University of Cantabria

Master's thesis.

Masters degree in Mathematics and Computing

Diameter of simplicial complexes, a computational approach

Author:
Francisco Criado Gallart

Advisor:
Francisco Santos Leal

2015-2016

Desvarío laborioso y empobrecedor el de componer vastos libros; el de explayar en quinientas páginas una idea cuya perfecta exposición oral cabe en pocos minutos. Mejor procedimiento es simular que esos libros ya existen y ofrecer un resumen, un comentario.

Borges, *Ficciones*

Acknowledgements

First, I would like to thank my advisor, professor **Francisco Santos**. He has been supportive years before I arrived in Santander, and even more when I arrived.

Also, I want to thank professor **Michael Joswig** and his team in the discrete geometry group of the Technical University of Berlin, for all the help and advice that they provided during my (short) visit to their department.

Finally, I want to thank my friends at Cantabria, and my “disciples”: Álvaro, Ana, Beatriz, Cristina, David, Esther, Jesús, Jon, Manu, Néstor, Sara and Susana.

Diameter of simplicial complexes, a computational approach

Abstract

The computational complexity of the simplex method, widely used for linear programming, depends on the combinatorial diameter of the edge graph of a polyhedron with n facets and dimension d. Despite its popularity, little is known about the (combinatorial) diameter of polytopes, and even for simpler types of complexes. For the case of polytopes, it was conjectured by Hirsch (1957) that the diameter is smaller than $(n - d)$, but this was disproved by Francisco Santos (2010). In this thesis, we present two main results. First, a lower bound on the maximum diameter for two classes of simplicial complexes: pure simplicial complexes and pseudo-manifolds. Second, a topological improvement of Santos' counter example to the Hirsch Conjecture. This one is (slightly) smaller than the previously known smallest simplicial non-Hirsch sphere.

Keywords: Discrete Geometry, Computational Geometry, Simulated Annealing, Simplicial Complex, Combinatorial Diameter.

Diámetro de complejos simpliciales, un enfoque computacional

Resumen

La complejidad computacional del método del símplex, ampliamente usado en programación lineal, depende del diámetro combinatorio del grafo de aristas de un poliedro con n facetas y dimensión d. Pese a su popularidad, sabemos muy poco sobre el diámetro (combinatorio) de los politopos, incluso para clases más simples de complejos. Para el caso de los politopos, Hirsch (1957) conjeturó que el diámetro era menor que $n - d$, pero su conjetura fue refutada por Francisco Santos (2010). En esta memoria, presentamos dos resultados fundamentales. Primero, una cota inferior para el diámetro de dos clases de complejos simpliciales: complejos simpliciales puros y pseudo-variedades. Segundo, una mejora topológica del contraejemplo de Santos a la conjetura de Hirsch. Ésta es más pequeña que la anterior esfera simplical no Hirsch más pequeña que se conocía.

Palabras clave: Geometría discreta, Geometría computacional, Enfriamiento simulado, Complejo simplicial, Diámetro combinatorio.

Contents

1 Introduction

The problem of finding the combinatorial diameter of simplicial complexes, particularly polytopal simplicial complexes, arises naturally from the analysis of the simplex method. In the simplex algorithm, we start from a vertex of a polytope and we move from one vertex to a neighbour until we reach the "best vertex" of an objective function. Therefore, a natural question to ask is how long can this path of vertices be in the worst case or, equivalently, what is the maximum possible diameter of a polytopal simplicial complex, as a function of its diameter and number of vertices.

For the case of polytopal simplicial complexes, it was conjectured by Hirsch (1957) that the diameter of a polytope with n facets and dimension d is at most $n - d$. This was disproven by Francisco Santos Leal (2010) [17], who found a non-Hirsch polytope with 86 facets, dimension 43, and diameter 44. His construction, and the smaller ones subsequently constructed in [15], depend on certain lower dimensional polytopes called *prismatoids*. In particular, finding prismatoids satisfying certain property with a small number of vertices (or more significantly, with small difference "vertices minus dimension") will yield smaller counterexamples to the Hirsch Conjecture. It is also important to note it could still be that the diameter of all polytopes is linear on $n-d$, even if no polynomial upper bound is known [12]. It would be desirable to prove or disprove this fact, and find the linear constant if applicable.

Partially as a means to shed light on the Hirsch question, but also as a natural mathematical question in itself, it is interesting to study how big can the diameter of other classes of simplicíal complexes be, and learn from the examples that we may find in this greater generality. This approach was started by Adler and Dantzig in the early 70s and has been continued, for example, in [14, 2]. See [19] for a recent survey of results. This is also the approach taken in this work, in which we present two contributions:

- First, we show how to construct pure simplicial complexes of a given dimension $d-1$ and number of vertices n which have diameters equal (modulo a constant depending on d) to the trivial upper bound of $O(n^{d-1})$. We first do this for some particular complexes that we call corridors (Theorem 4) and then show how to go from any pure simplicial complex to a pseudo-manifold of the same dimension without significantly changing neither the diameter nor the number of vertices (Theorem 8 and Corollary 1). These constructions improve the previously best ones which were of type $\Theta(n^{2d/3})$ for general complexes, and $\Theta(n^2)$ (no d in the exponent) for pseudo-manifolds. Our constructions are algebraic and are based on a sequence produced by certain polynomials over a (large enough) finite field. They are inspired in the well-known constructions of *linear-feedback shift registers* of maximal length. See the details in Section 3.
- We then introduce the concept of *topological prismatoids*, which generalize the (geometric) prismatoids from [17, 15], and we implement a metaheuristic to search for topological prismatoids with the non-Hirsch property and smaller than the ones

in [17, 15]. Our program has been able to find topological prismatoids of dimension 5 with 16 vertices and diameter 6. If one of these spheres turns out to be polytopal, then we would have constructed non-Hirsch polytopes of dimension 11 and with 22 facets, while the current smallest ones have dimension 20 and 40 facets [15]. Our algorithm combines the ideas of *simulated annealing* (SA), a common metaheuristic technique widely used in optimization, with the notion of *bistellar flips*, elementary transformations that change a simplicial complex in a local manner preserving its topology. Geometric bistellar flips are widely used in computational geometry and polyhedral combinatorics (see for example [18]). The topological version that we use is very similar to the one used in [3] in the context of triangulated manifold simplification, the main difference being that, there only manifolds without boundary are used and here, we need a version for manifolds with boundary. See the details in Section 4.

Even if our example turned out to be nonpolytopal (which could happen because not all simplicial spheres can be obtained from a polytope), it implies the existence of a sphere of dimension 10 and with 22 vertices whose diameter is greater than 11. This is an example of a non-Hirsch sphere (a $d-1$-sphere with n vertices and diameter greater than $n-d$). The existence on non-Hirsch spheres has been known for about 35 years now [14], but the smallest ones previously known are slightly bigger than ours: dimension 11 and 24 vertices in [14]. Both the sphere in [14] and the one we construct are *shellable*, a purely combinatorial property meaning basically that the sphere can be constructed one simplex at a time in such a way that all the intermediate complexes are balls. Shellability is a necessary condition for polytopality, but it is not sufficient. In fact, the sphere of [14] was proved to be non-polytopal in [2]. For the sphere we construct polytopality is unknown.

Let us mention that, although we speak of our sphere as a single example, in fact the program has given as output 5 prismatoids with the same parameters (dimension 5, 22 vertices, and width 6) and 100 5-dimensional prismatoids of width six and number of vertices still smaller than the previously known non-Hirsch prismatoids with a number of vertices ranging from 17 and 22. (But we have checked shellability only for one of them, since no polynomial time algorithm to check shellability is known).

As a final motivation for (the second part of) our work, we bring here a quote by Gil Kalai about the role of examples in mathematics [11, p. 769]:

> It is not unusual that a single example or a very few shape an entire mathematical discipline. [...] And it seems that overall, *we are short of examples*. The methods for coming up with useful examples in mathematics (or counterexamples for commonly believed conjectures) are even less clear than the methods for proving mathematical statements.

2 Preliminaries

2.1 Definitions

Definition 1. A *simplicial complex* is a set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of sets, such that $\forall s \in \mathcal{S}$, $\forall f \subset s, f \in S$. The complex is *pure of dimension* $d-1$ (PSC) if every maximal element of $\mathcal{S}$ has d elements.

The elements of $\mathcal{S}$ are called *faces*, and some faces have special names:

Facet If $|s| = d$.

Ridge If $|s| = d-1$.

Edge If $|s| = 2$.

Vertex If $|s| = 1$.

Empty face If $s = \emptyset$.

Also note that the set of faces has a natural partial order, the inclusion. The *Hasse diagram* of the simplicial complex is the directed graph representing the inclusion relations between faces.

Example. *The boundary of an octahedron in the three-dimensional space is a pure simplicial complex of dimension* 2*, that is,* $d = 3$*. It has eight facets, twelve ridges (that are also edges) and six vertices. It is also homeomorphic to the 2-dimenisonal sphere* $\mathbb{S}^2$*.*

In general, given a basis for the affine space $\mathbb{R}^n$*,* $e_1, \ldots, e_n$*, the convex hull of* $\{\pm e_1, \ldots, \pm e_n\}$*, which is called a* crosspolytope*, is a PSC with* $2n$ *vertices,* 2^n *facets, and homeomorphic to a* $(n-1)$*-sphere.*

Definition 2. Given a $(d-1)$-dimensional simplicial complex $\mathcal{S}$ and a face $f \in \mathcal{S}$:

- The *star* of f is the set of faces of $\mathcal{S}$ that are supersets of f.
- The *ustar* of f is the union of all faces in the star of f. That is to say, it is the set of vertices that appear in some face containing f.
- The *link* of f is the set of faces of $\mathcal{S} \setminus f$ that, joined with f, belong to the complex. That is, the vertices of the link, joined with f, form the ustar of f.

Note that the ustar of a face f is the set of vertices v such that $f \cup \{v\}$ is still in the complex. We will use this definition to "move up" in the Hasse diagram, and we can "move down" by removing vertices of the faces.

Definition 3. A *polytope* of dimension d is an bounded intersection of affine half-spaces in $\mathbb{R}^d$. A polytope is *simple* if every point of it is at most in d of the defining hyperplanes at the same time. That is, a polytope can be defined by a set of n linear inequalities in d variables.

The intersection of one of the defining hyperplanes with the polytope (that is, the points satisfying one equation with equality) is called a *facet*. In a similar way, the set of point satisfying with equality two equations are a *ridge*.

The *dual graph* of the polytope is a graph with a vertex for each facet and an edge for each ridge, such that an edge connecting two vertices in the graph corresponds to a ridge in the intersection of two facets.

Definition 4. Given a d-dimensional simple polytope P defined by n equations, we can define its *dual simplicial complex* as the subsets of $\{1, \ldots, n\}$ such that there is an $x \in \mathbb{R}^d$ satisfying exactly the corresponding equations with equality. That is, its vertices are the facets of the polytope, and its facets are the vertices of P.

Observe that the dual simplicial complex of a polytope P is the face complex of proper faces of the poytope dual to P. For example, the dual complex of a cube is the boundary complex of an octahedron.

Definition 5. The *dual graph* of a pure simplicial complex $\mathcal{S}$ is a graph having the facets of $\mathcal{S}$ as vertices where two facets are adjacent if their intersection is a ridge. The *dual diameter* of $\mathcal{S}$ is the diameter of its dual graph.

We define also $H_{\mathcal{C}}(n, d)$ as the maximum dual diameter a simplicial complex with n vertices and dimension $d-1$ can have in a particular class $\mathcal{C}$ of simplicial complexes.

Definition 6. A *prismatoid* is a polytope Q with two parallel facets Q^+ and Q^-, that we call the *bases*, containing all the vertices. We call a prismatoid *simplicial* if all faces except perhaps Q^+ and Q^- are simplices. Observe that the faces of a prismatoid of dimension d, excluding the two bases, form a simplicial complex of dimension $d-1$ and homeomorphic to the product of $\mathbb{S}^{d-2}$ with a segment. We call this complex the prismatoid complex of Q.

Definition 7. The *width* of a prismatoid is the distance in the dual graph from one base to the other, or, to be more in agreement with our definitions above, it is two plus the minimum distance butween a facet adjacent to a base and a facet adjacent to the other base.

Theorem 1 (Strong d-step theorem for prismatoids[17])**.** *If Q is a d-prismatoid with width l and n vertices, there is another $n-d$-prismatoid Q' with $2n-2d$ vertices and width at least $l+n-2d$.*

In particular, if $l > d$ then the polytope dual to Q' violates the Hirsch Conjecture.

Theorem 2 (Matschke-Santos-Weibel' improved counterexample [15]). *There is a prismatoid with* 28 *vertices, dimension* 5 *and width* 6. *That is, there is a non-hirsch polytope with dimension* 23 *and* 46 *vertices.*

Remark. *The best example in [15] has actually 25 and not 28 vertices. But in our computations in section 4 we take the one with 28 vertices as the initial state for the simulated annealing, because it has much more symmetry and is thus easier to input. Part of our goal was in fact to see whether we could go from the 28 example to one smaller than 25, in order to compare our methods with the half-computational ones used in [15].*

One of our main results in this project is a simplification of the Matschke-Santos-Weibel example, but in a topological sense. We now define the main object we are working with:

Definition 8. A *($(d-1)$-dimensional) topological prismatoid* is a $(d-1)$-dimensional pure simplicial complex homeomorphic to $\mathbb{S}_{d-2} \times [0,1]$ (that is, it is homeomorphic to a cylinder), and such that every face with all its vertices in the same boundary component is fully contained in the boundary. Put differently, the induced subcomplex on each boundary component coincides with the boundary component itself. (Observe that these boundary components are, by definition, $(d-2)$-spheres).

2.2 Previous best bounds on the diameter of simplicial complexes

For many important classes of simplicial complexes it is open whether $H_C(n,d)$ is polynomial or not. For example, no manifolds are known in which the diameter grows more than linearly, but no polynomial upper bound is known even in the much smaller class of simplicial spheres. Our first main result is precisely an improvement on the best known lower bound of two classes of simplicial complexes: pure simplicial complexes (defined in the previous section) and pseudomanifolds:

Definition 9. A *pseudo-manifold* is a pure simplicial complex in which every ridge belongs to exactly two facets.

For the class PSC of all pure (connected) simplicial complexes, it was known that:

Theorem 3 (Santos [19, Corollary 2.12]).

$$\Omega\left(\frac{n}{d}\right)^{\frac{2d}{3}} \leq H_{PSC}(n,d) \leq \frac{1}{d-1}\binom{n}{d-1} \simeq \frac{n^{d-1}}{d!}.$$

Here the upper bound is obtained by counting the number of ridges in the complex, and it is the same for pseudo manifolds.

For the case of pseudo-manifolds, the best known lower bound was quadratic. The following table sums the previously known best lower bounds.

	Upper bound	Lower bound
PSC	$O(n^{d-1})$	$\Omega(n^{2d/3})$ (Santos, 2013)
P. Manifolds	$O(n^{d-1})$	$\Omega(n^2)$ (Todd, 1974)
Spheres	$2^{d-3}n$ (Larman 1970)	$1.08(n-d)$ (Walkup-Mani 1980)
Simplicial Polytopes	$2^{d-3}n$ (Larman 1970)	$1.05(n-d)$ (Matschke-Santos-Weibel 2015)

3 The maximum diameter of pure simplicial complexes and pseudo-manifolds

This section is based in our preprint [5]

3.1 pure simplicial complexes

Our bound for the pure simplicial complex case is as follows:

Theorem 4. *For every $d \in \mathbb{N}$ there are infinitely many $n \in \mathbb{N}$ such that:*

$$H_{PSC}(n,d) \geq \frac{n^{d-1}}{(d+2)^{d-1}} - 3.$$

Observe that this matches the upper bound in Theorem 3, modulo a factor in $\Theta(d^{3/2}e^{-d})$, since $d! \simeq e^{-d}d^d\sqrt{2\pi d}$.

Remark. *Our proof of Theorem 4 uses an arithmetic construction valid only when the number n of vertices is of the form $q(d+2)$ for a sufficiently large prime power q. But every interval $[m, 2m]$ contains an n of that form, because there is a power of 2 between $m/(d+2)$ and $m/(2(d+2))$). Hence, the theorem is also valid "for every d and sufficiently large n", modulo an extra factor of 2^{d-1} in the denominator.*

Our construction uses the following well-known result that can be found, for example, in [13, Theorem 33.16]:

Theorem 5. *Let $p(x) = x^d + a_1x^{d-1} + \cdots + a_d$ be a primitive polynomial of degree d over the field $\mathbb{F}_q$ with q elements, for some $d \in \mathbb{N}$ and some prime power q. Consider the sequence $(u_n)_{n\in\mathbb{N}}$ defined by the linear recurrence*

$$u_{n+d} + a_1u_{n+d-1} + \cdots + a_du_n = 0,$$

starting with any non-zero vector $(u_1, \dots, u_d) \in \mathbb{F}_q^d$. *Then,* $(u_n)_{n\in\mathbb{N}}$ *has period* $q^d - 1$. *In particular, its intervals of length* d *cover all of* $\mathbb{F}_q^d \setminus \{(0, \dots, 0)\}$. *That is:*

$$\left\{(u_i, \dots, u_{i+d-1}) : i \in \{1, \dots, q^d - 1\}\right\} = \mathbb{F}_q^d \setminus \{(0, \dots, 0)\}.$$

Remember that a primitive polynomial of degree d is the minimal polynomial of a primitive element in the degree d extension $\mathbb{F}_{q^d}$ of $\mathbb{F}_d$. The number of monic primitive polynomials of degree d over $\mathbb{F}_q$ equals $\phi(q^d - 1)/d$, since $\mathbb{F}_{q^d}$ has $\phi(q^d - 1)$ primitive elements, and each primitive polynomial is the minimal polynomial of d of them. In our construction we will need the coefficients of $p(x)$ to be all different from zero. Primitive polynomials with this property do not exist for all q, but they exist when q is sufficiently large with respect to d, which is enough for our purposes:

Lemma 1. *For every fixed* $d \in \mathbb{N}$ *and every sufficiently large prime power* q*, there is a primitive polynomial of degree* d *over* $\mathbb{F}_q$ *with all coefficients different form zero.*

Proof. This follows from the fact that the number of primitive monic polynomials of degree d is greater than the number of monic polynomials of degree d with at least one zero coefficient, for q large.

Indeed, the latter is $q^d - (q - 1)^d \le dq^{d-1}$. The former equals $\phi(q^d - 1)/d$, which is greater than $(q^d - 1)^{1-\epsilon}/d$, for every $0 < \epsilon < 1$ and sufficiently large q. Letting $\epsilon = \frac{1}{d^2}$ we get:

$$\frac{\phi(q^d - 1)}{d} > \frac{(q^d - 1)^{1-\frac{1}{d^2}}}{d} > \frac{(q^d/2)^{1-\frac{1}{d^2}}}{d} = \frac{q^{(d^2-1)/d}}{2^{1-\frac{1}{d^2}}d} > \frac{q^{d-\frac{1}{d}}}{2d} > dq^{d-1}.$$

$\square$

With this we can now show our first construction proving Theorem 4.

Theorem 6. *Suppose that* $p(x) \in \mathbb{F}_q[x]$ *is a primitive polynomial of degree* $d - 1$ *with no zero coefficients. Then, there is a pure simplicial complex* C *of dimension* $d - 1$*, with* $n = (d + 2)q$ *vertices and at least* $\frac{n^{d-1}}{(d+2)^{d-1}} - 1$ *facets whose dual graph is a cycle.*

Proof. Our set of vertices is $V = \mathbb{F}_q \times [d + 2]$. That is, we have as vertices the elements of $\mathbb{F}_q$ but each comes in $d + 2$ different "colors". In the sequence $(u_i)_{i\in\mathbb{N}}$ of Theorem 5 we color its terms cyclically. That is, call

$$v_i = (u_i,\ i \mod (d + 2)).$$

Let C be the simplicial complex consisting of the intervals of length d in the sequence $(v_i)_{i\in\mathbb{N}}$. That is, we let:

$$F_i = \{v_i, \dots, v_{i+d-1}\}, \qquad C = \left\{F_i : i \in \{1, \dots, \operatorname{lcm}(q^{d-1} - 1, d + 2)\}\right\}.$$

Observe that the sequence $\{F_i\}_{i\in\mathbb{N}}$ is periodic of period $\mathrm{lcm}\{q^{d-1}-1, d+2\} \geq q^{d-1}-1 = \frac{n^{d-1}}{(d+2)^{d-1}} - 1$. Also, by construction, $\mathrm{G}(C)$ contains a Hamiltonian cycle. We claim that, in fact, $\mathrm{G}(C)$ equals that cycle.

For this, observe that ridges in C are of two types: some are of the form $\{v_i, \ldots, v_{i+d-2}\}$ and some are of the form $\{v_i, \ldots, v_j, v_{j+2}, \ldots, v_{i+d-1}\}$. We will study the facets that these types of ridges may belong to.

For a ridge $R = \{v_i, \ldots, v_{i+d-2}\}$ to be contained in a facet F we need the color of the vertex in $F \setminus R$ to be either $i-1$ or $i+d-1$ (modulo $d+2$).

Once we have the color c of the new vertex $v = (u, c) \in F \setminus R$, the recurrence relation (and the fact that the first and last coefficients of p are non-zero) gives us only one choice for u. Thus, R is only contained in the two contiguous facets F_{i-1} and F_i.

The same argument applies to a ridge $\{v_i, \ldots, v_j, v_{j+2}, \ldots, v_{i+d-1}\}$. Now the color of the new vertex must be $j+1 \mod d+2$ and the recurrence relation (and the condition $a_i! = 0 \forall i = 1 \ldots d-1$) implies the vertex to be precisely v_{j+1}. □

Proof of Theorem 4. Delete a facet in the complex C of Theorem 6. □

A complex whose dual graph is a path, such as the one in this proof, is called a *corridor* in [19]. It is a general fact that the maximum diameter $H_{\mathrm{PSC}}(n, d)$ is always attained at a corridor ([19, Corollary 2.7]). That is to say, $H_{\mathrm{PSC}}(n, d)$ equals the maximum length of an induced path in the *Johnson graph* $J_{n,d}$: the dual graph of the complete complex of dimension $d-1$ with n vertices. Induced paths in graphs are sometimes called *snakes*. In this language Theorem 4 can be restated as:

Theorem 7. *There is a constant $c > 0$ such that for every fixed d and sufficiently large n the Johnson graph $J_{n,d}$ contains snakes passing through a fraction c^{-d} of its vertices.*

A stronger statement is known for the graph of a d-dimensional hypercube: it contains snakes passing through a positive, independent of d, fraction of the vertices [1].

3.2 Pseudo-manifolds

Theorem 8. *For every strongly connected pure $(d-1)$-dimensional simplicial complex (that is, a PSC with connected dual graph) with n vertices and diameter δ there is a $(d-1)$-dimensional pseudo-manifold without boundary with $2n$ vertices and diameter at least $\delta + 2$.*

Let C be the simplicial complex in the statement and V its vertex set. By [19, Corollary 2.7] there is no loss of generality in assuming that C is a *corridor*. That is, its dual graph is a path, so its facets come with a natural order $F_0.\dots,F_\delta$.

We now construct a simplicial complex C' in the vertex set $V' = V \times \{1,2\}$. For a vertex $v \in V$ we denote v^1 and v^2 the two copies of it in V', and refer to the superscripts as "colors". Let a_i and b_i be the unique vertices in $F_i \setminus F_{i+1}$ and $F_i \setminus F_{i-1}$, respectively. (For F_0 and F_δ we choose a_0 and b_δ arbitrarily, but different from b_0 and a_δ). We define C' as the complex containing, for each F_i, the 2^{d-1} colored versions of it in which a_i and b_i have the same color. The diameter of C' is at least the same as that of C. Let us see that C' is almost a pseudo-manifold:

- If a ridge R in C' is obtained from a colored version of F_i by removing a vertex v different from a_i or b_i, then the only other facet containing R is the copy of F_i in which the color of v is changed to the opposite one. This is so because the "uncolored" version of R is a ridge of only the facet F_i of C, by assumption.
- If a ridge R in C' is obtained from a colored version of F_i $(i < \delta)$ by removing a_i then the only other facet containing R is obtained by adding to it the vertex b_{i+1} with the same color as a_{i+1} has in R.
- Similarly, if a ridge R in C' is obtained from a colored version of F_i $(i > 0)$ by removing b_i then the only other facet containing R is obtained by adding to it the vertex a_{i-1} with the same color as b_{i-1} has in R.

That is, the only ridges of C' that do not satisfy the pseudo-manifold property are the 2^{d-1} colored versions of $R_1 := F_0 \setminus \{b_0\}$ and the 2^{d-1} colored versions of $R_2 := F_\delta \setminus \{a_\delta\}$, which form two $(d-2)$-spheres, (each with the combinatorics of a cross-polytope, the generalization of the octahedron). Choose a vertex a in R_1 and a vertex $b \in R_2$, different from one another (which can be done since $R_1 \neq R_2$). Consider the complex C'' obtained from C' adding to it all the colored versions of $R_1 \setminus a$ joined to $\{a^1, a^2\}$ and all the colored versions of $R_2 \setminus b$ joined to $\{b^1, b^2\}$. The effect of this is glueing two $(d-1)$-balls with boundary the two $(d-2)$-spheres we wanted to get rid off, so that C'' is now a pseudo-manifold. (Observe that the new ridges introduced in C'' all contain either $\{a^1, a^2\}$ or $\{b^1, b^2\}$ so they were not already in C'). $\square$

Remark. *In some contexts it may be useful to apply Theorem 8 to* closed corridors, *that is, pure complexes whose dual graph is a cycle. The construction in the proof works exactly the same except now C' is already a pseudo-manifold, with no need to glue two additional balls to it as we did in the final step of the proof.*

Putting together Theorems 4 and 8 we get the main result in this section:

Corollary 1. *For every $d \in \mathbb{N}$ there are infinitely many $n \in \mathbb{N}$ such that:*

$$H_{PM}(n,d) \geq \frac{n^{d-1}}{(d+2)^{d-1}} - 3,$$

where PM denotes the class of all pseudo-manifolds.

Our construction also works for a particularization of pseudo-manifolds, called *duoids*[20]. A duoid is a pseudomanifold with no induced crosspolytopes. For duoids, the previous lower bound was quadratic in n.

4 Computational search for non-Hirsch topological prismatoids

The purpose of this second part is to find a non-Hirsch topological prismatoid by starting with the Matschke-Santos-Weibel prismatoid with 28 vertices and doing operations preserving its width while reducing its number of vertices.

We will use the metaheuristic method of simulated annealing modifying the initial example vie topological bistellar flips.

4.1 Bistellar flips

Bistellar flips are basic operation we perform in our complexes to transform them into hopefully simpler ones. The initial definition of a (topological) flip is as follows, first introduced in [16] and then used in [3] for sphere simplification:

Definition 10 (Bistellar flips in manifolds [3, 16])**.** In a pure simplicial complex $\mathcal{C}$ homeomorphic to a manifold (without boundary), a *bistellar flip* is a transformation of $\mathcal{C}$ defined by a pair (f, l) where $f \in \mathcal{C}$ and $l \notin \mathcal{C}$ is a minimal nonface of $\mathcal{C}$ (every subset of l is a face but l is not), and $\text{link}(f) = \partial(l)$ ($\partial(l)$ here represents the topological boundary, which for a face l coincides with the simplicial complex of proper faces of l). The changes produced on the facets of $\mathcal{C}$ are:

- Every face of $\mathcal{C}$ containing f is removed.
- Every subset of $f \cup l$ containing l but not containing f is inserted as a new face.

Observe that the definition implies $f + l = d + 1$, since $f \cup l \setminus \{v\}$ is a facet in $\mathcal{C}$ for every $v \in l$.

In a prismatoid, which is a manifold with boundary, we want to allow also boundary flips, so we are interested in flips of two types:

Interior flips These have exactly the definition above, except we need to require the new face l to be introduced to contain vertices of both bases of the prismatoid. Without this condition the flip introduces a face in the interior of our complex but with all vertices in the same boundary component, contradicting the requirement that the subcomplex induced by the vertices in each boundary component equals that boundary component.

Boundary flips The boundary of a prismatoid (or of any manifold with boundary) is itself a manifold without boundary. We want to allow for the flips in one of the boundary components to be considered flips in the prismatoid, but this only makes sense if all facets to be removed by the flip contain one and the same vertex v in the other boundary component. If this happens we can modify the prismatoid by performing a cone over a flip in the boundary. Observe that in this case $|f|+|l| = d$ instead of $d+1$, both f and l are contained in the same base. The vertex v in the other base is called the apex of the flip.

The following definition considers together these two types of flips:

Definition 11. A *bistellar flip* in a topological prismatoid $\mathcal{C}$ is a triple (f, l, v) such that f is a face, l is a minimal nonface, and either:

- $|f| + |l| = d + 1$ and $v = \emptyset$, in which case l is required to have vertices from both bases and $\text{link}(f) = \partial(l)$.
- $|f|+|l| = d$ and v is a vertex, in which case f and l are required to be contained in the base opposite to v and $\text{link}(f) = \partial(l)*v$. Here $*$ denotes the operation of *join* of two simplicial complexes, but in the case where v is a single vertex $\text{link}(f) = \partial(l)*v$ is just a cone over $\partial(l)$.

In both cases, performing a flip makes the following changes:

- Every face containing f is removed.
- Every subset of the $f \cup l \cup v$ containing l but not f is added as a face.

As a side remark, if $\mathcal{C}$ is a geometric simplicial complex and some additional geometric conditions on l are satisfied both cases of our flips are special cases of the usual *geometric bistellar flips* used in computational geometry and polyhedral combinatorics [18].

Only a boundary flip can modify the boundary. That means that only boundary flips can add or remove vertices, and only boundary flips can add new faces to the boundary.

One remark that is important for the implementation is that from the support of a flip (the set $f \cup l \cup v$ of vertices involved in the flip), we can determine if the flip is a boundary or an interior flip and recover the sets f, l and v:

- In a boundary flip, the support has a single vertex (v) in one of the components, and d in the other one, while in an interior flip it has at least two vertices in each: l has a vertex from each component by definition, and f has at least another from each base because the condition $\text{link}(f) = \partial(l)$, with $|f| + |l| = d + 1$, implies that f is an interior face.

- In both cases, the set $f \cup v$ equals the intersection of all facets of $\mathcal{C}$ contained in $f \cup l \cup v$. This allows us to recover f, and hence l, once we know v by the provious point.

Then, given a possible support u with $d + 1$ vertices (d vertices if we want to allow flips that introduce a new vertex, that is flips in which l is a single vertex and f a single boundary ridge) , we can check if it corresponds to a flip preserving the prismatoid definition:

1. First, u has to be the ustar of a ridge.

2. $\text{ustar}(f)$ must have $d + 1$ vertices (d vertices if we are adding a new one).

3. An interior flip can not add new faces to the boundary. That is, the bases must preserve their topology as spheres.

4. The corresponding l is not a face of $\mathcal{C}$.

Note that these conditions are necessary and sufficient for a flip to be valid in our context.

4.2 Simulated annealing

Simulated annealing is a very common metaheuristic for optimization problems, used when we have a search space and a "neighbourhood relation" between two feasible solutions. It has been used successfully in combinatorial topology to simplify simplicial complexes while preserving a condition (typically their homeomorphism type) [3]. It is also used in conjunction with other strategies to tackle the problem of sphere recognition [10].

The idea of simulated annealing is that we perform a random walk through the neighborhood graph of our feasibility space, but favouring moves that improve the objective function over moves that do not. There is a parameter, the *temperature*, regulating the probability assigned to each move as a function of how much it improves or worsens. At higher temperatures, the move selection is more random, and when the system cools down it converges to accepting only improving moves.

Then, areas of the graph with smaller values of the objective function will be more likely. As the temperature cools down, the random walk will focus on these areas, and make optimizations with more detail.

Simulated annealing requires some tweakings and adjustments for each particular problem:

- An appropiate objective function.
- A cooling schedule.
- A definition of the graph.

The first point requires a detailed analysis and is covered in the next subsection.

There is a lot of research on cooling schedules for each problem. It is known that SA converges to the global optimum for a particular cooling schedule [8], but it is too slow for any practical application. Since the best schedule depends on the problem, several adaptative schedules have been proposed too [9]. However, the most common approach is to define a geometric cooling schedule, of the form $T_t = t_0 * e^{st}$, where t_0 (initial temperature), s (cooling speed) and the number of iterations are chosen by hand. Since we are very fast at flipping the simplicial complex, we have chosen a slow schedule with a high number of iterations.

In our particular problem, the third point is already covered, every prismatoid is a state, and two prismatoids are neighbours if there is a flip connecting them. We just need to get random flips with uniform probability.

Note that uniform probability between the neighbours is very important, since SA works by defining custom probabilities for each flip. Modifying the "a priori" probability could make the search biased in unexpected ways.

4.3 The objective function

We initially had two independent goals with this project:

- Finding prismatoids of given dimension d and diameter $d + 1$.
- Finding prismatoids of dimension 5 starting with Matschke-Santos-Weibel's example, and reduce the number of vertices.

The first case is called `Plan_Z` in the source files, and it uses the number of minimal paths from base to base as objective function, starting from a crosspolytope.

The second case is far more interesting. We add a restriction of preserving the witdh after each flip, but we need an objective function that helps us removing vertices. Note that to remove a vertex we require to have a flip of type $(1, d)$, that is, a vertex whith $|f| = 1$. Then, the objective function to minimize is the number of vertices, plus a smaller function to push the algorithm in the right direction. Some heuristics we have implemented for the latter are:

A Number of facets.

B Geometric mean of the sizes of the ustars.

C Arithmetic mean of the sizes of the three vertices with the smallest ustars.

D Number of faces.

E Generalized mean for (relatively) large negative k ($k = -3$) of the sizes of the ustars.

The common idea in (most of) these functions is to make the algorithm to focus on the vertices that are easier to remove, that is, reducing the ustar of the vertex with the smallest ustar is a priority. Using just the minimum is not the best option since it will make the algorithm less sensible to reducing the ustars of other vertices that are not minimum.

We have obtained our best results with the last one. That is because the generalized mean with a negative parameter is heavily influenced by the smallest elements of the sample.

4.4 Data structures

Since we want to work with topological prismatoids, a type of simplicial complexes, we need a proper data structure with decent performance. It should have these operations:

- Construction from the list of facets.
- Check if a set of vertices is a face.
- Iterate through the maximal subfaces of a face.
- Iterate through the minimal superfaces of a face.
- Perform a flip.
- Compute the width of the prismatoid.
- Get a valid random flip (with uniform probability).

A natural way of implementing this data structure would be to have the whole Hasse diagram as a graph. That is, every face is a "node" and has a list of which nodes are subsets and which are supersets. It is also required to have some form of indexing to get which subsets of V are actually facets.

This approach has the potential disadvantage of storing every face. Some other data structures are discussed in [4].

Since we need to check for membership to the complex (see 10), and our complexes are small, we will take this approach, with some tweaking to reduce its memory load while improving its performance.

Taking a face as a set, iteration through maximal subsets is the same as iterating through the set, and remove a vertex each time. The hard part is to iterate through minimal supersets. Here is where the idea of the *ustar* is useful. If a face f has ustar u, every superfacet of f is of the form $f \cup \{v\}$ where $v \in u \setminus f$. Therefore, having the ustar of every face stored, it is very easy to iterate through supersets too. Thus, we implement the simplicial complex as a dictionary of pairs (face,ustar), indexed by the face.

A flip is implemented simply by removing the old faces and inserting the new inserted faces

There is no need to store the dual graph, because it is implicit in the complex. The neighbours of a face can be computed from the ustar of the corresponding ridge, even if the ridge is a boundary ridge. However, we do store, for each facet, the distance to the first base, and the number of paths achieving that distance. The reason is that we don't want to iterate through all the facets when performing a flip, we just want to update the values that have changed.

Therefore, when we perform the flip, the new facets are inserted into a queue, and the distances are updated by cascading through the prismatoid.

Finally, it is very important to have an unbiased generation of random flips. We imitate to some extent the techinque used in polymake [7]. In polymake, there is a set of pairs (f, l), called "options" satisfying some conditions for flipability, in particular conditions 1 and 2 of subsection 4.1. The flips are categorized by dimension of f. Since every flip has the ustar of a ridge as support, we planned to simplify this by using ridges instead of the pairs (f, l). However, several ridges may have the same ustar (making them correspond to the same potential flip). This makes this idea biased and some flips more likely than others.

To avoid this bias, we store in `options` the support instead of the ridge that has that support as its ustar.

This is very convenient, as it also makes it very easy to spot vertex-adding flips (which are represented by having a support of d vertices). And it is also very easy to update, because in the “fl-model”, after every flip, some potential flips may change their f and l while having the same support. And, since there is a bijection between a particular set of ustar of ridges and the set of flips (we have defined this bijection with the construction of (f, l, v)), it is easy to choose a random flip we just need to choose a random ustar and filter non-valid ones. So our approach is more stable, requires less changes to the data structure and it is a bit cleaner.

4.5 Implementation details

Along the course of the project we have developed several versions of the simulated annealing algorithm.

1. One written in pure C used bitmask techniques with a pool of faces, for fast addressing and modification. This limited the maximum number of facets, making it impossible to use it for high dimension. We did not implement boundary flips, and we used it to try to find non-Hirsch prismatoids starting with a nonHirsch one (in fact a cross-polytope, the dual of a hypercube).
2. A second one written in C++98, using the open-source project polymake [7] as a library. After severe problems with polymake's implementation of the graph data structure, we decided to implement our own simplicial complex structure from scratch.
3. Finally, we developed a version in C++11, using bitmask techniques from the first version and the versatility of C++ templates. This version can make boundary flips with great speed and has a correct implementation of everything needed for a simulated annealing strategy. As it is common for SA, it may require some tweaking to get good solutions.

I will focus on the last one, which can be found at my github [6]. Basically, it has three source files:

annealing.cpp The main file, implements the simulated annealing strategy.

prismatoid.hpp Is the header file for the prismatoid class, and includes some useful macros and functions for working with bitmasks.

prismatoid.cpp Implements the simplicial complex. It is conveniently partitioned as follows:

1. Constructors and functions to read and write prismatoids.
2. Functions dealing with choosing and executing flips.

3. Functions dealing with the dual graph, objective function and feasibility of the prismatoid.
4. Error handling.

The class `prismatoid` has some interesting attributes:

- `map<mask,mask> SC` represent the set of faces of the simplicial complex as a c++ map. Its first argument is the bitmask representing the face, and the second one is the union of all the faces in its star. Using this information, we can iterate through its subsets and supersets, easily traversing the complex in logarithmic time.
- `map<mask,il> dists` has, for every facet, its distance to the first base, and the number of shortest paths from the base to that facet.
- `map<mask,int> options` is the set of ustars of ridges. There is a bijection between the set of possible flips and options, as explained previously. The second argument is for reference counting, it represents how many ridges have this ustar. It speeds up the execution of a flip.

The most relevant source files can be found also as an appendix. There are some smaller scripts for polymake, verification and file manipulation, written in perl, c++ and python that are not included here.

4.6 Experimental results

Using our approach, we have found 105 non-Hirsch topological prismatoids improving the best (geometrical) non-Hirsch prismatoid previously known [15], which had 25 vertices. 5 of them have 16 vertices, (the minimum we have been able to get). Here is one of those. Vertices $0, 1, 2, 3, 4, 5, 6, 7$ are in one base and vertices a, b, c, d, e, f, g, h in the other one. The list of facets is arranged by “layers”, meaning by this the number of vertices in one and the other base:

(1,2,3,4,g)	(6,1,3,g,h)	(0,7,1,g,f)	(0,7,g,h,a)	(2,3,h,a,f)	(2,a,b,c,f)
(2,5,3,4,g)	(0,5,4,b,c)	(0,2,5,g,f)	(0,2,g,h,a)	(0,2,h,b,f)	(4,h,c,e,f)
(0,7,3,4,b)	(0,3,4,b,c)	(6,1,3,g,f)	(6,3,g,h,a)	(0,5,h,b,f)	(5,b,c,e,f)
(7,6,3,4,b)	(0,7,1,g,d)	(2,5,3,g,f)	(2,3,g,h,a)	(7,3,h,c,f)	(7,g,c,d,f)
(6,1,3,4,b)	(0,1,3,g,d)	(0,1,4,g,f)	(0,7,h,a,c)	(3,4,h,c,f)	(7,h,c,d,f)
(0,5,3,4,d)	(0,2,3,g,d)	(0,5,4,g,f)	(0,2,h,a,c)	(0,7,a,c,f)	(7,g,a,c,f)
(0,1,2,3,g)	(0,7,1,h,d)	(1,3,4,g,f)	(0,2,h,b,c)	(0,2,a,c,f)	(2,h,a,b,c)
(0,1,2,4,g)	(0,2,5,h,d)	(5,3,4,g,f)	(0,5,h,b,c)	(0,2,b,c,f)	(2,h,a,b,f)
(0,2,5,4,g)	(0,1,3,h,d)	(7,6,1,h,f)	(0,7,g,h,d)	(0,3,b,c,f)	(5,h,b,c,e)
(0,2,5,3,d)	(2,5,3,h,d)	(0,2,5,h,f)	(6,1,g,h,d)	(5,4,b,c,f)	(5,h,b,e,f)
(0,7,1,4,b)	(0,5,4,h,d)	(7,6,3,h,f)	(0,2,g,h,d)	(3,4,b,c,f)	(6,g,h,a,f)
(0,7,1,3,e)	(0,3,4,h,d)	(2,5,3,h,f)	(1,3,g,h,d)	(7,1,g,d,f)	(7,g,h,a,c)
(7,6,1,4,b)	(5,3,4,h,d)	(5,3,4,h,f)	(2,3,g,h,d)	(6,1,g,d,f)	(6,g,h,d,f)
(7,6,1,3,e)	(0,7,1,h,e)	(0,7,1,b,f)	(0,7,h,c,e)	(7,1,h,d,f)	(7,g,h,c,d)
	(7,6,1,h,e)	(7,6,1,b,f)	(0,5,h,c,e)	(6,1,h,d,f)	
	(7,6,3,h,e)	(0,7,3,b,f)	(7,3,h,c,e)	(5,4,h,e,f)	
	(0,1,3,h,e)	(7,6,3,b,f)	(3,4,h,c,e)	(5,4,c,e,f)	
	(6,1,3,h,e)	(6,1,3,b,f)	(0,7,g,a,f)		
	(0,5,4,h,e)	(0,1,4,b,f)	(0,2,g,a,f)		
	(0,3,4,h,e)	(0,5,4,b,f)	(6,3,g,a,f)		
	(0,7,3,c,e)	(1,3,4,b,f)	(2,3,g,a,f)		
	(0,5,4,c,e)	(0,7,3,c,f)	(6,3,h,a,f)		
	(0,3,4,c,e)				

Checking whether this prismatoid is polytopal or not seems out of reach computationally, at least within the scope of this master's thesis. (We mention ideas on how to try to do it in the conclusions section). But we have done two partial checks.

- We have checked that the two subcomplexes in the bases of the prismatoid are indeed 3-dimensional polytopal spheres, as they should if the whole thing is polytopal.
- We have checked that the complex obtained by filling-in a triangulation of each of the bases is a shellable 4-sphere.

Shellability of a complex is a necessary condition for polytopality, and has interesting topological implications.

Definition 12. A pure simplicial complex of dimension $d-1$ is *shellable* if there is an ordering of the facets such that the intersection of each facet with the complex consisting of the previous ones is pure $(d-2)$-dimensional (that is, a union of ridges).

Checking shellability is belived to be NP-complete, and general purpose methods (including the one implemented in polymake) failed to do it for this example. But a semiautomatized method based on first ordering the facets by layers and distance to the initial base, and then iteratively swapping facets that violated the shelling definition with later ones, succeeded. We believe this to be because our initial ordering was sufficiently close to a shelling. It is however interesting to note that no shelling order is compatible with the layers, because the union of the first three layers is not a ball, while every initial segment in a shelling of a sphere is a ball.

5 Conclusions

We have obtained simplified prismatoids preserving a property (*non d-step*, that is, its width is greater than d), using a common technique. 5 of them achieve the minimum, 16 vertices. We are interested in finding if these topological prismatoids correspond to a polytope. If one of these prismatoids is polytopal, according to Theorem 1, there is a non-Hirsch polytope in dimension 11 with 22 vertices.

Thus, an interesting idea for the future is to check if our prismatoids are polytopal. There is some research on the topic, and a necessary condition for polytopability is shellability. It is possible (but not trivial) to check the shellability of our prismatoids using their "layer" structure as a guide.

Another possibility to get a polytopal prismatoid is to repeat our procedure with *geometric flips*. That is, instead of only using topological information, we could keep track of the coordinates of each vertex and allow only flips that are realizable geometrically. Originally we did not consider this because of its computational complexity. But since topological flips are achieved in 150ms and (in the topological case) it is easy to improve the starting prismatoid, with the information we have now (a reasonable objective function and cooling schedule) it looks like a promising idea.

We could also repeat our simulated annealing strategy, starting from our smallest prismatoids, and trying to improve heuristics for polytopability while preseving the number of vertices.

If polytopal, these smaller examples of non d-step prismatoids could shed some light on the subject of the combinatorial diameter of polytopes. Studying them could help us understand the properties of combinatorial diameter, maybe even giving us information about the polynomial Hirsch conjecture.

References

[1] H. L. Abbott and M. Katchalski, On the snake in the box problem, *Journal of Combinatorial Theory, Series B* **44** (1988) 12–24.

[2] A. Altshuler. "The Mani-Walkup spherical counterexamples to the W v-path conjecture are not polytopal." Mathematics of operations research 10.1 (1985): 158-159.

[3] A. Björner, and F. H. Lutz. "Simplicial manifolds, bistellar flips and a 16-vertex triangulation of the Poincaré homology 3-sphere." Experimental Mathematics 9.2 (2000): 275-289.

[4] J.D. Boissonnat, and S. Tavenas. "Building efficient and compact data structures for simplicial complexes." arXiv preprint arXiv:1503.07444 (2015).

[5] F. Criado, F. Santos. The maximum diameter of pure simplicial complexes and pseudo-manifolds. Preprint, 2016. `https://arxiv.org/abs/1603.06238`

[6] `http://github.com/criado/tfm`

[7] E. Gawrilow and M. Joswig, Polymake: A software package for analyzing convex polytopes, in *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, DMV Sem. **29**, Birkhäuser, 2000, pp. 43–73. Software available at `http://polymake.org`

[8] V. Granville, K. Mirko, and J-P. Rasson. "Simulated annealing: A proof of convergence." IEEE transactions on pattern analysis and machine intelligence 16.6 (1994): 652-656.

[9] L. Ingber, et al. "Adaptive simulated annealing." Stochastic global optimization and its applications with fuzzy adaptive simulated annealing. Springer Berlin Heidelberg, 2012. 33-62.

[10] M. Joswig, F. H. Lutz, and M. Tsuruga. "Sphere recognition: Heuristics and examples." arXiv preprint arXiv:1405.3848 (2014).

[11] G. Kalai, *Combinatorics with a Geometric Flavor*. In "Visions in Mathematics, GAFA 2000 Special volume, Part II", N. Alon, J. Bourgain, A. Connes, M. Gromov, V. Milman eds., 2010, pp. 742–791.

[12] G. Kalai et al., *Polymath 3: Polynomial Hirsch Conjecture*, September 2010, http://gilkalai.wordpress.com/2010/09/29/polymath-3-polynomial-hirsch-conjecture.

[13] R. Lidl and G. Pilz, *Applied Abstract Algebra*, Springer New York, 1997.

[14] P. Mani, and D. W. Walkup. "A 3-sphere counterexample to the Wv-path conjecture." Mathematics of Operations Research 5.4 (1980): 595-598.

[15] B. Matschke, F. Santos, C. Weibel, The width of 5-dimensional prismatoids arXiv:1202.4701 Proc. London Math. Soc., Vol 110 (3) (2015), 647-672. DOI: 10.1112/plms/pdu064

[16] U. Pachner. "PL homeomorphic manifolds are equivalent by elementary shellings." European journal of Combinatorics 12.2 (1991): 129-145.

[17] F. Santos. A counter-example to the Hirsch Conjecture. *Ann. Math. (2)*, 176 (July 2012), 383–412. DOI: 10.4007/annals.2012.176.1.7

[18] F. Santos, Geometric bistellar flips. The setting, the context and a construction. arXiv:math.CO/0601746 In Proceedings of the International Congress of Mathematicians, Madrid, August 22-30, 2006 (Marta Sanz-Solé, Javier Soria, Juan Luis Varona, Joan Verdera, eds.), European Mathematical Society, 2006, Vol III, pp. 931-962. ISBN 978-3-03719-022-7

[19] F. Santos, Recent progress on the combinatorial diameter of polytopes and simplicial complexes, *TOP* **21**:3 (2013), 426–460. DOI: 10.1007/s11750-013-0295-7

[20] M. J. Todd. A generalized complementary pivoting algorithm, *Math. Programming*, **6**:1 (1974), 243–263. DOI: 10.1007/bf01580244.

A Source code.

Here is the source code for reference. An updated version can be found at my github [6].

A.1 annealing.cpp

```
#include "prismatoid.hpp"
#include <fstream>
#include <functional>
#include <iomanip>
#include <cmath>

double schedule(int k) {
  // "a ojo"-selected annealing schedule
  double t0=1000.0;
  return t0*pow(0.99997, k);
  /*
  return 1e-10;

  double t0=500.0;
  return t0/log(k+2);
  /**/
}

int main() {
  unsigned seed=chrono::system_clock::now().time_since_epoch().count();
  rng generator(seed);
  uniform_real_distribution<double> dist(0.0,1.0);
  auto dice=bind(dist,generator);
  double totaltime=0.0; int totalflips=0;

  flip fl; double record=28;

  for(int experiment=1; experiment>=1; experiment++) {
    int maxk=500000;
    double cost, oldCost, avgCost=0.0, bestCost=1e30, numPrisms=0.0;

    #ifndef PLAN_Z
      ifstream file("./28prismatoid"); prismatoid p(file); file.close();
    #else
      //prismatoid p(9);
      ifstream file("./outputs1/sol83"); prismatoid p(file); file.close();
    #endif

    for(int k=0; k<maxk; k++) {
      oldCost=p.cost();
```

```
    double t=schedule(k);

    numPrisms++; avgCost+=oldCost; bestCost=min(bestCost,oldCost);
    if(((k-1)%1000)==0) {

      cout<<"Experiment "<<experiment
          <<" ("<<setw(4)<<100*double(k-1)/maxk<<"%): "
          <<setw(5)<<" temp= "<<t<<": "
          << oldCost <<" "<<avgCost/numPrisms<<" "<<bestCost
          <<" flip time: "<<totaltime/totalflips/1000.0<<"us."<<endl;
      cout<<"  vertices:" <<countBits(p.base1|p.base2)
          <<" diameter:"<<p.distBase2.first<<endl;
      totaltime=0.0; totalflips=0;
      numPrisms=0.0; bestCost=1e30; avgCost=0.0;
    }

    auto start=chrono::steady_clock::now();
    while(true) {
      fl=p.execFlip(generator);
      cost=p.cost();

      swap(fl.f, fl.l);

      if(!p.feasible()) {
        p.execFlip(fl);
        assert(p.cost()==oldCost);
      }
      else break;
    }
    auto end=chrono::steady_clock::now();

    totaltime+=chrono::duration<double,nano>(end-start).count();
    totalflips++;

    double delta=cost-oldCost;

    if(delta>0.0 && dice()>exp(-delta/t)) {
      p.execFlip(fl);
      assert(oldCost==p.cost());
    }

    t=schedule(k);
  }

  ofstream index("./outputs/index", ofstream::app);
  index<<"Experiment "<<experiment
       <<" ("<< oldCost
       <<") Vertices: "<<countBits(p.base1|p.base2)
       <<" Diameter: "<<p.distBase2.first<<endl;
```

```
    index.close();

    ofstream sol("./outputs/sol"+to_string(experiment));
    p.write(sol);
    sol.close();
  }

  return 0;
}
```

A.2 prismatoid.hpp

```
#ifndef PRISMATOID
#define PRISMATOID     // Decaf ninja style.  Bit ninjustu but with class.
                       // My humble interpretation of what SimplicialComplex.hpp
#include <map>         // should be.
#include <vector>      //
#include <algorithm>   // Author:  Francisco "Paco" Criado
#include <iostream>    //
#include <fstream>     // I think my coding style is evolving into something
#include <iomanip>     // more readable after this.
#include <queue>       // Brief reminder:  A simplex has dim vertices.
#include <ctime>
#include <chrono>
#include <random>
#include <cstdlib>
#include <set>
#include <string>
#include <bitset>
#include <cassert>
#include <cmath>

#define N 14
#define LAYER2 ((1<<N)-1)
#define LAYER1 (((1<<N)-1)<<N)

//PLAN_A is minimizing the facets
//PLAN_B is minimizing the geometric mean of card(ustar(v))
// (Actually it just uses the product)
//PLAN_C is minimizing the ustar of the smallest vertex
//PLAN_D minimizes the number of faces
//PLAN_E is generalized mean (k=3)
//PLAN_Z should be starting with crosspolytope

//define DEBUG
#define PLAN_E

```

```
using namespace std;

typedef unsigned int mask;
typedef pair<int, long long unsigned> il;
typedef vector<int> vi;
typedef default_random_engine rng;

class flip { public:               // f:  face to remove
  mask f,l,v;                      // l:  maximal face to add
};                                 // v:  apex of the cone (frontier flip)
class prismatoid { public:
  //////////////////////////////////////////////////////////////////////////////
  // Public stuff (that should be private)
  //////////////////////////////////////////////////////////////////////////////

  mask base1, base2;               // The actual vertices in each base.
  int dim;                         // A facet has dim vertices
  int numFacets;                   // Number of facets.
  bool changeBases=true;           // Can we add/remove vertices?

  map<mask,mask> SC;               // Face and ustar of face
  map<mask, il> dists;             // Pair <distance, width> of each facet
  il distBase2;                    // (distance,width) for base2
  set<mask> adyBase2;              // The set of the ridges adyacent to base2

  map<mask,int> options;           // Set of ustars of ridges.  That's it.
                                   // Frontier flips have dim vertices.
                                   // The int represents reference counting.

  //////////////////////////////////////////////////////////////////////////////
  // Public methods
  //////////////////////////////////////////////////////////////////////////////

  // S1:  Constructors and IO
  prismatoid(int _dim);            // Crosspolytope
  prismatoid(istream& input);      // Reads prismatoid from file
  void write(ostream& output);     // Writes prismatoid to file

  // S2:  Flippin' magic
  flip execFlip(rng& gen);         // These two update everything.
  void execFlip(flip fl);          // The first choses flip at random.

  // S3:  Costs and graph stuff
  double cost();                   // Cost of this prismatoid (various options).
  bool feasible();                 // Do we want this prismatoid?
  pair<vi,vi> statsForSantos();    // f-vector and layers

  // S4:  Dont panic
  bool everythingIsOK();           // Is everything OK?
```

```

  private:
  ///////////////////////////////////////////////////////////////////////////////
  // Private stuff
  ///////////////////////////////////////////////////////////////////////////////

  void cascadeFacets();           // Completes the construction from the facets

  void initOptions();             // Inits the options list
  void initGraph();               // Inits graph and dists
  void updateDists(queue<mask>& q);// Updates dists by cascading

  flip findFlip(rng& gen);        // Finds a flip or crashes tryin'
  bool checkFlip(mask u, flip& fl);// Checks the validity of u as option,
};                                // returns the flip fl (by reference)

/////////////////////////////////////////////////////////////////////////////////
// S0:  Ancient bit-jutsu techniques
/////////////////////////////////////////////////////////////////////////////////
inline uint countBits(mask i) {
  i= i-((i>>1)&0x55555555); i= (i&0x33333333)+((i>>2)&0x33333333);
  return (((i+(i>>4))&0x0F0F0F0F)*0x01010101) >> 24;
}

// for (mask x=firstElement(f); x!=0; x=nextElement(f,x))
inline mask firstElement(mask f) {return f&-f;}
inline void nextElement(mask f, mask& x) {x= ((x|~f)+x)&f;}

// mask x=0; do stuff  while(x=nextSubset(f,x), x!=0)
inline void  nextSubset(mask f, mask& x) {x= ((x|~f)+1)&f;}

inline void printMask(mask f) {
  for(int i=0; i<2*N; i++) if(((1<<i)&f)!=0) cout<<i<<" "; cout<<endl;
}
inline mask readMask() {
  string str; getline(cin, str); mask res=0;
  for(int i=0; i<str.size(); i++) res=2*res+((str[i]=='1')?1:0);
  return res;
}

inline bool in(mask a, mask b) {return !(a&(~b));}

#endif
```

A.3 prismatoid.cpp

```
#include "prismatoid.hpp"
```

```cpp

///////////////////////////////////////////////////////////////////////////////
// S1:  Constructors and IO
///////////////////////////////////////////////////////////////////////////////

// Assumes that in SC we have only facets of dim dim.  Builds everything else.
void prismatoid::cascadeFacets() {
  queue<mask> q; base1=base2=0; numFacets=0;

  for(auto& it: SC) q.push(it.first),
                    base1|= LAYER1 & it.first,
                    base2|= LAYER2 & it.first,
                    numFacets++;

  while(!q.empty()) {
    mask f=q.front(); q.pop();
    for(mask x=firstElement(f); x!=0; nextElement(f,x)) {
      SC[f^x]|=f; q.push(f^x);
  } }

  initOptions(); initGraph();
  #ifdef DEBUG
    assert(everythingIsOK());
  #endif

  cout<<"built"<<endl;
}

// Crosspolytope constructor
prismatoid::prismatoid(int _dim) {
  dim=_dim; SC=map<uint,uint>(); mask f; base2=(1<<dim)-1; base1=base2<<N;

  for(uint i=1; i<base2; i++) f=(i|((i<<N)^base1)), SC[f]=f;
  cascadeFacets();
}

// Constructor reading from file.  The format is:
// -first line:  dim, number of facets
// -next lines:  each line is a new facet.
// The vertices 0..N-1 are in base2 and N..2N-1 in base1 (N=10).
prismatoid::prismatoid(istream& input) {
  SC=map<mask,mask>();

  input>>dim>>numFacets;

  for(int i=0; i<numFacets; i++) { //readFacet?
    mask f=0, aux; for(int j=0; j<dim; j++) input>>aux, f|=(1<<aux);
    SC[f]=f;
  }
```

```
  cascadeFacets();
}

// Write prismatoid to file in the same format.
void prismatoid::write(ostream& output) {
  output<<dim<<" "<<numFacets<<endl;
  for(auto& it: SC) if(countBits(it.first)==dim) {
    for(int i=0; i<2*N; ++i) if((it.first&(1<<i))!=0) output<<i<<" ";

    output<<endl;
  }
}

//////////////////////////////////////////////////////////////////////////////////
// S2:  Flippin' magic.
//////////////////////////////////////////////////////////////////////////////////

// Inits the option set.
void prismatoid::initOptions() {
  options=map<mask,int>();
  for(auto &it:SC) if(countBits(it.first)==dim-1) options[it.second]++;
}

// Finds a move or crashes tryin'.
flip prismatoid::findFlip(rng& gen) {
  flip fl;

  uniform_int_distribution<int> dis(0, options.size()-1);
  auto origin = options.begin(); advance(origin, dis(gen));

  for(auto it= origin; it!= options.end(); ++it)
    if(checkFlip(it->first, fl)) return fl;
  for(auto it=options.begin(); it!=origin; ++it)
    if(checkFlip(it->first, fl)) return fl;

  assert(false); return fl;
}

#ifdef DEBUG
  int numflips=0;
#endif
// Makes the flip (f,l,v).
void prismatoid::execFlip(flip fl) {
  queue<mask> q; const mask f=fl.f, l=fl.l, v=fl.v, u=f|l|v;
  // Note that the star (within the support) for a new face is:
  // -If it has exactly one 0 in f, the support without that zero
  // -If it has more than one 0 in f, u

  #ifdef DEBUG
```

```
    assert(everythingIsOK());
  #endif
  mask x=0; do {

    // The forbidden facet
    if(x==(f|l)) continue;

    // The supersets of f are disappearing faces.
    if(in(f,x)) {
      if(countBits(x)==dim) dists.erase(x),--numFacets;
      if(countBits(x)==dim-1) {
        if(in(x,base2)) adyBase2.erase(x);
        if(--options[SC[x]]==0) options.erase(SC[x]);
      }
      SC.erase(x);
    }

    // The supersets of l are appearing faces.
    else if(in(l,x)) {
      SC[x]= (countBits(f&~x)==1)? ((f&x)|l|v): u;
      if(countBits(x)==dim) q.push(x),++numFacets;
      if(countBits(x)==dim-1) {
        if(in(x,base2)) adyBase2.insert(x);
        ++options[SC[x]];
    } }

    // Stuff that remain the same.
    else {
      if(countBits(x)==dim-1) if(--options[SC[x]]==0) options.erase(SC[x]);
      SC[x]&=~u; SC[x]|= ((countBits(f&~x)==1)? ((f&x)|l|v): u);
      if(countBits(x)==dim-1) ++options[SC[x]];
    }

  } while(nextSubset(u,x), x!=u);

  base1=SC[0]&LAYER1; base2=SC[0]&LAYER2; updateDists(q);

  #ifdef DEBUG
    if(!everythingIsOK()) {
      cout<<"Panic Attack at flip "<<numflips<<endl;
      printMask(f);
      printMask(l);
      printMask(v);
      assert(false);
    }
    numflips++;
  #endif
}

```

```
// Makes a random move.  Returns its (f,l,v).
flip prismatoid::execFlip(rng& gen) {
  flip fl=findFlip(gen); execFlip(fl); return fl;
}

// Allow a flip with support u if:
// - It is the ustar of a ridge (assumed by pertenence to options)
// - There's room to add a new vertex (when required).
// - It does not add faces to the frontier (unless it is a frontier flip)
// - It does not change the set of vertices (under changeBases==false)
// - The ustar of f has exactly dim+1 vertices
// - The corresponding l is not in the complex.
// Returns the flip by reference.
bool prismatoid::checkFlip(mask u, flip& fl) {
  mask f,l,v;

  if(countBits(u)==dim) { // Add a vertex to the support when required.
    mask newv, LAYER;

    if     (countBits(u&LAYER2)==1) newv= firstElement(LAYER1 &~base1);
    else if(countBits(u&LAYER1)==1) newv= firstElement(LAYER2 &~base2);
    else cerr<<"Error 841: Not enough cheese in buffer."<<endl;

    if(newv==0) return false; u|=newv;
  }

  f=u;
  for(mask x=firstElement(u); x!=0; nextElement(u,x))
    if(SC.find(u^x)!=SC.end()) f&=u^x;
  l=u^f;

  if      (countBits(u&base1)==1) v=(u&base1), f^=v;
  else if (countBits(u&base2)==1) v=(u&base2), f^=v;
  else v=0;

  // Interior flips should not add new frontier faces.
  if(v==0 && (in(l,base1) || in(l,base2))) return false;

  // Am I adding/removing a vertex?
  //if(!changeBases && (countBits(l)==1 || countBits(f)==1)) return false;

  // Correct size of the support
  if(v==0 && countBits(SC[f])!=dim+1) return false;
  if(v!=0 && countBits(SC[f])!=dim)   return false;

  // l must not be in SC.
  if(SC.find(l)!=SC.end()) return false;

  fl.f=f; fl.l=l; fl.v=v; return true;
```

```
}

//////////////////////////////////////////////////////////////////////////////
// S3:  Costs and graph stuff
//////////////////////////////////////////////////////////////////////////////

// Inits graph and dists
void prismatoid::initGraph() {
  adyBase2=set<mask>(); queue<mask> q;

  for(auto &it:SC) {
    if(countBits(it.first)==dim-1) {
      if     ((LAYER1 & it.first)==0) adyBase2.insert(it.first);
      else if((LAYER2 & it.first)==0) q.push(it.second);
  } }
  dists=map<mask,il>(); updateDists(q);
}


// Stuff for the computation of the diameter and width.
inline void relaxPair(il& me, il& other) {
  if(other.first+1<me.first)       me.first = other.first+1,
                                   me.second= other.second;
  else if(other.first+1==me.first) me.second+=other.second;
}
void prismatoid::updateDists(queue<mask>& q) {

  while(!q.empty()) {
    mask f=q.front(), f2; q.pop();

    if(countBits(base1&f)==dim-1) {
      if(dists[f]==il(1,1)) continue;
      dists[f]=il(1,1);

      for(mask x= firstElement(f); x!=0; nextElement(f,x))
        if(SC.find(f^x)!=SC.end())
          if(countBits(f2=SC[f^x]^x)==dim) q.push(f2);
    }
    else {
      // if(dists[f]==ii(0,0)) dists[f]=ii(201,0); // needed?
      il aux(200,0);
      for(mask x= firstElement(f); x!=0; nextElement(f,x))
        if(SC.find(f^x)!=SC.end())
          if(countBits(f2=SC[f^x]^x)==dim && dists.find(f2)!=dists.end())
            relaxPair(aux, dists[f2]);

      if(aux!=dists[f]) {
        dists[f]=aux;
        for(mask x=firstElement(f); x!=0; nextElement(f,x))
```

```
          if(SC.find(f^x)!=SC.end())
            if(countBits(f2=SC[f^x]^x)==dim) q.push(f2);
  } } }

  il aux(200,0);
  for(auto &it: adyBase2) relaxPair(aux, dists[SC[it]]);
  assert(aux.first!=1); distBase2=aux;
}

// Number of vertices, distance and width
double prismatoid::cost() {
  #ifdef PLAN_A
  return double(numFacets);
  #endif
  #ifdef PLAN_B
  mask vertices=base1|base2;
  double avg=0.0;
  for(mask x=firstElement(vertices); x!=0; nextElement(vertices,x))
    avg+=log(countBits(SC[x]));
  avg/=double(countBits(vertices));

  return exp(avg);
  #endif
  #ifdef PLAN_C
  mask vertices=base1|base2;
  vector<double> ustars;

  for(mask x=firstElement(vertices); x!=0; nextElement(vertices,x))
    ustars.push_back((double)countBits(SC[x]));
  sort(ustars.begin(),ustars.end());
  return ustars[0]+ustars[1]+ustars[2]+ustars[3];
  #endif
  #ifdef PLAN_D
  return SC.size();
  #endif
  #ifdef PLAN_E
  mask vertices=base1|base2;
  double avg=0.0, k=-3.0;
  for(mask x=firstElement(vertices); x!=0; nextElement(vertices,x))
    avg+=pow(countBits(SC[x])-dim,k)/double(countBits(vertices));

  return countBits(vertices)*600 + (pow(avg,1/k)+dim);
  #endif
  #ifdef PLAN_Z
  return (distBase2.first>dim)?-1e10:distBase2.second;
  #endif
}
bool prismatoid::feasible() {
  #ifndef PLAN_Z
```

```
    return distBase2.first>dim;
  #else
    return true;
  #endif
}

// f-vector and layers.
pair<vi, vi> prismatoid::statsForSantos() {
  vi fvector(dim+1), layers(dim+2);
  for(auto &it: SC) fvector[countBits(it.first)]++;
  for(auto &it: dists) layers[it.second.second]++;

  return make_pair(fvector, layers);
}

////////////////////////////////////////////////////////////////////////////////
// S4:  Don’t panic.  Please don’t panic.
////////////////////////////////////////////////////////////////////////////////

// Can I panic now?
#ifdef DEBUG
bool prismatoid::everythingIsOK() {

  // 1:  is SC a simplicial complex?
  //*
  for(auto& it: SC)
    for(mask x=firstElement(it.first); x!=0; nextElement(it.first,x))
      if(SC.find(it.first&~x)==SC.end()||!in(it.second,SC[it.first&~x])) {
        cout<<"I find your lack of queso annoying"<<endl;
        cout<<"face and ustar"<<endl;
        printMask(it.first);
        printMask(it.second);
        cout<<"son and ustar"<<endl;
        printMask(it.first&~x);
        printMask(SC[it.first&~x]);
        return false;
      }
  /**/

  // 1.5:  Pure simplicial complex
  //*
  for(auto &it: SC)
    if(it.first==it.second && countBits(it.first)!=dim) {
      cout<<"sssh, no tears. Only "<<numflips<<" dreams now"<<endl;
      return false;
    }
  /**/

  // 2:  Every ridge must be internal xor in a base.
```

```
//*
for(auto& it: SC)
  if(countBits(it.first)==dim-1)
    if((countBits(it.second)==dim+1) ==
       (in(it.first,base1) || in(it.first,base2))) {
      cout<<"Non-specified excuse at "<<numflips<<"! *flips table*"<<endl;
      printMask(it.first);
      printMask(it.second);
      return false;
    }
/**/

// 3:  adybase2 is in fact adybase2
// 4:  options is options
//*
map<mask, int> otherOptions; set<mask> otherAdyBase2;
for(auto& it: SC)
  if(countBits(it.first)==dim-1) {
    otherOptions[it.second]++;
    if(in(it.first, base2)) otherAdyBase2.insert(it.first);
  }
if(options!=otherOptions) {
  cout<<"Number of cows backflipping: "<<numflips<<endl;
  return false;
}
if(adyBase2!=otherAdyBase2) {
  cout<<numflips<<" people thinking in piranhas right now"<<endl;
  return false;
}
for(auto& it: adyBase2){
  assert(SC.find(it)!=SC.end());
  if(dists.find(SC[it])==dists.end()) {
    cout<<numflips<<" likes in Facebook"<<endl;
    printMask(it);
    printMask(SC[it]);
    assert(SC.find(SC[it])!=SC.end());
  }
}
/**/

// 5:  dists
//*
for(auto& it:SC)
  if(countBits(it.first)==dim) {
    mask f=it.first, f2; il aux(200,0);

    if(dists.find(f)==dists.end()) {
      cout<<numflips<<" weird errors yet to be invented"<<endl;
      return false;
```

```
      }

      if(countBits(f&base1)==dim-1) {
        if(dists[f]==il(1,1)) continue;
        else {
          cout<<numflips<<" bottles standing at the wall"<<endl;
          return false;
        }
      }

      for(mask x= firstElement(f); x!=0; nextElement(f,x))
        if(SC.find(f^x)!=SC.end())
          if(countBits(f2=SC[f^x]^x)==dim && dists.find(f2)!=dists.end())
            relaxPair(aux, dists[f2]);

      if(aux!=dists[f]) {
        cout<<"Please reset the Universe "<<numflips<<" times."<<endl;
        return false;
      }
    }
  il aux(200,0);
  for(auto &it: adyBase2) relaxPair(aux, dists[SC[it]]);
  if (aux!=distBase2) {
    cout<<numflips<<" heroes saving the day"<<endl;
    return false;
  }

  /**/
  return true;
}
#endif
```