8-1-2021

Why Moving Fast and Breaking Things Makes Sense?

Francisco Zapata
The University of Texas at El Paso, fcozpt@outlook.com

Eric Smith
The University of Texas at El Paso, esmith2@utep.edu

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

Comments:

Technical Report: UTEP-CS-21-75

Why Moving Fast and Breaking Things Makes Sense?

Francisco Zapata, Eric Smith, and Vladik Kreinovich

Abstract In the traditional approach to engineering system design, engineers usually come up with several possible designs, each improving on the previous ones. In coming up with these designs, they try their best to make sure that their designs stay within the safety and other constraints, to avoid potential catastrophic crashes. The need for these safety constraints makes this design process reasonably slow. Software engineering at first followed the same pattern, but then realized that since in most cases, failure of a software test does not lead to a catastrophe, it is much faster to first ignore constraints and then adjust the resulting non-compliant designs so that the constrains will be satisfied. Lately, a similar "move fast and break things" approach was applied to engineering design as well, especially when designing autonomous systems whose failure-when-testing is not catastrophic. In this paper, we provide a simple mathematical model explaining, in quantitative terms, why moving fast and breaking things makes sense.

1 Formulation of the Problem

General engineering problems: reminder. Whatever we design, we have an objective function that we want to optimize:

- From the business viewpoint, we want to maximize profit.
- When we design computers, we want computations to be as fast as possible.
- When we are interested in saving environment, we want to make sure that the corresponding chemical process produces as little pollution as possible, etc.

What is also important is that there are always constraints, limitations. Here are some examples:

Francisco Zapata, Eric Smith, and Vladik Kreinovich
University of Texas at El Paso, El Paso, Texas 79968, USA
e-mail: fcozpt@outlook.com, esmith2@utep.edu, vladik@utep.edu

- A construction company contracted to build a federal office building may want to maximize its profit and use the cheapest materials possible – but in this desire, the company is restricted by the contract, and it is also restricted by the building regulations: according to these regulations, the building must be able to withstand strong winds, floods, and/or earthquakes that can occur in this area.
- A company that sets up a chemical plant may want to save money on filtering – which is often very expensive – but it has to make sure that the resulting pollution does not exceed the thresholds set by national and local regulations.
- Designers of a fast small plane aimed at customers interested in speed may sacrifice a lot of weight to gain more speed – but they still need to make sure that the plane is sufficiently safe.

How engineering problems used to be solved. To find the optimal solution, a company – or an individual investor – go from one design to another, trying to come up with the best possible design. In this process, maximum efforts were undertaken to make sure that at all the stages, the corresponding design satisfies all the needed constraints. These efforts make sense: whether we tests a car or a plane or a chemical process, a violation of safety and health constraints may be disastrous and may even lead to loss of lives.

Sometimes, an experimental plane would crash, often killing a pilot. Sometimes, a bridge would collapse – all these catastrophes would remind other designers of the importance to stay within the constraints.

These constraints provided sufficient safety, but they also have a negative effect: the need to make sure that each new design is within the constraints drastically slows down the progress.

Software design first followed the same pattern. At first, folks who designed software followed the same general patterns: whenever they changed a piece of a big software package to a more efficient (and/or more effective) one, they first patiently made sure that this change will not cause any malfunctions. This "making sure" slowed down the progress.

New idea: move fast and break things. With time, software designers realized that they do not necessarily need to be so cautious. Unless the software they design is intended for life-critical systems like nuclear power stations or airplane control, a minor fault is tolerable and usually does not lead to catastrophic consequences. They realized that they can move much faster if they come up with designs that may not necessarily satisfy all the constraints at first – corresponding corrections can be done later, and that even with some time spent on these corrections, still this new paradigm led to faster design.

This new practice was explicitly formulated as "move fast and break things" by Mark Zuckerberg, then CEO of Facebook. This phrase even became (for several years) the motto of the Facebook company – and, informally, of the whole Silicon Valley; see, e.g., [1, 2].

This idea moved to engineering design. Interestingly, this idea – first originated in software design and first intended for software design only – eventually moved to general engineering as well. The reason for this transition is that with the increased automation, most test crashes stopped being dangerous to humans: if a self-driving car or a pilotless plane crashes, there is no immediate danger to people – unless they accidentally happen to be nearby.

The main pioneer of using this idea in engineering was Elon Musk, who used this successfully, in particular, in his space exploration efforts.

How can we explain the success of this idea? The idea of moving fast and breaking things seem to work for many projects. (Sometimes, it does not work – but, on the other hand, sometimes projects based on the traditional engineering design techniques do not work either.)

But why does this moving-fast idea work? Understanding why it works for many projects is important: this way, we will be able:

- to better understand when it works and when it does not, and
- in situations where this idea works, come up with the best possible way of using this idea.

This is what we do in this paper: we provide a natural simple quantitative model explaining why this idea, in general, works.

2 Description of the Model

What we want to optimize. As we have mentioned, in engineering design, we need to select some quantities $x_1,\ldots,x_n$ – parameters of the design – for which a given objective function $f(x_1,\ldots,x_n)$ attains its largest possible value among all the tuples $x=(x_1,\ldots,x_n)$ that satisfy all the given constraints.

How to describe constraints. In general, constraints have the form of inequalities $\ell_i(x_1,\ldots,x_n)\le r_i(x_1,\ldots,x_n)$ for some quantities ℓ_i and r_i. Each such constraint can be equivalently reformulated as $g_i(x_1,\ldots,x_n)\ge 0$, where we denoted

$$g_i(x_1,\ldots,x_n)\stackrel{\text{def}}{=}r_i(x_1,\ldots,x_n)-\ell_i(x_1,\ldots,x_n).$$

So, we have a finite numbers of constraints that the desired design must satisfy:

$$g_1(x_1,\ldots,x_n)\ge 0,\ldots,g_m(x_1,\ldots,x_n)\ge 0.$$

Several numbers are non-negative if and only if the smallest of these numbers is non-negative. Thus, satisfying the above m constraints is equivalent to satisfying a single constraint

$$g(x_1,\ldots,x_n)\ge 0 \tag{1}$$

where we denoted

$$g(x_1,\ldots,x_n)\stackrel{\text{def}}{=}\min(g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n)).$$

Resulting formulation of the general problem. We want to maximize a function $f(x_1,\ldots,x_n)$ under the constraint (1).

Additional complexity: need to take uncertainty into account. At first glance, this sounds like a usual optimization problem, for which many algorithms are known. However, in many practical situations, there is an additional complexity – that both the objective function $f(x_1,\ldots,x_n)$ and the function $g(x_1,\ldots,x_n)$ that describes the constraints are only approximately known.

Indeed, if both these functions were exactly known, there would be no need for testing different designs – we would just be able to solve the corresponding constrained optimization problem and implement it.

Mathematical fact: the solution is usually on the edge of constraints. In general, the largest value of the objective function is attained:

- either inside the area where the constrains are satisfied, i.e., where $g_i(x_1,\ldots,x_n)>0$ for all i,
- or at the border of this area, i.e., when $g_i(x_1,\ldots,x_n)=0$ for some i, and thus,

$$g(x_1,\ldots,x_n)=0. \tag{2}$$

In the first case, the solution is a local maximum. So, in looking for this maximum, we can simply ignore all the constraints, they are automatically satisfied.

An important case where we do need to take constraints into account is the second case, when the maximum is attained at the border. This is the case that we will consider in this paper.

How the corresponding problem is solved under uncertainty: possible preliminary stage. We start with the first design $x^{(1)}=\left(x_1^{(1)},\ldots,x_n^{(1)}\right)$.

In some cases, this first design is already on the border (2) – or at least close to this border.

In many other cases, however, this design is usually rather far from the area where the constraints are not satisfied. So at first, we can kind-of ignore the constraints and try to modify the values x_i so as to increase the value of the objective function $f(x_1,\ldots,x_n)$. After several consequent improvements, we get closer and closer to the optimal design – and thus, closer and closer to the border (2). At the end of this possible preliminary stage, we get so close to the border that the constraints can no longer be ignored.

Main stage of the design process. Once we have reached a point close to the border, for which the value $\varepsilon\overset{\text{def}}{=}g(x_1,\ldots,x_n)>0$ is small, the main stage of the design process starts: finding the optimal solution while taking constraints into account.

Let us analyze how this main stage is performed in general, and what is different when we perform this stage in the traditional engineering methodology and in the moving-fast software-motivated methodology.

3 The Main Stage of Optimization: General Analysis

What we know and what we want. At the beginning of this main stage, we have a design $(x_1,\ldots,x_n)$ which is close to the border, i.e., for which – at least approximately – the condition (2) is satisfied. We known that this design is not optimal, so we want to find a modified design

$$(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)$$

for which the value of the objective function is larger.

Ideal case, when we have the exact knowledge of the objective function and of the constraints. Let us first consider the ideal case, when we know the exact expressions both:

- for the objective function $f(x_1,\ldots,x_n)$ and
- for the function $g(x_1,\ldots,x_n)$ that describes the constraints.

It is known that, in general, the constraint optimization problem is equivalent to the unconstrained optimization problem of maximizing the expression

$$f(x_1,\ldots,x_n)+\lambda\cdot g(x_1,\ldots,x_n) \tag{3}$$

for an appropriate value λ; this value is known as the *Lagrange multiplier*.

For unconstrained optimization, one of the most natural optimization techniques is *gradient method*, when, by choosing the values Δx_i, we follow the direction in which the objective function increases the most – i.e., the direction of its gradient. For the equivalent objective function (3), this means that we select

$$\Delta x_i=\alpha\cdot\frac{\partial}{\partial x_i}(f(x_1,\ldots,x_n)+\lambda\cdot g(x_1,\ldots,x_n))$$

for an appropriate value $\alpha>0$, i.e.,

$$\Delta x_i=\alpha\cdot(f_i+\lambda\cdot g_i), \tag{4}$$

where we denoted

$$f_i\stackrel{\text{def}}{=}\frac{\partial f}{\partial x_i}\text{ and }g_i\stackrel{\text{def}}{=}\frac{\partial g}{\partial x_i}.$$

The value of the Lagrange multiplier λ must be determined from the condition that we will be moving along the border, i.e., that the original zero value of the function $g(x_1,\ldots,x_n)$ – that describes this border as the set of all the tuples x for which $g(x_1,\ldots,x_n)=0$ – should not change. In precise terms, we should have

$$g(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)=0,$$

i.e.,

$$0 = g(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) \approx g(x_1, \ldots, x_n) + \sum_{i=1}^{n} g_i \cdot \Delta x_i = \sum_{i=1}^{n} g_i \cdot \Delta x_i.$$

Substituting the expression (4) into this formula, we get

$$\sum_{i=1}^{n} f_i \cdot g_i + \lambda \cdot \sum_{i=1}^{n} g_i^2 = 0,$$

hence

$$\lambda = -\frac{\sum\limits_{i=1}^{n} f_i \cdot g_i}{\sum\limits_{i=1}^{n} g_i^2}. \tag{5}$$

Realistic case, when we take uncertainty into account. As we have mentioned, in practice, we only know the objective function $f(x_1, \ldots, x_n)$ and the function $g(x_1, \ldots, x_n)$ (that describes the constraints) only approximately. As a result, we only know the approximate values of the corresponding derivatives, i.e., we know the values $\widetilde{f}_i \approx f_i$ and $\widetilde{g}_i \approx g_i$ for which

$$\widetilde{f}_i = f_i + \Delta f_i \text{ and } \widetilde{g}_i = g_i + \Delta g_i$$

for some reasonably small values Δf_i and Δg_i.

Let $\delta > 0$ be the accuracy with which we know these derivatives, This means that for each i, we have $|\Delta f_i| \le \delta$ and $|\Delta g_i| \le \delta$.

Of course, since these approximate values are the only information we have, we use these approximate values when we decide on which parameters to use for the next design. In other words, we take

$$\Delta x_i = \alpha \cdot (\widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i), \tag{6}$$

where

$$\widetilde{\lambda} = -\frac{\sum\limits_{i=1}^{n} \widetilde{f}_i \cdot \widetilde{g}_i}{\sum\limits_{i=1}^{n} (\widetilde{g}_i)^2}. \tag{7}$$

The only remaining question is what value α we should use. Let us show how the choice of α depends on which of the two methodologies we use: the traditional engineering methodology or the moving-fast software motivated methodology.

4 The Main Stage of Optimization: Case of the Traditional Engineering Methodology

Idea. In the traditional engineering methodology, we select the value α so as to make sure that, not matter what the actual values f_i and g_i are, we remain within the safe domain, i.e., that we still have $g(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)\ge 0$.

Based on this idea, what value α do we choose. For relatively small changes Δx_i we have

$$g(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)\approx g(x_1,\ldots,x_n)+\sum_{i=1}^n g_i\cdot\Delta x_i. \tag{8}$$

Since $\widetilde{g}_i=g_i+\Delta g_i$, we have $g_i=\widetilde{g}_i-\Delta g_i$, therefore

$$g(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)\approx g(x_1,\ldots,x_n)+\sum_{i=1}^n \widetilde{g}_i\cdot\Delta x_i-\sum_{i=1}^n \Delta g_i\cdot\Delta x_i. \tag{9}$$

Because of our selection (6) of the values Δx_i, we have

$$\sum_{i=1}^n \widetilde{g}_i\cdot\Delta x_i=0,$$

thus

$$g(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)\approx g(x_1,\ldots,x_n)-\sum_{i=1}^n \Delta g_i\cdot\Delta x_i. \tag{10}$$

We have denoted the current value of $g(x_1,\ldots,x_n)$ by ε, so

$$g(x_1+\Delta x_1,\ldots,x_n+\Delta x_n)\approx \varepsilon-\sum_{i=1}^n \Delta g_i\cdot\Delta x_i. \tag{11}$$

We want to make sure that this value remains non-negative for all possible combinations of values Δg_i for which $|\Delta g_i|\le\delta$, i.e., we want to make sure that for all these combinations, the sum

$$\sum_{i=1}^n \Delta g_i\cdot\Delta x_i \tag{12}$$

remains smaller than or equal to ε. In other words, we want to make sure that the largest possible value of the sum (12) is smaller than or equal to ε.

The sum (12) attains its largest possible value when each term $\Delta g_i\cdot\Delta x_i$ in this sum is the largest possible. Each of these terms is a linear function of Δg_i. So:

- when $\Delta x_i\ge 0$, this term is an increasing function of Δg_i and therefore, its largest value is attained when the variable Δg_i attains its largest possible value $\Delta g_i=\delta$; the corresponding largest value of this term is therefore equal to $\delta\cdot\Delta x_i$;
- when $\Delta x_i\le 0$, this term is a decreasing function of Δg_i and therefore, its largest value is attained when the variable Δg_i attains its smallest possible value $\Delta g_i=-\delta$; the corresponding largest value of this term is therefore equal to $-\delta\cdot\Delta x_i$.

Both cases can be describe by a single formula $\delta \cdot |\Delta x_i|$. Thus, the largest value of the sum (12) is equal to

$$\sum_{i=1}^{n} \delta \cdot |\Delta x_i| = \delta \cdot \sum_{i=1}^{n} |\Delta x_i|.$$

Substituting the expression (6) into this formula, we conclude that this largest value is equal to

$$\delta \cdot \alpha \cdot \sum_{i=1}^{n} \left| \widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i \right|.$$

Thus, the condition that this largest value is smaller than or equal to ε takes the form

$$\delta \cdot \alpha \cdot \sum_{i=1}^{n} \left| \widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i \right| \le \varepsilon,$$

i.e., equivalently, that

$$\alpha \le C \cdot \varepsilon, \tag{14}$$

where we denoted

$$C \stackrel{\text{def}}{=} \frac{\varepsilon}{\delta \cdot \sum\limits_{i=1}^{n} \left| \widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i \right|}.$$

Conclusion of this section. The desired optimal solution is located on the border. In the process of optimization, we get closer and closer to the optimal solution – and thus, the closer and closer to the border.

According to the formula (14), in the traditional engineering approach, the smaller the distance ε from the current solution to the border, the smaller our next modification can be – and thus, the slower our progress towards the optimal design.

5 The Main Stage of Optimization: Case of the Moving-Fast Software-Motivated Methodology

In the case of moving-fast methodology, we are not limiting our next design by any constraints, so we can make big step – probably violating the constraints, but then moving them back. Here, we do not having any slowing-down inequality like (14), so we get to the optimal solution much faster.

Thus, we indeed explained, in quantitative terms, why the moving-fast methodology is much faster than the traditional engineering one.

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), and by the AT&T Fellowship in Information Technology.

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, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

References

1. D. Parzych, "The fallacy of move fast and break things", 2020, https://devops.com/the-fallacy-of-move-fast-and-break-things/
2. J. Tuplin, *Move Fast and Break Things: How Facebook, Google, and Amazon Cornered Culture and Undermined Democracy*, Little, Brown, and Company, New York, 2017.