4-2012

Research-related Projects for Graduate Students as a Tool to Motivate Graduate Students in Classes Outside Their Direct Interest Areas

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

Comments:
Technical Report: UTEP-CS-12-08
Published in: Arunkumar Pennathur, Vivek Tandon, and Louis Everett (eds.), *Proceedings of the 2012 Annual Gulf Southwest Conference of the American Society of Engineering Education ASEE-GSW'2012 "Bridging Theory and Practice in Engineering and Technology Education"*, El Paso, Texas, April 4-6, 2012, pp. 1-9.

RESEARCH-RELATED PROJECTS FOR GRADUATE STUDENTS AS A TOOL TO MOTIVATE GRADUATE STUDENTS IN CLASSES OUTSIDE THEIR DIRECT INTEREST AREAS

Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@utep.edu

Abstract

In most graduate programs, students are required to take both "depth" classes – classes in the areas of the student's direct interest – and "breadth" classes, classes outside their direct interest areas. Naturally, the student's interest in "breadth" classes is often naturally lower than their interest in the "depth" classes. To enhance the students' interest in the "breadth" classes, a natural idea is to make research-related project an important part of the class, a project in which the student can apply the skills that he or she learns in the class to the research area of direct interest to this student. In this paper, we describe results of using this idea in Theory of Computation classes.

1. Formulation of the Problem

Research is an important part of graduate studies. For all PhD programs and for many Master's programs, one of the most important outcomes are students doing research, i.e., coming up with ideas and results which – for a selected problem – are better than everything that was known before.

Usually, students select a research topic about which they feel passionate. The students willingly (and usually successfully) study for the classes which are directly related to this topic.

Need for "breadth" classes. In most graduate programs, in addition to the "depth" classes directly related to the main student's research topic, the students are also required to take "breadth" classes, classes whose relation to the student's research topic is indirect – and may not be clear to the student.

Example. In our graduate Computer Science (CS) programs, all the students are required to take Theory of Computation. Students interested in other CS areas often do not understand the need for theory.

Problem. Since students often do not appreciate the need for the "breadth" classes, they often do not do their best in the "breadth" classes, and do not get as much knowledge as we faculty would like to have.

This deficiency often affects them later on, when it turns out that they do need the corresponding skills in their research.

2. Possible Solution: General Idea

To solve the above problem, we make a project an important part of the class (and of the class grade); a strongly encouraged project option is to perform class-relevant research related to the topic of their future thesis or dissertation. This win-win idea:

- helps students master the class,
- helps with their research – and
- sometimes even (eventually) leads to publications.

For students who have not yet selected their research topic, another option is help students who are doing research in the course's area; this has also led to eventual publications.

In the paper, we present examples of such projects and related publications. These examples come from the Theory of Computation classes taught in 2010 and 2011.

3. When Theory Is Useful: A General Description

Before we start listing the results of individual research-related projects in the Theory of Computation classes, let us provide a general explanation of why theoretical research is useful in computer science and related areas.

In many practical situations, we have empirically successful *heuristic* algorithms and methods. These method are heuristic in the sense that their success has no clear theoretical explanation. The problem with such methods is that since there is no theoretical explanation for this success, there is no guarantee that the corresponding method will work well in new situations – and no way to predict when this method will work well and when this method won't work.

Also, because of the lack of theoretical justification, it is not clear whether a modification or generalization of this method will work.

In such cases, a theoretical justification can help:

- it can lead to a better understanding of when this method works and when it does not; this understanding helps avoid wasting time on applying this method to situations where it does not work;
- it can also lead to a better understanding of when a proposed modification or a generalization of a method will work.

In the above description, we made a simplifying assumption that the question is whether to use the method or not. In many practical cases, the situation is more complex – namely, in order to apply a method, we first need to select the values of several parameters. Usually, the quality of the result depends on this parameter selection. However, because of the heuristic nature of the method, it is not clear how to predict the quality corresponding to given parameter values. Once we have a theoretical explanation for the method, we can not only use this theoretical description to predict the method's quality for given parameter values, we can also use the known optimization techniques to find the values of the parameters which are *optimal* for a given practical problem.

4. Towards Practical Applications of Computing

The ultimate objective of computing is to help in solving practical problems.

Computations help a lot – but they have an important limitation: computations are very precise, they process well-defined data according to well-defined algorithms. Thus, to be able to apply computing to a practical problem, we first need to formalize this problem, i.e., describe the problem in precise terms. This is an important *first stage* in solving the practical problem.

Once this problem is formalized, we need to come up with *an* algorithm for solving this problem. Designing such an algorithm is an important *second stage* of solving a practical problem.

The algorithm designed on the second stage is not always the most efficient one. Thus, once we have *an* algorithm, the next step is to come up with faster, more efficient algorithms for solving this problem. This optimization forms an important *third stage* of solving a practical problem.

On all three stages, we need to formalize heuristic methods, and to find optimal values of the parameters of these methods. In this paper, we give examples of research-related student projects from all three stages of applied computing.

5. First Stage: How to Formalize the Problem

Intelligent control. Several projects were related to *intelligent control.* The main objective of intelligent control is to transform the knowledge of an expert controller into an algorithm for an automatic controlling device. The problem is that the experience of a human controller is rarely formulated in precise terms; often, the expert controller uses imprecise words from natural language (such as "small") to describe his or her control strategy.

Many semi-heuristic approaches have been proposed for intelligent control. These techniques usually consist of several stages:

- first, they formalize the meaning of the words like "small";
- second, they combine these meanings into the meaning of the corresponding rules, like "if the car in front is close, it starts braking, and we are traveling at a high speed, it is necessary to break hard"; in describing this combination, it is important to translate the logical operations such as "and", "or", and "if . . . then" into appropriate combination rules;

- third, we need to combine these rules and transform these combined rules into an exact control strategy.

On the second stage, one of the natural criteria for selecting the corresponding "and"- and "or"-operations is related to the fact that we deal with the imprecise expert knowledge. Once we convert the natural language words – which are difficult for a computer to handle – into easier-to-handle numbers, it is clear that these numbers are imprecise, that a slightly different approach would lead to somewhat different numbers. It thus makes sense to require that the result of the corresponding "and"- and "or"-operation be not very sensitive to the small changes in the input numbers.

The less sensitive this result, the less dependent it is on the original knowledge elicitation method, the more adequate are the results. It is therefore reasonable to select the operations which are the least sensitive (= the most robust) with respect to such changes. In [6], for an important practical situation, the authors produced an explicit solution to the corresponding optimization problem and came up with a new expression for the "or"-operation describing "exclusive or".

On the third stage, several successful heuristics are known (Mamdani's approach, logical approach, etc.). In [1], the authors provided a theoretical explanation for the success of these heuristics. Interestingly, this theoretical explanation revealed that, in addition to the existing approaches, there is another possibly optimal approach – based on "exclusive or" operations, an approach that it worth analyzing and testing.

As an application of these techniques, we showed, in [4], that the formalized use of expert knowledge can be combined with the observed failure data to produce more realistic estimates for the failure rates of complex systems.

From formalizing the experience of expert controllers to formalizing the experience of expert researchers. Similar problems occur when we instead of formalizing the skills of expert controllers, we try to formalize the skills of expert researchers. The possibility to use this approach in formalizing physicists reasoning is justified in [5], where it is shown that this approach can indeed lead from the physicists' intuition to known equations of physics such as Newton's equations and equations describing electromagnetic interactions.

This intelligent technique, of course, goes beyond justifying well-known equations. For example, when analyzing physical processes, physicists use a lot of mathematical equations. However, in addition to these equations, they also use their physical intuition, an intuition that allows them to dismiss physically meaningless ("abnormal") mathematical solutions and only concentrate on physically meaningful ones. To avoid unnecessary mathematical computations, it is therefore desirable to formalize this notion of "abnormality" – so that computers be able to automatically detect possibly abnormal solutions. This is a very challenging task; an important step towards such formalization is presented in [7].

6. Second Stage: Designing Algorithms and Coming up with Theoretical Justification of Heuristic Algorithms

Geophysics: first example. One of the main objectives of *geophysics* is to find how density ρ and other physical characteristics depend on a 3-D location (x, y, z). In general, in

numerical methods, a way to find the dependence $\rho(x, y, z)$ is to discretize the space, and to consider, as unknown, e.g., values $\rho(x, y, z)$ on a 3-D rectangular grid. In this case, the desired density distribution is represented as a combination of point-wise density distributions. In geophysics, it turns out that a more efficient way to find the desired distribution is to represent it as a combination of thin vertical line elements that start at some depth and go indefinitely down. While this approach is empirically known to be successful, the theoretical explanation for this success was missing.

In [2], this empirical success was theoretically explained based on constraints theory: namely, it was shown that the empirical success of such vertical line element techniques can be explained if we recall that, in addition to the equations which relate the observations and the unknown density, we also take into account geophysics-motivated constraints, namely, the constraint that typically, the density increases with depth: if $z < z'$ then $\rho(x, y, z) < \rho(x, y, z')$. In the traditional representation, this constraint can be represented as relating the values of two variables $\rho(x, y, z)$ and $\rho(x, y, z')$. It turns out that in the vertical element representation, each of the corresponding constraints restricts the value of only one variable – and it is known that problems with one-variable constraints are much easier to solve than the problems with two-variable constraints.

Geophysics: second example. In many practical problems, especially in engineering problems, we have a well-defined objective function. For example, when we design a bridge, we have constraints on how much traffic it should safely carry, how much wind etc. it should withstand, how long it should serve without repairs, and within these constraints, we are looking for the cheapest design.

In science, the objective function is often not clear. For example, in geosciences, when we reconstruct the dependence of the density $\rho(x, y, z)$ on the spatial coordinates x, y, and z, we get different models depending on what is our objective. For example, if we use the seismic data, data based on the propagation of sound waves through Earth, we get a reasonably accurate description of the density at different depths and different locations – but only at those depth and locations which happen to be on the path from the seismic event to the recording station. On the other hand, if we use the gravity data, i.e., the result of measuring the gravity at different locations, we get a better understanding of densities at bigger depth – but the corresponding values represent averages over large areas and are, thus, not as accurate as describing individual values of density as the seismic measurements.

When we combine these two types of data, then, depending on how exactly we combine them, we can get either a model with more accurate individual values (as in the seismic model) or a model with more accurate averages (as in the gravity model).

This is similar to taking a photo: if we focus on faces, these faces come out nicely, but the background may be fuzzy. On the other hand, if we take a photo of a historic building and we focus on this building, this building comes up nice, but the faces on the pedestrians passing by are fuzzy.

In computer science terms, there is a *trade-off* between clarity of different parts of the image: if the clarity of the nearby part increases, the quality of a faraway part decreases. In scientific analysis, which of the possible images should we select? In general, instead of a single objective function, we have several objective functions. When we minimize the value

of one of them, the value of the other function increases, and vice-versa.

There are semi-heuristic methods for solving such *multi-optimization problems* (e.g., for selecting an image). Some of these methods consist of plotting a curve describing the dependence between the values of the two objective functions – and by selecting a point on this curve at which the curvature is the largest. The paper [12] provides a (partial) theoretical explanation for this heuristic method. This explanation not only explained the existing formulas, it enabled us to come up with a generalization of the known heuristic – a generalization which is worth trying.

7. Third Stage: Making Computations Faster

How to make computations faster: general analysis. We start with a general algorithm for solving a class of problems. Ideally, it is desirable to come with a faster algorithm for solving this class of problems.

Once we have reached the limit to this *general* improvement, once the general algorithm is already close to optimal, then the natural next step is to come up with a better algorithm for solving an *individual* problem, an algorithm that takes into account specific features of this problem.

Once this is achieved, once the individual algorithm is (almost) as fast as possible, a natural next step is to optimize the way this algorithm is implemented on a computer.

Finally, when this is optimized, the natural next step is to speed up the computers themselves.

The projects dealt with all the stages of this optimization process.

How to make algorithms faster: case of constraint satisfaction problems. In many practical problems, we need to find a solution that satisfies a given set of constraints (equations and inequalities). In most applications, the corresponding program is written by a team of human programmers. In many areas, however, we face new computational problems, and we often do not have enough people to come up with the corresponding algorithms and to program then.

It is therefore desirable to have an automatic system that, given the knowledge that we have about a situation, would automatically generate ("synthesize") the corresponding program. There exist methods and packages for such program syntheses. For examples, such methods have been successfully used by NASA in deep-space exploration, where unusual situations frequently occur that require new algorithms.

However, the programs generated by most existing programming synthesis packages are often not as efficient as program generated by skilled human algorithm developers and programmers. It is therefore desirable to go from program synthesis to *optimal* program synthesis – generating program that not only solve the original problems, they solve these problems in a most efficient way. An important step towards such optimal program synthesis packages was done in [11].

How to make algorithms faster: case of optimization problems. In many practical problems, we would like to find the *best* alternative. In other words, many practical problems

are optimization problems.

In solving optimization problems, we usually deal with sets of a fixed simple (easy-to-handle) type. For example, we start with a rectangular box containing the location of the desired optimum, and we try to decrease the size of this box until we can accurately locate the optimum. Alternatively, instead of boxes, it is sometimes beneficial to consider spheres or, more generally, ellipsoids.

One of the main computational advantages of an ellipsoid is related to the fact that, in contrast to a box which has non-smooth vertices, an ellipsoid is it smooth. As a result, an optimization of a smooth objective function over an ellipsoid can be handled by the usual calculus criteria – and is, therefore, computationally simpler.

On the other hand, ellipsoids have a disadvantage: if we divide a box into two equal subboxes, the volume of each subbox is exactly one half of the original volume. On the other hand, if we try to come up with two smaller spheres that cover the original sphere, these smaller spheres have to intersect and therefore, the volume of each of them is larger than the half of the original volume. As a result, while computations on each step are faster for ellipsoids, the region decreases slower when we use ellipsoids and thus, we need more iterations to locate the optimum.

It is therefore desirable to combine the advantages of these methods. One possibility is to use half-ellipsoids: they are much smoother than boxes and at the same time an ellipsoid can be divided into two half-ellipsoids of exactly half-volume. To be able to use half-ellipsoids, we need efficient algorithms for using their "almost-smoothness" in optimization. Such algorithms are described in [10].

How to optimize the way algorithms are implemented on the current computers. Computations are often slow because there is a delay in access to computer resources. Everyone is familiar with such delays. The good news is that these delays are rarely universal: usually, when there is a delay in access to one of the computers, other computer around the world are under-utilized.

It is therefore natural to combine many of the world's computers into a single "super-computer" so that for each user's task, the system will decide where to process it, where to store the data, etc. This idea is known as *cloud computing*.

One of the main challenges of cloud computing is that instead of a simpler (but still challenging) problem of optimally regulating requests to a single computer, we now face as much more difficult problem of optimally regulating billions of requests from all over the world. Some optimization ideas for the corresponding problems are described in [8].

How to make computers themselves faster. To make computations faster, we must use faster and faster processes. It is known that the speed of all the processes are limited by the speed of light. As a result, simply to send a signal from one side of a 30 cm wide computer to another side takes one nanosecond – the time of one operation on a current Gigahertz processor.

To make computations much faster, we therefore need to drastically reduce the size of the processor and the memory. Since we still want the computers to store at least the same amount of information – and even more – we therefore need to drastically decrease the size

of each individual element. Already now, these elements are so small that some of them consists of several hundred atoms. To decrease the size further, we need to reach the size of individual atoms – and since atoms are described by quantum physics, quantum effects needs to be taken into account when designing new computers. Quantum computing is therefore a promising area of research.

At present, many ideas in quantum computing are semi-heuristic. In line with our general explanation of why theoretical analysis can be beneficial, it is desirable to come up with a theoretical justification for these heuristic ideas and methods. In [3], such an explanation is provided for a specific mathematical idea – to the selection of topology (crudely speaking, the notion of closeness) in the usual description of quantum systems. In [9], such an explanation is provided for Feynman integration – a useful equivalent way to describe the dynamics of quantum systems.

References

[1] Bravo, Samuel, and Nava, Jaime, "Mamdani Approach to fuzzy control, logical approach, what else?", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011*, El Paso, Texas, March 18–20, 2011 (2011)

[2] Cardenas, Ronald, and Ceberio, Martine, "Efficient Geophysical Technique of Vertical Line Elements as a Natural Consequence of General Constraints Techniques", *Journal of Uncertain Systems*, **6**(2), to appear (2012)

[3] Culellar, Chris, Longpré, Evan, and Kreinovich, Vladik, "Why L^2 Topology in Quantum Physics", *Journal of Uncertain Systems*, **6**(2), to appear (2012)

[4] Ferregut, Carlos, Campos, F. Joshua, and Kreinovich, Vladik, "Reducing overconservative expert failure rate estimates in the presence of limited data: a new probabilistic/fuzzy approach", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011*, El Paso, Texas, March 18–20, 2011 (2011)

[5] Gutierrez, Eric, and Kreinovich, Vladik, "Fundamental physical equations can be derived by applying fuzzy methodology to informal physical ideas", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011*, El Paso, Texas, March 18–20, 2011 (2011)

[6] Hernandez, Jesus E., and Nava, "Least sensitive (most robust) fuzzy 'exclusive or' operations", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011*, El Paso, Texas, March 18–20, 2011 (2011)

[7] Jalal-Kamali, Ali, Nebesky, Ondrej, Durcholz, Michael H., Kreinovich, Vladik, and Longpr'e, Luc, "Towards Towards a 'Generic' Notion of Genericity: From 'Typical' and 'Random' to Meager, Shy, etc.", *Journal of Uncertain Systems*, **6**(2), to appear (2012)

[8] Lerma, Octavio, Gutierrez, Eric, Kiekintveld, Chris, and Kreinovich, Vladik, "Towards Optimal Knowledge Processing: From Centralization Through Cyberinsfrastructure to Cloud Computing", *International Journal of Innovative Management, Information & Production (IJIMIP)*, **2**(2), 67–72 (2011)

[9] Nava, Jaime, Ferret, Juan, Kreinovich, Vladik, Berumen, Gloria, Griffin, Sandra, and Padilla, Edgar, Why Feynman Path Integration?", *Journal of Uncertain Systems*, **5**(2), 102–110 (2011)

[10] Portillo, Paden, Ceberio, Martine, and Kreinovich, Vladik, "Towards an Efficient Bisection of Ellipsoids", *Proceedings of the ITEA Live-Virtual-Constructive Conference "Test and Evaluation"*, El Paso, Texas, January 24–27, 2011 (2011)

[11] Reyna, Joaquin, "From program synthesis to optimal program synthesis", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011*, El Paso, Texas, March 18–20, 2011 (2011)

[12] Sosa Aguirre, Uram Anibal, Ceberio, Martine, and Kreinovich, Vladik, "Why Curvature in L-Curve: Combining Soft Constraints", *Proceedings of the Fourth International Workshop on Constraint Programming and Decision Making CoProD'11*, El Paso, Texas, March 17, 2011 (2011)