How to Generate Worst-Case Scenarios When Testing Already Deployed Systems Against New Situations

Francisco Zapata\textsuperscript{1}, Ricardo Pineda\textsuperscript{1}, and Martine Ceberio\textsuperscript{2}

\textsuperscript{1}Research Institute for Manufacturing & Engineering Systems (RIMES)  
\textsuperscript{2}Department of Computer Science  
University of Texas at El Paso  
500 W. University, El Paso, TX 79968, USA  
\textsuperscript{1}\texttt{fazapatagonzalez@utep.edu, rlpineda@utep.edu},  
\textsuperscript{2}\texttt{mceberio@utep.edu}
1. Traditional Systems Engineering Approach

- Traditionally, a system of interest (SOI) is developed by eliciting requirements from the stakeholders.
- These requirements are analyzed to build an architectural design that will drive the system development.
- Through an iterative process the system is constantly refined via:
  - elicitation and update of requirements,
  - design,
  - development, and
  - testing.
- Eventually, a final product is obtained.
- In this approach, the development of the SOI is limited to the requirements specified by the stakeholders.
- Here, emergent behavior is not welcomed.
2. Systems of Systems

- Since the 1990s:
  - advances in Information and Communication Technologies (ICT)
  - have enabled greater capabilities to exchange information between systems in near real-time.

- The integration of these independently developed systems required:
  - communication interface standards,
  - information models, and
  - inter-operatibility standards.

- This integration need has given birth to a new kind of meta-systems called Systems of Systems (SoS).

- Example: an airplane contains navigation, propulsion, GPS, communication, and other systems.
3. Systems of Systems (cont-d)

• A SoS is a system of interest which is:
  – a collection of large-scale, heterogeneous systems,
  – that inter-operate to achieve a greater common objective.

• A SoS is characterized by the following attributes:
  – operational independence,
  – managerial independence,
  – SoS evolutionary development,
  – SoS incremental functionality (knowledge domains),
  – geographical distribution.

• For constituent systems, new behavior is not welcomed.

• But for the meta-system, some new emerging behavior may be welcomed.
4. Formulation of the Problem

• Before a complex system is deployed:
  – Integration, Verification, Validation, Test and Evaluation (IVVT&E) methodologies
  – are applied to known well-defined operational scenarios.

• Once the system is deployed, new possible scenarios may emerge.

• It is desirable to develop methodologies to test a system against such emergent scenarios:
  – an unmanned Aircraft System (UAS) encounters new scenarios that were not predicted;
  – a health care monitoring system may encounter a new illness that was not known before.
5. **Specific Example**

- In this paper, we start the analysis by considering the simplest example.

- As such an example, we take an automatic system that helps prevent a car from getting too close to the walls of a freeway.

- At first glance, all we need for this is a sensing system that measures a distance $x$ from a car to an obstacle.

- There are usually several distance sensors, and the system is set up to work well in the expected situations.

- The problem starts when we encounter a new unexpected situation, e.g., a hole in the nearby wall.
6. Specific Example (cont-d)

- In the case of a hole in the wall:
  - some sensors measure the distance to a wall, while
  - other sensors measure the distance to a next far-away wall (located very far from the road).
- As a result, the existing algorithms may under-estimate the distance to the obstacle.
- So, even when the car is very close to the wall, the system may operate under the false impression of safety.
7. Need for Generating Worst-Case Scenarios

- Once the system designers realize that novel situations are possible:
  - they can come up with methods to improve the system’s performance on non-standard situations;
  - then, they need to test these methods.

- A usual way of testing a system is to test it on worst-case scenarios.

- So, we face a question of generating such worst-case scenarios.

- In this talk, we explore:
  - the ways of generating worst-case scenarios to validate system behavior under unexpected scenarios
  - on the example of the above car problem.
8. How the Distance-Measuring System Is Set Up

Now: A Simplified Description

- The distance-measuring system usually involve several sensors to account for robustness (redundancy).
- Each of the sensors produces a measurement result $x_i$.
- So, we need to estimate the actual distance $d$ based on these measurement results $x_1, \ldots, x_n$.
- Because of the measurement noise, for each distance $d$, we get slightly different values $x_i \approx d$.
- In many cases, the measurement error is normally distributed, with a standard deviation $\sigma$.
- In other words, for each result $x_i$, we have a probability distribution with the probability density

$$
\rho_{d,i}(x_i) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp \left( -\frac{(x_i - d)^2}{2\sigma^2} \right).
$$

9. How the Distance-Measuring System Is Set Up Now (cont-d)

- Measurement errors corresponding to different measurements are usually independent.
- So, the probability density \( \rho_d(x) \) for the vector \( x = (x_1, \ldots, x_n) \) of measurement results is a product:
  \[
  \rho_d(x_1, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp \left( -\frac{(x_i - d)^2}{2\sigma^2} \right).
  \]
- As a desired estimate \( d \) for the distance, it is reasonable to select the most probable value \( d \),
- In other words, we select the value \( d \) for which the probability \( \rho_d(x_1, \ldots, x_n) \) is the largest possible.
- Equating the derivative to 0, we get an estimate
  \[
  \bar{x} = \frac{x_1 + \ldots + x_n}{n}.
  \]
10. Criterion for Selecting a Worst-Case Scenario

- **Reminder:** a reasonable way to estimate the distance $d$ is to take the average $\bar{x}$ of measured values $x_1, \ldots, x_n$.
- This average works well in standard situations.
- In non-standard situations, an alert is needed when the smallest $m$ of the distances is dangerously small:
  \[ m \overset{\text{def}}{=} \min(x_1, \ldots, x_n) \ll d_{\text{min}}. \]
- When the minimum $m$ is close to the average $\bar{x}$, the situation is not so bad.
- Situation is bad when there is a drastic difference between $\bar{x}$ and $m$.
- The worst-case scenario is when the difference $\bar{x} - m$ is the largest:
  \[ \bar{x} - m = \frac{x_1 + \ldots + x_n}{n} - \min(x_1, \ldots, x_n) \to \max. \]
11. Crisp Case

- First, we consider the crisp case.
- In this case, each distance $x_i$ can take arbitrary value from the interval $[0, D]$, for some constant $D$.
- In this case, we need to maximize the difference $\bar{x} - m$ under the constraints that $0 \leq x_i \leq D$.
- In this case, we assume:
  - that we know the exact bound $D$ on the possible distances $x_i$, and
  - that we have no information about which combinations $x = (x_1, \ldots, x_n)$ are more probable.
12. Case Study: Algorithmic Analysis

- The problem of exactly maximizing a given f-n is computationally difficult (NP-hard), i.e., we cannot have:
  - an efficient (feasible) algorithm
  - that always provides an exact solution to the optimization problem.

- Since *exact* optimization is difficult, we need to use *approximate* optimization algorithms $A$.

- Most known optimization algorithms $A$ (e.g., gradient descent) use derivatives of the objective function.

- In our case, the objective function is not differentiable, since $\min(x_1, x_2)$ is not differentiable when $x_1 = x_2$.

- We thus need $A$ which do not require derivatives; the simplest such algor. $A$ is *component-wise optimization*. 

- We start with some initial values $x_1^{(0)}, \ldots, x_n^{(0)}$.
- Then, we fix all the values but $x_1$, i.e., we take
  $$x_2 = x_2^{(0)}, \ldots, x_n = x_n^{(0)}.$$  
- We find the value $x_1^{(1)}$ for which the following expression is the largest possible:
  $$f \left( x_1, x_2^{(0)}, \ldots, x_n^{(0)} \right).$$
- Then, we fix all the values but $x_2$, i.e., we take
  $$x_1 = x_1^{(1)}, x_3 = x_3^{(0)}, \ldots, x_n = x_n^{(0)}.$$  
- We find the value $x_2^{(1)}$ for which the following expression is the largest possible:
  $$f \left( x_1^{(1)}, x_2, x_3^{(0)} \ldots, x_n^{(0)} \right).$$

- Once $x_1^{(1)}$ and $x_2^{(1)}$ are found, we perform similar computations to find new values
  
  $$x_3^{(1)}, x_4^{(1)}, \ldots, x_n^{(1)}.$$ 

- Once the new values $x_1^{(1)}, \ldots, x_n^{(1)}$ of all the variables $x_1, \ldots, x_n$ are found, we repeat the whole cycle.

- Thus, we find the new value
  
  $$x_1^{(2)}, \ldots, x_n^{(2)}.$$ 

- If needed, we repeat the whole cycle again, getting the values
  
  $$x_1^{(3)}, \ldots, x_n^{(3)}.$$ 

- If necessary, we repeat this cycle several times.

- We stop when we do not get any improvement.
15. Component-Wise Optimization: A Formal Description

• We start with some initial values $x_1^{(0)}, \ldots, x_n^{(0)}$.
• Each iteration consists of $n$ stages $i = 1, \ldots, n$.
• On each stage $i$:
  – we fix the previously obtained values of all the variables except for $x_i$;
  – as $x_i^{(k+1)}$, we take a value $x_i$ for which the following expression is the largest:
    $$f \left( x_1^{(k+1)}, \ldots, x_{i-1}^{(k+1)}, x_i, x_{i+1}^{(k)}, \ldots, x_n^{(k)} \right).$$
• We stop when for some appropriate $\varepsilon > 0$, for all $i$, we have:
  $$\left| x_i^{(k+1)} - x_i^{(k)} \right| \leq \varepsilon.$$
16. Applying the Algorithm to Our Problem

- Let us start with equal values:
  \[ x_1^{(0)} = \ldots = x_n^{(0)} = d_0 \text{ for some appropriate } d_0. \]

- 1st stage: select \( x_1 \) that maximizes the difference:
  \[
  D(x_1) = \frac{x_1 + d_0 + \ldots + d_0}{n} - \min(x_1, d_0, \ldots, d_0) = \frac{x_1 + (n - 1) \cdot d_0}{n} - \min(x_1, d_0).
  \]

- When \( x_1 \in [0, d_0] \), we have \( \min(x_1, d_0) = x_1 \) and thus,
  \[
  D(x_1) = \frac{x_1 + (n - 1) \cdot d_0}{n} - x_1 = \frac{n - 1}{n} \cdot d_0 - \frac{n - 1}{n} \cdot x_1.
  \]

- This function decreases with \( x_1 \), so the difference is the largest when \( x_1 = 0 \), and is equal to \( D(0) = \frac{n - 1}{n} \cdot d_0. \)
17. Algorithm: 1st Stage (cont-d)

- **Reminder:**
  - \( D(x_1) = \frac{x_1 + (n - 1) \cdot d_0}{n} - \min(x_1, d_0); \)
  - when \( x_1 \in [0, d_0] \), the max is \( D(0) = \frac{n - 1}{n} \cdot d_0. \)
  - When \( x_1 \in [d_0, D] \), we have \( \min(x_1, d_0) = d_0 \), so the difference equals \( D(x_1) = \frac{x_1 + (n - 1) \cdot d_0}{n} - d_0. \)
  - This function increases with \( x_1 \), so its largest value is when \( x_1 = D \), and equals to
    \[
    D(D) = \frac{n - 1}{n} \cdot d_0 + \frac{1}{n} \cdot D - d_0 = \frac{1}{n} \cdot (D - d_0).
    \]
  - \( x_1 = 0 \) leads to the larger difference if \( D \leq d_0 \cdot n \); so:
    - if \( D \leq d_0 \cdot n \), then we take \( x_1^{(1)} = 0; \)
    - if \( D \geq d_0 \cdot n \), then we take \( x_1^{(1)} = D. \)
18. Algorithm: Next Stages and Conclusion

- On the 2nd stage, we fix $x_1 = x_1^{(1)}$ and find $x_2$ that maximizes the difference.
- On the 3rd stage, we fix $x_2 = x_2^{(1)}$ and find $x_3$, etc.
- When $x_1^{(1)} = 0$, we get $x_2^{(1)} = \ldots = x_n^{(1)} = D$; the corresponding difference is equal to $\frac{(n-1) \cdot D}{n}$.
- When $x_1^{(1)} = D$ we get $x_2^{(1)} = \ldots = x_{n-1}^{(1)} = D$ and $x_n^{(1)} = 0$, with the same difference $\frac{(n-1) \cdot D}{n}$.
- One can prove that this is actually the global maximum of the difference.
- *Conclusion*: we recommend to use component-wise optimization to find the worst-case scenario.
19. From Crisp Case to a More Realistic Case of Soft Constraints

- In practice, we only know such bounds $D$ with uncertainty.
- We also have some information about which combinations are more probable and which are less probable.
- This information is usually described in imprecise terms, by using words from a natural language.
- It is therefore reasonable to use fuzzy techniques to describe this information.
- In the fuzzy approach, we assign, to every combination $x$, a degree $\mu(x)$ to which $x$ is probable.
- Then, to find the worst-case scenario, we optimize the objective function under such soft constraints.
20. How to Optimize Under Soft Constraints

• For this optimization, we can use known techniques of optimizing a (crisp) function $f(x)$ over fuzzy sets.

• For example, we can use Bellman-Zadeh techniques in which we maximize the expression

$$g(x) \overset{\text{def}}{=} \min \left( \frac{f(x) - y}{\bar{y} - y}, \mu(x) \right),$$

where:

• $y$ and $\bar{y}$ are the minimum and maximum of the function $f(x)$ over the entire domain,

• the ratio $\frac{f(x) - y}{\bar{y} - y}$ describes to what extent the vector $x$ is optimal, and

• $g(x)$ means that $x$ is optimal and satisfies the constraints – with min corresponding to “and”.