Why Moving Fast and Breaking Things Makes Sense?

Francisco Zapata, Eric Smith, and Vladik Kreinovich

University of Texas at El Paso, El Paso, Texas 79968, USA
fcozpt@outlook.com, esmith2@utep.edu, vladik@utep.edu

1. 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.
- Let us give some examples.

2. General engineering problems: reminder (cont-d)

- A construction company contracted to build a federal office building may want to maximize its profit.
- So, it may want to use the cheapest materials possible.
- However, in this desire, the company is restricted by the contract.
- 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.
- However, it has to make sure that the resulting pollution does not exceed the thresholds set by national and local regulations.

3. General engineering problems: reminder (cont-d)

- Designers of a fast small plane aimed at customers interested in speed may sacrifice a lot of weight to gain more speed.
- However, they still need to make sure that the plane is sufficiently safe.

4. How engineering problems used to be solved

- To find the optimal solution, a company – or an individual investor – go from one design to another.
- They try 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.
- It may even lead to loss of lives.

5. How engineering problems used to be solved (cont-d)

- 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.
- However, they also have a negative effect.
- Namely, the need to make sure that each new design is within the constraints drastically slows down the progress.

6. 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.

7. 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 even with some time spent on these corrections, still this new paradigm led to faster design.

8. New idea: move fast and break things (cont-d)

- 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.

9. This idea moved to engineering design

- Interestingly, this idea:
  - while 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.
- For example:
  - 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.
- He used this successfully, in particular, in his space exploration efforts.

10. How can we explain the success of this idea?

- The idea of moving fast and breaking things seem to work for many projects.
- Yes, sometimes, it does not work.
- However, 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.

11. How can we explain the success of this idea (cont-d)

- 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 talk:
  - we provide a natural simple quantitative model
  - that explains why this idea, in general, works.

12. 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.
- We want a given objective function $f(x_1, \ldots, x_n)$:
  - to attains its largest possible value
  - among all the tuples $x = (x_1, \ldots, x_n)$ that satisfy all the given constraints.

13. 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 $g(x_1, \ldots, x_n) \ge 0$, where we denoted
$$g(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \min(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$
- So, we want to maximize a function $f(x_1, \ldots, x_n)$ under this constraint.

14. 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.
- Namely, both the objective function $f(x_1, \ldots, x_n)$ and the function $g(x_1, \ldots, x_n)$ 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.

15. 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, $g(x_1, \ldots, x_n) = 0$.
- 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 talk.

16. 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 – 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.
- At the end of this possible preliminary stage, we get so close to the border that the constraints can no longer be ignored.

17. Main stage of the design process

- We have reached a point close to the border, for which the value $\varepsilon \stackrel{\text{def}}{=} g(x_1, \ldots, x_n) > 0$ is small.
- Now, the main stage of the design process starts.
- We need to find 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.

18. 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.
- For this design – at least approximately – the condition $g = 0$ is satisfied.
- We know 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.

19. 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.
- In general, a constraint optimization problem is equivalent to the unconstrained optimization problem of maximizing the expression

$$f(x_1, \ldots, x_n) + \lambda \cdot g(x_1, \ldots, x_n).$$

- The appropriate value λ is known as the *Lagrange multiplier*.
- For unconstrained optimization, one of the most natural optimization techniques is *gradient method*.
- There, when choosing the values Δx_i, we follow the direction in which the objective function increases the most.

20. Ideal case (cont-d)

- This is the direction of its gradient.
- For the equivalent objective function, this means that we select

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

$$\Delta x_i = \alpha \cdot (f_i + \lambda \cdot g_i), \text{ where } f_i \stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i} \text{ and } g_i \stackrel{\text{def}}{=} \frac{\partial g}{\partial x_i}.$$

- The value of the Lagrange multiplier λ must be determined from the condition that we will be moving along the border.
- This border is the set of all the tuples x for which $g(x_1, \ldots, x_n) = 0$.
- So this means that the original zero value of the function $g(x_1, \ldots, x_n)$ should not change.

21. Ideal case (cont-d)

- 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 for Δx_i into this formula, we get

$$\sum_{i=1}^{n} f_i \cdot g_i + \lambda \cdot \sum_{i=1}^{n} g_i^2 = 0; \text{ thus } \lambda = -\frac{\sum\limits_{i=1}^{n} f_i \cdot g_i}{\sum\limits_{i=1}^{n} g_i^2}.$$

22. Realistic case, when we take uncertainty into account

- In practice, we only know the objective function $f(x_1, \ldots, x_n)$ and the function $g(x_1, \ldots, x_n)$ only approximately.
- As a result, we only know the approximate values of the corresponding derivatives.
- In other words, we know the values $\widetilde{f}_i \approx f_i$ and $\widetilde{g}_i \approx g_i$ for which

$$\widetilde{f}_i = f_i + \Delta f_i \text{ and } \widetilde{g}_i = g_i + \Delta g_i \text{ for some small } \Delta f_i \text{ and } \Delta g_i.$$

- 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 $|\Delta g_i| \le \delta$.
- These approximate values are the only information we have.
- So, we have to use these approximate values when we decide on which parameters to use for the next design.

23. Realistic case (cont-d)

- So, we take $\Delta x_i = \alpha \cdot (\widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i)$, where $\widetilde{\lambda} = -\dfrac{\sum\limits_{i=1}^{n} \widetilde{f}_i \cdot \widetilde{g}_i}{\sum\limits_{i=1}^{n} (\widetilde{g}_i)^2}$.
- 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.

24. Case of the traditional engineering methodology

- In the traditional engineering methodology, we select the value α so as to make sure that:
  - no matter what the actual values f_i and g_i are,
  - we remain within the safe domain, i.e., that we still have

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

25. Based on this idea, what value α do we choose

- For relatively small changes Δx_i we have

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

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

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

- Because of our selection of Δx_i, we have $\sum_{i=1}^{n} \widetilde{g}_i \cdot \Delta x_i = 0$, thus

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

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

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

26. What value α do we choose (cont-d)

- We want to make sure that this value remains non-negative for all possible combinations of values Δg_i for which $|\Delta g_i| \leq \delta$.
- So, we want to make sure that for all these combinations, the sum $\sum_{i=1}^{n} \Delta g_i \cdot \Delta x_i$ remains smaller than or equal to ε.
- In other words, we want to make sure that the largest possible value of this sum is smaller than or equal to ε.
- The sum 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.
- When $\Delta x_i \geq 0$, this term is an increasing function of Δg_i.
- 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 thus $\delta \cdot \Delta x_i$.

27. What value α do we choose (cont-d)

- When $\Delta x_i \le 0$, this term is a decreasing function of Δg_i.
- 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 thus $-\delta \cdot \Delta x_i$.
- Both cases can be describe by a single formula $\delta \cdot |\Delta x_i|$.
- So, the largest value of the sum is $\sum\limits_{i=1}^{n} \delta \cdot |\Delta x_i| = \delta \cdot \sum\limits_{i=1}^{n} |\Delta x_i|$.
- Substituting the expression for Δx_i into this formula, we conclude that this largest value is $\delta \cdot \alpha \cdot \sum\limits_{i=1}^{n} \left| \widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i \right|$.
- Thus, the condition that this largest value is smaller than or equal to ε takes the form $\delta \cdot \alpha \cdot \sum\limits_{i=1}^{n} \left| \widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i \right| \le \varepsilon$.

28. What value α do we choose (cont-d)

- This is equivalent to $\alpha \le C \cdot \varepsilon$, where $C \stackrel{\text{def}}{=} \dfrac{\varepsilon}{\delta \cdot \sum\limits_{i=1}^{n} \left| \widetilde{f}_i + \widetilde{\lambda} \cdot \widetilde{g}_i \right|}$.
- So, 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.

29. Case of the moving-fast software-motivated methodology

- In this case, we are not limiting our next design by any constraints.
- So we can make a big step – probably violating the constraints, but then moving them back.
- Here, we do not having any slowing-down inequality like before.
- 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.

30. 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 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.