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

Martine Ceberio and
Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968, USA
Emails: mceberio@utep.edu
vladik@utep.edu

Hung T. Nguyen
Department of Mathematical Sciences
New Mexico State University
Las Cruces, New Mexico 88003, USA
and Faculty of Econmics
Chiang Mai University, Thailand
Email: hunguyen@nmsu.edu

Songsak Sriboonchitta[1] and
Rujira Ouncharoen[2]
[1]Faculty of Economics
[2]Department of Mathematcs
Chiang Mai University, Thailand
Emails: songsakecon@gmail.com
rujira.o@cmu.ac.th

***Abstract*—In the general case, most computational engineering problems are NP-hard. So, to make the problem feasible, it is important to restrict this problem. Ideally, we should use the most general context in which the problem is still feasible. In this paper, we start with a simple proof that finding such most general context is itself an NP-hard problem. Since it is not possible to find the appropriate context by utilizing a general algorithm, it is therefore necessary to be creative – i.e., in effect, to use computational intelligence techniques. On three examples, we show how such techniques can help us come up with the appropriate context. These examples explain why it is beneficial to take knowledge about causality into account when processing data, why sometimes long-term predictions are easier than short-term ones, and why often for small deviations, a straightforward application of a seemingly optimal control only makes the situation worse.**

I. In Engineering, It Is Important to Come up with an Appropriate Context: an Informal Formulation of the Problem

Formulation of the problem. One of the main objectives of engineering is to come up with a design and/or control that satisfies required functionality. It is know that in most general form, the problem of finding a design (control) that satisfies a given constraint is NP-hard, i.e., crudely speaking, cannot be solved by a feasible algorithm.

Thus, to be able to use feasible algorithms, it is necessary to restrict the formulation, i.e., to formulate the corresponding problem in an appropriate context.

If this context is too narrow, if the corresponding algorithm can be extended to a more general context without losing feasibility, then why not do it? If we use a context which is too narrow, then in similar situations, instead of using the same algorithm, we would have to solve the problem again. Thus, it is desirable to find the most general context in which the corresponding problem is still feasible. Finding such context is the problem that we will consider in this paper.

What we do in this paper. In Section 2, we provide exact definitions and prove the main (simple) theoretical result of this paper – that finding the optimal context is itself an NP-hard problem. In a short Section 3, we explain that this result necessitates the use of computational intelligence. In the following sections, we give examples of how computational intelligence can help, examples that cover all the stages of solving an engineering problem.

II. Finding the Optimal Context Is NP-Hard: Proof (by a Simple Reduction)

Brief reminder: what is a feasible algorithm. Some algorithms – e.g., algorithms whose running time grows linearly or quadratically with the size of the input – are practically useful (*feasible*). Other algorithms – e.g., many exhaustive search algorithms – require computation time that grows exponentially as the input size. For example, many exhaustive-search algorithms require that we try all 2^n binary (0-1) sequences of length n.

For such exponential-time algorithms, even when the bit size of the input is reasonable, e.g., $n = 300$, the corresponding number of computational steps 2^{300} exceeds the lifetime of the Universe. Thus, such algorithms are not feasible.

This distinction is usually described as follows (see, e.g., [7]:

- algorithms $\mathcal{A}$ whose worst-case running time $t_{\mathcal{A}}(n)$ of inputs of bit size n (a.k.a. computational complexity) is bounded by a polynomial $P(n)$ of n are called *feasible*, while
- all other algorithms are called *non-feasible*.

Comment. The above definition is not perfect, since it does not fully capture the intuitive notion of a feasible algorithm:

- an algorithm whose computation time grows, with the bit size n of the input, as $10^{100} \cdot n$ is feasible in the above sense, but it is not practically feasible, since even for $n = 1$, it needed more time than the lifetime of the Universe;
- on the other hand, an algorithm with an exponentially increasing running time $\exp(10^{-40} \cdot n)$ – an algorithm which is not feasible according to the above definition – runs very fast on all inputs of realistic bit size n, and is, thus, feasible from the practical viewpoint.

However, in spite of this limitation, no better definition of a feasible algorithm is known. So, in this paper, we will use this definition of a feasible algorithm.

NP-hard problems: a brief reminder. In computer science, 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 known as the class NP.

It is still not known (2015) whether it is possible to solve all the problems from the class NP in feasible time, i.e., whether the class NP coincides with the class P of all the problems that can be feasibly solved. Most computer scientists believe that this is not possible, i.e., that P≠NP. What *is* known is that there are some problems $\mathcal{P}_0$ from the class NP such that solving any other problem from the class NP can be feasibly reduced to solving this problem. In this sense, these problems are the hardest in the class NP. Such hardest problems are called *NP-hard* [7].

Many important practical problems are NP-hard. Many important practice-related problem are NP-hard if we consider them in their most general form; see, e.g., [7].

Crudely speaking, the NP-hardness of a problem $\mathcal{P}$ means that unless P=NP (which, as we have mentioned, most computer scientists believe to impossible), no feasible algorithm is possible that would solve all particular cases of this problem.

Need for restriction. Since the most general problems are NP-hard, to make the problem feasible, it is important to restrict the problem.

It is desirable to consider restrictions which are as general as possible. Within the limitation of feasibility, it is desirable to come up with restrictions which are as general as possible.

What we do in this section. In this section, we prove that finding such most general feasible generalization is itself an NP-hard problem. In other words, we prove that finding the optimal context for feasibility is an NP-hard problem.

How to find the most general restriction, under which the problem remains feasible: analysis of the problem. Our objective is to restrict the original generic NP-hard problem so that it becomes feasible – and to find the most general of such restrictions.

It is known that the more we restrict the original problem, the more reasonable it is to expect that this restriction will lead to the possibility of a feasible algorithm for solving thus restricted problem.

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. In this case, if we simultaneously impose two or more restrictions i, j, etc., then the fraction of problems that satisfy all these restrictions is equal to the product $p_i \cdot p_j \cdot$ of the corresponding fractions.

Under this independence assumption, if from the class $\{1, \ldots, m\}$ of all possible restrictions, we select a subclass $I \subset \{1, \ldots, m\}$, then the fraction of problems that satisfy all these restrictions is equal to $\prod\limits_{i \in I} p_i$.

The more we restrict the problem, the more probable it is that the resulting restricted problem can be solved by a feasible algorithm. Let us denote the largest fraction for which the problem becomes feasible solvable by p_0. In this simplified description:

- if we have a class of restrictions that forms a fraction p_0 (or smaller) of the original class of problems, then it is possible to have a feasible algorithm for solving all the problems from this class;
- on the other hand, if some class of problem forms a fraction of $> p_0$, then no such feasible algorithm is possible.

We are interested in selecting the restrictions in such a way that the resulting problems form the feasible subclass of the largest possible size (p_0). Let us formulate this problem in precise terms.

How to find the most general restriction under which the problem remains feasible: a formalization of the main problem. We are *given:*

- fractions $p_1, \ldots, p_m \in (0, 1)$ that describe possible constraints, and
- the fraction p_0 with the property that:
  - subclasses of size $\le p_0$ are feasibly solvable, while
  - subclasses of size $> p_0$ are not feasible solvable.

The problem is *to check* whether it is possible to select a set of constraints $I \subseteq \{1, \ldots, m\}$ for which the resulting class of problem is of the largest feasibly solvable size, i.e., for which

$$\prod_{i \in I} p_i = p_0. \tag{1}$$

The above problem is NP-hard: a proof. Let us prove that the above problem is, in general, NP-hard, i.e., that it is NP-hard to find the most general restriction under which the problem remains feasible.

In precise terms, we need to prove that the problem of finding a subset I for which the equality (1) is true is NP-hard. Indeed, it is known that the following *subset sum* problem is NP-hard [7]:

- *Given:* m positive integers $s_1, \ldots, s_m$ and a positive integer s_0,
- *to find* a subset $I \subseteq \{1, \ldots, m\}$ for which

$$\sum_{i \in I} s_i = s_0. \tag{2}$$

This problem is also known as the *problem of exact change*: if we have coins of denominations $s_1, \ldots, s_m$, is it possible to form an amount s_0 by using some of these coins?

Let us prove that our problem is NP-hard by reducing it to the subset problem, i.e., by showing that:

- if we can feasibly solve our problem,
- then we will be able to feasibly solve the subset sum problem.

Since the subset sum problem is known to be NP-hard, this will prove that our problem is NP-hard as well.

The desired reduction can be obtained as follows: take $p_i = 2^{-s_i}$ and $p_0 = 2^{-s_0}$. Then, the equality (1) takes the form

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

By taking the binary logarithm of both sides of this equality (and taking into account that the logarithm of the product is equal to the sum of the logarithms), we conclude that $\sum\limits_{i \in I} s_i = s_0$. Vice versa, if this equality holds, then, for $p_i = 2^{-s_i}$ and $p_0 = 2^{-s_0}$, the equality (2) also holds.

This reduction shows that our problem is indeed NP-hard.

III. The Above Result Necessitates the Use of Computational Intelligence

The meaning of the above result: (feasible) algorithms are not enough. The result of the previous section shows that it is not possible to have an automatic algorithm that would always solve all the particular cases of the general context-finding problem. The fact that a general algorithm is not possible means that to solve this problem, we must use our creativity, we must use our intelligence.

We need to use computational intelligence. We need to use intelligence, and we also need to use computers. Thus, we need to translate intelligent techniques into computer-understandable form – and this is exactly what computational intelligence is about.

Let us give examples. In the following sections, we will give examples of how (computational) intelligence can help find an appropriate context for an engineering problem.

IV. We Will Provide Examples from All Stages of Solving Engineering Problems

How engineering problem are solved? We will show that computational intelligence can be helpful on all the stages of solving the engineering problem. To show this, let us recall what are the main stages of solving such problems.

In mathematical terms, the goal of an engineering problem is to change the values of some quantities: transportation means changing the spatial coordinates of an objects, heating means changing the temperature inside a building, etc.

First stage: finding the dependence. Rarely can we directly change the desired quantity. Usually, this can be achieved by changing some easier-to-change related quantities. For example, since we cannot directly drag a load across the sea, we place it in a container, place such containers into a specially designed ship, burn oil in a ship's engine, etc. To come up an appropriate engineering scheme, we first need to find the dependence between the desired quantity y and the corresponding easier-to-change quantities $x_1, \ldots, x_n$. This is an importance *first stage* of the process of solving the engineering problem.

Second stage: analysis. Once the dependence is found, for each engineering design, we can predict the future values of different quantities – and thus, we can check how well the given design satisfies our requirements. This analysis of possible solutions forms the *second stage* of the solving the engineering problem – engineers (or computers trained by engineers) come up with possible designs, and then we check these designs once by one until we find a satisfactory one.

Third stage: optimization. Once we have found a satisfactory design, a natural next step is *optimization*: trying to come up with a perfect design, with optimal control, etc. This optimization forms the last *third stage* of engineering design.

What we do in the following sections. In the following sections, we show that computational intelligence can help to find the proper context on all these three stages of engineering design. In each of the three cases, we will illustrate our point on a simple example of a linear system.

What is known and what is new. All three examples describe phenomena which are (largely) known in computational intelligence. However, to the best of our knowledge, prior to this paper, these examples have not been brought together as examples of need for proper context on all the stages of engineering design.

Besides, while these examples may be known in computational intelligence community, they are not well known outside this community. For example, many economists still view the possibility of long-term predictions in situations in which short-term predictions are not possible as an important theoretical challenge.

V. First Stage: Prior Knowledge about Casuality Can Help to Find the Dependence

What is the first stage: a brief reminder. On the first stage of solving an engineering problem, we try to find the dependence between the desired difficult-to-directly-change quantity y and some easier-to-change quantities x_i. This dependence is then used to come up with an engineering design that helps us change the values of the desired quantity.

In this paper, we consider the simplest possible example. As we have mentioned, in this paper, we consider the simplest possible situations – i.e., linear models – and the simplest possible solutions – e.g., linear controls. Therefore, in this section, we consider the approximation in which the dependence of y on the appropriate easier-to-change quantities is linear, i.e., when:

$$y = a_0 + a_1 \cdot x_1 + \ldots + a_n \cdot x_n \tag{3}$$

for appropriate coefficients a_i.

What does it mean to find the dependence. For a linear model, finding the dependence means finding the corresponding coefficients a_i.

Ideal case, when we know which quantities are relevant and which are not. Let us first consider the ideal case when we know the list of quantities $x_1, \ldots, x_n$ which are important to determine y, then we can use the known observations $(x(k)_1, \ldots, x^{(k)}, y^{(k)})$, $1 \le k \le E$ for which we should have

$$y^{(k)} \approx a_0 + a_1 \cdot x_1^{(k)} + \ldots + a_n \cdot x_n^{(k)}, \tag{4}$$

and then use the usual Least Squares technique to find the values a_i for which the mean square error is the smallest possible:

$$\sum_{k=1}^{E} \left(y^{(k)} - \left(a_0 + a_1 \cdot x_1^{(k)} + \ldots + a_n \cdot x_n^{(k)} \right) \right)^2 \to \min_{a_0, \ldots, a_n} \tag{5}$$

What happens in real life, when we do not know which quantities are relevant. In real life, we often do not have guaranteed information re which quantities x_i are relevant and which are not. In such situations, instead of considering dependence of y only on relevant quantities $x_1, \ldots, x_n$, we have to consider the dependence on all possible easy-to-change quantities $x_1, \ldots, x_N$, with $N \gg n$. In this case, we use a similar Least Square approach to find the coefficients of a general linear equation:

$$y = a_0 + a_1 \cdot x_1 + \ldots + a_N \cdot x_N, \tag{6}$$

by findings the values $a_0, \ldots, a_N$ for which

$$\sum_{k=1}^{E} \left(y^{(k)} - \left(a_0 + a_1 \cdot x_1^{(k)} + \ldots + a_N \cdot x_N^{(k)} \right) \right)^2 \to \min_{a_0, \ldots, a_n}. \tag{7}$$

In the limit $E \to \infty$, the additional coefficients $a_{n+1}, \ldots, a_N$ tends to 0, but in practice, based on the measurement results, we get non-zero values for all the coefficients. Yes, we can eliminate those coefficients which are too small (e.g., smaller than a certain threshold), but still, when N is large, some of the unnecessary coefficients turn out of be greater than this threshold and thus remain.

As a result, instead of the original ideal dependence (4), we get a dependence (6) with additional terms $a_i \cdot x_i$ ($i > n$), which decrease the accuracy of the resulting model.

This decrease in accuracy may be significant. At first glance, it may seem that this decrease in accuracy causes by almost-zero coefficients a_i is small, but there may be many such terms, and, as a result, this decrease in accuracy can be significant. The possible for a drastic decrease has been illustrated, e.g., on the example of hydrological analysis in [1].

Natural solution: use (imprecise) intelligence. Since using all possible easy-to-change quantities x_i may lead to a drastic decrease in accuracy, a natural idea is to restrict ourselves only to relevant quantities x_i.

Since we do not have a guaranteed knowledge of which variables are relevant and which are not, we have to use imprecise ("fuzzy") expert knowledge about relevance. A natural way to describe such intelligence is by using fuzzy techniques [2], [5], [11], in which, to each quantity x_i, we assign a degree d_i to which, according to the expert, this quantity is relevant.

We then sort the quantities in the decreasing order of their relevance, and find the smallest value n for which the use of Least Squares leads to the approximation error below the expected threshold – e.g., by using the χ^2 criterion (see, e.g., [8]). This can done, e.g., by increasing n one by one until we reach the first value with statistical fit.

So what is an appropriate context here. Here, an appropriate context is ignoring the dependence of the desired quantity y on the irrelevant easy-to-change quantities $x_{n+1}, \ldots, x_N$.

VI. Second Stage: How Come Long-Term Predictions Are Possible but Short-Term Predictions Are Often Not?

Second stage: reminder. On the second stage of solving an engineering problem, we use the relation that we found on the first stage to predict the future behavior of the system.

A strange phenomenon: often, long-term predictions are possible but short-term predictions are not possible. 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, but it is next to impossible to predict technology in the next century. It is possible to predict tomorrow's weather, but it is practically impossible to accurately predict weather in ten years.

However, all these examples are about social and natural phenomena, where the problem is that 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. In this example, the possibility of long-term vs. short-term predictions sounds reasonable, but in general, this phenomenon 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?

How can we explain this seemingly counterintuitive phenomenon. Let us show, on a very simple example, that, while this phenomenon may sound counterintuitive, it actually naturally follows from the corresponding equations.

Let us try to come up with equations which are as simple as possible. In general, dynamical equations that describe how the state of system change with time describe the time derivatives of the parameters describing the system. These derivatives depend on the current state, on the control applied to the system, and on the unpredictable (random) processes that also affect the system.

The simplest case is when the state of the system is described by a single quantity y; as an example of such quantity, we can take the distance between the plane and its destination (London). For simplicity, we also assume that:

- the control is described by a single parameter u, and
- that there is a single random process with 0 mean whose value at each moment t will be denoted by $r(t)$.

For simplicity, let us assume that the random values corresponding to different moments of time are independent and identically distributed.

In this simple case, the corresponding differential equation takes the form

$$\frac{dy}{dt} = f(y, u, r). \tag{8}$$

As we have mentioned earlier, we consider linear systems. When the function $f(y, u, r)$ is linear, the equation (8) takes the following form:

$$\frac{dy}{dt} = b_0 + b_1 \cdot y + b_2 \cdot u + b_3 \cdot r. \tag{9}$$

In the engineering system, usually, unless we apply a control and/or unless there is a random perturbation, the state of the system does not change. In other words, when $u = r = 0$, we should have $\dfrac{du}{dt} = 0$. This requirement implies that

$$b_0 + b_1 \cdot u = 0$$

for all possible states u. Thus, the equation (9) takes an even simpler form

$$\frac{dy}{dt} = b_2 \cdot u + b_3 \cdot r. \tag{10}$$

As usual, to solve this equation, we discretize time, i.e., select a time step Δt, and and consider the value

$$y_0 = y(t_0), \ \ y_1 = y(t_0 + \Delta t), \ \ \ldots, \ \ y_k = y(t_0 + k \cdot \Delta t), \ \ \ldots$$

Then, the relation between each value $y_k = y(t_0 + k \cdot \Delta t)$ and the next value $y_{k+1} = y(t_0 + (k+1) \cdot \Delta t)$ can be described as

$$y_{k+1} - y_k \approx \Delta t \cdot \frac{dy}{dt}, \tag{11}$$

thus, as

$$y_{k+1} - y_k \approx A \cdot u + B \cdot r(t), \tag{12}$$

where we denoted $A \stackrel{\text{def}}{=} \Delta t \cdot b_2$ and $B \stackrel{\text{def}}{=} \Delta t \cdot b_3$.

When we add up the equations (12) corresponding to $k = 0, 1, \ldots, K$, we conclude that the change $y_K - y_0$ in the values of y between the starting state y_0 and the state y_K at the K-th moments is equal to:

$$y_K - y_0 \approx K \cdot (A \cdot u) + B \cdot \sum_{k=1}^{K} r(t_k). \tag{13}$$

The random values are unpredictable, the only effect that we can predict is the joint effect of controls, i.e., the term $K \cdot u$.

The error term $\sum\limits_{k=1}^{K} r(t_k)$ is a sum of K independent identically distributed random variable with 0 mean. According to the Central Limit Theorem [8], when K is large, the distribution of this sum is close to Gaussian (normal), with the variance equal to the sum of K variances – and thus, with the standard deviation that grows as $\sqrt{K}$. It is known that with a very high confidence 99.9%, the value of a normally distributed random variable with 0 mean is bounded by 3σ (6σ if require an even higher confidence $1 - 10^{-8}$). Thus, the error term grows with K as $\sqrt{K}$, while the main term grows as K. As a result, the relative error of the estimate $K \cdot (A \cdot u)$ for the difference $y_K - y_0$ decreases with K as $\dfrac{\sqrt{K}}{K} = \dfrac{1}{\sqrt{K}}$.

The farther in the future we want to predict, i.e., the larger K, the more accurate our prediction. When $r \gg u$, no accurate prediction is possible for the next moment of time, but as time goes by, predictions become more and more accurate.

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.

What can we do to avoid inaccurate predictions: use (computational) intelligence. A natural solution is *not* to make short-term predictions – since these predictions are inaccurate anyway, to only make prediction for a time period that exceeds a certain threshold.

Ideally, we should know the statistical characteristics of the random factors and use these characteristics to generate the value of the corresponding threshold.

In practice, however, we often do not know these probabilities. In this case, when we do not know a guaranteed knowledge of the random quantity r, a natural idea is to use imprecise expert knowledge of this quantity to generate the value of the appropriate threshold.

So what is an appropriate context here. Here, an appropriate context is considering predictions only after a certain moment of time – since accurate short-term predictions are not possible.

VII. Third Stage: If It Ain't Broke, Don't Fix It

Third stage: reminder. On the third stage of solving an engineering problem, we use the second-stage analysis to come up with an optimal control.

At first glance, the situation is straightforward. 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, but overall, we expect some improvement.

In practice, the results are not always good. Somewhat surprisingly, in practice, an application of the seemingly optimal control only makes the situation worse.

Our main example – that motivated this section – comes from medicine, where the 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 [6]. This seemingly counterintuitive result has been motivated by an analysis of the corresponding mathematical model, but on the phenomenological level, such phenomena are ubiquitous in medicine. 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.

Similar phenomena are known for engineering problems as well. For example, when we control a robot, it makes sense to promptly correct robot's deviations from the desired trajectory – provided that these deviations are large enough. However, if we apply similar corrections for small deviations, the robot will start wobbling and its motion will be less efficient; see, e.g., [3], [4], [9], [10].

How can we explain this seemingly counterintuitive phenomenon. Similar to the second-stage explanation, let us show, on a very simple example, that, while this phenomenon may sound counterintuitive, it actually naturally follows from the corresponding equations.

Indeed, as we mentioned in the previous section, in the simplest approximation, if we start at a state y_0, then at the next moment of time, we get a new state

$$y_1 = y_0 + A \cdot u + B \cdot r. \tag{14}$$

When we select a control, we do not know the value r of the random process, we only know that $y_1 \approx y_0 + A \cdot u$. In this case, it is reasonable to select a control for which the corrected state $y_0 + A \cdot u$ is equal to the desired state Y: $y_0 + A \cdot u = Y$. For this control, due to the random error $B \cdot r$, the actual state will be, in general, different from Y, namely, it will be equal to

$$y_1 = Y + B \cdot r. \tag{15}$$

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.

How can we avoid such situation when a seemingly optimal control only makes situation worse: use (computational) intelligence. A natural solution is *not* to apply control when the deviation from the ideal state is small, and to only apply control when the deviation from the ideal state exceeds a certain threshold.

Ideally, we should know the statistical characteristics of the random factors and use these characteristics to generate the value of the corresponding threshold.

In practice, however, we often do not know these probabilities – this was exactly the case with the 1996 AAAI-award-winning robot [3]. In this case, when we do not know a guaranteed knowledge of the random quantity r, a natural idea is to use imprecise expert knowledge of this quantity to generate the value of the appropriate threshold.

So what is an appropriate context here. Here, an appropriate context is applying control only when the deviation from the ideal state exceeds a certain threshold value.

VIII. How This Can Be Used in a Practical Context?

Since in general, most engineering problems are computationally intractable (NP-hard), it is important to find an appropriate context that will enable us to make the corresponding problem feasible.

Due to computational intractability of these problems, 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 – covering all three stages of engineering design – show that in many practical situations, expert knowledge indeed helps us find the appropriate context.

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.

The authors are greatly thankful to the anonymous referees for valuable suggestions.

References

[1] H. V. Gupta and G. S. Nearing, "Debates – the future of hydrological sciences: a (commn) path forward? Using models and data to learn: a systems theoretic perspective on the future of hydrological science", *Water Resources Research*, 2014, Vol. 51, pp. 5351–5359.

[2] G. Klir and B. Yuan, "Fuzzy Sets and Fuzzy Logic", Prentice Hall, Upper Saddle River, New Jersey, 1995.

[3] D. Morales and Tran Cao Son, "Interval methods in robot navigation", *Reliable Computing*, 1998, Vol. 4, No. 1, pp. 55–61.

[4] H. T. Nguyen and V. Kreinovich, "Interval-valued fuzzy control in space exploration" *BUlletin for Studies and Exchanges on Fuzziness and its AppLications (BUSEFAL)*, 1997, Vol. 71, pp. 55–64.

[5] H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2006.

[6] R. Ouncharoen, S. Intawichai, T. Dumrongpokaphan, and Y. Lenbury, "A mathematical model for HIV apheresis", *International Journal of Mathematical Models and Methods in Applied Sciences*, 2013, Vol. 7, No. 9, pp. 810–819.

[7] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.

[8] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman & Hall/CRC, Boca Raton, Florida, 2011.

[9] K. C. Wu, "A robot must be better than a human driver: an application of fuzzy intervals", In: L. Hall, H. Ying, R. Langari, and J. Yen (eds.), *NAFIPS/IFIS/NASA'94, Proceedings of the First International Joint Conference of The North American Fuzzy Information Processing Society Biannual Conference, The Industrial Fuzzy Control and Intelligent Systems Conference, and The NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic*, San Antonio, December 18–21, 1994 (IEEE Press, Piscataway, New Jersey), pp. 171–174.

[10] K. C. Wu, "Fuzzy interval control of mobile robots", *Computers and Electrical Engineering*, 1996, Vol. 22, No. 3, pp. 211–219.

[11] L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.