CONSTRAINT REASONING BASED ON INTERVAL ARITHMETIC[1]

Eero Hyvönen
Technical Research Centre of Finland[2]
Lehtisaarentie 2A, 00340 Helsinki
FINLAND

Abstract

Current numerical constraint propagation systems accept as input only problems represented by exact numerical values and correspondingly produce only crisp solutions as output. In order to remove this limitation we have designed and implemented a generalized constraint propagation scheme based on interval arithmetic instead of conventional arithmetic. By using intervals instead of exact values we may express inexact numerical constraints in a well-defined way and compute necessary conditions for consistency in inconsistent underconstrained or overconstrained situations. If only singleton intervals are used our system produces similar results as conventional exact value systems.

1 Introduction

The problem

In this paper two major limitations of current numerical constraint propagation systems (Steele, 1980; Kanopasek and Jayaraman, 1984; Leler, 1988) are discussed:

1. The problem of exact input/output. The systems accept as input only exact values and correspondingly produce only crisp values for output. In many applications, however, data may be noisy, uncertain or incomplete and exact values cannot be determined for input. In such cases the output should reflect the inexactness of the input: more exact input values should result in more exact output values. Furthermore, in many cases it impossible make the distinction between input and output parameters beforehand. If a given parameter value is modified by the propagation engine the parameter is simultaneously input and output parameter. The type of the parameter is determined dynamically.

2. Reasoning in under- and constrained situations. The triggering condition for propagation, i.e. problem solving, is that the problem is perfectly (or over-) constrained. If the situation is underconstrained, i.e. too few parameter values are known, the systems typically halt without external help (like relaxation). When more than enough parameters are given as input propagation may proceed but usually just finds the situation inconsistent. In both cases the propagation systems cannot maintain consistency properly in the constraint network.

As a solution to these problems the application of interval arithmetic in numerical constraint reasoning instead of conventional exact value arithmetic is proposed. By using intervals we may express inexact, incomplete, and general numerical problems in a well-defined way and compute conditions for consistency in inconsistent situations.

For example, in current exact value systems the temperature conversion between Celsius (C) and Fahrenheit (F) degrees F=C*1.8+32 can be represented by the constraint net of figure 1.1. If C is known (input) F can be computed (output) and wise versa. However, problems in which both C and F are partly known cannot be dealt with properly. For instance, we may have two measurements C:=[1,5] and F:=[27,35] and want to know under what circumstances the measurements are coherent. In this kind of problems parameters are regarded simultaneously as input and output. The problem is underconstrained because C and F are not known exactly. However, the situation is also overconstrained because the measurements are partly conflicting. As result, exact value propagation techniques are inapplicable even for propagating the limits of the intervals. By our scheme we are able to determine as conditions of consistency: C:=[1,1.666...] and F:=[33.8, 35].

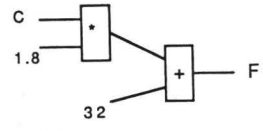

***Figure 1.1** A constraint net for temperature conversion.*

The idea of propagating simultaneous alternative values is called *tolerance propagation* (TP). TP can be applied in discrete logical domains, too, resulting in a generalization of clausal and multiple valued logics, *tolerance logics* (Hyvönen, 1988). A detailed presentation of the general TP scheme can be found in (Hyvönen, 1989).

Related work

Current inexact reasoning systems typically deal with crisp numerical values. In Bayesian systems (Pearl, 1988), for example, probabilities are exact numbers; inexactness is due to the user's interpretation of the values. However, in (Quinlan, 1983) a mechanism is developed for propagating probability intervals in an inference network. To this category of systems fall also systems based on Dempster-Shafer theory (Shafer, 1976). In these systems parameters refer to sets of alternative values and inexactness is embedded -- in a sense -- explicitly in the semantics of the systems. However, the mechanisms discussed are devised for probabilistic or uncertain reasoning only. We will develop a general approach for reasoning with numerical intervals that makes no commitment to a particular application domain of numerical reasoning like probabilistic or uncertain inference.

The problem of underconstrained situations can be approached in two major ways. Firstly, we can try to avoid underconstrained situations in the first place. Elias (1986), for example, has developed a technique to help the user in

[1] This work is a part of the FINPRIT research programme, funded mainly by the Technology Development Centre of Finland. Financial support was provided also by DEC, Finnish PTT, Nokia, VTKK, and Wärtsilä Marine.

[2] 15.3.1989-15.3.1990: Electrotechnical Laboratory, Cognitive Science Section, 1-1-4 Umezono, Tsukuba, Ibaraki 305, JAPAN.

selecting correct input parameter set for computing desired output parameters. For dealing with underconstrained problem formulations several techniques has been developed (Leler, 1988) like using defaults or heuristically determined values, applying relaxation (Newton-Raphson method and others) (Kanopasek, Jayaraman, 1984), or exploiting redundant views and algebraic transformations (Gosling, 1983). In these proposals underconstrained situations are made perfectly constrained by some strategy external to propagation. Furthermore, exact value arithmetic is used and inexact, possibly inconsistent, problem formulations (initial situations) or solutions (situations after propagation) cannot be represented. In our system problems and solutions can be represented in inexact terms and conditions for consistency be determined in inconsistent situations.

2 Interval Constraints

In this chapter basic notions of (rational) interval arithmetic (IA) (Moore, 1966) are presented and representation of interval constraints in our system is discussed.

Rational interval arithmetic

The *interval* X is a continuum (set) of real numbers represented as a number pair:

$$X=[a, b]=\{x \mid a\leq x\leq b\}$$

A real number x is in interval X=[a,b], i.e $x\in X$, iff $a\leq x\leq b$. Interval $A=[a_1,a_2]$ is a *subinterval* of $B=[b_1,b_2]$, i.e. $A\subseteq B$, iff $b_1\leq a_1\leq a_2\leq b_2$.

In IA algebraic basic operations +, -, *, and / can be defined as:

$$A+B = \{a+b \mid a \in A, b \in B\}$$
$$A-B = \{a-b \mid a \in A, b \in B\}$$
$$A*B = AB = \{a*b \mid a \in A, b \in B\}$$
$$A/B = \{a/b \mid a \in A, b \in B\} \quad , \text{if } 0 \notin B$$

We will use same function symbols for both interval and exact value operations. Which operation is in question can be determined by the arguments. When possible, capital letters will be used in interval functions.

For computational purposes, definitions relating the limits of argument intervals to the limits of the value interval can be derived:

$$[a,b]+[c,d] = [a+c, b+d] \qquad (2.1)$$
$$[a,b]-[c,d] = [a-d, b-c]$$
$$[a,b]*[c,d] = [\min(ac,ad,bc,bd),\max(ac,ad,bc,bd)]$$
$$[a,b]/[c,d] = [a.b]*[1/d,1/c] \text{ , if } 0 \notin [c,d]$$

IA differs from exact value arithmetic with respect to some basic algebraic laws. Distributive law does not hold in interval arithmetic. However, for any intervals I,J, and K the *subdistributivity law* $I(J+K)\subseteq IJ+IK$ holds. For example:

$$[1,2]*([1,2]-[1,2])=[-2,2]$$
$$\subseteq [1,2]*[1,2]-[1,2]*[1,2]=[-3,3]$$

An important property of interval arithmetic is *inclusion monotonicity* stating that if $I\subseteq K$ and $J\subseteq L$ then $I+J\subseteq K+L$, $I-J\subseteq K-I$, $I*J\subseteq K*L$, and $I/J\subseteq K/L$. This applies also more generally (Moore, 1966, theorem 3.1): If $F(X_1,...,X_n)$ is a rational expression in the interval parameters $X_1,...,X_n$, i.e. a finite combination of $X_1,...,X_n$ and a finite set of constant intervals with IA operations, then

$$X_1'\subseteq X_1, X_2'\subseteq X_2,...,X_n'\subseteq X_n \Rightarrow F(X_1',...,X_n')\subseteq F(X_1,...,X_n)$$

for every set of interval numbers $X_1,...,X_n$ for which the IA operations in F are defined.

Representation of constraints

In this paper we consider closed intervals. For notational convenience, a slightly larger and slightly smaller number than x is represented as x+ and x-, respectively. By this way open intervals can be approximately represented. For example:

$$[1+,2-]\approx[1.00...1, \quad 1.99...9]\approx(1,2)$$

Meaning of "slightly" depends on the accuracy we want or can use in computations. For example, if two numbers x and y are considered equal iff $|x-y|<d$ (absolute criterion), then we can use conventions x+=x+d, x-=x-d. For "infinitely" large and small numbers symbols $+\infty$ and $-\infty$ will be used, respectively. In practical systems, an infinitely large/small number will have some finite value.

In our experimental system an IA equation is represented in prefix notation. For example, the third order equation constraint $PX^3+QX=-R$ is represented as:

(ADD= (TEXP3 X) (MULT Q X) (OPP R)) (2.2)

Equations are parsed into sets of primitive constraints. For example, (2.2) is transformed into a set of $A^3=B$ (TEXPT3), A*B=C (MULT), -A (OPP), and A+B=C (ADD) primitives:

((TEXP3 X G1)(MULT Q X G2) (OPP R G3) (ADD G1 G2 G3)) (2.3)

The last argument represents the value of the function; if it is missing a dummy parameter (G*i*) is generated. Problems (to be called *propositions*) are represented by substituting each parameter an interval value. The value assignment defines the *situation* of the proposition. For example, the problem of solving the equation pair

$$E^X+X*Y=.2 \qquad (2.4)$$
$$SIN(X*Y)+X+Y=.4$$

is represented as:

(ADD= (TEXP X) (MULT X Y XY) A)
(ADD= (TSIN XY) X Y B)
$X:=[-\infty,\infty]$, $Y:=[-\infty,\infty]$, A:=[.2,.2], B:=[.4,.4]

3 Conditions of Consistency

In this chapter a technique for computing the conditions of consistency of an interval proposition is developed.

Admissibility and consistency

Let $C(P_1,...,P_n)$, $P_i:=X_i$, i=1...n be a proposition of a a single constraint C. The proposition is *admissible* iff

$$\exists\, x_1,...,x_i,...,x_n,\ x_i\in X_i,\ i=1...n:\ C(x_1,...,x_i,...,x_n) \text{ is satisfied}$$

Admissibility means that there is a way of selecting exact values consistently for the parameters within the tolerances. We say that a parameter $P_i:=X_i$, i=1...n, is *consistent* in the proposition iff:

$$\forall\, x_i\in X_i:\ \exists\, C(x_1,...,x_i,...,x_n) \text{ is satisfied,}$$
$$x_j\in X_j,\ j=1,...,i+1,i+1,...,n$$

For example, in the proposition

$$A+B=C \qquad (3.1)$$
$$A:=[1,2],\ B:=[3,4],\ C:=[4,6]$$

C is consistent. If C:=[4,6.1], then C is *inconsistent* because values $6<c\leq 6.1$ cannot be obtained by any pair of exact values for A and B.

A single contraint proposition is *consistent* iff its every parameter is consistent. Consistency of a proposition means that if we select any exact value for any parameter within its tolerance, it is possible to assign exact values to the other parameters within their tolerances in such a way that the constraint is satisfied. For example, (3.1) is consistent.

With *constraint nets*, i.e. sets of constraints, we will use two notions of consistency: A proposition is *locally consistent* if its every constraint is consistent independently from each other,

and *globally consistent* if its every constraint is consistent simultaneously.

Solution functions

In IA, arithmetic functions are used in one direction for computing the combined interval from the arguments. For example, if A=[1,2] and B=[3,4] we know that A+B=[4,6]. However, interval functions constrain values in other "directions", too. Symmetrically, the value condition of each parameter P_i:=X_i in a constraint can be represented by the *solution function* $X_i=F_i(X_1,...X_{i-1}, X_{i+1},...,X_n)$. For example, the solution functions for the interval addition constraint A+B=C are:

$$A=F(B,C)=C-B, \quad B=F(A,C)=C-A, \quad C=F(A,B)=A+B$$

In conventional arithmetic the relationships expressed by the solution functions hold simultaneously if one of them holds (excluding undefined exceptional cases, like division by zero). For example, a=c-b, b=c-a, and c=a+b in situation {a:=1, b:=2, c:=3}. IA is different in this respect. For example, in situation (3.1) we get C=A+B=[4,6], but A=C-B=[0,3]≠[1,2] and B=C-A=[2,5]≠[3,4].

It can be shown (Hyvonen, 1989) that:

Theorem 3.1. A constraint proposition $C(P_1,...,P_n)$, P_i:=X_i, i=1...n, is consistent iff its every solution function F_i, i=1...n, satisfies:

$$X_i \subseteq F_i(X_1,...X_{i-1}, X_{i+1},...,X_n)$$

It is assumed here that each F_i is *sound* and *complete*, i.e. it produces exactly the actual interval of values for P_i. It is easy to see that the rational IA functions of (Moore, 1966) (2.1), for example, satisfy this condition. Hence, for instance, (3.1) A=[2,5]⊇[3,4], and C=A+B=[4,6]⊇[4,6]. In our system we can also define solution functions for more complex constraints, like exponential, logarithmic, trigonometric, and hyperbolic constraints.

Hierarchical problem space

A situation S={X_1:=[...],...,X_n:=[...]} is called more *general* than S'={X_1':=[...],...,X_n':=[...]}, i.e. S'$\subseteq_S$S, iff $X'_i \subseteq X_i$, i=1...n. The situations of a constraint net form mathematically a hierarchical lattice defined by the $\subseteq_S$ relation. Based on this observation *the tolerance constraint satisfaction problem* (TCSB) can be stated as: Determine the least general common consistent situation (*solution*) of the given proposition. Intuitively, the task is to find the most constrained interval values still containing exactly the same exact value solutions as the original proposition. This formulation generalizes the conventional *Boolean constraint satisfaction problem* (BCSB) formulation (Macworth, 1987) (with its extensions to continuous domains) in which the situation space defined by the value assignments is non-hierarchical. A BCSB (in continuous domains) can be represented as a TCSB in which singleton tolerances (e.g. [3.14, 3.14]) are used as values for known parameters and value [-∞,∞] is used for unknowns.

A proposition may have none, one or infinitely many (sub)solutions. Generating all consistent subsolutions for a proposition is in the general case impossible without some kind of external heuristic preferences for selecting between the solutions. In our scheme such heuristics are not needed. The problem of finding a consistent solution that generalizes every consistent solution of the original proposition can be accomplished by purely mathematical means and without producing infinitely many individual subsolutions.

Tolerance propagation procedure

By theorem 3.1 a constraint is satisfied iff its solution functions are satisfied (by ⊆). This provides a practical means for computing the conditions of consistency for propositions. The procedure for doing this is represented below. The idea is to cautiously constrain tolerance values by the solution functions until each solution function of each constraint is satisfied or inadmissibility is found. In the procedure we have assumed that within the intervals of the original proposition situation the solution functions are defined, sound, and complete.

Procedure 3.1

Input: C, A set of constraints (eg. {SIN(X Y), ...})
Ass, Value assignment for the parameters in C
(eg. {X:=[.3,2],...})
Output: Locally consistent Ass

```
Agenda:=The solution functions of the constraints in C
For each Xi =Fi(...) in Agenda
   until Agenda={} do
      Xi':=Fi(...)
      Int:= Xi'∩ the value of Xi in Ass
      If Int={}
         then return 'Inadmissible'
      If Xi ⊆Xi'
         then remove Xi =Fi(...) from Agenda
         else set Xi:=Int in Ass
               Agenda:= Agenda U {Fi's with Xi as argument}
```

The procedure converges - at least asymptotically - towards a unique solution, if there exists one, or towards an inadmissible situation. The computational complexity of procedure 3.1 depends on the topology of the constraint net, constraints, initial values, the strategy used for updating the Agenda, and the accuracy demanded.

Consider as example the constraint equation

$$X^3-9X=-4 \qquad (3.2)$$

as an instance of (2.2). From situation X:=[0+,∞] the procedure 3.1 converges towards solution X:=[0.46.., 2.75..]. The actual positive roots of the equation are x:≈0.46, and x:≈2.75. By tolerance propagation we could constrain the tolerance of X as far as this was possible without losing any exact value solutions of the original proposition. If we consider subintervals X:=[0.46, 1.6] and X:=[1.6, 2.75] separately the procedure converges towards the two positive roots.

Proposition decomposition

However, if initial value X:=[-∞,∞] is used in (3.2) problems arise because the solution function Q=G2/X of multiplication constraint (MULT Q X G2) (2.3) is not defined if 0∈X. Our solution is to divide the problematic situation into an equivalent set S of less general *(sub)situations* for which procedure 3.1 can be applied. It is demanded that the division S is *complete* and *sound*. In such a division, each exact value solution of the original situation is a solution of one situation in S and each exact value solution of the situations in S is a solution of the original situation (i.e. exact value solutions are neither lost nor introduced). Similar decomposition strategy is used in situations in which a solution function produces multiple interval values (like in square or sinus constraints). In both cases multiple solutions may result. For example, if X:=[-∞,∞] in (3.2) the negative root X:=[-3.20.., -3.20..] is obtained, too.

We have used following criteria for situation decomposition:

Definability. Given a constraint C defined by solution functions F_i, each F_i must be defined within the argument intervals.

Monotonicity. For each (exact value) solution function f_i only one of the following situations holds within the argument intervals:

$\forall x_1,\ldots,x_n: x_j \le x'_j,\ j=1\ldots n \Rightarrow$
$f_i(x_1,\ldots,x_{i-1}, x_{i+1},\ldots,x_n) \le f_i(x'_1,\ldots,x'_{i-1}, x'_{i+1},\ldots,x'_n)$
$\forall x_1,\ldots,x_n: x_j \ge x'_j,\ j=1\ldots n \Rightarrow$
$f_i(x_1,\ldots,x_{i-1}, x_{i+1},\ldots,x_n) \ge f_i(x'_1,\ldots,x'_{i-1}, x'_{i+1},\ldots,x'_n)$

In the continuous case monotonicity assumption means that the partial derivatives of the solution functions with respect to each argument parameter

$$\frac{d\, f_i(x_1,\ldots,x_{i-1}, x_{i+1},\ldots,x_n)}{d\, x_j}, \quad j=1,\ldots,i-1,i+1,\ldots,n$$

do not change sign within the corresponding interval $x_j \in X_j$.

The reason for making this assumption is that the interval value [min,max] of the corresponding interval function F_i can now be determined easily without deeper functional analysis by:

$$\min=\min\{f_i(\underline{x}_1,\ldots,\underline{x}_{i-1}, \underline{x}_{i+1},\ldots,\underline{x}_n)\}$$
$$\max=\max\{f_i(\underline{x}_1,\ldots,\underline{x}_{i-1}, \underline{x}_{i+1},\ldots,\underline{x}_n)\}$$

Here each $\underline{x}_j$, $j=1,\ldots,i-1,i+1,\ldots,n$, is either $\min(X_j)$ or $\max(X_j)$.

As an example, consider the general logarithmic/exponential relationship $Y={}^B\log(X)$ defined by the solution functions:

$$y={}^b\log(x),\quad x=b^y,\quad b=x^{1/y}$$

By the definability assumption we get limitations $b>0$, $b\neq 1$, $x>0$, $y\neq 0$. By considering the derivatives of the solution functions one additional division point, $x=1$, is found:

$$dy/db=-\ln(x)/b*\ln(b)^2 \begin{array}{ll} \ge 0 & \text{if } 0<x\le 1 \\ <0 & \text{if } 1<x \end{array}$$

Altogether, the assumptions mean that parameter values must be considered separately within following limits: Y: {[-i,0-], [0+,+i]}, X: {[0+,1-], [1,+i]}, B: {[0+,1-], [1+,+i]}. In the worst case each argument interval contains a division point and 8 cases must be computed separately. If the original intervals do not contain division points, exactly one solution (or none, if the proposition is inadmissible) emerges. For example, to satisfy $Y={}^B\log(X)$ in {B:=[0.5,3], X:=[2,7], Y:=[-100,89]}, cases B:=[0.5,1-] and B:=[1+,3] must be considered separately with both positive and negative Y-values (four different subsituations).

Due to the monotonicity assumption the solution functions are defined easily as:

$$Y={}^B\log(X) = [\min\{{}^{b_i}\log(x_i)\},\max\{{}^{b_i}\log(x_i)\}],\quad i=1,2$$
$$X=B^Y = [\min\{b_i^{y_i}\},\max\{b_i^{y_i}\}],\quad i=1,2$$
$$B=e^{\ln(X)/Y}=[\min\{x_i^{1/y_i}\},\max\{x_i^{1/y_i}\}],\quad i=1,2$$

Monotonicity assumption is a sufficient condition for computing function value limits from argument limits. It is not a necessary condition because a function may have absolute maximum and minimum points at interval limits but be nonmonotonic in between. We apply monotonicity assumption due to its simplicity and generality although it in some cases may entail unnecessarily many substitutions to be considered separately.

4 Interval constraint reasoning

In this chapter some properties of the TP scheme are discussed.

Reasoning in under/overconstrained situations

In current propagation systems (Leler, 1988) a solution function cannot be computed before its arguments are known For example, for the addition constraint x=y+z two out of three parameters must be known. If less than enough parameters are known the situation *underconstrained* and the system keeps on waiting until enough values are provided from the outside. In many situations such values may never come and local propagation halts. We call this problem the *problem of underconstrained situations*, PUS. On the other hand, if more than necessary parameters are known the situation is *overconstrained*. Such situations are usually inconsistent and conditions for consistency cannot be determined in a well-defined way. Instead, ad hoc heuristics are used for forcing the situation consistent. We call this the *problem of overconstrained situations*, POS.

In our scheme both PUS and POS (partly) can be solved. PUS cannot occur because all parameters always have a value. Situations are actually always "overconstrained" in the above sense. However, the POS does not occur in our scheme as severely as in the exact value case: Conditions for consistency can be determined for inconsistent situations. Only if the situation is inadmissible POS is encountered.

Consider as an example of the PUS the circular equations

$$\begin{aligned} x+t&=y \\ y+t&=z \end{aligned} \qquad (4.1)$$

constraining y to be the average of x and z. By local propagation alone we cannot determine either y or t from the situation {x:=1, z:=11 t:=unknown, y:=unknown} because both addition constraints are locally underconstrained. However, at the global level the problem is perfectly constrained (two independent equations with two unknowns) with result {y:=6,t:=5}. In the TP scheme the PUS is solved without introducing special techniques, like relaxation. The problem can be represented by the following interval value assignment:

{X:=[1,1], , Z:=[11,11], T:=[-∞,∞], Y:=[-∞,∞]}

The difference with the exact value case is that our procedure does not halt but actually verifies consistency. The tighter the initial intervals Y and T are the better estimates we get. For example, if T:=[0,∞] we get T:=[0,11] and Y:=[1,11].

Local and global consistency conditions

The IA rules of (Moore, 1965) (2.1) often produce larger intervals than one would intuitively expect. There seems to a kind of mismatch between algebraic and arithmetic equalities. For example, consider equations:

$$A=X(Y+Z)$$
$$B=X*Y+X*Z$$

Our algebraic intuition suggests that the values of A and B should be always equal because their syntactic expressions are equivalent due to the distributivity law. However, in IA the distributivity law is substituted by the subdistributivity law $X(Y+Z)\subseteq X*Y+X*Z$ leading to conclusion $A\subseteq B$. The mismatch is due to the fact that IA rules treat the multiple instances of parameters as if they were totally different parameters with equal values. The rules are local and fail to notice that the instances refer globally to the same parameter with identical *simultaneous* exact value interpretation. The rules are too "cautious", i.e. produce too large intervals as result, but they never rule out correct exact value interpretations.

The problem of multiple parameter instances can be encountered (1) *locally* within a single constraint when defining the solution functions and (2) *globally* in cyclic constraint nets.

As an example of (1) consider the square constraint $X^2=Y$. If interval multiplication (2.1) is used for defining the solution function $Y=X^2$ we get, for example, from X=[-2,3] value Y=[-2,3]*[-2,3]=[-6,9], although the actual interval is Y=[0,9]. In such cases, special functions must be used that take into account of the global constraints. In the above case, for instance, we use the interval function:

$$Y=[\sqrt{x_1},\sqrt{x_2}] \qquad ;\ X=[x_1,x_2],\ 0\le x_1\le x_2$$

The problem (2) means that in a cyclic network a locally consistent solution produced by our procedure (3.1) is not necessarily globally consistent, although it always has the global solution as a subsolution, if there exists one. (Intuitively, we get a solution that is not the best one with the tightest conditions). For example, in example (4.1) we could not find the global solution {T:=[5,5], Y:=[6,6]} by TP. The problem can be approached by transforming the equation set algebraically into acyclic form. Gosling (1983) describes a technique for performing such transformations by eliminating critical parameters in the cycles. As result the constraint net is transformed to a single higher level constraint -- to be called *universal relation*. Gosling's technique can be generalized for the interval case. However, instead of applying it to PUS - as Gosling did - we apply its generalized version to solving the problem of finding the globally consistent solution by the local propagation procedure (3.1). In our scheme, the universal constraint corresponding to a set of equations S with parameters X_i is defined (as any constraint) by a set of (universal) solution functions Fu_i. In general, there may be several algebraic solutions $Fu_{i,j}$ for X_i. In order to respect them all the intersection of the $Fu_{i,j}$ values must be used as the universal solution function:

$$X_i = Fu_i(X_1,\ldots,X_{1-i},X_{1+i},\ldots X_n)$$
$$= \cap \{Fu_{i,j}(X_1,\ldots,X_{1-i},X_{1+i},\ldots X_n)\},\ j=1\ldots m$$

For example, by eliminating T from the equations (4.1) we get universal AVE-constraint with solution functions X=2Y-Z, Y=(X+Z)/2, and Z=2Y-X. (In this simple case only one partial function $Fu_{i,j}$ can be derived for each parameter.) By the AVE-constraint we always get the global conditions for consistency for X, Y, and Z. In our example (4.1), procedure 3.1 finds the solution with {T:=[5,5], Y:=[6,6]}.

Algebraic transformations can be applied to making interval reasoning more efficient, too. They also give us a tool for defining more abstract constraints. A major problem with the approach is that in some cases the solution functions may be difficult to obtain or cannot be determined at all by algebraic means. Another problem is that transformations are computationally costly. In problematic situations local tolerance propagation can always be applied and benefits of algebraic and constraint based arithmetic reasoning can be combined. For example, the non-linear equations (2.4) are hard to manipulate algebraically. However, by procedure 3.1 we obtain X:=[-1.61.., 0-], Y:=[.4, 2.73..] as local conditions of consistency without using any global techniques. By dividing these intervals further and propagating values, the exact solution x:=.385.., y:=1.247.. is easily found with enough precision in a few iterations.

Interval relaxation

Tolerance propagation can be used instead of exact value relaxation in under/overconstrained situations. On the other hand, relaxation techniques can combined with interval reasoning. For example, interval reasoning can be applied to solving the problem of determining the rough magnitude of the initial guess values in exact value relaxation. (After finding a local interval solution exact value relaxation can be applied for the global solution.) A major benefit of the TP approach is completeness: Tolerance propagation does not "lose" solutions. For example, with exact value relaxation techniques, like Newton-Raphson method, we cannot always determine whether there exists a root within an interval. Consider as example the problem of finding the roots of a fourth order equation $X^4+2X^3+3X^2+4X+5=0$, X:=[-∞,∞]. By procedure 3.1 we first get the (locally consistent) solution X:=[-2.34.., -.02]. We know that *if* the original equation has roots they must be within this interval. By considering, for example, the intervals X:=[-2.34.., -1.67] and X:=[-1.67 -.02] inadmissibility is found in both cases immediately. After this we can be *sure* that (within the accuracy used) the original equation has no real roots. With TP we can consider infinite numbers of exact value solution candidates simultaneously. By reasoning with more general structures (intervals) we can obtain more general results.

5 Example of Application

In this chapter we apply our scheme as example to a mathematically simple but yet non-trivial design problem. The goal is to illustrate the practical applicability of the TP scheme.

Our problem is to allocate the working time of r researchers in p projects. The working time XY of each researcher X in project Y is constrained as follows: (1) Each researcher X has an individual total capacity of X hours. (2) Each project Y needs resources of Y hours. This basic problem is quite common in many organizations. Problems of similar structure are met in various design tasks. However, satisfactory tools for solving it have not been developed.

In the problem, the mathematical relations between the parameters are simple additions. For example, the arithmetic constraints for three researchers A, B, and C and three projects P1, P2, and P3 are:

A+B+C=TOTAL	; Total working hours
P1+P2+P3=TOTAL	
AP1+BP1+CP1=P1	; Project resources
AP2+BP2+CP2=P2	
AP3+BP3+CP3=P3	
AP1+AP2+AP3=A	; Researcher resources
BP1+BP2+BP3=B	
CP1+CP2+CP3=C	

Figure 5.1 depicts the situation as a graph.

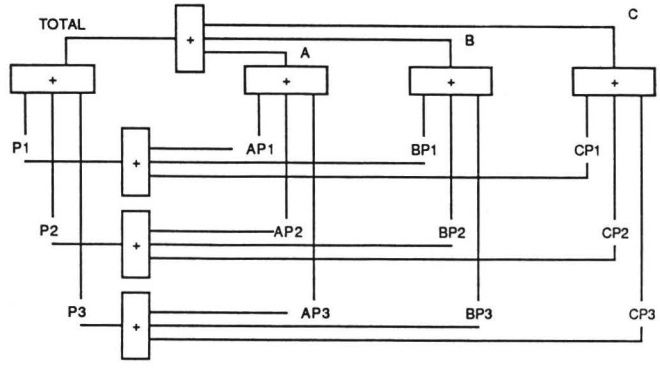

Figure 5.1. *A simple constraint network*

By current spread sheets or exact value propagation systems arithmetic computations cannot be performed unless the problem is perfectly constrained by the user (enough and right parameter values are known). This not usually possible, especially in the beginning of the allocation design process when most allocations are unknown and are to settled more or less simultaneously. Furthermore, some parameter values cannot perhaps be represented by exact numbers. It is also quite hard to see what are the input parameter combinations from which a set of desired output parameter could be computed. A change in any parameter value may affect any other parameter's value.

With exact value techniques we end up in a tedious iteration or relaxation where some set of values (typically XY's) are guessed, others (A, B, C and P1,P2,P3) are computed, and results evaluated (with respect to the problem constraints) until a satisfying assignment of parameter values emerges. The solution - if found - is always exact and cannot reflect inexactness of input data. By our scheme the user can initially

set the intervals for the parameters as wide or as strict as he wants. In any inconsistent (but admissible) design state suggested by the user the system can derive a consistent solution for the problem. Problem solving proceeds not by iteration but by *stepwise refining* the constraints for the problem until a satisfying solution of acceptable precision emerges. Consultation mode is *mixed initiative:* The user proposes a refinement to the problem formulation or a solution and the system determines additional interval constraints derivable.

For example, the intervals of the parameters may initially be

{A:=B:=C:=[0,160],
P1:=P2:=P3:=[0,480]
AP1:=AP2:= AP3:=[0,160]
BP1:=BP2:=BP3:=[0,160]
CP1:=CP2:=CP3:=[0,160]
TOTAL:=[0,480]}

If the user increases the lower limit of project P2 up to 160, i.e. P2:=[160,480], the system can infer {P1:=[0,320], P2:=[0,320], TOTAL:=[160,480]}. If we accept this and constrain the problem further by saying that A should not work in project P1 less that 120 hours, i.e. A1:=[120,160], then the system can infer seven necessary modifications for the other parameters: {P1:=[120,320], P2:=[160,360], P3:=[0,200], TOTAL:=[280,480] AP2:=[0,40], AP3:=[0,40], A:=[120,160]}. In this case only one limit of one parameter was changed in a quite simple constraint model. In the general case we may modify both lower and higher limits of several parameters simultaneously. Furthermore, more complex arithmetical relationships than addition can be applied.

If the problem in some phase of designing is found to be inadmissible the system can easily propose one potential way of making the system consistent again, i.e. a suggestion is made for making some interval larger than it was in the original problem formulation.

6 Conclusions

In this paper we have generalized numerical exact value propagation into interval propagation. Semantically, the main difference between exact value and tolerance propagation is: In TP, the propositions/solutions constitute a hierarchy (lattice) defined by the partial generality relation between their situations. Tolerance satisfaction means determination of the least general common consistent situation that generalizes each exact value solution of the problem. Major contributions of this view are: We can represent inexact and general problems in terms of intervals, perform computations at a more abstract level by interval functions, perform numerical reasoning in underconstrained situations and with inconsistent data, and represent inexact and generalized solutions. The consistency maintenance scheme can be used to support several intelligent functions, like stepwise refinement strategy in problem solving and mixed initiative consultation mode.

Acknowledgements

Fruitful discussions with Seppo Linnainmaa, Matti Hämäläinen, Aarno Lehtola, and Tapio Niemelä are acknowledged.

References

Elias, A. L. 1986. Knowledge Engineering of the Aircraft Design Process. In Kowalik, J. S. (ed.): Knowledge Based Problem Solving. Englewood Cliffs, New Jersey, Prentice-Hall, pp. 213-256.

Gosling J. 1983. Algebraic Constraints. Doctoral Dissertation. Pittsburgh, Pennsylvania, Department of Computer Science, Carnegie-Mellon University.

Heintze, N. C. et. al. 1987. Constraint Logic Programming: A Reader. Fourth IEEE Symposium on Logic Programming, San Francisco, 31.8.-4.9.1987.

Hyvönen, E. 1988. Truth Maintenance of Incomplete Knowledge. Paper, Helsinki, Technical Research Centre of Finland, Lab. for Information Processing, 10 pp.

Hyvönen, E. 1989. Tolerance Propagation. An Approach to Inexact Constraint Reasoning. Working paper, Helsinki, Technical Research Centre of Finland, Lab. for Information Processing, 150pp.

Jaffar, J., Lassez, J-L. 1986. Constraint Logic Programming. Clayton, Victoria, Australia, Monash University, Dept. of Computer Science, Technical Paper.

Kanopasek, M., Jayaraman, S. 1984. The TK!Solver book. Berkeley, California, McGraw-Hill, 458 pp.

Leler, W. 1988. Constraint Programming Languages. Their Specification and Generation. Reading, Massachusetts, Addison-Wesley, 202 pp.

Macworth, A. K. 1987. Constraint Satisfaction. In Shapiro, S. (Ed.): Encyclopedia of Artificial Intelligence. New York, John Wiley & Sons.

Moore, R. E. 1966. Interval Arithmetic. Englewood Cliffs, New Jersey, Prentice-Hall.

Pearl, J. 1988. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Palo Alto, California, Morgan Kaufmann, 552 pp.

Quinlan, R. 1983. Inferno: A Cautious Approach to Uncertain Inference. The Computer Journal, Vol. 26, No. 3, August, pp. 255-269.

Shafer, G. 1976. A Mathematical Theory of Evidence. Princeton, New Jersey, Princeton University Press.

Steele G. L. 1980. The Definition and Implementation of a Computer Programming Language Based on Constraints. Doctoral Dissertation, Cambridge, Massachusetts, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.