What is the Right Context for an Engineering Problem: Finding Such a Context is NP-Hard

Martine Ceberio¹, Vladik Kreinovich¹
Hung T. Nguyen²,³, Sngsak Sriboonchitta³, and Rujira Ouncharoen⁴

¹Department of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA, mceberio@utep.edu, vladik@utep.edu,
²Department of Mathematical Sciences, New Mexico State University
Las Cruces, New Mexico 88003, USA, hunguyen@nmsu.edu
³Faculty of Economics and ⁴Department of Mathematics
Chiang Mai University, Thailand
songsakecon@gmail.com, rujira.o@cmu.ac.th
1. In Engineering, It Is Important to Come up with an Appropriate Context

- One of the main objectives of engineering is to come up with a design or control with required functionality.
- In general, this problem is NP-hard.
- Thus, to be able to use feasible algorithms, we must restrict the problem to an appropriate context.
- Ideally, we should use the most general context – to help solve future problems as well.
- Thus, it is desirable to find the most general context in which the corresponding problem is still feasible.
- We prove that finding the optimal context is itself an NP-hard problem, so Comput. Intel. (CI) is needed.
- We show how CI can help on all the stages of solving an engineering problem.
2. Brief Reminder: What Is a Feasible Algorithm

- Some algorithms are practically useful (feasible).
- However, some exhaustive-search algorithms try all $2^n$ binary (0-1) sequences of length $n$.
- For a reasonable bit size $n = 300$, the running time $2^{300}$ exceeds the lifetime of the Universe.
- Thus, exhaustive search algorithms are not feasible.
- Usually, an algorithm $\mathcal{A}$ is called feasible if its running time $t_\mathcal{A}(x)$ is bounded by a polynomial $P(\text{len}(x))$.
- This definition is not perfect: e.g., $10^{100} \cdot n$ is feasible in the above sense, but it is not practically feasible.
- However, this is the best definition we have.
3. NP-Hard Problems: A Brief Reminder

• In computing, we usually consider problems for which:
  – once we have a candidate for a solution,
  – we can feasibly check whether this candidate is indeed a solution.

• The class of all such problems is denoted NP.

• It is still not known whether it is possible to feasibly solve all the problems from the class NP: $\text{NP} \neq \text{P}$.

• What is known is that:
  – some problems $\mathcal{P}_0$ are the hardest in the class NP,
  – meaning that any $\mathcal{P} \in \text{NP}$ can be reduced to $\mathcal{P}_0$.

• Such hardest problems are called $\text{NP-hard}$. 
4. Many Practical Problems Are NP-Hard

- Many general practical problems are NP-hard.
- This means that most probably, no feasible algorithm can solve all particular cases.
- To make the problem feasible, it is important to restrict the problem.
- It is desirable to consider restrictions which are as general as possible.
- Let \( m \) be the number of possible ways of restricting the problem.
- For each of these ways \( i = 1, \ldots, m \), let \( p_i \) denote the fraction of the problems that satisfy this restriction.
- It is reasonable to consider restrictions which are independent from each other.
5. First Result

- Then, the fraction \( p(I) \) of problems that satisfy all restrictions \( i \in I \) is \( p(I) = \prod_{i \in I} p_i \).

- The more we restrict the problem, the more probable it is that the restricted class is feasibly solvable.

- Let us denote the largest fraction for which the problem becomes feasible solvable by \( p_0 \).

- Simple description: \( I \) is feasible \( \iff p(I) \leq p_0 \).

- Resulting problem:
  - we are given the values \( p_0, p_1, \ldots, p_m \),
  - we want to find a set \( I \subseteq \{1, \ldots, m\} \) for which \( p(I) \rightarrow \max \) under condition \( p(I) \leq p_0 \).

- Our first result is that this problem is NP-hard.

- So, it is NP-hard to find the most general restriction under which the problem remains feasible.
6. The Above Result Necessitates the Use of Computational Intelligence

- It is not possible to have an automatic algorithm that would always solve the context-finding problem.

- This means that to solve this problem, we must use our creativity, we must use our intelligence.

- We need to use intelligence, and we also need to use computers.

- Thus, we need to translate intelligent techniques into computer-understandable form.

- This is exactly what computational intelligence is about.

- Let us give examples how (computational) intelligence can help on all stages of solving a problem.
7. Stages of Solving Engineering Problems

- In precise terms, the goal of an engineering problem is to change the values of some quantities $y$:
  - transportation means changing the spatial coordinates of an object,
  - heating means changing the temperature inside a building, etc.
- Rarely can we directly change the desired quantity.
- Usually, this can be achieved by changing some easier-to-change related quantities $x_1, \ldots, x_n$.
- Thus, we need to find the dependence between $y$ and $x_1, \ldots, x_n$.
- This is an important first stage of the process of solving the engineering problem.
8. Stages (cont-d)

- Once the dependence is found:
  - for each engineering design,
  - we can predict the future values of different quantities.

- Thus, we can check how well the given design satisfies our requirements.

- This analysis of possible solutions forms the second stage.

- Once we have found a satisfactory design, a natural third stage is optimization.
9. First Stage: Prior Knowledge about Casuality Can Help to Find the Dependence

- Let us consider the simplest possible linear dependence models
  \[ y = a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n. \]

- Usually, the parameters \( a_i \) are found from the observations \( x_i^{(k)} \) and \( y^{(k)} \) by Least Squares:
  \[
  E \sum_{k=1}^{E} \left( y^{(k)} - \left( a_0 + a_1 \cdot x_1^{(k)} + \ldots + a_n \cdot x_n^{(k)} \right) \right)^2 \to \min.
  \]

- In practice, we often do not know which quantities \( a_i \) are relevant, so we consider \( N \gg n \) variables
  \[ y \approx a_0 + a_1 \cdot x_1 + \ldots + a_N \cdot x_N. \]

- Ideally, we should get \( a_{n+1} = \ldots = 0 \), but due to measurement errors, \( a_{n+1} \neq 0 \).

- The resulting noise \( a_{n+1} \cdot x_{n+1} + \ldots \) decreases the accuracy of the resulting model.
10. First Stage (cont-d)

• The noise \( a_{n+1} \cdot x_{n+1} + \ldots \) decreases the accuracy of the resulting model.

• If we know which quantities \( x_i \) are irrelevant, we can drastically increase the model’s accuracy.

• Often, experts can only provide degrees \( d_i \) to which each \( x_i \) is relevant.

• What we can then do is select \( x_i \) with highest degrees \( d_i \geq d_0 \).

• We can try all possible \( d_0 \) and see which value leads to the most accurate model.
11. Second Stage: Long-Term vs. Short-Term Predictions

- On the second stage, we predict the future behavior of the system.

- In general, the further we in the future we want to predict, the more difficult this prediction.

- It is possible to predict technological advances for the new few years.

- However, it is next to impossible to predict technology in the next century.

- It is possible to predict tomorrow’s weather.

- However, it is practically impossible to accurately predict weather in ten years.
12. Second Stage: A Problem

- In the above examples, we have only a very crude knowledge of the system’s dynamics.

- In engineering, often, we have an exactly opposite phenomenon:
  - we can predict the long-term consequences really well, but
  - it is difficult to make short-term predictions.

- For example, if we trace a flight going from Cape Town to London, then
  - we can safely predict that in a few hours, it will be approaching the English Channel, but
  - where the plane will be an hour after the flight depends heavily on the winds, turbulence zones, etc.
13. Second Stage: A Problem (cont-d)

• In general, this is very counter-intuitive:
  – if we cannot accurately predict the state of a system short-term,
  – how come we can reasonably accurately predict its long-term behavior?

• Let’s consider the simplest dynamical model:
  – the state of the system is described by a single quantity \( y \);
  – the control is described by a single parameter \( u \),
  – there is a single random process \( r(t) \) with 0 mean, and
  – the dependence \( \frac{dy}{dt} = f(y, u, r) \) is linear:

\[
\frac{dy}{dt} = b_0 + b_1 \cdot y + b_2 \cdot u + b_3 \cdot r.
\]
14. Long Term vs. Short-Term Explained

- Here, \( \frac{dy}{dt} = b_0 + b_1 \cdot y + b_2 \cdot u + b_3 \cdot r \).

- In engineering, when \( u = r = 0 \), the state does not change, so \( \frac{dy}{dt} = b_2 \cdot u + b_3 \cdot r \).

- Thus, \( y_K - y_0 \approx K \cdot (A \cdot u) + B \cdot \sum_{k=1}^{K} r(t_k) \).

- The error term \( \sum_{k=1}^{K} r(t_k) \) is a sum of \( K \) independent identically distributed random variables with 0 mean.

- So, its variance grows as \( K \), and this term – as \( \sqrt{K} \).

- Thus, the relative error of the estimate \( K \cdot (A \cdot u) \) for the difference \( y_K - y_0 \) decreases with \( K \) as \( \frac{\sqrt{K}}{K} = \frac{1}{\sqrt{K}} \).
15. Long Term vs. Short-Term Explained (cont-d)

- The relative error of the estimate $K \cdot (A \cdot u)$ for the difference $y_K - y_0$ decreases with $K$ as $\frac{\sqrt{K}}{K} = \frac{1}{\sqrt{K}}$.

- So, the farther in the future we want to predict, i.e., the larger $K$, the more accurate our prediction.

- This explains why in many engineering systems:
  - it is possible to make long-term predictions, but
  - it is not possible to make short-term ones.

- CI can help estimate the random error and thus, to avoid inaccurate short-term predictions.
16. Third Stage: If It Ain’t Broke, Don’t Fix It

- On the third stage of solving an engineering problem, we try to come up with an optimal control.
- At first glance, it seems like a very natural idea:
  - we use the (approximate) model to find the optimal control, then
  - we apply this optimal control, and
  - we expect the situation to improve.
- Yes, in practice, we expect some deviations from optimality, since the model is approximate.
- However, overall, we expect some improvement.
- Surprisingly, sometimes, an application of the seemingly optimal control only makes the situation worse.
17. Third Stage (cont-d)

- For example, sometimes, a medical treatment:
  - which is beneficial when the state is very different from the norm
  - becomes harmful when the difference from the normal state is small.

- For example, when a patient has high fever, it beneficial to give him/her medicine that reduces this fever.

- However, in case of a slight fever:
  - such medicine will only reduce the body’s ability to fight the disease and
  - thus, delay the patient’s recovery.
18. Similar Phenomena Are Known for Engineering Problems

- When we control a robot, it makes sense to promptly correct robot’s deviations from the desired trajectory.
- However, if we apply similar corrections for small deviations, then:
  - the robot will start wobbling and
  - its motion will be less efficient.
- We show, on a very simple example, that:
  - while this phenomenon may sound counterintuitive,
  - it actually naturally follows from the corresponding equations.
- In the simplest approximation, if we start at a state $y_0$, then at the next moment of time, we get a new state
  \[ y_1 = y_0 + A \cdot u + B \cdot r. \]
19. Analysis of the Problem

- We know that \( y_1 = y_0 + A \cdot u + B \cdot r \).

- When we select a control \( u \), we do not know the value \( r \), we only know that \( y_1 \approx y_0 + A \cdot u \).

- So, it is reasonable to select \( u \) for which the corrected state \( y_0 + A \cdot u \) is equal to the desired state \( Y \):
  \[
  y_0 + A \cdot u = Y.
  \]

- Due to the random error \( B \cdot r \), the actual state will be, in general, different from \( Y \): \( y_1 = Y + B \cdot r \).

- When \( y_0 \) is very close to \( Y \),
  - but the standard deviation of \( r \) is large,
  - we may end up much further away from the desired state \( Y \) that we originally were.

- In this case, indeed, a seemingly optimal control only makes things worse.
20. How Can We Avoid Such Situations?

- *Natural idea:* only apply control when the deviation from the ideal state exceeds a certain threshold $t$.
- Often, we do not know much about $r$.
- In this case, a natural idea is to use expert knowledge to estimate $t$. 
21. How Can This Be Used in a Practice?

- In general, most engineering problems are computationally intractable (NP-hard).
- So, it is important to find a context that will enable us to make the corresponding problem feasible.
- The problem of finding the optimal context is computationally intractable.
- Thus, it is not possible to come up with a general method for finding such context.
- This context has to come from the expert’s analysis of the problem.
- Our examples show that in many practical situations, expert knowledge indeed helps.
- These examples cover all three stages of engineering design.
22. Acknowledgment

- This work is supported by Chiang Mai University, Thailand.
- In particular, we acknowledge the support of the Center of Excellence in Econometrics, Faculty of Economics, Chiang Mai University, Thailand.
- This work was also supported in part by the National Science Foundation grants:
  - HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and
  - DUE-0926721.
23. Appendix: Proof of NP-Hardness

- **Known**: the following *subset sum* problem is NP-hard:
  - *Given*: positive integers $s_0, s_1, \ldots, s_m$,
  - *to find* $I \subseteq \{1, \ldots, m\}$ for which $\sum_{i \in I} s_i = s_0$.

- By definition of NP-hardness, every $\mathcal{P} \in \text{NP}$ can be reduced to subset sum.

- So, if we reduce subset sum to our problem, this will prove its NP-hardness.

- The reduction is $p_i = 2^{-s_i}$ and $p_0 = 2^{-s_0}$.

- Then, $\prod_{i \in I} p_i = \prod_{i \in I} 2^{-s_i} = 2^{-s_0} = p_0$ iff $\sum_{i \in I} s_i = s_0$.

- So, if the subset sum problem has a solution, we get an optimal context.

- Thus, our problem is indeed NP-hard.