Searching for elliptic curves with high rank in the PARI/GP software package

Vinko Petričević
vpetrice@math.hr

PMF-MO Zagreb

December 15, 2022.

Let E be an elliptic curve.

It is well known that the group $E(\mathbb{Q})$ is a finitely generated Abelian group (Mordell 1922).

$$E(\mathbb{Q}) \cong E(\mathbb{Q})_{tors} \times \mathbb{Z}^r.$$

In 1928, Weil generalized it to Abelian varieties over number fields.

In 1978, Mazur proved that there are exactly 15 possible torsion groups for elliptic curves over $\mathbb{Q}$:

$$\begin{aligned} \mathbb{Z}/k\mathbb{Z}, &\quad \text{for } k = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, \\ \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/k\mathbb{Z}, &\quad \text{for } k = 2, 4, 6, 8. \end{aligned}$$

Kenku and Momose (1988) and Kamienny (1992) proved that in quadratic fields there can occur precisely the following 26 groups:

$$\begin{aligned} \mathbb{Z}/k\mathbb{Z}, &\quad 1 \leq k \leq 18, k \neq 17, \\ \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2k\mathbb{Z}, &\quad 1 \leq k \leq 6, \\ \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3k\mathbb{Z}, &\quad k = 1, 2, \\ \mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}. & \end{aligned}$$

Motivation

The questions concerning the rank of elliptic curves are much more difficult than those which concern the torsion group, and satisfactory answers are still unknown. For a long time, it was believed that the rank can be arbitrarily large, i.e. that for any $M \in \mathbb{N}$, there is an elliptic curve E over $\mathbb{Q}$ such that $rank(E) \geq M$. However, some recent papers provide different heuristic arguments on why the rank might actually be bounded.

Details on the record curves can be found on the web pages
`https://web.math.pmf.unizg.hr/~duje/tors/tors.html`
`https://web.math.pmf.unizg.hr/~duje/tors/torsquad.html`

In the last year, we have found curves with the same rank as the record for $E(\mathbb{Q})$ and torsion group $\mathbb{Z}/9\mathbb{Z}$.

In $E(\mathbb{K})$ we broke the record for torsion groups $\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$ and $\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$.

In the book Number theory of prof. Dujella, can be found formulas for elliptic curves with given torsion group.

The problem is that we need to calculate the rank for large number of curves. In various math packages there are commands that we can use to calculate the rank more or less well.

In the newer versions of PARI/GP software package, they added the command `ellrank` which in some situations calculate the rank of the elliptic curve very well.

All programs I will show you next, can be found at `https://web.math.pmf.unizg.hr/~vpetrice/radovi/graz22/`.

Stable versions can be found on web-side
`https://pari.math.u-bordeaux.fr/download.html`

Development version of PARI/GP can be found
`http://pari.math.u-bordeaux.fr/pub/pari/snapshots/`.
I will show how to install it on Linux:

```
wget http://pari.math.u-bordeaux.fr/pub/pari/snapshots/
    pari-2.15.1-5-g209df6ce3e.tar.gz

tar xf pari-2.15.1-5-g209df6ce3e.tar.gz
cd pari-2.15.1-5-g209df6ce3e/

./Configure --prefix=GPDIR --mt=pthread
make -j4 gp
make install

./gp
```

Since 2020. PARI has support for multithreading. If we want to use it, we write `--mt=pthread` or `--mt=MPI`. There is quich tutorial for multithreading `https://pari.math.u-bordeaux.fr/pub/pari/manuals/2.13.3/parallel.pdf`.

We can use static version, which is somewhat faster.

```
make install-bin-sta
```

```
Olinux-x86_64/gp-sta
```

More details on installing PARI can be found on slides `https://pari.math.u-bordeaux.fr/Events/PARI2020/talks/sources.pdf`.

There is quick tutorial `https://pari.math.u-bordeaux.fr/Events/PARI2022/talks/ellrank.pdf` for function `ellrank`. For example, lets try:

```
e = ellinit([1, 0, 0, -844569946329969667402928554315 5,
             94467055910851185410161129206761573021778530 25]);
ellrank(e)
ellrank(e, 1)
ellrank(e, 5)
```

PARI, like other modern math programs, we can easily use and effectively do quite advanced things, but such programs are not very fast.

One of the reasons is that it is an interpreter, unlike, for example, C, which is a compiler.

The good thing is that PARI has the ability to write part of the code in C.

Let's look at the following simple program (which calculates the sum of first 2^{25} positive integers):

```
ff(d)=my(s=0);{
  for(i=1,d,s=s+i);
  return(s);
}
print(ff(2^25)); quit
```

On my laptop it takes $\approx$ 4.8 sec. On our server `euler.math.hr` it takes about 2.45sec).
On my laptop in Wolfram Mathematica it takes almost 20 sec, while on our server magma takes about 2.4 sec.
Program written in C-u takes only 0.02 sec!!! (pay attention to loading the data, otherwise the compiler will calculate it during compilation).

The result is always the same: 562 949 970 198 528.

Development versions of PARI are somewhat slower. Normal (dynamically linked) multithreaded (running on one core) takes 6.8 sec, singlethreaded 6.3 sec, while statically linked takes about 3.6 sec in both cases.

Along with the PARI comes the tool `gp2c` with which we can transfer the `.gp` program to a C program and edit/compile it.

Within it we have the program `gp2c-run` with which we can start the `.gp` program. The program generates C code, compiles it automatically, and starts gp with those new commands loaded. Now the same program is executed in a little less than 1.5 seconds.

If we don't want *loss* of memory, start the program with the -g option, and it will create the so-called repile code, which frees the memory it was using. (by default it does not release it!)

If we look at the contents of the folder after starting the gp2c-run program, it also created some additional files (.run, .so).

From the gp (installed now, from this folder), we can run the contents of the `.gp.run` file, and use the newly compiled commands without recompiling.

However, there is still a big difference in the execution time of a program called directly from C and GP.

Let's look at another example first. Let's calculate n!.

To calculate $2^{17}!$, PARI takes about 2.5sec (the fastest version about 1.9sec)

An identically written C program executes again extremely quickly, but returns 0.

This happens because C by default calculates with numbers up to 2^{64} (that is, with built-in types it could do up to 2^{128}). However, the exact result is slightly higher than $10^{600\,000}$. In order to get an exact result for an arbitrarily large number, we need a library for working with large numbers.

Package `gmp` comes standard with Linux, and it can be installed on Windows as well. Now instead of `long` (or `__uint128_t`) we use `mpz_class`.

When we run this program in C, it takes about 1.4 seconds, and the result is correct.

Now let's go back to the first example. `gp2c-run`, in addition to running `.gp` programs, also supports running `.c` programs, so let's say since we know that the result will be *enough* small, we can replace the PARI data type `GEN` type with `long`.

A new program executes in 14ms, which is the same as in C.

The reverse is also possible, to call PARI commands from C, and to compile and link such a program directly from the terminal. We can call all PARI commands, and they will be executed as fast as in PARI (and depends on which version is currently installed, the C-program will also work in this version).

When looking for elliptic curves of high rank, given torsion groups, we first need to look at what such curves look like (for example in Dujella's book).

After choosing a family, in order not to search all the curves, it would be good to first *sieve* the good candidates (using Mestre-Nagao sums).

After we wrote the initial version of the program in PARI, there is a lot of room for acceleration. And for all torsion groups, the programs are very similar.

Let us look torsion group $\mathbb{Z}/9\mathbb{Z}$. The general form of an elliptic curve is $[A, B, C, 0, 0]$

$$\begin{aligned} A &= u2^3 - u1^3 + u1^2 * u2, \\ B &= -u1^2 * (-u2 + u1) * (u2^2 - u1 * u2 + u1^2) * u2, \\ C &= -u1^2 * (-u2 + u1) * (u2^2 - u1 * u2 + u1^2) * u2^4, \end{aligned}$$

where $u1$ and $u2$ are coprime numbers (i.e. $u1/u2 \in \mathbb{Q}$).

The current record is 4, and in order not to wait too long, rank 3 curves are also printed.
Let's see what one possible optimization would look like:

The program `z9pari1.gp` searches for all curves for $u1 + u2 \in [3..50]$. The calculation takes about 8 seconds(we usually calculated such things up to 5000, so the differences in time increase significantly).

Some things may be calculated unnecessarily (for example mod p reductions), so those parts could be thrown out. There we get about half a second of savings. `z9pari2.gp`
However, some things are unnecessarily calculated multiple times, so we can calculate and remember those parts in advance (`z9pari3.gp`) On a sample up to 50, you can't see the difference, it even works a little slower.

First of all, we can use multiple cores, and run a multi-threaded form (`z9pari4.gp`).
The longest calculation is calculated at the end, so we could change the order of calculation and leave the shortest for the end (`z9pari5.gp`).

In other torsion groups, apart from a_ps, we could also remember the Kronecker symbol, and if we have a multi-parameter family, it would be good to pre-calculate *gcd*s as well.

We can use the gp2c tool to transfer the entire program to C, and we can also copy only parts and use parts from C.

```
../GPDIR/bin/gp2c -g file.gp > file.cpp
```

It depends on our skills, we can optimize a good part of things in C++, and in order to successfully compile and link it, we need to write, something like this:

```
g++ -O3 -Wall -ffp-contract=off -fno-strict-aliasing -fPIC
-I/home/vpetrice/graz22/pari-2.15.1-5-g209df6ce3e/GPDIR/include
z9.cpp -fPIC -lc -lm
-L/home/vpetrice/graz22/pari-2.15.1-5-g209df6ce3e/GPDIR/lib
-lpari -lgmp -lgmpxx -o z9 -Wl,-rpath
/home/vpetrice/graz22/pari-2.15.1-5-g209df6ce3e/GPDIR/lib
```

For example, for comparison, just calculating sums up to 1000 in normal PARI (gp) takes 73 seconds, in statically linked (`gp-sta`) about 50 seconds, while in C it takes about 2 seconds (and slightly less than 3 seconds for initialization – which we can also speed up).

The entire program can also be written in C, but perhaps it would be simpler in this part to write the results to some file, and to rewrite the PARI program that only reads that data and calculates the rank of the corresponding curve.
Example `z9citaj.gp`

Thank you for your attention