To be published by the International Journal on Interval Computations in the Proceedings of the International Workshop on Applications of Interval Arithmetic, El Paso, Texas, 1995.

Interval Constraint Spreadsheets: An Implementation for Microsoft® Excel

Eero Hyvönen
(eero.hyvonen@vtt.fi)

Stefano De Pascale
(stefano.depascale@vtt.fi)

Technical Research Centre of Finland (VTT)
Information Technology, Information Systems
P.O. Box 1201, 02044 VTT
(Tekniikantie 4B, Espoo)
FINLAND

Abstract

This paper concerns application of interval constraint satisfaction techniques in spreadsheet computing. The idea is to generalize the underlying computational paradigm of spreadsheets by interval propagation techniques. As a demonstration, an interval constraint extension to commercial spreadsheet program Excel by Microsoft has been implemented. Our implementation is based on three general purpose C++ libraries for interval reasoning tasks. Their technical novelty lies in a combined use of branch-and-bound interval algorithms, computer algebra, and consistency techniques for obtaining better than locally consistent solutions in an efficient way.

Keywords: Constraint satisfaction, spreadsheets, interval arithmetic, computer algebra.

1 Why interval spreadsheets?

Interval constraint satisfaction techniques (Cleary, 1987; Davis, 1987; Hyvönen, 1989, 1992) were proposed as a generalized computational paradigm for spreadsheet computing in (Hyvönen, 1991, 1994). Interval constraint spreadsheets would have several attractions when compared with current products such as MS Excel, Quattro Pro, or Lotus 1-2-3:

- Interval arithmetic (Moore, 1966, 1979) makes it possible to deal with inexact data and rounding errors.
- Consistency techniques make it possible to compute (or at least bound) values for any variable in a set of related functions or more generally constraint equations. No distinction between input variables (cells) and output variables is needed.
- Top-down refinement can be used as a new problem solving paradigm. In this scheme, a solution is found by constraining cell intervals stepwise until exact solutions emerge. This is often more useful than the trial-and-error scheme used in ordinary spreadsheets.

- Interval computations are safe and sound — possible solutions are never lost.

- Ordinary numerical computation in spreadsheets is a special case of interval constraint satisfaction. Interval techniques offer new possibilities without giving up the possibility of using traditional exact arithmetic.

In the following, the notion of interval constraint satisfaction problem (ICSP) is first presented and three C++ libraries for solving it are introduced. After this, problems of incorporating interval computations with a spreadsheet program are discussed. As a test and demonstration, the libraries have been applied to extending MS Excel for constraint satisfaction.

2 Interval CSPs

The general interval constraint satisfaction problem (ICSP) (Hyvönen, 1992) can be formulated as:

$$E_1,...,E_n \qquad (2.1)$$
$$P_1 \in X_1,...,P_m \in X_m$$

Here E_i are functions, equations (or inequalities, that can be transformed into interval equations), P_j are variables used in them, and X_j are real intervals (or discontinuous real intervals). The equations constitute a *constraint net* that consists of a set of variables and equational constraints connecting them. For example:

$$\sin(Z)=X+Y+0.4,\ e^X+Z=-0.2, \qquad (2.2)$$
$$X*Y=Z,\ X\leq 0$$
$$Y\in[0,1000],\ Z\in[-100,-0.1]$$

The *interval (tolerance) situation* $\{P_1 \in X_1,...,P_m \in X_m\}$ in an ICSP (2.1) refers to a set of exact value situations $\{P_1=x_1,...,P_m=x_m \mid x_i \in X_i,\ i=1...m\}$. An ICSP is *feasible* iff its interval situation has an exact subsituation satisfying all constraints, i.e., the equations have a solution within the intervals:

$$\exists\{P_1=x_1\in X_1,...,P_m=x_m\in X_m\}:$$
$$E_1,...,E_n \text{ satisfied}$$

A variable $P_i \in X_i$ is *consistent* iff each interpretation $P_i=x$, $x\in X_i$, can be satisfied with respect to all constraints by some exact subsituation:

$$\forall\ x\in X_i$$
$$\exists\ \{P_1=x_1\in X_1,...,P_i=x,...,P_m=x_m\in X_m\}:$$
$$E_1,...,E_n \text{ satisfied}$$

We say that an interval situation is *(globally) consistent*, i.e., it is a *(global) solution*, iff its every variable is consistent. Intuitively, an interval in a solution should not contain "extra" values that cannot be satisfied within the given intervals. For example, it turns out that equations (2.2) have a single (global) solution

$$X\approx -1.25,\ Y\approx 0.387,\ Z\approx -0.485 \qquad (2.3)$$

within in the given range. Hence (2.2) is feasible.

A weaker but computationally more easily obtainable and hence useful notion of consistency is *local consistency*. A variable is *locally* consistent iff it is consistent with respect to all directly connected constraints in the constraint net. Local consistency of an interval situation (solution) means that all variables are locally consistent. For example, the local solution of

$$X+T=Y,\ Y+T=Z \qquad (2.4)$$
$$X=1,\ T\in[0,inf),\ Y\in[0,inf),\ Z=11$$

is $T\in[0,10]$, $Y\in[1,11]$. Only at the global level one can determine that actually Y must be the mean value of X and Z, i.e., that the global solution is Y=6, T=5. Intervals in the local solution may have extra width but always correctly bound the global solutions.

In the general case, the solution of an ICSP may not be represented by a single interval solution, but as a set S of them. For reasons of convenience, these solutions can be generalized by the situation. $G=\{P_1 \in X_1,...,P_n \in X_n\}$ such that for i=1...n, $\min(X_i)$ is the minimum and $\max(X_i)$ is the maximum of all values X_i of P_i in the solutions S. Hence, the solution G is the most constrained situation that can be obtained by refining the tolerances from both ends without losing exact value solutions. If discontinuous intervals are employed, the generalized solution can be represented more accurately by a single discontinuous interval situation.

3 Evaluating constraints globally

In our implementation, an extension to local tolerance propagation (Hyvönen, 1992) is used for solving ICSPs. The main new idea is to treat constraint equations $E(X_1,...,X_n)$ as primitives and not to decompose them into primitive constraints (addition, multiplication etc.). The benefit of this approach is that individual constraint equations can be solved globally, i.e., the strictest possible interval bounds can be found at the constraint level.

In this technique, each variable X_i is solved algebraically from E and represented by a *solution function* $Y_i=F(X_1,...,X_n)$. Although such functions cannot always be solved in closed form, there are ways for determining their actual (global) value range:

1. Recursive solution functions Firstly, it is possible to derive at least a recursive solution function for each variable. For example, consider equation:

$$Y=X^5-7*X^4+3*X^2 \qquad (3.1)$$

It is not possible to solve X algebraically from the equation. However, it is easy to obtain a recursive solution such as:

$$X=\pm\mathrm{sqrt}(Y-X^5+7*X^4)/3 \qquad (3.2)$$

Here X is unfolded from term 3*X^2.

2. Interval Newton method Another even more effective technique is to transform the constraint equation L=R into function form Z=F(...)=L-R=0 and use the multidimensional interval Newton method (Hansen, 1992) for bounding the zeros of F w.r.t. each variable involved. It is easy to see that these ranges are the actual (global) values of the corresponding solution functions.

After deriving the solutions functions $Y_i=F(X_1,...,X_n)$ of a constraint E we know that E is consistent iff (Hyvönen, 1992):

$$Y_1 \subseteq F_1(X_1,...,X_n),...,Y_n \subseteq F_n(X_1,...,X_n) \qquad (3.3)$$

In the tolerance propagation algorithm, interval solutions are found by evaluating the solution functions of all constraint equations until a fixed point is rearched. The problem of using complicated interval functions such as (3.1-2) is how to evaluate them. Fortunately, there are numeric interval arithmetic algorithms for the task (Skelboe, 1972; Asaithambi et al., 1982; Ratscheck, Rokne, 1988; Moore et al., 1992). These techniques are based on branch-and-bound searching. The general idea is to search for the minimum/maximum of a function by inspecting the interval values in ever smaller parts. Various mathematical and other heuristics (value limits, monotonicity, convexity, search order, precision,...) have been developed for pruning/bounding the search space.

By these techniques solutions up to given precision can be determined in theory. In practice, problems arise. Computational time and space needed in global interval function evaluation is largely unpredictable. It depends both on the algebraic form and on the argument values of the function. In the general case, it is not possible to see whether the function at hand can be evaluated easily.

We have implemented C++ libraries LIA InC++ and GIA InC++ (Hyvönen, De Pascale, 1994a-b) for global interval function evaluation based on the branch-and-bound approach. From the mathematical view point, main novelties of these libraries are (1) use of an algebraic preprocessor for speeding up interval computations, (2) capability of evaluating recursive interval functions and (3) possibility of using discontinuous intervals as input. These extensions are quite essential when applying the libraries to constraint satisfaction:

1. In our experiments, algebraic speeding techniques have turned out to be quite necessary from the computational complexity view point.

2. Recursive functions such as (3.2) and recursive interval Newton formulas are frequently encountered in global interval constraint satisfaction

3. Generalization into discontinuous interval arithmetic is important because algebraic solution functions may have undefined points not present in the original equation. For example, constraint X*Y=1 is defined everywhere but solution function X=1/Y is not in points with Y=0. Our solution to this problem is to use enhanced definitions of inverted functions obtained when solving the constraint equation. For example, the inverse operator of multiplication above would be one that accepts $0 \in Y$ and evaluates then a discontinuous interval, not the traditional interval division.

4 Tolerance propagation engine ICE InC++

In the general case, an ICSP consists of a set of constraints. Locally consistent solutions for the general ICSP can often be determined within reasonable precision fairly easily, although even this task can be computationally quite expensive. For example, the local tolerance propagation algorithm converges reasonably fast to the solution (2.3) for (2.2).

The hard problem is how to find global solutions. For example, in

$$A=X*Y,\ B=Y/X \qquad (4.1)$$
$$A=1,\ B=1,\ X \in [1/10,10],\ Y \in [1/10,10]$$

both constraints are globally consistent in separation. The net is locally consistent but not globally; the global solution would be X=Y=1. An avenue towards finding global solutions is global tolerance propagation (Hyvönen, 1992). The key idea here is that global problems are due to the cycles of the underlying constraint net. If the solution functions for each variable are defined not only w.r.t. neighbouring constraints (local solution functions) but also w.r.t. cycles going through the variable (global solution functions), then global solutions can be obtained. For example, in (4.1) globalization actually means that variables X and Y are solved from the equations. A problem of the approach is that a large number of complicated global functions may be needed in networks with many cycles. Also their algebraic derivation can be difficult.

In our work, local and global tolerance propagation algorithms have been implemented as a C++ library ICE InC++ (Hyvönen, De Pascale, 1994c). By default, this system uses local propagation but individual constraints are solved globally by the techniques of the previous chapter. If desired, additional global solution functions can be generated for the engine. Globalization process can be controlled by the programmer by setting an upper limit for the length and number of the cycles to be globalized. Length limit is often useful because long cycles are computationally expensive to process and global interactions in them are likely to be not so important.

ICE InC++ is designed to be used as a general purpose ICSP library in C++ programming (Hyvönen et al., 1994). From the programmer's view point, it is implemented as a C++ class Ice. Objects of this class correspond to ICSPs. Communication with Ice-objects is based on simple member functions that define ICSPs, set computational parameters, propagate tolerances, and read changed variable values.

5 Interfacing MS Excel

As a demonstration, a version of ICE InC++ has been integrated with MS Excel by a DLL library. This chapter discusses modifications that need to be made in a spreadsheet program such as MS Excel when extending it with interval constraints. Important issues are reviewed mainly from the user's point of view and solution approaches taken in our demo implementation ICExcel (Interval Constraint Excel) are described.

Inserting equations. Each spreadsheet is associated with an Ice-object (an instance of class Ice) that maintains internally the ICSP corresponding to the spreadsheet definitions. Interval values and constraints are defined by an Excel-function I("xxx") that sends string "xxx" to the underlying object updating its internal state accordingly. For example, when the user types I("[2,3]") into cell A13, the value of variable A13 is updated in the Ice-object and the statuses of related constraints and solution functions are set undetermined. These constraints will later be processed during tolerance propagation. In the same way, I("A1*SIN(B2)=A1+C2") would insert the equation as a constraint, update the internal constraint net, solve local and global solution functions, and redefine other affected global solution functions. After inserting/modifying constraints and values and setting computational parameters (like desired precision), tolerance propagation can be executed, and changed variable values are transmitted to corresponding cells on the sheet.

In ordinary MS Excel, output cells have function definitions such as =A2+B3+7 and the function value is used and displayed as the value of the cell. A problem here is that when an interval constraint equation is inserted, such as A1+B3=C3*B2, there is no unique cell where to display the value, but all related variable cells must be considered. In contrast to the function case, the position of an equation in the sheet is immaterial but should be respected as a decision of the user.

ICExcel works exactly like MS Excel when interval functions are inserted. When a constraint equation is inserted, ICExcel shows the equation itself in the cell, unless the cell has a value, in which case the value is shown.

Setting output cell value. In spreadsheet programs, it is not possible for the user to set the value of an output cell, because output cell values are defined by function expressions. In interval constraint satisfaction, it must be possible to set the value of any cell. ICExcel extends MS Excel in the following way: when double clicking a cell, a special DLL function gets activated and shows the value of the cell in a dialogue box for editing. The possible formula attached to the cell remains as it was.

Control of computation. Control of tolerance propagation is quite different from the evaluation mechanism used in MS Excel. As a result, all interval computations have to be done by shifting control to the underlying Ice-object. In ICExcel, the special I-function described above makes this shift possible. I-function does not activate MS Excel's own value propagation algorithm, but only communicates with the underlying Ice-object.

Dynamic interrupts. Since interval computations can take an unpredictable time to be completed, the user must be supported with a dynamic interrupt facility. In GIA InC++ library, computation can be terminated interactively and the resulting

semi-global result used as the solution. ICExcel makes use of the facility: a lengthy computation can terminated by hitting a control key.

Relaxation. Since all cell values are user settable, the user can easily end up in an infeasible situation. In such situations it is not always easy to see what cells should be modified in order to make the situation feasible again. In ICExcel, a local relaxation facility is available. In infeasible situations, computation is carried on until all constraints have been evaluated. After this, cell values can be enlarged locally in order to satisfy all related constraints, if demanded by the user. Often, but not always, this suffices to make an infeasible situation feasible.

Special computational parameters. Interval computations can be controlled by a set of parameters. ICExcel makes use of those available in ICE InC++, especially desired absolute/relative precision and response time limit.

Compatibility. An important motivation behind the idea of interval constraint spreadsheets is full compatibility with traditional spreadsheet computing. In ICExcel, the same spreadsheet can be used simultaneously for both ordinary computations and interval constraint satisfaction. When intervals change into numbers, traditional MS Excel features like advanced graphics become automatically available.

6 ICExcel: an example

Consider the resource allocation problem where the task is to allocate the working time of *n* researchers in *m* projects during some time period. Let us assume that three researchers X, Y, and Z are allocated to three projects, as shown in Figure 1. By applying (local) constraint propagation the system gives the response shown in Figure 2. The user has now the possibility to constrain tolerances. Assume that he/she demands that researcher X should not work on project 1 less than 120 weeks, i.e., C3=[120,160], and that the total resources of project 2 are exactly 150 weeks, i.e., D6=150 (Figure 3). Notice that the formula D6=I("D3+D4+D5") remains in force. The system applies again (local) constraint propagation to refine values. The response is shown in Figure 4, where the modified values are in bold font. Based on the new situation, the user can then further refine the tolerances, the system will infer necessary consequences, and so on until some exact solution satisfying the user's demands for the problem is found.

In Figure 4 the user simultaneously changed one input and one output variable (italic font). Although only two cell values were altered in this arithmetically quite simple system, it was possible to infer eight value modifications. For the user it is difficult to see every consequence of his/her changes immediately. In the general case, it is possible to change any combination of cell values simultaneously. Furthermore, more complicated nonlinear algebraic constraints are often present in real-world problems of this type.

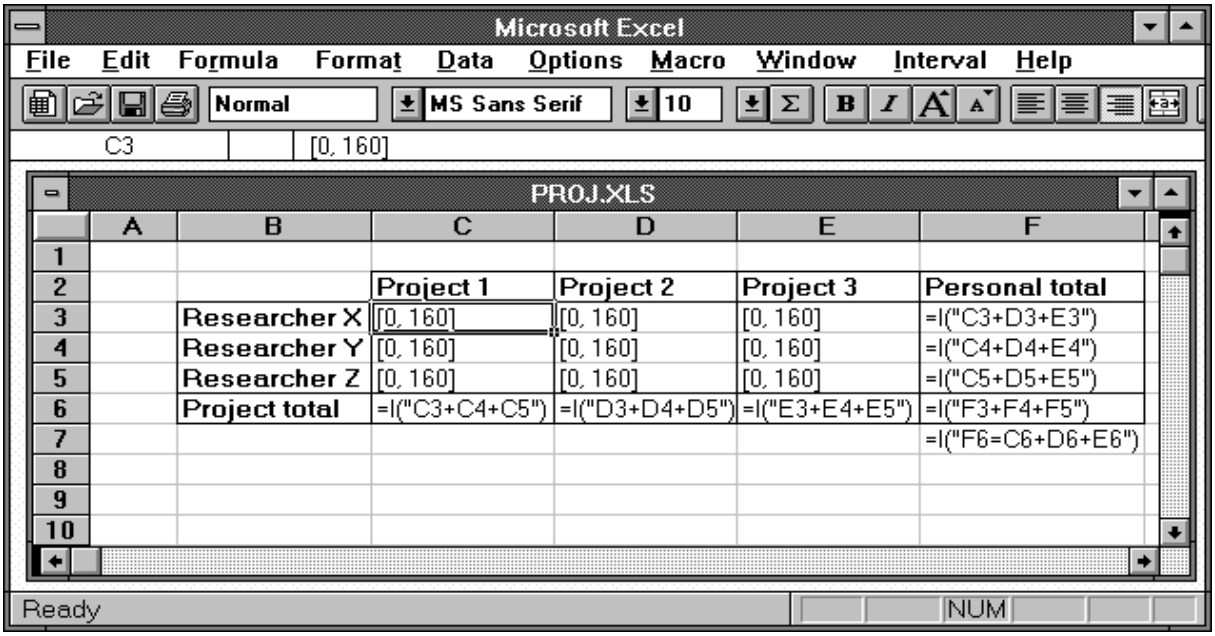

Figure 1. The resource allocation problem. Initial values and formulas.

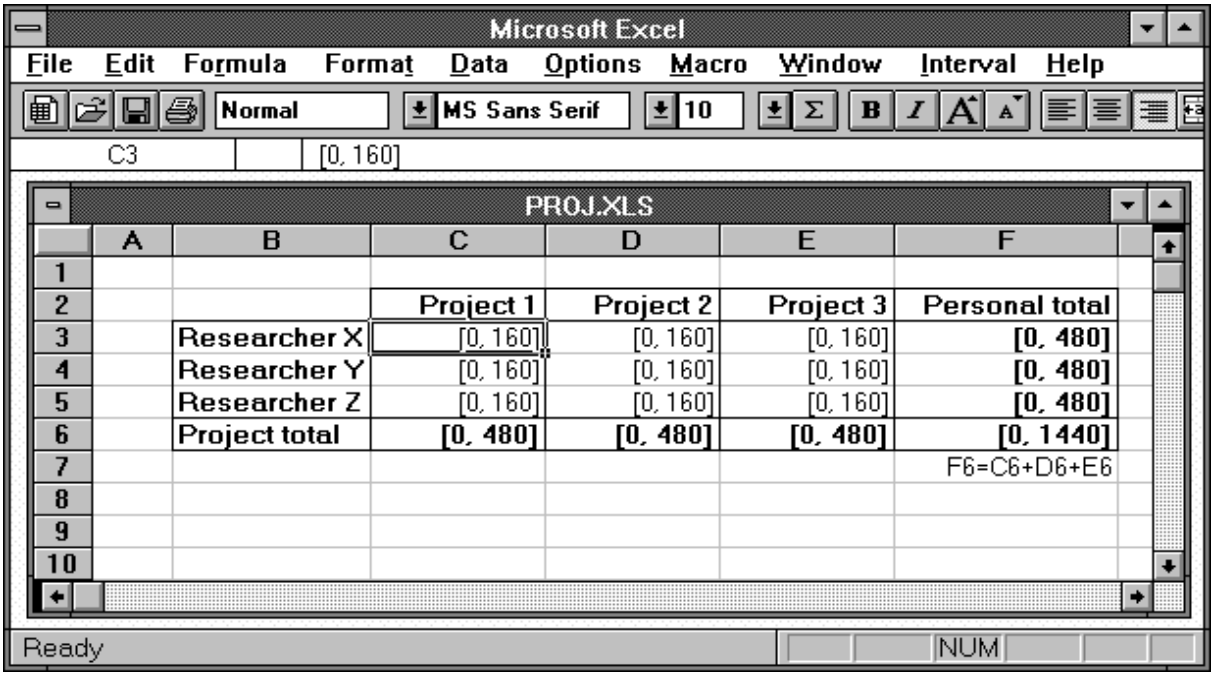

Figure 2. The system's response to the initial state of Figure 1. Modified cells are in bold font.

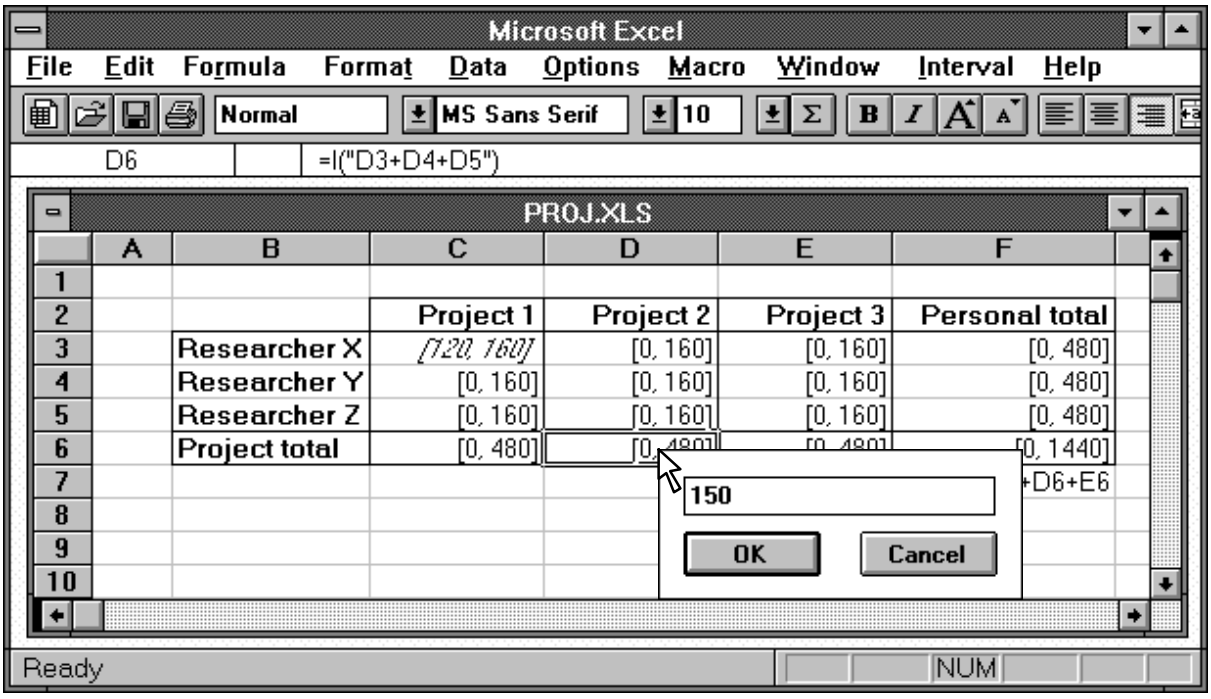

Figure 3. Changing the total time for the resources of Project 2 to 150 weeks.

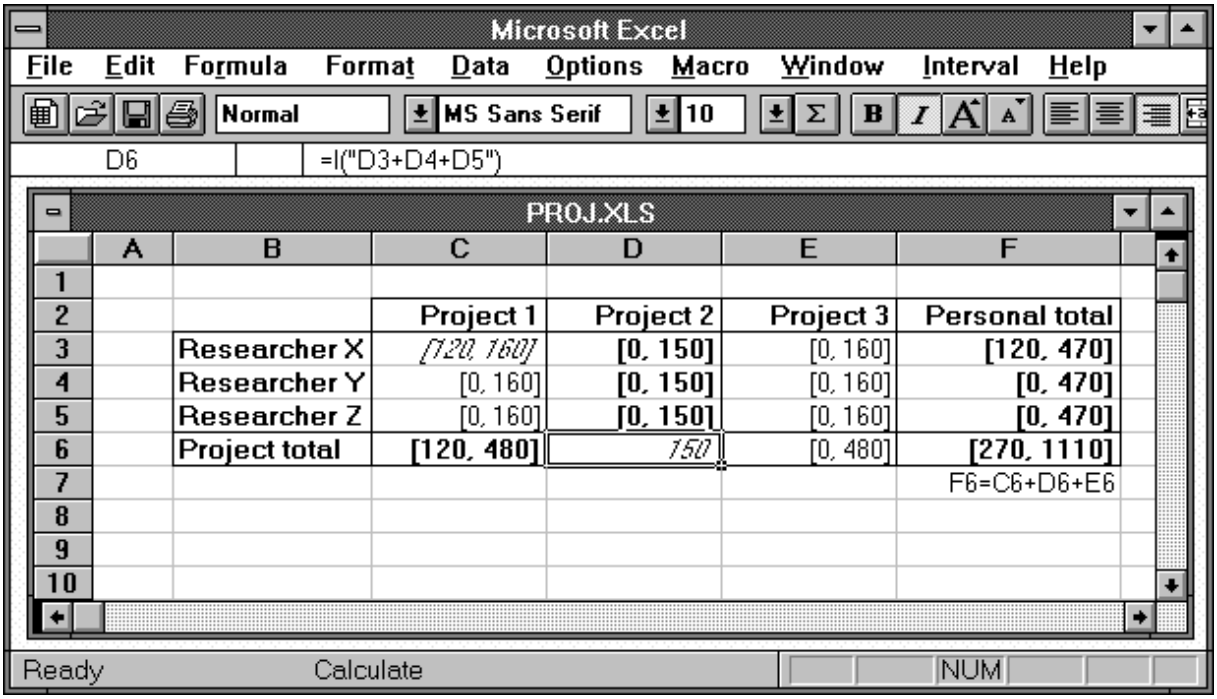

Figure 4. The user has made two changes (italic font). The system's modifications in response are shown in bold font.

7 Discussion

The main theoretical innovation of LIA InC++, GIA InC++, and ICE InC++ libraries is to apply a combination of interval analysis, computer algebra and tolerance propagation techniques for determining local and, especially, global interval solutions in an efficient way. Other implementations of interval constraints, such as (Older, Vellino, 1993; Lee, vanEmden, 1991ab, Sidebottom, Havens, 1992; Babitchev, 1993, Lhomme, 1993) do not make use of algebraic

techniques but rely on numerical ones. They also decompose computations into primitive functions and cannot determine in the general case global values. However, in the recent work of Benhamou et al. (1994) a variant of the interval Newton method is used for solving individual constraints globally.

Our experiments indicate that algebraic optimization of the functions to be evaluated often dramatically shortens execution times — even when the time needed for algebraic manipulation is taken into account of. From the numerical view point, the libraries seem to be first interval constraint tools to make use of numerical branch-and-bound algorithms developed in interval arithmetic literature. As a result, they have the power of determining global interval function values. This is in contrast with other implementations that usually deal with purely local techniques or employ numerical "splitting" schemes that may find better than purely local solutions, but that do not generally find global solutions and that may end up splitting intervals infinitely without good termination criteria.

A major problem one has to live with when dealing with interval computations is that their time complexity is to a large extent unpredictable: It heavily depends on the functions evaluated and argument values used. Our solution approach to difficult situations is—in addition to trying to make the implementation as efficient as possible—the use of precision and time limits, dynamic interrupts, and the possibility of getting excess width estimates for the returned value. Good news is that tolerance propagation makes it possible to trade computational time resources with the precision of results. Results, at least imprecise ones with possible extra width, can always be obtained in time.

The work of this paper demonstrates that interval function evaluation and constraint satisfaction techniques offer a promising avenue towards next generation interval spreadsheet programs.

Acknowledgements

This work has been supported financially by Technology Development Centre of Finland (TEKES). We thank Aarno Lehtola and Eero Peltola for discussions and support to our research. Thanks are also due to Trema Inc. and ViSolutions Inc. for co-operation.

References

Asaithambi, N., Zuhe, S., Moore, R. (1982) On computing the range of values, *Computing* **28** 225-237.

Babichev, A., Kadyrowa, O., Kashevarove, T., Semenov, A. (1993) Unicalc — An intelligent solver for problems with imprecise and subdefinite data. International Conference on Numerical Analysis with Automatic Result Verification, Lafayette, USA, March 1993.

Benhamou, F., McAllester, D., Van Hentenryck, P. (1994) CLP (Intervals) Revisited, Research Report (LIM University of Marseille, France).

Bohlender, G., Ulrich, C., von Gudenberg, J., Rall, J. (1987) *Pascal-SC, a Computer Language for Scientific Computation* (Academic Press, New York, NY).

Cleary, J.C. (1987) Logical arithmetic, *Future Computing Systems* **2** (2) 125-149.

Davenport, J.H., Siret, Y., Tournier, E. (1993) *Computer Algebra - Systems and Algorithms for Algebraic Computation* (Academic Press, New York, NY).

Davis, E. (1987) Constraint propagation with interval labels, *Artificial Intelligence* **32** 99-118.

Hyvönen, E. (1989) Constraint Reasoning Based on Interval Arithmetic, *Proceedings of IJCAI-89* (Morgan Kaufmann, Los Altos, USA) 1193-1198.

Hyvönen, E. (1991) Interval constraint spreadsheets for financial planning, *Proceedings of the first International Conference on Artificial Intelligence Applications on Wall Street* (IEEE Press, New York).

Hyvönen, E. (1992) Constraint reasoning based on interval arithmetic — The tolerance propagation approach, *Artificial Intelligence* **58** 71-112.

Hyvönen, E., (1994) Spreadsheets based on interval constraint satisfaction, *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* **8** 27-34.

Hyvönen, E., De Pascale, S., Lehtola, A. (1993) Interval constraint satisfaction tool INC++, *Proceedings of 5th International Conference on Tools with Artificial Intelligence*, Boston (IEEE Press, New York).

Hyvönen, E., De Pascale, S., (1994a) LIA InC++: A local interval arithmetic library, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland).

Hyvönen, E., De Pascale, S. (1994b) GIA InC++: A global interval arithmetic library, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland).

Hyvönen, E., De Pascale, S. (1994c) ICE InC++: A library for interval constraint equations, Research Report, (VTT, Technical Research Centre of Finland, Information Technology, Espoo, Finland).

Hyvönen, E., De Pascale, S., Lehtola, A. (1994, in press) Interval constraint programming in C++. In B. Mayoh, E. Tyugu, J. Penham (Eds.), *Constraint Programming,* NATO Advanced Science Institute, Series F (Springer-Verlag, Berlin).

Lee, J., vanEmden, M. (1991a) Numerical computation can be deduction in CHIP, paper (Department of Computer Science, University of Victoria, Canada).

Lee, J., vanEmden, M. (1991b) Adapting CLP(R) to floating point arithmetic, paper (Department of Computer Science, University of Victoria, Canada).

Lhomme, O. (1993) Consistency techniques for numeric CSPs. *Proceedings of IJCAI-93* (Morgan Kaufman, San Mateo, CA) 232-238.

Moore, R. (1966) *Interval Arithmetic* (Prentice-Hall, Englewood Cliffs, NJ).

Moore, R. (1979) *Methods and Applications of Interval Analysis*, SIAM Studies in Applied Mathematics (SIAM, Philadelphia).

Moore, R., Hansen, E., Leclerc, A. (1992) Rigorous methods for global optimization, In: *Recent Advances in Global Optimization* (Princeton University Press, Princeton) 321-342.

Older, W., Vellino, A. (1993) Constraint arithmetic on real intervals, in: F. Benhamou and A. Colmerauer, eds., *Constraint Logic Programming, Collected Research* (MIT Press, Cambridge, MA).

Ratscheck, H., Rokne, J. (1984) *Computer Methods for the Range of Functions* (Ellis Horwood, Chichister, England).

Ratscheck, H., Rokne, J. (1988) *New Computer Methods for Global Optimization* (Ellis Horwood, Chichister, England).

Sidebottom, G., Havens, W. (1992) Hierarchical arc consistency for disjoint real intervals in constraint logic programming, *Computational Intelligence* **8** (4).

Skelboe, S. (1974) Computation of rational interval functions, *BIT* **14** (1) 87-95.