[Re] Volume computation for polytopes: Vingt ans après

Andreas Enge

RESCIENCE C

Replication / Discrete Mathematics

[Re] Volume computation for polytopes: Vingt ans après

Andreas Enge[1]

[1]INRIA, Université de Bordeaux, CNRS, LFANT, F-33400 Talence, France

Edited by
Thomas Arildsen

Reviewed by
Dmitrii Pasechnik

Received
10 April 2020

Published
04 December 2020

DOI
10.5281/zenodo.4242972

Abstract This article endeavours to reproduce the experimental study of B. Büeler, A. Enge, K. Fukuda: *Exact Volume Computation for Polytopes: A Practical Study*, 2000, in which a variety of algorithms for volume computation are applied to a variety of different higher-dimensional polytopes. The original software is used on a modern machine to redo the computations. It turns out that due to Moore's law, running times go down, but the original conclusions are still valid.

Dedicated to Komei Fukuda, who taught me the values of free software and scientific ethics

1 The original study

1.1 Volume computation for polytopes

A convex polytope of dimension d can be given in two forms: by its *vertex* or $\mathcal{V}$*-representation* as the convex hull of n points $v_1, \ldots, v_n \in \mathbb{R}^d$; or by its *halfspace* or $\mathcal{H}$*-representation* as the bounded intersection of m halfspaces given by inequalities $a_i x \leqslant b_i$ with $a_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$, $i = 1, \ldots, m$, or in matrix notation $Ax \leqslant b$ with $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$. When both are available, this is called the *double description*, which also provides the incidence information which point v_j lies on the boundary of which halfspace given by the equality $a_i x = b_i$. The elements of the (unique) minimal $\mathcal{V}$-representation are called *vertices*; the boundaries $\{a_i x = b_i\}$ of the (unique unless the polytope is contained in a strict affine subspace) minimal $\mathcal{H}$-representation are called *facets*.

Algorithmically these representations are not equivalent, since the size of one may be exponential in the size of the other. For instance, the d-dimensional hypercube can be given by $m = 2d$ inequalities $-1 \leqslant x_j \leqslant 1$ for $j = 1, \ldots, d$, but it has 2^d vertices x with $x_j \in \{-1, 1\}$ for $j = 1, \ldots, d$. There is a notion of duality, which "swaps" vertices and facets. So the dual of a hypercube, a cross polytope, has the exponential number 2^d of facets and the linear number $2d$ of vertices.

The *volume* of a polytope is understood with respect to the standard measure of $\mathbb{R}^d$; for instance, the hypercube has volume 2^d. Algorithms for exact volume computation generally decompose the polytope into smaller ones, for which a volume formula is available. These can be *simplices*, a generalisation of 2-dimensional triangles or 3-dimensional tetrahedra, given by $d + 1$ facets or $d + 1$ vertices $v_0, \ldots, v_d$, which are the intersections of any subset of d facets; the volume of such a simplex is given by $\frac{1}{d!} \left|\det(v_1 - v_0, \ldots, v_d - v_0)\right|$. In the case of a decomposition into simplices, one speaks of a *triangulation*. For instance, by triangulating the facets and augmenting each of the $d - 1$-dimensional simplices by an interior point of the polytope, one obtains a *boundary triangulation*. Alternatively, there are *signed decompositions* into simplices, usually containing also points exterior to the polytope, such that by an inclusion–exclusion principle every point in the interior of every simplex is counted exactly once. Other methods decompose the polytope into "pyramids", the convex hull of a facet and an additional vertex, with volume $\frac{1}{d}$ times the height times the $d - 1$-dimensional volume of a facet. All these algorithms eventually recurse over the dimension, so for general polytopes they end up having an exponential complexity in the dimension.

Correspondence should be addressed to Andreas Enge (andreas.enge@inria.fr)
The authors have declared that no competing interests exist.
Code is available at https://doi.org/10.5281/zenodo.4293820 – DOI 10.5281/zenodo.4293820..
Data is available at https://doi.org/10.5281/zenodo.4293875 – DOI 10.5281/zenodo.4293875.
Open peer review is available at https://github.com/ReScience/submissions/issues/27.

1.2 Experimental results

The subject of [1] was to implement all algorithms we could find in the literature and to compare their suitability for different classes of polytopes. We also suggested practical improvements to some algorithms and developed a new one.
Table 1 from [1, §5.3] summarises our computations; it records for each algorithm (the columns) and each polytope (the rows, sorted by decreasing ratio n/m) the running time in seconds, as well as reasons why the computation may not have succeeded. The last two columns give the timings for switching from one representation to the other. The rough conclusion was that the "upper right" and "lower left" corners of the volume computation part correspond to good combinations. For detailed descriptions of the algorithms and the polytopes, see the original article [1].

					Triangulation				Signed decomp.				
					$\mathcal{V}$-rep.		$\mathcal{V}$- and $\mathcal{H}$-rep.		$\mathcal{H}$-rep.				
	$\frac{n}{m}$	d	m	n	Bnd	Del	CH	HOT	Lnd	Ld	rL	$\mathcal{H}\to\mathcal{V}$	$\mathcal{V}\to\mathcal{H}$
cube-9	28.4	9	18	512	25e3	270	82	4.0	.3	1.6	.3	1.1	12
cube-10	51.2	10	20	1024	**[c]	**[a]	940	18	.6	3.3	.5	2.7	36
cube-14	585	14	28	16384	**[c]	**[a]	**[c]	3300	13	90	10	88	3500
rh-8-20	56	8	20	1115	**[c]	**[b]	93	7.8	1.5	28	14	28	1900
rh-8-25	104	8	25	2596	**[c]	**[b]	430	27	3.5	80	120	80	4600
rh-10-20	109	10	20	2180	**[c]	**[b]	3000	63	5.2	90	31	91	9500
rh-10-25	309	10	25	7724	**[c]	**[b]	34e3	390	15	390	600	380	44e3
$CC_8(9)$	1.5	8	54	81	410	40	11	3.4	**[c]	75	140	72[d]	140[d]
$CC_8(10)$	1.43	8	70	100	1300	80	28	7.7	**[c]	210	880	200[d]	340[d]
$CC_8(11)$	1.38	8	88	121	3500	150	65	17	**[c]	550	4400	550[d]	800[d]
Fm-6	3	15	59	177	11e5	**[a]	25	12	**[c]	**[b]	6900[9]	13e3[d]	6600
ccp-5	0.3	10	56	16	.7	.5	.2	.2	**[c]	200	2900[6]	4.5	.7
ccp-6	0.09	15	368	32	520	150	43	23	**[c]	**[c]	**[c]	4400[d]	2000[d]
rv-8-10	0.42	8	24	10	.3	.2	.1	.1	**[c]	82	.2	16	.5
rv-8-11	0.2	8	54	11	.4	.1	.2	.1	**[c]	2100	79	33	.9
rv-8-30	0.007	8	4482	30	170	4.0	6.9	6.8	**[c]	**[c]	**[c]	7200	150
rv-10-12	0.34	10	35	12	.4	.2	.1	.2	**[c]	1300	.3	92	1.0
rv-10-14	0.08	10	177	14	1.9	.2	.2	.1	**[c]	**[c]	**[c]	510	5.1
cross-8	0.063	8	256	16	.5	.3	.2	.2	**[c]	2800	170	4.2	.5
cross-9	0.035	9	512	18	0.9	.5	.3	.3	**[c]	**[b]	2700	12	1.2

Table 1. Timing for the different methods and examples. [a]beyond memory limit, [b]volume computed incorrectly, [c]problem is intractable with this method, [d]'cdd' is faster by a factor of at least 100, [6,9]storage performed for 6 and 9 levels resp.

2 Software installation

The goal of this article, written in the context of the "Ten Years Reproducibility Challenge"[1], is to try and reproduce the results of [1], by running the old code on a current machine and comparing results. In fact, [1] is my first scientific article, written not ten, but more than twenty years ago. The first challenge is obviously to locate the old source code (and example data files) and to compile it with modern tools.

[1] https://rescience.github.io/ten-years/

2.1 Fast track

The volume computation software written during the project, called vinci, was published under the GPL and is still available on my web page [2], which also hosts the polytope data files. It is written in ANSI C, so I expected no problems redoing the computations, and indeed there were none: It is enough to download and uncompress the vinci source code and the polytope files, to type `make` and to run the generated binary. This could be the end of §2. However, it turns out that the version on the web is *not* the one used for carrying out the computations of [1]. For instance, the algorithm of column "Del" in Table 1, deemed numerically too unstable, has been removed before publication of the source code. So to meet the reproducibility challenge, I have attempted to get back as closely as possible to the historical situation when writing [1].

2.2 Historically informed performance

Software — The historical instrument, an HP 7000/735-99 server, has long gone; so the challenge is to run the historical software on a modern machine. First of all, where is the historical source code? The article [1] was submitted in January 1998, revised in August 1998, and published in a book with conference proceedings in 2000. Only the latest released version 1.0.5 of vinci is available on the web [2]. It contains a `ChangeLog` file that dates it to 2003, and gives 2001 as the release date for 1.0.1 (no date for release 1 is mentioned), which is clearly after writing the article. On my private hard disk I have kept an mbox file containing mails with the results of the batch submission system of the time, including the outputs of the vinci runs. In these, the software identifies itself as "VINCI - Version gamma as of" with three different dates, "27.09.1997", "18.10.1997" and "11.10.1997". On the same disk, there is indeed a file `vinci_gamma.tar.bz2`; uncompressing it produces files with time stamp October 12, 1997, a rather close match. Moreover, it turns out that the software identifies itself with the date "12.10.1997" defined in `vinci.h`.
In the following, I document precisely the steps taken to compile this source code as well as all auxiliary software.

```
$ wget https://zenodo.org/record/4293820/files/vinci_gamma.tar.bz2
$ tar xvf vinci_gamma.tar.bz2
$ cd vinci_gamma
$ make
vinci.c:750:1: warning: return type defaults to 'int' [-Wimplicit-int]
  750 | main (int argc, char *argv [])
      | ^~~~
```

Indeed, the return type `int` is missing in front of `main`, and by default the current gcc version 9.2.0 complains about it.

```
$ ./vinci square
```

prints 4, the volume of the hypercube in dimension 2, the files for which are included in the vinci tarball. So far, so good; the twenty years old software compiles and seems to run without problem.
According to the article, we need the auxiliary software packages lrs and qhull; they are used for computing a boundary triangulation (column "Bnd" of Table 1) or a Delaunay triangulation (column "Del"), respectively. The two binaries are compiled separately, put into the vinci directory and called from vinci via the C library `system` call; communication is done through input and output files. The article [1] provides URLs for the different programs, but the exact versions of the time were not recorded.
The current website of lrs[2] lists as the oldest available version `lrslib-0.4.0` of December 11, 2000, which is after our experiments. So the historically informed performance needs to compromise and rely on a slightly newer lrs version than available at the time.

```
$ wget http://cgm.cs.mcgill.ca/~avis/C/lrslib/archive/lrslib-040.tar.gz
$ tar xvf lrslib-040.tar.gz
```

[2] http://cgm.cs.mcgill.ca/ avis/C/lrslib/archive/

```
$ cd lrslib-040/
$ make
buffer.c:69:5: error: conflicting types for 'getline'
   69 | int getline(void)
      |     ^~~~~~~
In file included from buffer.c:6:
/home/enge/.guix-profile/include/stdio.h:616:18: note:
  previous declaration of 'getline' was here
  616 | extern __ssize_t getline (char **__restrict __lineptr,
```

The function `getline` is declared in `buffer.c` and used only there, and it clashes with a function of the same name in the C library `glibc-2.29`. Such a function is not part of the C standard, but a GNU extension[3]; running `git blame` on its source[4] shows that it has been around since at least 1995. So while the `makefile` of lrs hardcodes gcc as the C compiler, the binary was probably linked at the time against the C library of the HP system without that extension. The function `getline` is renamed to `mygetline`, and its three occurrences (declaration, definition and usage) are replaced in the file by the following commands:

```
$ wget https://zenodo.org/record/4293820/files/lrs-getline.patch
$ patch < lrs-getline.patch
```

where the content of the file `lrs-getline.patch` is given in Table 2.

```
diff -u lrslib-040/buffer.c lrslib-040/buffer.c
--- lrslib-040/buffer.c        2020-09-11 14:06:05.504948673 +0200
+++ lrslib-040/buffer.c        2020-09-11 14:07:01.532951018 +0200
@@ -9,7 +9,7 @@
 char *line;

 int maxline;
-int getline(void);
+int mygetline(void);
 void notimpl(char s[]);

 main(int argc, char *argv[])
@@ -38,7 +38,7 @@
         bufsize= -1;  /*upper index of buffer size*/
         count=-1;     /* count lines output "begin" before "end" minus 1*/
         counton=0;
-        while ( getline() > 0 )
+        while ( mygetline() > 0 )
         {
                 i=0;
                 if(strncmp(line,"end",3)==0) counton=0;
@@ -66,7 +66,7 @@
 }

 /* getline from KR P.32 */
-int getline(void)
+int mygetline(void)
 {
         int c,i;
         extern int maxline;
```

Table 2. `lrs-getline.patch`

Then compilation with `make` goes through. (It turns out that reproducing a reproducibility experiment is a tricky affair: Between writing the first version of this article and revising it a few months later, compilation of lrs actually stopped working; `gcc-10.2.0` complains about a lacking inclusion of `stdlib.h` and then about duplicate definitions, whereas `gcc-9.3.0` emits warnings, but does compile the project nevertheless.)
So build lrs (with a suitable compiler version) and copy the binary into the correct directory:

[3] Search for "getline" in the documentation at https://archive.softwareheritage.org/browse/content/sha1_git:c48e3e692f6f4a9c9dfd8e51ebb1ecf18e756e28/raw/?filename=stdio.texi

[4] https://sourceware.org/git/?p=glibc.git;a=blob;f=stdio-common/getline.c;h=9b1641f23e1ae527de54a206a7e7b3f49bead0f3;hb=HEAD

```
$ make
$ cp lrs ../
$ cd ../
```

(The current lrs version `lrslib-070` has also renamed the internal function; so here I have simply redebugged a problem that was already solved in a later version.)
The qhull website[5] distributes code from 2019 and `qhull-1.0` from 1993 according to the time stamps of the files of the tarball and their copyright notices. So it is either too old or too new. The git repository of the project provides some more information:

```
$ git clone https://github.com/qhull/qhull.git qhull-git
$ cd qhull-git
$ git tag -l
```

The last command prints tags ranging from v3.0 to v7.3.2, and then from 2002.1 to 2019.1. The file `src/Changes.txt` states that version 3.0 was released on February 11, 2001; at the time, we probably used version 2.4 of 1997, but not having its source code, I again compromised in my historically informed performance by using release 3.0

```
$ git checkout v3.0
$ cd src
$ make
$ cp qhull ../../
$ cd ../../
```

Now the `vinci_gamma` directory contains the three binaries `vinci`, `lrs` and `qhull`. The latter two can be tested on the square via

```
$ export PATH=.:$PATH
$ ./vinci square -m lrs
...
Using 'lrs' for computing a boundary triangulation.
...
Volume:  4

$ ./vinci square -m qhull
...
Running qhull with the options d i Q0 Qz.
precision problems
      2 coplanar half ridges in output
      1 coplanar horizon facets for new vertices
Output file of qhull opened.
2 pseudo-simplices to be computed.
...
Volume:   4.000000000000e+02
```

So it looks like we are operational for vinci.
Table 1 also contains timings for switching between the $\mathcal{V}$- and the $\mathcal{H}$-representations; the "easy" direction (from fewer vertices/facets to many facets/vertices) is computed by lrs, already installed above; the "hard" direction is computed by the additional external software pd ("primal-dual"), which is no more available at the web page given in the old paper. However, it can be found using a quick web search:

```
$ wget http://www.cs.unb.ca/~bremner/software/pd/pd.tar.gz
$ mkdir pd-build
$ cd pd-build
$ tar xvf ../pd.tar.gz
```

The time stamp of the extracted files is May 14, 2013, but the main C file gives as date November 20, 1997. Depending on the example (here, `rv_10_12.ine`), the computation may fail with an error message such as

[5] http://www.qhull.org/

```
 Mulint overflow for a*b
a=...b=... BASE: 10000     DIGITS:  250
 Maximum integer is BASE^(DIGITS-1)-1
Increase BASE and/or DIGITS and recompile!
```

From the code, it turns out we are in 32-bit mode; but larger constants are hard-coded for 64-bit arithmetic. To switch, one may add `-DB64` to `CFLAGS` in `makefile` as follows:

```
$ wget https://zenodo.org/record/4293820/files/primal-dual-64bit.patch
$ patch < primal-dual-64bit.patch
```

where the content of `primal-dual-64bit.patch` is reproduced in Table 3.
Then compile and copy the binary to the correct directory:

```
$ make
$ cp pd ../
$ cd ../
```

```
diff -u pd.old/makefile pd/makefile
--- pd.old/makefile     2020-09-16 16:03:22.724144656 +0200
+++ pd/makefile 2020-09-16 16:03:53.564145534 +0200
@@ -3,7 +3,7 @@
 # http://wwwjn.inf.ethz.ch/ambros/pd_man.html

 CC = gcc -Wall
-CFLAGS = -O -DNDEBUG -DOMIT_TIMES
+CFLAGS = -O -DNDEBUG -DOMIT_TIMES -DB64
```

Table 3. `primal-dual-64bit.patch`

With the patch applied, `BASE` becomes 10^9 and `DIGITS` becomes 112; so with or without the 64-bit patch, the maximal precision is about 1000 decimal digits, which is apparently not enough. Interestingly, the culprit is the input file (discussed in more detail in §2.2.2), which specifies a precision with the directive `digits 1000`. As a solution, one may multiply the number of digits in the file by 2 until the computation passes; this is necessary only for the "hard" direction using pd. For recomputing the corresponding entries of the table, the precision is thus increased as follows: For `rh_8_*.ext` and `rv_8_*.ine`, from 500 to 1000 digits; for `rh_10_*.ext`, from 500 to 2000 digits; and for `rv_10_*.ine`, from 1000 to 2000 digits.

Input files — It turns out that between the versions gamma and 1.0.5 of vinci, the requirements for the polytope input files changed. For algorithms working with the double description, both require the $\mathcal{V}$-representation in a `.ext`-file (for "extreme points") and the $\mathcal{H}$-representation in a `.ine`-file (for "inequalities"). But the older version also requires a `.icd`-file for the incidence information, while the newer one recomputes it on the fly from the other two files. The polytopes on the web [2] lack the `.icd`-files. These could be recreated using code from `vinci-1.0.5`, but luckily it turned out that I had also kept copied of the historical polytope files on my hard disk.

(Re-)debugging, or reverse debugging — The `-m` parameter of vinci specifies the method to use for volume computation; if it is not set, it uses the default `-m hhv` (for the gamma version, renamed to `-m hot` in release 1.0.5), corresponding to the column labelled "HOT" in Table 1. As seen above, this works on the square, but it fails already on a 4-dimensional hypercube with a segmentation fault. Using a gdb backtrace and comparing the offending function `tri_ortho` between the versions gamma and 1.0.5 reveals that indeed a null pointer is dereferenced; the bug is fixed by applying a patch as follows:

```
$ wget https://zenodo.org/record/4293820/files/vinci-dummy.patch
$ patch < vinci-dummy.patch
```

where the content of `vinci-dummy.patch` is reproduced in Table 4.

```
diff -u vinci_gamma.old/vinci_volume.c vinci_gamma/vinci_volume.c
--- vinci_gamma.old/vinci_volume.c      2020-09-16 17:31:16.512294710 +0200
+++ vinci_gamma/vinci_volume.c  2020-09-16 17:31:55.952295832 +0200
@@ -211,7 +211,7 @@
    rational         volume, *stored_volume;
    rational         distance, maxdistance = 0;
    boolean          i_balance = FALSE, store_volume = FALSE, compute_volume = TRUE;
-   T_Key            **dummy;
+   T_Key            *dummy;

    *V = 0;

@@ -227,7 +227,7 @@
          {  copy_set (face [d], &(key.vertices.set));
             key.vertices.d = d;
          }
-         tree_out (&tree_volumes, &i_balance, key, &stored_volume, dummy, key_choice);
+         tree_out (&tree_volumes, &i_balance, key, &stored_volume, &dummy, key_choice);
          if (*stored_volume < -0.5)  /* volume has not yet been computed and is -1 */
          {
             if (!G_OutOfMem) /* stored_volume points to a tree element where the volume */
```

Table 4. `vinci-dummy.patch`

Maybe the code worked at the time on the HP machine since, as its name indicates, the variable `dummy` is indeed not used after the call to `tree_out`; it is only required for a different volume computation method.
In 1997, we displayed timing information with two decimals; given Moore's law, this is not enough anymore, and the following patch switches to four decimals:

```
$ wget https://zenodo.org/record/4293820/files/vinci-print.patch
$ patch < vinci-print.patch
```

where the content of `vinci-print.patch` is shown in Table 5.

```
diff -u vinci_gamma.old/vinci.c vinci_gamma/vinci.c
--- vinci_gamma.old/vinci.c     2020-09-16 17:31:16.512294710 +0200
+++ vinci_gamma/vinci.c 2020-09-16 17:35:24.744301773 +0200
@@ -968,7 +968,7 @@

 #endif

-           printf ("\n\nTime passed with computation: %8.2f s\n", time_passed (0));
+           printf ("\n\nTime passed with computation: %9.4f s\n", time_passed (0));
            printf ("--------------------------------------------------------------------------------\n\n");

         }
```

Table 5. `vinci-print.patch`

Lawrence's method, a signed decomposition algorithm (corresponding to the columns "Lnd" and "Ld" in Table 1) uses a random hyperplane. In the original code, we tried to make it reproducible between different runs by calling `srand` with the fixed seed 4, then using `rand` to create several `int` values. However, the random numbers are not the same due to different values of `RAND_MAX`: $2^{31}-1$ on my modern 64-bit system, probably $2^{15}-1$ on the old 32-bit HP system (the old logs indeed show 15-bit random numbers, which is consistent with this hypothesis). This implies that the simplex volumes added and subtracted during the algorithm are not the same any more. Vinci prints the random numbers, and it would be possible to replace the calls to `rand` by using the fixed list of numbers of the time; given the numerical instability of the signed decomposition, where substractions may result in huge cancellations, I considered it was not worth the effort.
It turns out that an additional tiny patch is needed to reproduce the results for one of the volume computation algorithms. Lasserre's method requires to choose a pivot in a matrix; by default vinci uses a defensive partial pivoting strategy, choosing the largest pivot in absolute value, while the column "rL" of Table 1 was computed with an early abort strategy, choosing the first pivot larger than the value 0.01 of `MIN_PIVOT` if possible. There is no command line parameter for selecting the behaviour, so a trivial patch needs to be applied to `vinci.h`:

```
$ wget https://zenodo.org/record/4293820/files/vinci-pivoting.patch
$ patch < vinci-pivoting.patch
```

the content of which is reproduced in Table 6.

```
diff -u vinci_gamma.old/vinci.h vinci_gamma/vinci.h
--- vinci_gamma.old/vinci.h     2020-09-16 17:31:16.512294710 +0200
+++ vinci_gamma/vinci.h 2020-09-16 17:37:09.484304753 +0200
@@ -65,7 +65,7 @@
 #define QHULL_OPTIONS "d i Q0 Qz"
    /* location of external programmes with paths and options for qhull                */

-#define PIVOTING 1
+#define PIVOTING 0
 #define MIN_PIVOT 0.01
    /* for choosing a pivoting strategy whenever this is needed, e.g. for determinant  */
    /* computation                                                                     */
```

Table 6. `vinci-pivoting.patch`

This one-character change has an important influence on the running time, while still producing correct results; I initially forgot it, and to give an example, the running time of about 6s for rL on rh-10-25 in Table 8 rises to about 29s with partial pivoting. This is due to our strategy of dynamic programming in Lasserre's algorithm, where partial pivoting makes it less likely that volumes of subfaces may be retrieved from memory, as explained right above §5.1 in [1]. Pivoting also plays a role when computing a simplex volume as a determinant; but the article [1] mentions it only in the context of Lasserre's algorithm and not for triangulations. Indeed, a quick experiment with CH on rh-10-20 shows no noticeable difference in the running time. And looking at the output of the batch jobs used to produce Table 1 shows that partial pivoting is used for other algorithms than Lasserre's.
After all these little patches, it is time to build the vinci executable again:

```
$ make
```

2.3 Modern times

For good measure and to go with modern tools, I have also included vinci release 1.0.5 into GNU Guix[6], a modern, functional (in the sense of functional programming) GNU/Linux distribution with exact dependency tracking and the aim of easing reproducibility[7]. The full build recipe, added in git commit `b457f3cc16`, is shown in Table 7.
Besides some self-explaining metadata, the core content of the record are the fields `build-system` and `arguments`. To start with, the build system is chosen as `gnu-build-system`, which essentially runs

```
./configure && make && make check && make install
```

Of these four phases, only the second one, `make`, is actually kept. The check phase is disabled by the line `#:test? #f`. Installation is done "by hand", copying only the binary `vinci` into the output directory. The package depends on lrs, which is called `lrslib` in GNU Guix, via the `inputs` field. The configure phase is replaced by a phase in which the actual location of the `lrs` binary is coded into the `vinci.h` header file, instead of expecting it in the current directory.
GNU Guix has mechanisms for "going back in time"; the commands

```
$ guix pull --commit=b457f3cc16
$ guix environment --ad-hoc vinci
```

downgrade the Guix version on a machine to the moment where `vinci-1.0.5` was added to it and start an ephemeral environment with the vinci version of this commit. Unfortunately, `guix pull` modifies the user environment (otherwise said, it pollutes the environment with state, which is non-functional). Functional time travel is enabled by `guix time-travel`, which essentially combines the two previous commands into one without modifying state:

```
$ guix time-machine --commit=b457f3cc16 -- environment --ad-hoc vinci
```

[6] https://guix.gnu.org/
[7] See Konrad Hinsen's blog post at https://guix.gnu.org/blog/2020/reproducible-computations-with-guix/

```
(define-public vinci
  (package
    (name "vinci")
    (version "1.0.5")
    (source
     (origin
       (method url-fetch)
       (uri (string-append "https://www.math.u-bordeaux.fr/~aenge/software/"
                           "vinci/vinci-" version ".tar.gz"))
       (sha256
        (base32
         "1aq0qc1y27iw9grhgnyji3290wwfznsrk3sg6ynqpxwjdda53h4m"))))
    (build-system gnu-build-system)
    (inputs
     `(("lrslib" ,lrslib)))
    (arguments
     `(#:tests? #f                      ; no check phase
       #:phases
       (modify-phases %standard-phases
         (replace 'configure
           ;; register the lrs location in the config file
           (lambda* (#:key inputs #:allow-other-keys)
             (let* ((lrs (assoc-ref inputs "lrslib"))
                    (lrsexec (string-append lrs "/bin/lrs")))
               (substitute* "vinci.h"
                 (("#define LRS_EXEC      \"lrs\"")
                  (string-append "#define LRS_EXEC \"" lrsexec "\""))))
             #t))
         (replace 'install
           (lambda* (#:key outputs #:allow-other-keys)
             (let* ((out (assoc-ref outputs "out"))
                    (bin (string-append out "/bin")))
               (install-file "vinci" bin))
             #t)))))
    (home-page
     "https://www.math.u-bordeaux.fr/~aenge/?category=software&page=vinci")
    (synopsis "Volume computation for polytopes")
    (description
     "Vinci implements a number of volume computation algorithms for convex
polytopes in arbitrary dimension.  The polytopes can be given by their
V-representation (as the convex hull of a finite number of vertices), by
their H-representation (as the bounded intersection of a finite number of
halfspaces) or by their double description with both representations.")
    (license license:gpl2+)))
```

Table 7. Definition of the vinci package in GNU Guix

Of course it is not possible to travel to an alternative past: When the software versions of §2.2 were written, Guix did not yet exist, so there is no past in which they would have been available in Guix.
To make it possible to carry out the historically informed performance of §2.2 with modern tools, I added the older software versions to an additional "channel" of Guix, guix-past[8], which was introduced in the context of another article in this Ten Years Reproducibility Challenge [3, §2.3]. Channels are a way of defining "overlays" on top of the "official" Guix distribution. Using them requires a declaration in a configuration file. For instance, the guix-past channel is registered by putting exactly the following lines into `$HOME/.config/guix/channels.scm`:

```
(cons (channel
        (name 'guix-past)
        (url "https://gitlab.inria.fr/guix-hpc/guix-past")
        (introduction
         (make-channel-introduction
          "0c119db2ea86a389769f4d2b9c6f5c41c027e336"
          (openpgp-fingerprint
           "3CE4 6455 8A84 FDC6 9DB4  0CFB 090B 1199 3D9A EBB5"))))
      %default-channels)
```

Then

```
$ guix pull
```

updates GNU Guix (from `%default-channels`) to its latest version, and adds the `guix-past` channel. Finally

```
$ guix environment --ad-hoc vinci@0.97.10.12 \
  lrslib@4.0 primal-dual@0.97.11.20
```

creates an ephemeral environment in which the vinci, lrs and pd versions referenced in §2.2 are available.

3 Reproduced results

In line with the Ten Years Reproducibility Challenge, I have used the historically informed approach described in §2.2 to recompute Table 1. According to [1], the machine used at the time was an HP 7000/735-99 workstation with about 500MB of RAM. We did not record more detail; a web search reveals documentation about an HP 9000 (not 7000) 735/99 computing cluster[9], introduced in 1992, with a 32-bit PA-7100 CPU clocked at 99 MHz and a 30W power consumption. I now performed all computations on my laptop with an Intel Core i5-6300U CPU running at 2.4 GHz (with a boost mode up to 3.0 GHz) and a thermal design power of 15W[10] with 16GB of main memory. The results are recorded in Table 8. As in [1], I used a `MIN_PIVOT` of 0.01 for Lasserre's algorithm of column "rL" and partial pivoting for all other methods, cf. the discussion at the end of §2.2.3 on reverse debugging.
At first glance, the table looks quite similar to the old one: Volumes that were computed incorrectly are still incorrect; intractable computations are (mostly) still intractable; and while all running times are lower, the conclusions of [1] about which algorithms behave well or poorly on which classes of polytopes are confirmed.
Unlike in [1], all computations with Lawrence's formula in the degenerate case (column "Ld") fail. This may be due to bad interfacing with the used version of lrs, which is not the same as in [1]; but since already there we concluded that Lawrence's method is numerically unstable and should thus be avoided (at least in the way we implemented it, with a random hyperplane), I did not find it worthwhile to debug, and also did not try to recompute the examples that already failed in 1997.
A few entries in the table deserve special comment, since the computations can now be carried out on a machine with more memory. These are `cube-10` with lrs (column "Bnd") and qhull (column "Del"); the

[8] https://gitlab.inria.fr/guix-hpc/guix-past
[9] https://www.openpa.net/systems/hp-9000_735_755.html, https://www.hpmuseum.net/display_item.php?hw=431
[10] https://ark.intel.com/content/www/us/en/ark/products/88190/intel-core-i5-6300u-processor-3m-cache-up-to-3-00-ghz.html

	Triangulation				Signed decomp.				
	$\mathcal{V}$-rep.		$\mathcal{V}$- and $\mathcal{H}$-rep.		$\mathcal{H}$-rep.				
	Bnd	Del	CH	HOT	Lnd	Ld	rL	$\mathcal{H}\to\mathcal{V}$	$\mathcal{V}\to\mathcal{H}$
cube-9	180	3.3	1.2	.048	.0036	**[b]	.0020	.010	.10
cube-10	4400	5.7	16	.22	.0068	**[b]	.0033	.023	.26
cube-14	**[c]	**[a]	**[c]	35	.15	**[b]	.093	.56	63
rh-8-20	**[c]	**[b]	1.5	.11	.015	**[b]	.14	.26	19
rh-8-25	**[c]	**[b]	8.2	.47	.038	**[b]	1.1	.73	55
rh-10-20	**[c]	**[b]	42	.82	.044	**[b]	0.30	.89	110
rh-10-25	**[c]	**[b]	590	6.7	.15	**[b]	5.7	3.7	530
$CC_8(9)$	3.1	.43	.15	.044	**[c]	**[b]	1.1	.57	.94
$CC_8(10)$	10	.84	.39	.10	**[c]	**[b]	6.3	1.6	3.4
$CC_8(11)$	27	1.7	.88	.22	**[c]	**[b]	31	4.0	5.2
Fm-6	7800	**[a]	.32	.15	**[c]	—	83	100	180
ccp-5	.008	.005	.0004	.0003	**[c]	**[b]	37	.028	.001
ccp-6	4.4	1.8	.67	.32	**[c]	—	**[c]	22	4.4
rv-8-10	.004	.003	.0001	.0001	**[c]	**[b]	.0008	.21	.002
rv-8-11	.005	.002	.0001	.0001	**[c]	**[b]	0.89	.44	.003
rv-8-30	1.6	.042	.037	.034	**[c]	—	**[c]	110	1.6
rv-10-12	.004	.003	.0001	.0001	**[c]	**[b]	.0012	1.1	.003
rv-10-14	.019	.004	.0003	.0003	**[c]	—	**[c]	6.5	.015
cross-8	.005	.004	.0006	.0006	**[c]	**[b]	1.1	.026	.004
cross-9	.010	.001	.0011	.0011	**[c]	—	16	.091	.007

Table 8. Timing for the different methods and examples. [a]beyond memory limit, [b]volume computed incorrectly, [c]problem is intractable with this method

latter requires about 3GB of memory. Our implementation of Lasserre's algorithm (column "rL") uses a dynamic programming style approach to store and retrieve volumes of lower-dimensional faces, starting at dimension $d-2$, and going down to dimension 2 (since edges, of dimension 1, are always simplices, their length is easy to recompute and does not warrant wasting memory). For the complete cut polytope on five vertices, `ccp-5` of dimension 10, we needed in [1] to limit storage to six levels, from dimension 8 down to 3, while now we can also store 2-dimensional volumes. The same holds for the metric polytope `Fm-6` of dimension 15, where instead of storing face values of nine levels we can store all twelve levels from dimension 13 down to 2, using a bit less than 5GB of memory.

For a closer comparison, it is interesting to consider the factor by which the current computations are faster than in 1997. This is recorded in Table 9 for the combinations of polytopes and algorithms where this makes sense, that is, those that are computed correctly and with a running time that could be measured to two significant digits in both tables and, for Lasserre's method, with the same level of storage. Again, all numbers are rounded to two significant digits. Since numbers are divided that were already rounded to two significant digits, there is a total rounding error of up to 11%, but this should be enough for drawing some general conclusions.

The disparity between the numbers, ranging from 37 to 450, is somewhat surprising; in the following, I try to provide some explanations.

The results in the first column labelled "Bnd" are entirely computed by lrs. The discussion on `BASE` and `DIGITS` at the end of §2.2.1 shows that the software relies on its own implementation of multiprecision integers using machine integers; so it is reasonable to assume that the quite homogeneous factors in Table 9, between 100 and 140, measure the relative performance of integer arithmetic on the two machines. This is consistent with the upper half of column "$\mathcal{H}\to\mathcal{V}$", from `cube-9` to `Fm-6`, which is

	Triangulation				Signed d.			
	$\mathcal{V}$-rep.		$\mathcal{V}$- and $\mathcal{H}$-rep.		$\mathcal{H}$-r.			
	Bnd	Del	CH	HOT	Lnd	rL	$\mathcal{H}\to\mathcal{V}$	$\mathcal{V}\to\mathcal{H}$
cube-9	140	82	68	83	—	—	110	120
cube-10	—	—	59	82	—	—	120	140
cube-14	—	—	—	94	87	110	160	56
rh-8-20	—	—	62	71	100	100	110	100
rh-8-25	—	—	52	57	92	110	110	83
rh-10-20	—	—	71	77	120	100	100	86
rh-10-25	—	—	58	58	100	110	100	83
$CC_8(9)$	130	93	73	77	—	130	130	150
$CC_8(10)$	130	95	72	77	—	140	120	100
$CC_8(11)$	130	88	74	77	—	140	140	150
Fm-6	140	—	78	80	—	—	130	37
ccp-5	—	—	—	—	—	—	160	—
ccp-6	120	83	64	72	—	—	200	450
rv-8-10	—	—	—	—	—	—	76	—
rv-8-11	—	—	—	—	—	89	75	—
rv-8-30	110	—	190	200	—	—	65	94
rv-10-12	—	—	—	—	—	—	84	—
rv-10-14	100	—	—	—	—	—	78	340
cross-8	—	—	—	—	—	150	160	—
cross-9	—	—	—	—	—	170	130	—

Table 9. Factor gained in running times from 1997 to 2020.

also computed by lrs, while I have no explanation for the factors 340 and 450 in the lower half of column "$\mathcal{V}\to\mathcal{H}$", from `ccp-5` to `cross-9`, again obtained by lrs. The remaining results in the columns "$\mathcal{H}\to\mathcal{V}$" and "$\mathcal{V}\to\mathcal{H}$" are computed by pd with essentially the same multiprecision arithmetic, with a more spread variation of the factors; I do not know enough of the algorithm and implementation to venture an explanation.

The column "Del" is obtained by a mixture of a call to qhull for computing the Delaunay triangulation, and the computation of the resulting simplex volumes in vinci; both use double precision, so one might conclude that the rather homogeneous numbers measure the relative performance of floating point operations between the two machines. This is, however, also the case for "Lnd", with a somewhat larger gain on the modern machine. Lasserre's algorithm of column "rL" also makes heavy use of floating point arithmetic, not only for the volume computation formula, but also for the linear algebra when projecting faces onto affine subspaces. Additionally, it stores partial results for later reuse in a balanced search tree. So the higher factor in the "rL" column might be explained by a relatively better memory performance on the modern machine. However, also the algorithms behind the columns "CH" and "HOT" store partial results in a tree in memory, and there the factors are rather small. On the other hand, for the "combinatorial part" of going down by one dimension, they do not use floating point operations for projections, but symbolic intersection of faces with facets, both given by an ordered array of vertices specified by their labels of `int` type. So the relative performance of this operation may explain the consistently smaller factors in the two columns, except for the outlier `rv-8-30`. This polytope with 30 facets has by far the largest number of vertices, 4482, of all considered examples (cf. Table 1). So the combinatorial structure of the polytopes appears to not only have an influence on the relative performance of the different algorithms, but also on the performance of the same algorithm on different processors: While the factors do vary a lot over the complete table, they are quite similar inside a "block" of one algorithm applied to a fixed

class of polytopes.
Admittedly, these attempts at explanation are more speculation than proof; for a scientifically sounder comparison, one would have to carry out more detailed experiments, which would require access to the old machine.
All factors in Table 9 are (much) larger than the quotient of the nominal clocking speeds of the processors of about 24. Moreover, the energy efficiency of the computations has probably increased a lot: The thermal design power has been halved from the processor in the workstation to that in the laptop (while the figures for the complete system are unknown), and nevertheless the speed of the computations has gone up by a factor of about 100. This should be attributable to the miniaturisation of transistors as captured by Moore's law. Since more than 22 years have passed between the computations, even a conservative estimate of two years per "Moore cycle" leads to eleven cycles and a factor of $2^{11} \approx 2000$. Notice that this is very far from the factors recorded in Table 9, which is consistent with the general observation that Moore's law does not apply to the performance of serial code anymore, but that the increased number of transistors is used to place more computing cores onto the same die.
So in order to profit from more powerful computers, code needs to be parallelised. It looks at first as if this would not be too difficult for volume computation: By recursing over the dimension, the algorithms visit the polytope faces and perform a tree traversal. So different branches of the tree may be assigned to different processors. Since not all branches have the same size, a task based approach may prove to be fruitful. However, by storing and reusing partial results, the faster algorithms exploit the fact that the face lattice is not a tree, but a directed acyclic graph with cycles if directions are removed: Every edge is contained in at least two faces of dimension 2, for instance. In a fast parallel implementation, to profit from this structure would require additional communication between the computing cores.

4 Conclusion

Reproducing the results of Table 1 has been quite easy, due to the following observations.
All used software is free and readily available on the Internet. The task would have been easier if the software and example polytope files could have been published electronically alongside the article; or if all software projects (including my own…) had archived all releases on their websites[11]. Nowadays, developing source code in a publicly accessible, version controlled repository could also ease the task of finding a specific release in the future, assuming the technology (git, right now) remains sufficiently stable and in use or provides an upgrade path (as is available from subversion to git, for instance).
Secondly, all used software is implemented in standard C with a simple `makefile`. The choice of the C language seems to be optimal for reproducibility due to its ubiquity; compilers have been available for all platforms, notably gcc of the GNU project: The oldest version still available on the GNU ftp server[12] is release 1.3.5 from 1989. The build system of all used software is also very simple; again, GNU make is available on all platforms. (In the worst case, the C files could be compiled "by hand" and the resulting object files could all be linked together into an executable, the presence of `makefile` in vinci is a mere convenience.)
Once the software had been compiled, redoing the computations has not posed any problem either (on the contrary, the availability of more powerful machines has considerably reduced the waiting time!). However, one of the algorithms ("rL") has required patching a header file to select an option; it would have been easier if this had been realised by a command line option, prominently advertised next to the table of results.
I am pleased to see that while not all computations have been sped up by the exact same factor, our main observations of the time remain valid: The ordering of the algorithms by performance on the different polytope classes is still the same, making it possible to automatically select the best algorithm for a given polytope. Moreover the chosen example polytopes are still meaningful; even twenty years later, it is not possible to go much further due to the exponential complexity of the algorithms (and most likely of the problem itself). So this is an interesting case in which algorithms may remain relevant over a long period of time.

[11] This problem is addressed by the Software Heritage project, https://www.softwareheritage.org/.
[12] https://ftp.gnu.org/old-gnu/gcc/

References

1. B. Büeler, A. Enge, and K. Fukuda. "Exact Volume Computation for Polytopes: A Practical Study." In: **Polytopes — Combinatorics and Computation**. Ed. by G. Kalai and G. M. Ziegler. Vol. 29. DMV Seminar. Basel: Birkhäuser Verlag, 2000, pp. 131–154. URL: https://hal.inria.fr/hal-03029034/.
2. B. Büeler and A. Enge. `vinci`. 1.0.5. Distributed under GPL v2+. July 2003. URL: http://doi.org/10.5281/zenodo.4294009.
3. L. Courtès. "[Re] Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices." In: **ReScience C** 6.1 (2020), #12. URL: http://doi.org/10.5281/zenodo.3886739.