math. Verein. **38** (1929), 248–260.

[24] Sergei Matveev, *Algorithmic topology and classification of 3-manifolds*, Algorithms and Computation in Mathematics, no. 9, Springer, Berlin, 2003.

[25] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall, *The double description method*, Contributions to the Theory of Games, Vol. II (H. W. Kuhn and A. W. Tucker, eds.), Annals of Mathematics Studies, no. 28, Princeton University Press, Princeton, NJ, 1953, pp. 51–73.

[26] J. Hyam Rubinstein, *An algorithm to recognize the* 3*-sphere*, Proceedings of the International Congress of Mathematicians (Zürich, 1994), vol. 1, Birkhäuser, 1995, pp. 601–611.

[27] Marco Terzer and Jörg Stelling, *Parallel extreme ray and pathway computation*, Parallel Processing and Applied Mathematics, Lecture Notes in Comput. Sci., vol. 6068, Springer, Berlin, 2010, pp. 300–309.

[28] Jeffrey L. Tollefson, *Normal surface Q-theory*, Pacific J. Math. **183** (1998), no. 2, 359–374.

[29] Günter M. Ziegler, *Lectures on polytopes*, Graduate Texts in Mathematics, no. 152, Springer-Verlag, New York, 1995.

Benjamin A. Burton
School of Mathematics and Physics, The University of Queensland
Brisbane QLD 4072, Australia
(bab@maths.uq.edu.au)

Melih Ozlen
School of Mathematical and Geospatial Sciences, RMIT University
GPO Box 2476V, Melbourne VIC 3001, Australia
(melih.ozlen@rmit.edu.au)