3-1-2021

Mexican Folk Arithmetic Algorithm Makes Perfect Sense

Julio C. Urenda
The University of Texas at El Paso, jcurenda@utep.edu

Christian Servin
El Paso Community College, cservin1@epcc.edu

Olga Kosheleva
The University of Texas at El Paso, olgak@utep.edu

Vladik Kreinovich
The The University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-21-21a
Published in Julia Rayz, Victor Raskin, Scott Dick, and Vladik Kreinovich (eds.), Explainable AI and Other Applications of Fuzzy Techniques, *Proceedings of the Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'*2021, West Lafayette, Indiana, June 7-9, 2021, Springer, Cham, Switzerland, 2022, pp. 453-460

Mexican Folk Arithmetic Algorithm Makes Perfect Sense

Julio C. Urenda, Christian Servin, Olga Kosheleva, and
Vladik Kreinovich

Abstract Traditional algorithms for addition and multiplication – that we all study at school – start with the lowest possible digits. Interestingly, many people in Mexico use a different algorithm, in which operations start with the highest digits. We show that in many situations, this alternative algorithm is indeed more efficient – especially in typical practical situations when we know the values – that we need to add or subtract – only with uncertainty.

1 Formulation of the Problem

Standard arithmetic algorithms that everyone learns. At school, kids all over the world study the exact same addition, subtraction, and other arithmetic algorithms as everywhere in the world: we start with the lowest digits, add them. If the result is larger than 10, we carry a one to the next digit, add it to the result of adding the next digits of the two numbers, etc.

Julio C. Urenda
Department of Mathematical Sciences and Department of Computer Science
University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA
e-mail: jcurenda@utep.edu

Christian Servin
Computer Science and Information Technology Systems Department
El Paso Community College (EPCC), 919 Hunter Dr., El Paso, TX 79915-1908, USA
e-mail: cservin1@epcc.edu

Olga Kosheleva
Department of Teacher Education, University of Texas at El Paso, 500 W. University
El Paso, TX 79968, USA, e-mail: olgak@utep.edu

Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University
El Paso, TX 79968, USA, e-mail: vladik@utep.edu

For example, if we want to add 89 and 23, we first add the last digits 9 and 3, resulting in digit 2 and a carry:

```
   .
  89
+ 23
----
   2
```

and then add 8, 2, and the carried 1, resulting in 11

```
  89
+ 23
----
 112
```

How Mexico is different. Interesting, in Mexico, many parents teach their kids somewhat different "folk" arithmetic algorithms. One of us (JCU) who grew up in Mexico indeed learned different algorithms from his parents.

There are several such algorithms.

An example of a Mexican folk algorithm. One interesting Mexican folk arithmetic algorithm was recently described in a pedagogical paper [6]. In this algorithm, we start not with the lowest but with the highest digits. In the above example, we start by adding 8 and 2:

```
  89
+ 23
----
 10
```

Then, we add the next digits – and, since the result (12) is larger than 10, we add 1 to the previous digit, resulting – of course, in the same answer 112 as before:

```
  89
+ 23
----
 112
```

A similar idea can be (and is) used for subtraction.

So what is a problem? The main point of the paper [6] is that if a student uses a different algorithm, it does not necessarily mean that the student does not know to add or subtract: the teacher needs to ask the student how he or she (in this case, it was a she) came up with this answer, and make sure that the corresponding algorithm is correct.

From the pedagogical viewpoint, this probably solves the problem: the teachers are aware that students can use different algorithms, students are not frustrated – as they would be if they got a not-so-perfect grade for producing a correct answer by means of a correct (although not standard) algorithm. Implicit in this paper is

the assumption that the usual algorithm is optimal, and modified versions can be tolerated but definitely not recommended.

But our viewpoint is different. Most of the authors of this paper are computer scientists. As computer scientists, we are not surprised by the fact that there are many different algorithms for solving the same problem: this is a known fact in computer science; see, e.g., [2]. All computer science students learn that there are many different algorithms for sorting and searching, that there are many algorithms for subtraction and multiplication, etc. How do we select an algorithm? Usually, we select the most efficient one, i.e., the one that runs the fastest – otherwise, we would not be able to process as much information as computer currently do.

As computer scientists, we also know that the arithmetic algorithms that we all learn in elementary school are often *not* the most efficient ones, and that different, more efficient algorithms are implemented in computers. For example, to make subtraction more efficient, computers use 2's complement representation of negative numbers, in which – in contrast to the algorithms that we learn in elementary school – the same method applies both to positive and negative numbers. Similarly, to multiply two large numbers, a more efficient way is to use techniques based on Fast Fourier Transform; see, e.g., [4].

From this viewpoint, the very fact that Mexican folk algorithms have survived for so long, in contrast to other historic ethnic algorithms like the Russian peasant multiplication algorithm (see, e.g., [1, 4, 5, 7]), seems to us an indication that these algorithms may be – at least in some reasonable situations – more efficient than the algorithms that we all study at school.

And this is exactly what we show in this paper – that these algorithms *are*, in some reasonable sense, more efficient.

2 Let Us Look at How Computers Add Numbers

Why should we look at computers. As we have mentioned, when people design computers, they try to implement the most efficient algorithms.

So, to understand which algorithms are most efficient, a natural idea is to look at how computers perform the corresponding operations – i.e., in our case, how computers add and subtract.

So how do computers add: a simplified description. In this regards, usual introductory Computer Science textbooks provide a simplified version of how computers perform arithmetic operations. According to these textbooks, the main difference between how we add and how computer add is that computers use binary numbers.

Let us illustrate it on a simplified 5-bit example. So, if a computer wants to add $13 = 01101_2$ and $5 = 00101_2$, it first adds the last digits, getting a carry

```
      .
  01101
+ 00101
-------
      0
```

then it adds the carry and the next digits:

```
  01101
+ 00101
-------
     10
```

then it adds the next digits:

```
    .
  01101
+ 00101
------
    010
```

then it adds 1, 0, and the carry:

```
   .
  01101
+ 00101
-------
   0010
```

and finally, it adds 0, 0, and the carry:

```
  01101
+ 00101
-------
  10010
```

and gets the correct result $18 = 10010_2$.

We need to go beyond the simplified description. The above operation took quite a few steps already for 5-bit numbers, but in reality, modern computers deal with 64-but numbers. For such numbers, if each bit has to wait – as the simplified algorithm implies – for all previous bits, it will take us 64 sequential bit operations to add two numbers.

This was how very first computers were built, but this is definitely *not* how addition is performed now.

So how do computers actually add? In the actual computers, all bits are added at the same time, and all the carries are obtained at the same time; see, e.g., [3, 4]. Then, on the next iteration, carries are added to the result, etc.

In the above example, the first parallel step would result in the following:

```
  . .
  01101
+ 00101
-------
  01000
```

Then, we add the result 01000 and the binary number 01010 corresponding to carries: 0 means no carry, 1 means that there is a carry. By applying the same parallel procedure to the numbers 01000 and 01010, we get

```
  .
  01000
+ 01010
-------
  00010
```

Finally, we add the result 00010 and the binary number 10000 corresponding to carries:

```
  00010
+ 10000
-------
  10010
```

and we get the desired result $18 = 10010_2$.

This is really more efficient. Here, instead of 5 iterations, we use only 3, and the savings are even larger for 64-bit numbers.

How is this related to the Mexican folk algorithm. Computers are simple machines. Crudely speaking, they see and process one symbol at a time. For example, to understand a simple 1000×1000 picture, they need to process all million bytes one by one. In contrast, human perception is highly parallel: we look at the picture and we immediately see, e.g., that this is a cat (if it is indeed a picture of a cat).

Similarly, for addition, we can add two digits at the same time. From this viewpoint, whether we start with the lowest digits or with the highest digits, does not matter much. For example, if we want to add 89 and 23, we can first add all the digits, getting

```
 . .
  89
+ 23
----
  02
```

and then add – by adding all the digits at the same time – the result 02 and the number 110 corresponding to carries:

```
  02
+110
----
 112
```

This does not (yet) show that the Mexican folk algorithm is better, but it does show that the perceived importance of starting with the lowest digits is actually not that important – if we use an efficient algorithm instead of the one we learned in elementary school.

What about subtraction. A similar idea can be used for subtraction. Suppose that we want to subtract 89 from 112. We subtract each pair of digits, making all the needed borrowings:

```
 ..
 112
- 89
----
 133
```

then from the result 133, we subtract – again digit-by-digit – the number 110 corresponding to the borrowings:

```
 133
-110
----
 023
```

and we get the correct value 23.

3 Let Us Look at Practical Situations Where People Use Addition and Subtraction

Why do we need to look for practical situations. So far, we considered the problem as a pure theoretical one: we need to add or subtract numbers, and the question is how to best do it. But why do we need to add or subtract numbers?

By the way, this is not simply a rhetorical question: there are (unfortunately) many public figures who claim that arithmetic is not needed in real life and thus, should not be taught in schools (or, as many of them say it: "I was tortured in school by having to learn very complex and very useless stuff like fractions, I barely got a passing grade on that, this traumatized me for life, so let us stop torturing kids and ruining their lives").

An example of a practical situation. Let us give a real-life example of where subtraction is needed. You are buying something by paying cash (and yes, there are still many small places where you need to pay cash), you do not have the exact change, so you are expecting the change.

If you paid the amount a and you need to pay the amount $b < a$, then you need to get the difference $a - b$ back as change.

A reasonable idea. How would you prefer your change to be given to you? Take into account that In most countries, there are no 15-dollar or 15-peso bills, bills are either in tens, or in hundreds, etc.

In the past, in Russia, the sellers were *required* to first give you the largest nominations, then the smaller ones, etc.

For example, if you pay with a 100-rouble bill and the thing you bought costs 21.56 roubles (to be more precise, 21 roubles and 56 kopecks) – so that you should get 78.54 rubles change, then you are supposed to get first 70 roubles (e.g., $50 + 10 + 10$), then 8 roubles (e.g., $5 + 1 + 1 + 1$), then 50 kopecks (could be one coin) and, finally, 4 kopecks.

Why is this better than starting with kopecks? Because if you get distracted or simply forget about the last step, you still get the most of the change, while if this is done in the opposite order, and you are supposed to get 70 roubles last, you miss most of the change if distracted (as happened sometimes, which is why this requirement was imposed).

How is this related to the Mexican folk algorithm? At places where a machine computes the change automatically – as it is done now in most US stores – it does not matter that much how you compute the difference. But in some places – and in many places in the old days – the difference $a - b$ had to be computed orally.

From this viewpoint, it is indeed desirable to start with the largest digit, not with the smallest digit – exactly as the Mexican folk algorithm suggests.

Another practical example. A specific example of such a situation is a restaurant, where usually, at some point, after getting the largest nominations as a change, many customers suggest that the waiter keeps the remaining part of the change as a tip.

In this case, there is even more need for a Mexican-style algorithm, since in such situations, no one cares that much about the exact computation of a different: for example, I can leave 4 roubles and whatever kopecks remain as a tip, so there is no need to compute the exact amounts all the way to the lowest digits.

This brings us to another aspect of practical examples.

4 In Many Practical Situations, We Operate Under Uncertainty

In real life, situations when we need to compute the exact difference are somewhat rare. Much more frequently, we compute *approximate* differences – this is why this paper is submitted to a fuzzy conference.

For example, when we go shopping with cash, we rarely know exactly how much money exactly we have – for example, we may know that we have approximately 200 roubles. Then, when we see something that costs, say, 149 roubles, we need to check whether after buying this attractive (but someone expensive) object we will still have enough money to buy what we planned to buy from the very beginning.

In this case, we subtract 149 from "approximately 200". Of course, since we do not know the exact value of the original amount, it makes no sense to subtract exactly – "approximately 50" is a much more adequate answer than "approximately 51". In this case, not only we do not need to compute the lower digits, these digits will be misleading – letting us mistakenly think that the amount is between 50.5 and 51.5. In such situations, it is also more efficient to start with the higher digits – and not to go all the way to the lowest ones.

We may also be subtracting two approximate numbers: e.g., we have approximately 50 roubles left after downtown shopping, and we are thinking whether we have enough money left to have a nice lunch at a somewhat fancy downtown cafeteria. If, in our experience, lunch there costs approximately 40 roubles, we can risk it, but if our previous costs was about 50, we better not risk it: the difference may turn to out to be negative.

5 Conclusion

The Mexican folk arithmetic algorithms – where we start adding and subtraction from the highest digits, not from the lowest ones – may sound weird, but in many realistic situations, it is actually better than the traditional ones, especially in situations when we perform operations on fuzzily known quantities.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), and HRD-1834620 and HRD-2034030 (CAHSI Includes). It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478.

References

1. L. N. H. Bunt, P. S. Jones, and J. D. Bedient, *The Historical Roots of Elementary Mathematics*, Dover, New York, 1988.
2. Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2009.
3. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, Cambridge, Massachusetts, 2017.
4. D. E. Knuth, *The Art of Computer Programming. Vol. 2. Seminumerical Algorithms*, Addison Wesley, Reading, Massachusetts, 1969.
5. O. Kosheleva and K. Villaverde, *How Interval and Fuzzy Techniques Can Improve Teaching*, Springer Verlag, 2018.

6. G. Krause, L. A. Maldonado Rodríguez, and M. Adams-Corral, "Listening to and understanding students' algorithms", *Mathematics Teacher: Learning and Teaching PK-12*, 2021, Vol. 114, No. 2, pp. 139–141.
7. J. I. Vargas and O. Kosheleva, "Russian peasant multiplication algorithm, RSA cryptosystem, and a new explanation of half-orders of magnitude", *Journal of Uncertain Systems*, 2007, Vol. 1, No. 3, pp. 178–184.