Interval and Complexity Workshops Back-to-Back with 1997 ACM Symposium on Theory of Computing (STOC'97)

Workshops. On May 4–6, 1997, El Paso hosted the ACM STOC (Symposium on Theory of Computing). This conference is one of the most important theoretical computer science conference, and attracts researchers from all over the world every year. In conjunction with this conference, two workshops were organized for May 6–8:

- Workshop on computational complexity.
- Workshop on interval computations.

Attendees of the workshop were recruited through messages to the complexity and interval computations mailing lists. Due to mutual interest and a desire to encourage cross-fertilization of the two areas, participants of the two workshops decided to combine them into one workshop. In this report, we want to succinctly describe, from the interval computations viewpoint, the talks presented at this combined workshop.

Brief introduction: computational complexity problems of interest to interval computations. The ultimate *goal of computations*, be they interval or more general, is to *solve problems*. It is well known that some natural classes of problems are so computationally hard that (either provably or according to the belief of most computer scientists) no algorithm can solve all instances of problems from these classes in reasonable time. These classes of problems are called *computationally intractable*, and an important and well studied such class is the class of *NP-hard problems*. In interval computations, even the simplest problems are known to be NP-hard: e.g., the problem of computing the exact bounds for the values $y = f(x_1, \ldots, x_n)$ for a polynomial f when we know the exact interval bounds $\mathbf{x}_i$ for each of the variables x_i (i.e., if we know that $x_i \in \mathbf{x}_i$).

When a class of problems is computationally intractable, we still face the *practical* necessity to solve the problems from this class. Various approaches are taken:

- One possibility if to define a *sub-class* of the original (computationally hard) general class of problems. For example, while computing the exact bounds for the function f above is NP-hard, the special case of polynomials of one variable is tractable. Care must be taken to select an interesting sub-class: one that is large enough to include a significant portion of practical problems, yet one that offers a feasible algorithm.

- Another possibility is to look for algorithms that may not work well in the *worst case*, but at least is feasible *on average*. This is possible if the instances giving rise to bad running time have a low probability of occurrence. Formal definitions of problems that can be solved, on average, in reasonable time, were proposed some time ago, by L. Levin and Yu. Gurevich. These definitions, however, require that we know the frequencies (= probabilities) with which different cases of a general problem occur. The readers of this journal will not be surprised to know that in many practical cases, we only have *partial* information about these probabilities. It is, therefore, desirable to make predictions about the "average" computational complexity of different problems based on this partial information.

- Another approach is to consider *parallel computations*. Several areas of research in computational complexity are related to parallelization. To be able to use parallelism efficiently, one must break down the original sequential computation into computing steps that can be performed in parallel. Depending on the problem, this may or may not be possible. One area of complexity is to analyze which problems can be parallelized and which cannot. Another area is to compare different models of parallel machines. For example, one model of parallel computation considers parallel computers consisting of the simplest possible computational units working in parallel: *Boolean* operations. This model is the *Boolean circuit* model of computations. In this model, the elements of the computation are *Boolean gates*, like OR, AND and NOT gates. Inputs are fed to the circuit, the circuit computes the output. Typical complexity parameters considered are the *depth* of the circuit (which describes how much time the corresponding computations will take) and the number of gates needed, with various restrictions on which gates can be used.

- One limit to parallel computations is that the speed up is limited by the number of processors available. To have a very large number of processors, we must *decrease* their size. When the size drops to the size of a molecule, we reach the level of *molecular* (*chemical*) computations. Analysis of the corresponding *molecular (chemical)* computations model attracted many after Adleman successfully used DNA fragments to solve a particular case of an NP-hard problem.

- By definition, all NP-hard problems are difficult to solve, but some are more difficult than the others. It is important to know how difficult a given problem is, and how difficult problems can be. It is, therefore, desirable classify hard problems into subclasses of problems of different complexity.

All these areas of research were presented at the workshops. We will briefly describe these talks and comment on how some of them can be specifically related to interval computations.

Computational complexity talks and their relation to interval computations. The workshop started with a talk by MARTIN FÜRER on the *graph isomorphism* problem: given two graphs, is one isomorphic to the other one? Or in other words, is there a way to re-label the nodes in such a way that the two graphs become identical? Until now, no feasible algorithm is known to solve this problem in its generality. When this happens, the usual approach is to either develop heuristics that are *not guaranteed* to always give the correct solution, or to develop algorithms to solve some *particular cases* of the problem. For example, the graph isomorphism problem is feasibly solvable if the graphs have a bounded *degree* (number of neighbors to a node). Fürer has shown that there is a feasible algorithm if we bound the eigenvalue *multiplicity* of the graph's adjacency matrix ($a_{ij} = 1$ if the i-th and the j-th nodes are connected, otherwise it is 0).

ALAN SELMAN gave a presentation on *average case complexity.* Analysis of the average case complexity is, as we have mentioned, an important subfield of computational complexity. Most problems are classified by their *worst case* complexity: the complexity of an algorithm solving a problem of size n is that of the worst complexity among all instances of that size. *Average case* complexity considers the expected complexity, based on a probability distribution of all possible instances. An interesting overview of results in this field, especially a critical look at various definitions of *what is feasible on average*, was presented in this talk.

SAMIR DATTA in his talk showed that many known results about Boolean circuits can be naturally extended to a more realistic case in which we allow arithmetic elementary computer operations instead of Boolean operations.

> *Comment.* Datta only considered operations with *integers*. It would be interesting to generalize his results to arithmetic operations on *real numbers* and *intervals* that are of interest to interval computations community.

DESH RANJAN in his talk described an efficient dynamic data structure. The problem arose in the context of implementing a parallel version of Prolog, a programming language especially designed to solve problems formulated in terms of logic. In solving those problems, a special consideration must be given to the scope of each variable, and how changing the value of a variable in an equation affects computations which is occurring in parallel. Ranjan described a data structure to keep track of whether binding of variables have been changed, where most operations can be done in constant time.

> *Comment.* Ranjan's results can be applied, in particular, to systems (known to interval community) that combine intervals with logic programming.

STEVE MAHANEY first overviewed the *molecular (chemical)* computations model. As chemical computations become more and more technically possible, it is becoming more important to estimate the actual time that these computations take. For this, we must solve a system of differential equations (called equations of chemical kinetics) that describes how molecules interact. Mahaney proved very interesting results by analyzing the cases when the corresponding system of differential equations has an explicit analytical solution.

> *Comment.* As we turn to more complicated reactions, we will inevitably run into equations for which *no analytical solution* is possible. To get *guaranteed* time estimates for such equations, we will need *guaranteed* methods for solving systems of differential equations (and this is exactly what interval computations are for!).

STEVE HOMER talked about the hyperpolynomial hierarchies. This type of research is trying to classify NP-hard problems into subclasses of problems which are harder and harder. The polynomial hierarchy is well known to complexity theorist as an infinite (actually, just believed to be infinite) hierarchy of classes of problems, starting at NP, and consisting of harder and harder problems. Homer constructed a complexity hierarchy similar to the hierarchy of ordinal numbers $(1, 2, 3, \ldots, \omega, \omega+1, \ldots)$, and showed that, under some reasonable assumptions, there exist problems on each level of this hierarchy.

JUDY GOLDSMITH considered the problem of a robot in a statistically known environment. It is an empirical fact that even if we know the exact probabilities, the statistical robot planning problem is computationally hard. Goldsmith has shown that this planning problem is indeed hard: it is $\mathrm{NP}^{\mathrm{PP}}$-complete, which means, in particular, that it is more complicated than the problems in Homer's hierarchy.

> *Comment.* It is worth mentioning that the empirical computational complexity of statistical robot planning was one of the reasons why the award-winning 1996 University of Texas at El Paso robot team decided to use *interval* and not probabilities for planning.

Interval computation talks. The workshop continued with talks on interval computations. VLADIK KREINOVICH started this section with a general overview of which interval computation problems are NP-hard, which are solvable in feasible (polynomial) time, and which are hard in the worst case, but solvable in polynomial time on average. He mainly mentioned "static" problems such as computing the values of a given function, or solving a system of equations.

MARTIN BERZ talked about the complexity of interval approaches to *dynamical* systems. He was motivated by the problem of tracking particles traveling at almost the speed of light in a particle accelerator. We must guarantee that the particles will not get out of the camera after billions of cycles. To do this, we must solve systems of differential equations. Interval methods have been designed to provide guaranteed estimates for the solutions of such equations, but for this particular problem, traditional interval algorithms will really take too much time. If we want the result of integrating a system of differential differential equations over a time interval $[0, T]$ with the accuracy of k decimal digits (i.e., with an accuracy of $\varepsilon = 10^{-k}$), we must, in traditional methods, take an integration step to be of size ε and thus, we need at least $T/\varepsilon \sim 10^k$ computational steps. Berz described an approach based on the *Taylor series*. Indeed, if we restrict ourselves to n-th order terms in Taylor series, then the remainder is proportional to T^{n+1}. Therefore, to achieve an accuracy $\varepsilon = 10^{-k}$, we must choose n for which $T^{n+1} \approx 10^{-k}$, i.e., for which $n \approx \text{const} \cdot k$. Thus reducing the *exponential* time to *linear* time.

Continuing on the same vein, KYOKO MAKINO discussed *efficient verified integration schemes*. Instead of simulating the system of differential equations, starting at time t_0, and computing the desired parameters at each subsequent time step, she reformulated the original system as a *system of integral equations*, in which the desired functions form a *fixed point* of an appropriate integral operator, and then use iterative fixed point techniques to find the desired solution. In this case, at each computation step we have the approximate values $\tilde{x}_i(t)$ for *all moments of time* $t \in [0, T]$; but with the *accuracy* that is *not yet* as *good* as desired; the accuracy improves from iteration to iteration until it reaches the desired level ε. This allowed a further decrease in the computation time.

Interval papers presented at the workshop will be submitted to *Reliable Computing*.

Social events. The workshop had an informal atmosphere, conducive to discussions. Various social events were planned: Chinese food reception (imported from Mexico), and all-American cookout, attended by University of Texas at El Paso Provost and faculty. Thanks to Shinping Chyi for her tremendous job at organizing these events.

STOC'97. As an aside, STOC'97 was a very successful conference. About 200 participants from around the world attended the conference. As usual, technical contents of the conference, addressing the latest research in theoretical computation, was of very high quality, and many submissions that were not accepted for presentation at the conference were recommended for publication in major theoretical computer science journals.

Included in the registration was a banquet at *Cattleman's Steak House*, located on a ranch in the middle of the desert. This banquet included a hay ride around the desert (with a visit to the places where Hollywood companies recently made movies), a delicious brisket dinner on an authentic Southwestern farm and a performance by the “Viva El Paso!” dancing troupe whose dances illustrated the four centuries of El Paso history.

Report prepared by Luc Longpré, University of Texas at El Paso, and Martin Berz, Michigan State University.