White- and Black-Box Computing and Measurements under Limited Resources: Cloud, High Performance, and Quantum Computing, and Two Case Studies – Robotic Boat and Hierarchical Covid testing

Vladik Kreinovich[a], Martine Ceberio[a] and Olga Kosheleva[a]

[a] *University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA*

Abstract

In many practical problems, it is important to take into account that our computational and measuring resources are limited. In this paper, we overview main resource limitations for different types of computers, and we provide two case studies explaining how to best take this resource limitation into account.

Keywords

limited resources, computing and measurement under limited resources, cloud computing, high performance computing, quantum computing, and robotic boat, Covid testing

1. Formulation of the Problem

What are the goals of science and engineering. We want to know what will happen in the future, and we want to decide what to do to make this future better. In a nutshell:

- predicting the future is what science is about, and
- deciding what to do is more the area of engineering.

Need for measurements and computations. To predict the world's future and to decide how to improve this future, we need to understand how the world works. Some of the situations we can simply observe, but in most cases, we need to *measure* the values of different quantities.

Based on the measurement results, we deduce the algorithms describing dynamics of the world's processes – and formulate selection of the best action and/or device as an appropriate optimization problem. To use prediction algorithms and to solve the corresponding optimization problem, we often need to perform a large number of computations. For example, to reasonably accurately predict tomorrow's weather, we need to use high-performance computers.

Comment. Some problems require so many computational steps that we cannot solve them even on the fastest computers. For example, to predict in which direction a tornado will go, even the existing high-performance computers are not sufficient:

ICT&ES-2020: Information-Communication Technologies & Embedded Systems, November 12, 2020, Mykolaiv, Ukraine
vladik@utep.edu (V. Kreinovich); mceberio@utep.edu (M. Ceberio); olgak@utep.edu (O. Kosheleva)
0000-0002-1244-1650 (V. Kreinovich); 0000-0003-2587-4209 (O. Kosheleva)

- to predict where the tornado will do in the next 15 minutes, we need to spend several hours on a high-performance computer;
- by that time, the tornado has already moved and caused damage.

Our resources are limited. In principle, for the same computational problem, we may have different algorithms. Which algorithm should we select?

Traditional approach of theoretical computer science uses asymptotic analysis to compare different algorithms. For example, we analyze how the computation time grows with the size n of the input, and we select an algorithm that leads to the shortest possible time for large n. This analysis implicitly assumes that we can have an unlimited amount of resources:

- we can have inputs of arbitrary large size n,
- we can have an arbitrary large memory to store this data and to store all needed intermediate results,
- we can perform an arbitrary large number of computations, etc.

In practice, our resources are limited. It is therefore important to consider how this limitation affect computations and measurements.

What we do in this paper. In this paper:

- we provide a brief overview of the corresponding resource limitations, and
- we also present two case studies explaining, in some detail, how to take these resource limitations into account.

Comment. In most cases, we focus on our own related results – although, of course, in each of these problems, there are many other results and techniques. The reason for this selection is that the results that we cite are usually mathematically reasonably simple – this is why we selected them: for our own results, we know how we can simplify these results into easy-to-follow not-too-complex ones.

We believe that overall, these results provide a good introduction to the great variety of related problems.

White-box and black-box computing. In our analysis, we will distinguish between:

- traditional (white-box) computing and
- practically important black-box computing.

White-box computing corresponds to the usual analysis of theoretical computer science, when we know, step-by-step, what our algorithms are doing. This corresponds to:

- situations when we write our own code "from scratch", and
- situations when we only use open-source software.

In practice, however, we often use proprietary code, i.e., a code that its producer does not explain. We can use it, but we are not allowed to see the algorithm – we only see the results. Similarly, in defense applications, we may need to use classified code – e.g., the code used to control the corresponding devices.

2. White-Box Computing: Three Types of Situations and Related Resource Limitations

Three types of situations. In our analysis, we will distinguish between three different computational situations.

- One such situation is *regular-scale computing*, when we perform computations on a usual computer, be it a single laptop of an affordable computer cluster.
- Regular-size computations work well in many cases, but, as we have mentioned earlier, there are many practical situations where regular-scale computing is not sufficient, where we need *large-scale computing* – what is usually called *high-performance computing*.
- For some situations, the opposite is true: we do not need even the full computational power of a regular laptop, it is sufficient to use limited computational power of an easier-to-carry smaller device such as a cell phone or a smart watch. It is natural to call such situations *small-scale computing*.

Let us describe what are the typical resource limitations of these three types of situations – and what can we do to best takes these resource limitations into account.

Resource limitations of small-scale computing: energy. For small-scale computing – such as computing on cell phones – by definition, computational ability is not a problem. The main need for such small-scale devices comes from the fact that regular computers are not very portable: it is easier to carry a cell phone than a computer.

This portability is a problem: to perform computations, we need energy. For a more or less stationary device like a regular computer, this is not a problem: we just plug in the computer, and get the energy non-stop. For portable devices, energy is a problem: we can only plug it in once in a while, and in a small volume, we can only a limited amount of energy. So, for such devices, the main resource limitation is *energy*.

This require a serious change in algorithms – since most traditional algorithms are designed to minimize computation time, without taking into account energy limitations. In particular, this means that energy-needing auxiliary procedures like garbage collection – that periodically frees the no-longer-used part of computer memory – have to be performed only based on need, and not, as in usual computers, periodically or whenever this leads to faster computations; see, e.g., [1] and references therein.

For example, a regular laptop usually updates when it is idle, when it is not doing any computations:

- this preserves its computational speed when it is used, and
- no matter how many updates we make, it does not affect its computing abilities.

For portable devices, any such procedure uses some portion of stored energy, so it needs to be performed as rarely as possible.

The corresponding optimization of computations is a very challenging task, especially taking into account that some operating systems running on this devices are proprietary. Since the operating system uses a significant amount of energy, we thus face a partly-black-box problem even when the computational algorithms that we run on these devices are white-box ones; see, e.g., [2].

Resource limitations of regular-size computing: cost and cloud computing. For regular-size computing, computational ability is not a problem – otherwise, we would have needed a high-performance computer. As we have mentioned earlier, for such computers, energy is also not a problem: we just plug in. So is there any limited resource? Yes, a very mundane one: money.

The possibility to save money comes from the fact that the amount of needed computations changes with time:

- we have daily fluctuations, since most computations are performed at daytime,
- we have seasonal computations – e.g., a tax-advising company needs a lot of computations before the taxes deadline, stores need to perform more computations before Christmas, when many people are shopping for Christmas gifts, etc.

In all these cases, we have a choice:

- we can buy as many computers that we need at peak times, or
- we can buy fewer computers and at peak time, rent computation time on other computers.

Computations involving rented computation time are known as *cloud computing*; see, e.g., [3, 4, 5, 6, 7]. This name comes from the fact that we do not care where exactly these computations are performed – as long as they *are* performed. So, the possible locations of processors performing such computations do not form a clear shape, the resulting geographic shape is amorphous and often fast-changing, like a cloud in the sky.

So how much computing power should we buy and how much should we rent? Detailed answers to this question can be found in [8, 9, 10, 11]. In this paper, we show the optimal money-saving solution for the simplest case, when we have:

- full information about the costs and
- full information about our computational needs.

This means, in particular, that we know:

- the cost c_0 per computational step of in-house computations (including the corresponding part of the cost of buying the computers),
- the cost $c_1 > c_0$ per computational step of rented out-of-house computations, and
- probability distribution of the future computer needs x, which can be described, e.g., by the probability density function (pdf) $\rho(x)$.

Based on this information, we need to select the optimal amount x_0 of computational power that we should buy.

For each value $x > x_0$, we rent the missing amount $x - x_0$ of the computational power. Thus, the overall expected cost of all these computations is equal to

$$c_0 \cdot x_0 + c_1 \cdot \int_{x_0}^{\infty} c_1 \cdot (x - x_0) \cdot \rho(x)\, dx.$$

To find the optimal value x_0, we can differentiate this expression with respect to x_0 and equate the derivative to 0. As a result, we get the equation

$$F(x_0) = 1 - \frac{c_0}{c_1},$$

where $F(x)$ is the cumulative distribution function corresponding to the given pdf. This equation described the optimal amount of computing power that we should buy.

Comment. In [8, 9, 10, 11], one can also find:

- algorithms for solving this problem in more realistic settings, when we know the costs and needs with some uncertainty,
- algorithms for deciding when a long-term contract is beneficial, and
- algorithms that help companies providing cloud computing to find the optimal locations of their computers around the world.

Resource limitations of large-scale computing: energy again. For large-scale computing, as we have mentioned earlier, one of the main problems is that some important practical problems still cannot be solved on such computers – computer engineers are working on this. For the existing computers, what is the main limitation? Somewhat surprisingly, the main limitation is again energy, the same as for small-scale computing. Indeed, we can simply plug in a regular-size computer, but a usual company-wide or university-wide connection can only support a limited number of such plugged-in computers.

For a high-performance computer – which, crudely speaking, consists of hundreds and thousands of usual computers – we need to build a special power station, and even this may not be sufficient. How can we minimize the power needed for computations?

To answer this question, we need to go back to computer design:

- When we design a high-performance computer, we want to maximize the overall number of computations per second. From this viewpoint, a seemingly natural idea is to run all the processors at their maximum speed.
- It turns out that from the viewpoint of energy consumption, this is not the best arrangement.

To be more precise, if the power P_i used by each processor i was exactly proportional to this processor's computation speed f_i – i.e., to the number of computational operations per second, this would not matter:

- if $P_i = k \cdot f_i$,
- then, no matter how we distribute the task between processors, the overall power P would be similarly proportional to the overall number f of computational steps per second:

$$P = k \cdot f.$$

In practice, however, the linear dependence of P_i on f_i is only a rough approximation, the actual dependence $P_i = F(f_i)$ is non-linear.

As a result, instead of having all the processor work at their maximal speed, a better idea is:

- to have each processor work at the speed f_i at which the ratio $\frac{P_i}{f_i} = \frac{F(f_i)}{f_i}$ is the smallest possible, and
- to use more processors if needed.

If at some point, the task requires a smaller overall computation speed f, then, to minimize energy consumption:

- we make some processors idle, while
- others will work at their optimal speed (optimal in terms of energy consumption).

Quantum computing and its resource limitations. As we have mentioned several times, there is a practical need to increase computation speed beyond what we can do now. This need faces a fundamental physical limitation: that, according to relativity theory, the speed of all processes is limited by the speed of light. As a result:

- for a usual laptop of approximately 30 cm size, it takes 1 nanosecond for the light to travel from one side of the computer to another, and
- during this time, even the cheapest 4-GHz laptop performs 4 operations.

To drastically speed up processing, we need to drastically decrease the computer size – and this means to make memory cells drastically smaller. Already each cell consists of thousands of molecules. With further size decrease, we will get to the level of individual molecules – and at this level, we have to take into account physical laws of the micro-world – the law of quantum physics.

Computations that take quantum effects into account are know as *quantum computing*; see, e.g., [12]. One of the main features of quantum physics – see, e.g., [13, 14] – is that, in addition to classical states $s, s', \ldots$ – which, in quantum physics, are denoted by $|s\rangle$, $|s'\rangle$, etc. – we can also have *superpositions* of these states, i.e., states of the type

$$c_s|s\rangle + c_{s'}|s'\rangle + \ldots,$$

where c_s, $c_{s'}$, $\ldots$, are complex values for which

$$|c_s|^2 + |c_{s'}|^2 + \ldots = 1.$$

If we measure, in this superposition state, a classical variable v, then we get the value $v(s)$ with probability $|c_s|^2$, the value $v(s')$ with probability $|c_{s'}|^2$, etc. The sum of all these probabilities should add to 1 – which explains the above condition on the values c_s, $c_{s'}$, …

In particular, a quantum analogue of the usual bit – i.e., an element that can be in two possible states 0 and 1 – is a *quantum bit* (*qubit*, for short), i.e., the state of the type

$$c_0|0\rangle + c_1|1\rangle.$$

Similarly, a quantum analogue of a 2-bit state is a general 2-qubit state

$$c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle,$$

etc.

There are many efficient quantum algorithms, e.g., Grover's algorithm [15, 16, 12].

- This algorithm enables to find an element with a given property in an unsorted n-element array in time proportional to $\sqrt{n}$.
- On the other hand, for non-quantum algorithms, not to miss the desired element, we need to look at all n elements, so we need at least n computational steps, and $n \gg \sqrt{n}$.

Grover's algorithm requires n-qubit states. At this moment, the number of available qubits is the main limitation to quantum computing, the main limited resource. What happens if we only have $s < n$ qubits, i.e., if we can only implement s-qubit states? In this case, we can:

- divide the n-element array into n/s fragments of size s, and
- apply Grover's algorithms to each fragment; see, e.g., [17].

For each fragment, Grover's algorithm requires $\sqrt{s}$ steps, so the overall computation time will be $\frac{n}{s} \cdot \sqrt{s} = \frac{n}{\sqrt{s}}$. This amount:

- is still smaller than the time n needed in the non-quantum case, and
- decreases as the resource bound s increases (and reaches the Grover's value $t \sim \sqrt{n}$ when we reach the case $s = n$).

3. Black-Box Computing: Related Resource Limitations

General case. For black-box computing, when computations include calls to a "black-box" (proprietary or classified) program, a natural idea is to minimize the number of such calls:

- For proprietary programs, when we often need to pay for each class, this minimizes the overall costs.
- For classified programs, when each call is an extra vulnerability, this minimizes the risk.
- For programs that take a long time to compute, this also minimizes the overall computation time.

What can we compute in this manner? Usually, commercial software provides a turn-key solution to the corresponding problem, a solution that does not require additional programming. There exist commercial software for deep learning, for solving partial differential equations, etc.

This software produces the result $y = f(x_1, \ldots, x_n)$ of processing the inputs $x_1, \ldots, x_n$, but what is usually does not do is produce the accuracy of this result. To be more precise, most black-box packages compute the result implicitly assuming that the inputs are exactly known. In practice, the inputs come from measurements, and measurements are never absolutely accurate: the result $\tilde{x}_i$ of measuring the i-th input are, in general, different from the actual (unknown) value x_i of this input.

Thus, when we apply the black-box algorithm f to the measurement results $\tilde{x}_i$, the result $\tilde{y}$ of this computation will be, in general, different from the desired value $y = f(x_1, \ldots, x_n)$:

$$\tilde{y} = f(\tilde{x}_1, \ldots, \tilde{x}_n) \neq y = f(x_1, \ldots, x_n).$$

An important question is: how accurate is the computation result? In other words, how big can the difference $\Delta y \overset{\text{def}}{=} \tilde{y} = y$ be?

This question is very practically important. For example, if we are prospecting for oil and we learned that the given area contains $\tilde{y} = 200$ million tons, then:

- if this amount is 200 ± 20, this is good news and we should start drilling, but
- if it is 200 ± 300, then maybe there is no oil at all, and it is better to perform additional measurements before we spend a large amount of money on possibly useless drilling.

Let us formulate this problem in precise terms. Usually, the measurement errors $\Delta x_i \overset{\text{def}}{=} \tilde{x}_i - x_i$ are relatively small. So, taking into account that $x_i = \tilde{x}_i - \Delta x_i$, we can expand the expression

$$\Delta y = f(\tilde{x}_1, \ldots, \tilde{x}_n) - f(\tilde{x}_1 - \Delta x_1, \ldots, \tilde{x}_n - \Delta x_n)$$

in Taylor series and keep only linear terms in this expansion. As a result, we get the expression

$$\Delta y = \sum_{i=1}^{n} c_i \cdot \Delta x_i,$$

where $c_i \overset{\text{def}}{=} \dfrac{\partial f}{\partial x_i}$.

What do we know about the measurement errors?

- In some cases, we know the probability distribution of each measurement error Δx_i, and we know that these measurement errors are independent. In many cases, this distribution is normal (Gaussian) with mean 0 and known standard deviation σ_i.
- In other cases, we do not have any information about the probabilities, we only know the upper bound Δ_i on the absolute value of the corresponding measurement error: $|\Delta x_i| \le \Delta_i$; see, e.g., [18].

How can we use this information to find bounds on Δx_i?

Seemingly natural idea. If we know the values c_i, then, e.g., for the normal distribution, we can conclude that the linear combination $\sum\limits_{i=1}^{n} c_i \cdot \Delta x_i$ is also normally distributed, with mean 0 and standard deviation $\sigma = \sqrt{\sum\limits_{i=1}^{n} c_i^2 \cdot \sigma_i^2}$.

In case of upper bounds, when each measurement error Δx_i can take any value from the interval $[-\Delta_i, \Delta_i]$, the largest possible value of the sum $\sum\limits_{i=1}^{n} c_i \cdot \Delta x_i$ is attained when each term $c_i \cdot \Delta x_i$ in this sum is the largest possible.

- For $c_i \geq 0$, the term $c_i \cdot \Delta x_i$ is increasing, so its largest value is attained when Δx_i attains its largest value Δ_i. The corresponding largest value of the i-th term is $c_i \cdot \Delta_i$.
- For $c_i \leq 0$, the term $c_i \cdot \Delta x_i$ is decreasing, so its largest value is attained when Δx_i attains its smallest value $-\Delta_i$. The corresponding largest value of the i-th term is $-c_i \cdot \Delta_i$.

In both cases, the largest value of the i-th term is $|c_i| \cdot \Delta_i$, so the largest value Δ of the sum Δy is equal to

$$\Delta = \sum_{i=1}^{n} |c_i| \cdot \Delta_i.$$

Straightforward approach and its limitations. If we knew the values c_i, then we could use the above formulas and find the desired characteristic σ or Δ of the approximation error Δy.

The algorithm $f(x_1, \ldots, x_n)$ is given as a black box, we do not know the corresponding expression and thus, we cannot simply differentiate this expression. However, what we can do is use numerical differentiation, according to which, for sufficiently small value h_i, we have

$$c_i = \frac{\partial f}{\partial x_i} \approx \frac{f(\tilde{x}_1, \ldots, \tilde{x}_i + h_i, \tilde{x}_{i+1}, \ldots, \tilde{x}_n) - \tilde{y}}{h_i}.$$

This way, we get the desired expressions for σ and Δ – but we need to make $n + 1$ calls to f: the first call to compute $\tilde{y}$ and then n more calls to compute n partial derivatives c_i.

In many data processing algorithms, we use thousands of inputs, and each call to f may require minutes or even hours, so this is not a very feasible idea. What can we do?

Case of probability distributions: Monte-Carlo simulations. In situations when we know the probability distributions, the answer is well known: use Monte-Carlo simulations. Namely, for some integer N, we:

- N times simulate the variables $\delta x_i^{(k)}$, $k = 1, \ldots, N$, whose distribution is the same as the distribution of the corresponding measurement errors Δx_i, and then
- compute the values

$$\delta^{(k)} = f(\tilde{x}_1, \ldots, \tilde{x}_n) - f(\tilde{x}_1 - \delta x_1^{(k)}, \ldots, \tilde{x}_n - \delta x_n^{(k)}).$$

By using the same Taylor expansion as before, we conclude that

$$\delta^{(k)} = \sum_{i=1}^{n} c_i \cdot \delta x_i^{(k)}.$$

Since the values $\delta x_i^{(k)}$ have the same distribution as the measurement errors Δx_i, we can conclude that the distribution of the values $\delta^{(k)}$ is exactly the same as the desired distribution of the approximation errors Δy.

In particular, for the normal distribution, we can estimate σ as

$$\sigma \approx \sqrt{\frac{1}{N} \cdot \sum_{k=1}^{N} \left(\delta^{(k)}\right)^2}.$$

Good news is that the accuracy of this estimation is proportional to $\frac{1}{\sqrt{N}}$ (see, e.g., [19]) and does not depend on the number of inputs n.

For example, if we want to find σ with accuracy $\frac{1}{\sqrt{N}} = 20\%$, we need to take $N = 25$ iterations and thus, 25 calls for the program f – which is much smaller than thousands of calls needed for straightforward estimation.

What if we only know the upper bounds? At first glance, in this case the situation is hopeless: we do not know what distribution to use. However, interestingly, a Monte-Carlo approach is possible in this case as well; see, e.g., [20, 21]. The idea is to use Cauchy distribution, i.e., the distribution with probability density function

$$\rho(x) = \frac{1}{\pi \cdot \Delta} \cdot \frac{1}{1 + \left(\frac{x}{\Delta}\right)^2}.$$

The use of Cauchy distribution is based on the following property of this distribution:

- if independent random variables $\Delta x_1, \ldots, \Delta x_n$ are Cauchy-distributed with parameters Δ_i,
- then the linear combination $\sum\limits_{i=1}^{n} c_i \cdot \Delta x_i$ of these random variables is also Cauchy-distributed, with parameter $\Delta = \sum\limits_{i=1}^{n} |c_i| \cdot \Delta_i$.

This expression is exactly what we want.

Thus, to find Δ:

- we simulate $\delta x_i^{(k)}$ to be Cauchy-distributed with parameter Δ_i;
- then the values
$$\delta^{(k)} = f(\tilde{x}_1, \ldots, \tilde{x}_n) - f(\tilde{x}_1 - \delta x_1^{(k)}, \ldots, \tilde{x}_n - \delta x_n^{(k)})$$
are Cauchy-distributed with the desired value of the parameter Δ. We can then find this value Δ by applying, e.g., the Maximum Likelihood method [19].

How quantum computing can help. Sometimes, some parts of the input are irrelevant. For example, to predict tomorrow's weather, if the wind blows from the North, then what was to the south of this area does not matter. If we could find such irrelevant bits and not process them, this will make our computations faster. How can we find such irrelevant bits?

Here, quantum computing can help in the following way. In the simplest case, in which the input x is just one bit, and we consider one bit $f(x)$ of the output, the question is whether the input is relevant at all, i.e., whether $f(0) = f(1)$.

- In non-quantum computing, we can only have inputs 0 or 1. So, to check the equality $f(0) = f(1)$, we need to call f twice: for $x = 0$ and for $x = 1$.

- It turns out that in quantum computing, we can check this property by using only one call to f – but in this call, the input should be a superposition of 0 and 1; see, e.g., [22, 12].

A similar reduction is possible for some multi-bit cases [12].

4. Measurements under Limited Resources: Two Case Studies

Two types of situations. In addition to limited resources for computations, we can (and do) also have limited resources for measurements. In this section, we consider case studies covering two possible types of situations: when we can only measure individual quantities, and when we can measure combinations of different quantities.

Measuring individual quantities: general description and the case study of a robotic boat. Measurements require even more resources than computations: they require time, they require energy. It is therefore desirable to minimize the number of measurements that we need to describe some spatial or temporal-spatial phenomenon with a given accuracy.

In some cases, we need to develop a measurement schedule at the very beginning. For this case, general ideas on how to minimize the number of measurements, see, e.g., [23, 24, 25] and references therein. Further minimization can be achieved if we make the measurement plan adaptive: we will decide what to measure next based on the results of previous measurements. In this section, let us concentrate on the case of a single passive measuring device: e.g., a spaceship following a given trajectory or a robotic boat floating along the river.

This choice of examples may need some explanation. Probably no one will wonder why we need measurements performed by a spaceship, but why do we need a floating robotic boat? The answer to this question is very straightforward: with planes, satellites, and drones, most of the Earth's surface has been accurately mapped. We know the elevations at different locations, we know how much plants are there. Similarly, we have a reasonable understanding of the oceans, and of major rivers. The least explored geographic objects are small rivers. To get a complete picture of Earth's geography, we need to describe their depths and flows at different locations. This is important for ecological studies, this is important to understand how water circulates – since big rivers are usually fed by smaller ones. To get a good description of a small river, the usual way is to traverse the river and measure the corresponding parameters as we go. The problem is that there are many small rivers, and it is not realistic to expect human researchers to traverse all of them.

To solve this problem, researchers are working on designing robotic boats; see, e.g., [26] and references therein. The idea is to place this boat at the origin of the small river and let is flow along the river and measure the corresponding parameters. The problem is that measurements and transmission of the resulting data use some energy, and since boat is autonomous, it can only carry a limited amount of energy. So, the question is: how to get the desired map of the river by using the smallest possible number of measurements.

The main idea behind this minimization is straightforward:

- when we are passing through a part of the trajectory where the depth (or any other quantity of interest) is approximately constant, then there is no sense of measuring this quantity too frequently; but
- when we come to the part of the trajectory where the quantity starts changing, then we must measure it with the highest possible frequency.

Let us describe the corresponding problem in precise terms. We know reasonably well the trajectory followed by a spaceship or the trajectory followed by a floating robotic boat or by another similar device. Let x be the parameter describing the device's location on this trajectory – e.g., the length of the path that it has already covered along this trajectory. At some values x, we perform some measurements. We may want to measure some quantity in its location – e.g., the depth at this location. We may want to measure some quantity that depends not only x, but also on some other parameters $y_1, \ldots$. For example, for the boat, we may want, by sending a signal in several directions, to measure the depths not only right beneath the boat, but also on other locations (x, y_1) on the line orthogonal to the trajectory.

In general, based on the measurements, we want to find the while dependence $q(x, y_1, \ldots)$ of the measured quantity on all these parameters, and we want to measure all these values with accuracy ε. How can we reach this goal by using the smallest possible number of measurements?

A natural idea is to use a simple approximation

$$q(x, y_1, \ldots) = C_1 \cdot q_1(x, y_1, \ldots) + \ldots + C_k \cdot q_k(x, y_1, \ldots)$$

to describe the local behavior of the corresponding field. In this approximation:

- the functions $q_1(x, y_1, \ldots), \ldots, q_k(x, y_1, \ldots)$ are fixed, and
- the parameters $C_1, \ldots, C_k$ change from location to location.

In many cases, the field is smooth, so it can be locally approximated by linear or quadratic functions. For example, if we are interested in the depth $d(x)$ at different points along the trajectory, then we can use approximations $d(x) \approx C_1 + C_2 \cdot x$ or take $d(x) \approx C_1 + C_2 \cdot x + C_3 \cdot x^2$.

- In the first case, $k = 2$, $q_1(x) = 1$, and $q_2(x) = x$.
- In the second case, $k = 3$, $q_1(x) = 1$, $q_2(x) = x$, and $q_3(x) = x^2$.

How we proceed depends on:

- whether we know the probability distribution of the measurement error – e.g., Gaussian with standard deviation σ – or

- whether we only know the bound Δ on the measurement error.

In both cases, we first perform $k+1$ (or more) measurements corresponding to the values $x^{(i)}$, $y_1^{(i)}, \ldots$, where $i = 1, 2, \ldots, k+1$. Let $q^{(i)}$ be the corresponding measurement results.

In the probabilistic case, we form the resulting system of approximate equations:

$$q^{(1)} \approx C_1 \cdot q_1\left(x^{(1)}, y_1^{(1)}, \ldots\right) + \ldots + C_k \cdot q_k\left(x^{(1)}, y_1^{(1)}, \ldots\right);$$

$$\ldots$$

$$q^{(k+1)} \approx C_1 \cdot q_1\left(x^{(k+1)}, y_1^{(k+1)}, \ldots\right) + \ldots + C_k \cdot q_k\left(x^{(k+1)}, y_1^{(k+1)}, \ldots\right).$$

We can use the usual Least Squares method (see, e.g., [19]) to find the estimates $\widehat{C}_1, \ldots, \widehat{C}_k$ for the corresponding parameters $C_1, \ldots, C_k$. This method also provides us with the variance of these estimates, and with the covariances describing correlation between these estimates.

Based on these variances and covariances, for all future values x and for all corresponding values $y_1, \ldots$, we can compute the standard deviation for the predicted value

$$q(x, y_1, \ldots) = \widehat{C}_1 \cdot q_1(x, y_1, \ldots) + \ldots + \widehat{C}_k \cdot q_k(x, y_1, \ldots).$$

As long as this value is smaller than or equal to the desired accuracy ε, we do not perform any additional measurements. The next group of measurements is performed at the first future value x at which the standard deviation becomes equal to ε.

In the case of upper bounds, for each future value x and for all possible values of the variables $y_1, \ldots$, we find the range $[\underline{q}, \overline{q}]$ of possible values $q(x, y_1, \ldots)$ by solving the following two optimization problems. To find the upper bound $\overline{q}$, we maximize the expression

$$C_1 \cdot q_1(x, y_1, \ldots) + \ldots + C_k \cdot q_k(x, y_1, \ldots)$$

under the constraints

$$q^{(1)} - \Delta \le C_1 \cdot q_1\left(x^{(1)}, y_1^{(1)}, \ldots\right) + \ldots + C_k \cdot q_k\left(x^{(1)}, y_1^{(1)}, \ldots\right) \le q^{(1)} + \Delta;$$

$$\ldots$$

$$q^{(k+1)} - \Delta \le C_1 \cdot q_1\left(x^{(k+1)}, y_1^{(k+1)}, \ldots\right) + \ldots + C_k \cdot q_k\left(x^{(k+1)}, y_1^{(k+1)}, \ldots\right) \le q^{(k+1)} + \Delta.$$

To find the lower bound $\underline{q}$, we minimize the same expression under the same constraints. In both optimization problems, we optimize a function which is linear in terms of the unknowns $C_1, \ldots, C_k$ under constraints which are also linear in terms of the unknowns. Such optimization problems are known as *linear programming* problems. There exist efficient algorithms for solving such problems; see, e.g., [27].

The next measurement is performed at a location for which the prediction accuracy is ε, i.e., for which the interval $[\underline{q}, \overline{q}]$ has the form $[\widetilde{q} - \varepsilon, \widetilde{q} + \varepsilon]$ for some $\widetilde{q}$, i.e., equivalently, for which

$$\overline{q} - \underline{q} = 2\varepsilon.$$

Measuring combinations of quantities: general description and the case study of Covid-19 testing. When we measure, we are interested in some property. In some situations, only a few objects have the property of interest. In such situations, the possibility to measure combinations of quantities corresponding to individual objects can save on the number of measurements.

For example, if we are prospecting for oil, we save a lot of efforts if, instead of drilling at each possible location, we perform some measurements of the area as a whole and only drill in those areas which, according to these combined measurements, have oil.

Another example is Covid-19 testing. In the ideal world we should test everybody, but the number of available test kits is limited. If instead of testing all N people from a given area, we apply the test to a mixture of samples from a group of several (s_1) people, we can decrease the required number of tests kits – since:

- we can dismiss all the groups where no one was Covid-positive, and
- we need to continue testing only folks from the remaining groups; see, e.g., [28, 29, 30].

In [30], we considered the arrangement when after the group testing, we individually test everyone from still-suspicious groups. However, if there are many such folks, a reasonable idea is to combine these remaining folks into new groups of size $s_2 < s_1$, and test each new group, etc., until, after n such stages, on the following stage $n + 1$, we test all remaining possibly-positive folks individually. Let us analyze what is the optimal approach to such multi-stage group testing.

Let p denote the proportion of people in a given area that test Covid-positive. This number can be (and is) estimated based on the preliminary random testing. Thus, overall, we have $N \cdot p$ Covid-positive folks.

The value p is small, so for sufficiently small group sizes s_k, the probability that we have two Covid-positive folks in each group is small. Since most of $N \cdot p$ Covid-positive folks belong to different groups, after k stages, we will have $N \cdot p$ different groups of size s_k – and thus, we have $N \cdot p \cdot s_k$ possibly-positive folks.

On the first stage, we test $\dfrac{N}{s_1}$ folks. After k stages, we have $N \cdot p \cdot s_k$ possibly positive folks. We divide them into groups of size s_{k+1}. Thus, on the $(k + 1)$-st stage, we get $\dfrac{N \cdot p \cdot s_k}{s_{k+1}}$ testing groups. Hence, on the $(k + 1)$-st stage, we need to perform $\dfrac{N \cdot p \cdot s_k}{s_{k+1}}$ tests. The overall number of tests is therefore equal to

$$\frac{N}{s_1} + \frac{N \cdot p \cdot s_1}{s_2} + \ldots + \frac{N \cdot p \cdot s_{k-1}}{s_k} + \frac{N \cdot p \cdot s_k}{s_{k+1}} + \ldots$$

We want to select the values s_k that minimize this overall number of tests.

Towards finding the optimal testing strategy. Let us first select an integer $k > 1$ and minimize the overall number of tests with respect to s_k. Differentiating the above expression for the overall number of tests with respect to s_k and equating the derivative to 0, we conclude that

$$-\frac{N \cdot p \cdot s_{k-1}}{s_k^2} + \frac{N \cdot p}{s_{k+1}} = 0.$$

Dividing both sides by $N \cdot p$, multiplying both sides by s_k, and moving the negative term to the other side, we conclude that $\frac{s_k}{s_{k+1}} = \frac{s_{k-1}}{s_k}$. This is true for all k, so the value $\frac{s_k}{s_{k+1}}$ is the same for all k. Let us denote this ratio by q. Thus, the values s_k form a geometric progression with common ratio q.

At the end, we check individually, so $s_{n+1} = 1$. Thus, we get $s_n = q$, $s_{n-1} = q^2, \ldots,$ until we get $s_1 = q^n$. Thus, $n = \log_q(s_1) = \frac{\ln(s_1)}{\ln(q)}$. For these values s_k, each term $\frac{N \cdot p \cdot s_k}{s_{k+1}}$ in the expression for the overall number of tests is equal to $N \cdot p \cdot q$, and there are n such terms – as many as stages in the testing procedure. Thus, the overall number of tests is equal to

$$\frac{N}{s_1} + n \cdot N \cdot p \cdot q = \frac{N}{s_1} + \frac{\ln(s_1)}{\ln(q)} \cdot N \cdot p \cdot q.$$

Minimizing this expression with respect to q means minimizing the ratio $\frac{q}{\ln(q)}$. Differentiating this expression with respect to q and equating the derivative to 0, we conclude that $\ln(q) - q \cdot \frac{1}{q} = 0$, i.e., that $\ln(q) = 1$ and thus, $q = e$.

For $q = e$, the above expression for the overall number of tests takes the form $\frac{N}{s_1} + N \cdot p \cdot e \cdot \ln(s_1)$. There is only one parameter left undetermined – the value s_1. Differentiating the above expression with respect to s_1 and equating the derivative t 0, we get $-\frac{N}{s_1^2} + N \cdot p \cdot e \cdot \frac{1}{s_1} = 0$. Multiplying both sides by s_1^2, dividing both sides by $N \cdot p \cdot e$, and moving negative terms to the other side, we conclude that $s_1 = \frac{1}{p \cdot e}$.

Thus, we arrive at the following recommendation.

Optimal testing schedule. We first combine folks into groups of size $s_1 = \frac{1}{p \cdot e}$, and test combined samples from each group. All the people from groups whose combined sample tested negative are Covid-negative and thus, do not need any additional testing.

The remaining folks are combined in groups of size $s_2 = \frac{s_1}{e}$. We test combined samples from each group. All the people from groups whose combined sample tested negative are Covid-negative and thus, do not need any additional testing.

The remaining folks are combined in groups of size $s_3 = \frac{s_1}{e^2}$, etc. At the end, we reach the stage at which $s_{n+1} \approx 1$, i.e., where the remaining folks need to be individually tested.

How many tests do we need for the optimal testing schedule. Substituting the expression $s_1 = \frac{1}{p \cdot e}$ into the formula for the overall number of tests, and taking into account that $\ln(s_1) = |\ln(p)| - 1$, we conclude that we need

$$\frac{N}{s_1} + N \cdot p \cdot e \cdot \ln(s_1) = N \cdot p \cdot e + N \cdot p \cdot e \cdot (|\ln(p)| - 1),$$

i.e., we need

$$N \cdot p \cdot |\ln(p)| \cdot e$$

tests.

Is this method optimal? This is indeed the best we can do – at least asymptotically the best. Indeed, we need to find $N \cdot p$ elements out of N. The number of such possible arrangement is

$$\binom{N}{p \cdot N} = \frac{N!}{(p \cdot N)! \cdot ((1-p) \cdot N)!}.$$

Each test has 2 possible results, so after t tests, we have 2^t possible combinations of results. To get all $\binom{N}{p \cdot N}$ possible answers, we need to have $2^t \ge \binom{N}{p \cdot N}$, i.e., we need $t \approx \log_2 \binom{N}{p \cdot N}$. Asymptotically, for each integer q, we have $q! \approx \left(\frac{q}{e}\right)^q$, so

$$\log_2(q!) \approx q \cdot (\log_2(q) - \log_2(e)) = q \cdot \log_2(q) - q \cdot \log_2(e).$$

Thus,

$$t \approx \log_2 \binom{N}{p \cdot N} = \log_2(N!) - \log_2((p \cdot N)!) - \log_2((1-p) \cdot N)!) =$$
$$N \cdot \log_2(N) - N \cdot \log_2(e) - p \cdot N \cdot \log_2(p \cdot N) + p \cdot N \cdot \log_2(e) -$$
$$(1-p) \cdot N \cdot \log_2((1-p) \cdot N) + (1-p) \cdot N \cdot \log_2(e).$$

Terms proportional to $\log_2(e)$ cancel each other. So, taking into account that the logarithm of the product is equal to the sum of logarithms, we get

$$t \approx N \cdot \log_2(N) - p \cdot N \cdot \log_2(N) - p \cdot N \cdot \log_2(p) -$$
$$(1-p) \cdot N \cdot \log_2(N) - (1-p) \cdot N \cdot \log_2(1-p).$$

Terms proportional to $N \cdot \log_2(N)$ cancel each other, so

$$t \approx -N \cdot (p \cdot \log_2(p) + (1-p) \cdot \log_2(1-p)).$$

For small p, we have $1 - p \approx 1$ and $\log_2(1-p) \sim p$, so

$$(1-p) \cdot \log_2(1-p) \sim p \ll p \cdot \log_2(p),$$

thus the smallest possible number of tests is indeed proportional to

$$N \cdot p \cdot |\log_2(p)| = N \cdot p \cdot \frac{|\ln(p)|}{\ln(2)}.$$

This expression differs from our number of tests by a factor $e \cdot \ln(2) \approx 1.88$; so, modulo a small multiplicative constant, our method is indeed asymptotically optimal.

5. 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).

The authors are greatly thankful to the conference organizers for their encouragement and support, and to Laura Alvarez for helpful discussions.

References

[1] G. Arellano, E. Hudgins, D. Pruitt, A. Veliz, E. Freudenthal, V. Kreinovich, How to gauge disruptions caused by garbage collection: towards an efficient algorithm, Journal of Uncertain Systems 10 (2016) 4–9.

[2] Y. Zhang, Y. Liu, L. Zhuang, X. Liu, F. Zhao, Q. Li, Accurate cpu power modeling for multicore smartphones (2015).

[3] B. Furht, A. Escalante, Handbook of cloud computing, volume 3, Springer, 2010.

[4] D. C. Marinescu, Cloud computing: theory and practice, Morgan Kaufmann, 2017.

[5] P. Mell, T. Grance, et al., The NIST definition of cloud computing (2011).

[6] J. Rhoton, Cloud computing explained, Recursive Press London, 2010.

[7] T. Velte, A. Velte, R. Elsenpeter, Cloud computing, a practical approach, McGraw-Hill, Inc., 2009.

[8] V. Kreinovich, Towards optimizing cloud computing: an example of optimization under uncertainty, in: S. U. Khan, A. Y. Zomaya, L. Wang (Eds.), Scalable Computing and Communications: Theory and Practice, John Wiley & Sons and IEEE Computer Society Press, 2013, pp. 613–627.

[9] V. Kreinovich, E. Gallardo, Optimizing cloud use under interval uncertainty, in: R. Wyrzykowski, E. Deelman, J. Dongarra, K. Karczewski, J. Kitowski, K. Wiatr (Eds.), Parallel Processing and Applied Mathematics, Springer, 2016, pp. 435–444.

[10] O. Lerma, E. Gutierrez, C. Kiekintveld, V. Kreinovich, Towards optimal knowledge processing: from centralization through cyberinsfrastructure to cloud computing, International Journal of Innovative Management, Information & Production (IJIMIP) 2 (2011) 67–72.

[11] L. O. Lerma, V. Kreinovich, Towards analytical techniques for optimizing knowledge acquisition, processing, propagation, and use in cyberinfrastructure and big data, volume 29, Springer Verlag, 2018.

[12] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2000.

[13] R. P. Feynman, R. B. Leighton, M. Sands, The Feynman Lectures on Physics: Definitive Edition, Pearson Addison-Wesley, 2005.

[14] K. S. Thorne, R. D. Blandford, Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics, Princeton University Press, 2017.

[15] L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 212–219.

[16] L. K. Grover, Quantum mechanics helps in searching for a needle in a haystack, Physical review letters 79 (1997) 325.

[17] L. Longpre, V. Kreinovich, Can quantum computers be useful when there are not yet enough qubits?, Bulletin of the EATCS 79 (2003) 164–169.

[18] S. G. Rabinovich, Measurement errors and uncertainties: theory and practice, Springer Science & Business Media, 2005.

[19] D. J. Sheskin, Handbook of Parametric and Non-Parametric Statistical Procedures, Chapman & Hall/CRC, 2011.

[20] V. Kreinovich, S. A. Ferson, A new cauchy-based black-box technique for uncertainty in risk analysis, Reliability Engineering & System Safety 85 (2004) 267–279.

[21] H. T. Nguyen, V. Kreinovich, B. Wu, G. Xiang, Computing statistics under interval and fuzzy uncertainty, volume 130, Springer, 2012.

[22] O. Kosheleva, V. Kreinovich, How to introduce technical details of quantum computing in a theory of computation class: using the basic case of the deutsch-jozsa algorithm, International Journal of Computing and Optimization 3 (2016) 83–91.

[23] H. T. Nguyen, O. Kosheleva, V. Kreinovich, S. Ferson, Trade-off between sample size and accuracy: Case of dynamic measurements under interval uncertainty, in: V.-N. Huynh, Y. Nakamori, H. Ono, J. Lawry, V. Kreinovich, H. T. Nguyen (Eds.), Interval/Probabilistic Uncertainty and Non-Classical Logics, Springer, 2008, pp. 45–56.

[24] H. T. Nguyen, O. Kosheleva, V. Kreinovich, S. Ferson, Trade-off between sample size and accuracy: case of measurements under interval uncertainty, International Journal of Approximate Reasoning 50 (2009) 1164–1176.

[25] H. T. Nguyen, O. Kosheleva, V. Kreinovich, S. Ferson, Trade-off between sample size and accuracy: Case of static measurements under interval uncertainty, in: V.-N. Huynh, Y. Nakamori, H. Ono, J. Lawry, V. Kreinovich, H. T. Nguyen (Eds.), Interval/Probabilistic Uncertainty and Non-Classical Logics, Springer, 2008, pp. 32–44.

[26] L. Alvarez, H. Moreno, A. Segales, T. Pham, E. Pillar-Little, P. Chilson, Merging unmanned aerial systems (uas) imagery and echo soundings with an adaptive sampling technique for bathymetric surveys, Remote Sensing 10 (2018) 1362. URL: http://dx.doi.org/10.3390/rs10091362. doi:10.3390/rs10091362.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms, MIT press, 2009.

[28] T. S. Perry, Researchers are using algorithms to tackle the coronavirus test shortage: The scramble to develop new test kits that deliver faster results - [spectral lines], IEEE Spectrum 57 (2020) 4–4. doi:10.1109/MSPEC.2020.9099916.

[29] N. Shental, S. Levy, S. Skorniakov, V. Wuvshet, Y. Shemer-Avni, A. Porgador, T. Hertz, Efficient high throughput SARS-CoV-2 testing to detect asymptomatic carriers (2020). URL: https://doi.org/10.1101/2020.04.14.20064618. doi:10.1101/2020.04.14.20064618.

[30] J. Urenda, O. Kosheleva, M. Ceberio, V. Kreinovich, How mathematics and computing can help fight the pandemic: two pedagogical examples, in: Proceedings of the Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2020, 2020.