Journal of the Korean Institute
of Industrial Engineers
Vol.20, No.1, March., 1994.

Interval을 이용한 Conditional Constraints의 Propagation 알고리듬

A PROPAGATION ALGORITHM FOR INTERVAL-BASED CONDITIONAL CONSTRAINTS

김경택*

Kyeong Taek Kim*

Abstract

Conditional constraints are frequently used to represent relations. To use these conditional constraints, it is necessary to develop an appropriate logic in which these conditional constraints can be represented and manipulated. Nevertheless, there has been little research that addresses interval-based conditional constraints. The proposed approach addresses the use of conditional constraints involving intervals in constraint networks. Two algorithms are presented: (1) a propagation algorithm for an interval-based conditional constraint, which is similar to one for an exact-value conditional constraint; (2) a propagation algorithm for interval-based conditional constraints which satisfy some conditions. The former can be applied to any conditional constraint. However, with the former algorithm, conditional constraints are usually categorized into the cases that they cannot be propagated. After investigating several methods in which most conditional constraints can be propagated, we propose the latter algorithm under certain condition that usually results in smaller resulting design space comparing to the former.

* 한국전자통신연구소

1. Introduction

As engineering design progresses from the early stages of requirement formulation toward the detailed design stage, the constraints grows not just in volume but in complexity. Growth in volume means that an object which is represented by a few aggregate constraints at the early design stage is elaborated into tens or hundreds of detail constraints in the detailed design stage. The growth in complexity comes from inserting interrelationships that are missing at early stages. In such complex designs, it is difficult to keep track of all relevant constraints and variables, and to understand the interrelationships among variables and their tradeoffs. In such complex design environments, constraint networks can help the designer in the following functions:

- to test the truth value of each constraint
- to maintain global consistency
- to find values for unknown variables from the set of given variable values
- to reduce the design space without loss of any feasible solution.

So far constraint networks have successfully supported the above functions for the areas where they have been applied.

The main inference used in engineering design varies with the design stage. At the earlier design stages, it is common that not only some design parameters but also some variable values are imprecise. Therefore intervals are probably used frequently. As the design progresses, more specified alternatives will emerge and replace continuous domains. At this stage, descrete label inference is more likely actively used. At the detailed design stage, specific value for each variable should be picked. Therefore exact-value inference is more appropriate at the latter stage.

Most research on constraint problems provides solution techniques for handling and propagating constraints with exact-value variables. But there are some limitations to apply these techniques to engineering design problems.

First, one of the characteristics of engineering design problems is inequality constraints. In exact-value constraint system, the upper bound and the lower bound of a variable are used only to test if a specified or deduced value falls between these bounds. There is, therefore, the potential of relevant information being lost.

Second, the design process in an engineering design problem expressed as a constraint network means the process of finding a set of variable values which satisfies all the constraints in the network. During the design process it may be impossible to pinpoint the presumed values for some variables. A design, which is incomplete, may have some variables which cannot be assigned exact values because the designer only has a vague idea for the values of those variables. The system should utilize the vauge idea.

Third, the engineering problem usually includes not only variables with discrete domains

but also variables with continuous domains. Pruning continuous domain by the process which is used to prune discrete domains would be computationally prohibitive.

The above would therefore indicate that there are some considerable benefits that can result from the use of an interval approach to engineering design.

2. Interval Approaches in Related Areas

A few systems have used interval arithmetic to deal with continuous real numbers. CHIP (Constraint Handling In Prolog) is a language on the finite domain and linear rational number [1,2,3]. CHIP can deal with linear equations and inequalities such as $\leq$, $<$, $\geq$, $>$, over natural numbers. ALICE [4] is a problem-solver where problems are stated in a mathematical language based on set-theory, first-order logic, and graph theory. Combinatorial problems can be solved by ALICE. It has a procedure for dealing with linear equations and inequalities($\leq$). In ALICE the procedure results in addition of several constraints to its program. Hyvönen [5] defines the term 'interval-consistent' for a constraint and constraint network. He describes the characteristics of interval arithmetic compared with value arithmetic. LIC(Labeled Interval Calculus) [6,7,8,9] is a formal system that performs interval inference about sets of artifacts under several operation conditions. By the use of labels, it represents the meaning of the relationship between variables and its intervals. LIC is used as the basis of a program called "mechanical design compiler." The compiler provides the user with a language in which design problem for systems to be built of cataloged components can be quickly and easily formulated. Navinchandra and Rinderle [10] show how interval propagation can help the designer by two-bar truss example. Rinderle and Krishnan [11] propose an interval-based approach, which is augmented with monotonocity and dominance principles. The goal is to identify active constraints in complex design problems with objctive functions. CASCADE-T [12] is a computer-based tolerance synthesis system that employs a constraint propagation similar to CONSTRAINTS [13].

It is found from the above liteature review that conditional constraints are not allowed on the basis of intervals. Current interval-based systems would stop or ignore if they meet conditional constraints. However, conditional constraints are frequently used to represent some relations. In this paper, we propose two algorithms: (1) a propagation algorithm for an interval-based conditional constraint, which is similar to one for an exact-value conditional constraint; (2) a propagation algorithm for interval-based conditional constraints which satisfy some conditions.

3. Conditional Constraints

Conditional information of the form "if ω then Ψ", is referred to as implication, conditional, or conditional constraint. ω is called the antecedent and Ψ is called the consequent. The interpretation of the conditional constraint "if ω then Ψ" is hardly obvious. Intuition and common sense do not provide a clear answer. Nevertheless an interpretation is required to use this operator. In Boolean Logic, "if ω then Ψ" is interpreted as being equivalent to "(not ω) or Ψ", the material implication. The material implication is represented by "$\Rightarrow$". Table 1 shows the truth value of "$\omega \Rightarrow \Psi$" in Boolean Logic [14]. Boolean Logic is a two-valued logic, where the truth value of any logic formula is either *true* or *false*.

Table 1. Truth Tables of $\omega \Rightarrow \Psi$ in Boolean Logic

ω	Ψ	$\omega \Rightarrow \Psi$
T	T	T
T	F	F
F	T	T
F	F	T

3.1 Definitions on the Satisfiability Values

It is pointed out that two-valued logic is not appropriate for interval-based approach [15]. Kim [15] suggests a three-valued logic for an interval-based approach. In three-valued logic, the following satisfaibility values are used:

Definition 1: *valid, satisfiable, unsatisfiable*

A constraint is *valid* in the interval-based logic if it is *true* for any tuples in the current design space in the exact-value logic with single numbers.

A constraint is *satisfiable* in the interval-based logic if it is *true* for some tuples and *false* for other tuples in the current design space in the exact-value logic with single numbers.

A constraint is *unsatisfiable* in the interval-based logic if it is *false* for any tuples in the current design space in the exact-value logic with single numbers.

Since a three-valued logic rather than a two-valued logic is employed, the satisfiability value of the conditional constraints should be defined. For further discussion, subset relation between two interpretations is defined as follows:

Definition 2: *subset relation between two interpretations*

An interpretation I is a subset of an interpretation J if and only if for each variable x, the interval of x under I is a subset of the interval of x under J.

3.2 Satisfiability Value of a Conditional Constraint

The satisfiability value of a conditional constraint is determined by the satisfiability value of its antecedent and its consequent as

follows:

- A conditional constraint is *valid* if and only if
  the antecedent is *unsatisfiable*, or
  both the antecedent and the consequent are *valid*.
- A conditional constraint is *satisfiable* if and only if
  the antecedent is *valid* and consequent is *satisfiable*, or
  the antecedent is *satisfiable*.
- A conditional constraint is *unsatisfiable* if and only if
  the antecedent is *valid* and the consequent is *unsatisfiable*.

3.3 Satisfiability Values of Antecedents and Extension of Interpretation

For a conditional constraint in which the satisfiability value of the antecedent is *valid* under the current interpretation, there is no more need to re-evaluate the antecedents again in the following cycles as long as the given constraints and intervals are deleted or modified. This is because under *any* interpretation, which is a *subset* of the current interpretation, the satisfiability value of the antecedent remains *valid* as long as the given constraints and intervals are deleted or modified. Addition of constraints neither change the satisfiability value of the *valid* antecedent nor affect the fact that *valid* antecedent does not change any interpretation.

If the antecedent is *valid*, the consequent is evaluated to determine the satisfiability value of the conditional constraint and whether the current interpretation is extendible.

For a conditional constraint in which the antecedent is *unsatisfiable* under the current interpretation, there is no need to re-evaluate the antecedent and the consequent again as long as it is under *any* interpretation which is a subset of the current interpretation. This is because under *any* interpretation which is a subset of the current interpretation, the satisfiability value of *unsatisfiable* antecedent does not change.

For a conditional constraint in which the antecedent is *satisfiable*, the antecedent should be re-evaluated at the every cycle as long as any variable value in the antecedent changes. For the constraint with a *satisfiable* antecedent, extending the interpretation using its consequent may restrict unnecessarily the design space. For example, consider the following constraints:

C1: $w=[2,4]$
C2: $x=[2,4]$
C3: $y=[3,5]$
C4: $z=[1,3]$
C5: $x \geq y \Rightarrow z=w$

From the above, the antecedent of C5 is *satisfiable*. If the current interpretation is extended using the consequent of C5, then $z=[2,3]$ and $w=[2,3]$ will be deduced. Under the extended interpretation, the reduced design space is:

$w=[2,3]$

$x=[2,4]$

$y=[3,5]$

$z=[2,3]$.

However, there are tuples, such as $\langle w,x,y,z\rangle=\langle 4,2,5,1\rangle$ which satisfy all the above constraints and are out of the reduced design space. This example shows that the interpretation extended using a consequent may exclude some feasible solutions if its antecedent is *satisfiable*.

3.4 An Algorithm for Extending Interpretation using a Conditional Constraint

The algorithm shown in Figure 1 is proposed to determine the satisfiability value of an interval-based conditional constraint and to extend the interpretation where appropriate. Most conditional constraints have *satisfiable* antecedents. In this algorithm conditional constraints with *satisfiable* antecedents cannot be propagated.

If a consequent with a *satisfiable* antecedent cannot be used for extending the current interpretation, it will be disadvantageous in that no further information would be deduced from the conditional constraints with satisfiable antecedents. What is more desirable is that a method is developed to extend the interpretation using a consequent with a *satisfiable* antecedent under some circumstances. In the following section the methods in which conditional constraints with *satisfiable* antecedents are utilized will be discussed.

4. Extended Interpretation using Constraints with Satisfiable Antecedents

There are two methods in which the extension of the current interpretation using a consequent with a *satisfiable* antededent does not exclude any feasible solution. The first method is to partition each interval into several sub-intervals based on the antecedents, and to compose exhaustive combinations and to propagate each combination. For example consider the following constraints:

C1: $x=[0,7]$

C2: $y=[10,20]$

C3: $z=[0,5]$

C4: $v=[3,6]$

C5: $x\leq 2\Rightarrow z=v$.

C6: $x>2$ and $x\leq\Rightarrow z=2\otimes v$

C7: $y\leq 15\Rightarrow w=y$.

C8: $y>15$ and $y\leq 20\Rightarrow w=2\otimes y\ominus 15$.

C9: $y>20\Rightarrow w=3\otimes y\ominus 35$.

Using the antecedents of the above constraints, the following exhaustive combinations can be made:

$x=[0,2]$ and $y=[10,15]$,

$x=[0,2]$ and $y=[15,20]$,

$x=[0,2]$ and $y=[20,25]$,

$x=[0,2]$ and $y=[10.15]$,

$x=[0,2]$ and $y=[15,20]$,

$x=[0,2]$ and $y=[20,25]$.

For each combination, one propagation proceeds. This method is a kind of generate-and-test method. Although it seems to be very simple, the expected computing time is combi-

For each interval-based conditional constraint, find the satisfiability value of its antecedent.

- For a conditional constraint with an *unsatisfiable* antecedent, record the conditional constraint as being *valid* and do not consider it any more until the user changes constraints or intervals of variables.
- For a conditional constraint with a *valid* antecedent, find the satisfiability value of its consequent.
  1. If the consequent is *valid*, record each resulting variable value and record the conditional constraint as being *valid*.
  2. If the consequent is *unsatisfiable*, exit this algorithm and report that a constraint violation is detected.
  3. If the consequent is *satisfiable*, extend the current interpretation using the consequent and find the satisfiability value of the consequent under the extended interpretation.
     3.1 If the consequent becomes *valid*, record each resulting variable value and record the conditional constraint as being *valid*.
     3.2 If the consequent is still *satisfiable*, record each resulting variable value and record the conditional constraint as being *satisfiable*.
     3.3 If the consequent becomes *unsatisfiable*, exit this algorithm and report that a constraint violation is detected.
- For a conditional constraint with a *satisfiable* antecedent, record the conditional constraint as being *satisfiable*.

Figure 1. An Algorithm for Extending Interpretation using a Conditional Constraint

natorially explosive. For example, for the n interval variables in which each interval is divided into m sub-intervals, the whole constraints should be evaluated approximately m^n times, if the interpretation is extended no more. This method may therefore be intractable not only in terms of computation time, but also in terms of requiring space for recording the whole set of combinations and each set of results.

The second method is to partition an interval into several sub-intervals based on the antecedents, and to propagate the first sub-interval, and after the propagation is finished, to propagate the next sub-interval and so on. For example, assume that the above constraints are given. Then, the propagation will be proceed as follows:

instantiation 1: $x=[0,2]$,
instantiation 2: $x=[0,2]$ and $y=[10,15]$,
instantiation 3: $x=[0,2]$ and $y=[15,20]$,
instantiation 4: $x=[0,2]$ and $y=[20,25]$,
instantiation 5: $x=[2,5]$,
instantiation 6: $x=[2,5]$ and $y=[10,15]$,
instantiation 7: $x=[2,5]$ and $y=[15,20]$,
instantiation 8: $x=[2,5]$ and $y=[20,25]$.

If the propagation of instantiation 1 results in a constraint violation, instatiation 2 through 4 will not be propagated. In the same way, if the propagation of instantiation 5 results in a constraint violation, instantiation 6 through 8 will not be propagated. Therefore if this

method is applied to the example, the number of porpagation will be at least two and at most eight.

This method is a kind of backtracking method. Compared to the first method, this method can utilize the intermediate results if it is saved. However, if this method is applied to constraints with n interval variables in which each interval is divided into m sub-intervals as a result of conditional constraints, each constraint should be evaluated approximately $m(m^n-1)/(m-1)$ times in the worst case, if the interpretation is extended no more. In the case that further restrictions are imposed on the current constraint network as the designer's knowledge for the artifact is accumulated, all the set of sub-intervals that are not proved to be infeasible should be kept. If all sets of feasible sub-intervals are recorded, large recording space is usually required.

4. An algorithm for Extending an Interpretation Using Grouped Conditional Constraints

The two approaches described in the previous section suffer from computational complexity and large memory requirements. The approach proposed in this paper is to maintain only one interval for each variable while the design space can be reduced by the extension of interpretation using a group of conditional constraints.

Definition 3: *Sufficient Antecedent Space*

A group of conditional constraints has a *sufficient* antecedent space when every tuple in the space, which is Cartesian product of variable values under the current interpretation, satisfies at least one antecedent.

Note that when a group of conditional constraints consists of a single conditional constraint with *valid* antecedent, it has a *sufficient* antecedent space. For this constraint, there will be no difference in their results between the case that the algorithm shown in Figure 1 is applied and the case that the algorithm that will be proposed in the next section is applied. If a group of conditional constraints consists of a single conditional constraint with *satisfiable* or *unsatisfiable*, it does not have a *sufficient* antecedent space. For this conditional constraint, the algorithm that will be proposed in the next section cannot be applied. Instead, the algorithm shown in Figure 1 can be applied.

For the case that a group of conditional constraints consists of two or more conditional constraints, consider the following constraints:

C1: $(10 \leq x) \Rightarrow (y = x + 10)$.

C2: $(5 \leq x < 10) \Rightarrow (y = 2x)$.

The group of conditional constraints, C1 and C2 under an interpretation where $x = [0, \infty]$, has an *insufficient* antecedent space. If another constraint C3: $x = [7,15]$ is added to the above constraints, then under the extended interpretation x can be set to $[7,15]$. Under this

extended interpretation, the group of conditional constraints, C1 and C2, has a *sufficient* antecedent space.

In this paper, the algorithm shown in Figure 2 is proposed for the extension of an interpretation using a group of conditional constraints having a *sufficient* antecedent space. It is based on the premise that the extension of an interpretation is performed using the group of conditional constraints as a whole.

After the current interpretation is extended using a group of conditional constraints, normal propagation is continued, if necessary. To illustrate the effect of applying the above algorithm on the final result, consider the same constraints as in the previous section:

C1: $x=[0,7]$

C2: $y=[10,20]$

C3: $z=[0,5]$

C4: $v=[3,6]$

C5: $X\leq 2\Rightarrow Z=V$.

C6: $x>2$ and $x\leq 5\Rightarrow z=2\otimes v$

C7: $y\leq 15\Rightarrow w=y$.

C8: $y>15$ and $y\leq 20\Rightarrow w=2\otimes y\ominus 15$.

C9: $y>20\Rightarrow w=3\otimes y\ominus 35$.

Under the interpretation where

$x=[0,5]$,

$y=[10,25]$,

$z=[0,5]$,

$v=[3,6]$

a group of conditional constraints, C5 and C6 has a *sufficient* antecedent space. Therefore the above algorithm can be applied to the group of conditional constraints. By applying the above algorithm to C5 and C6, the interpretation I will be extended, and under the extended interpretation, J,

$x=[0,2]$,

$y=[10,25]$,

$z=[3,5]$,

$v=[3,5]$,

Under the interpretation J, a group of conditional constraints, C7, C8, and C9, has a *sufficient* antecedent space. Thus, the above algorithm can be applied to C7, C8, and C9, and the final result is as follows:

$x=[0,2]$,

$y=[10,25]$,

$z=[3,5]$,

$v=[3,5]$,

$w=[10,40]$.

For the n interval variables in which each interval is divided into m sub-intervals, each constraint will be used only one times if the interpretation is extended no more. The merit of applying this method to conditional constraints is that the number of propagation is significantly reduced comparing to the previous two methods, i.e., generate-and-test method, and backtracking method. The another merit is that there is no need to partition the variable intervals. However, if this method is applied, the result is the filled union of each sub-intervals resulting from applying one of the previous two methods. The filled union of two intervals results in an interval where the lower

For a group of conditional constraints with *sufficient* antecedent space:

1. For each conditional constraint, find the satisfiability value of its antecedent.
   - 1.1 For a conditional constraint with an *unsatisfiable* antecedent, record the conditional constraint as being *valid* and do not consider it any more.
   - 1.2 For a conditional constraint with a *valid* antecedent, find the satisfiability value of the consequent.
     - 1.2.1 If the consequent is *valid*, record each resulting local variable value and record the conditional constraint as being *valid*.
     - 1.2.2 If the consequent is *unsatisfiable*, report that a constraint violation is detected.
     - 1.2.3 If the consequent is *satisfiable*, extend the current interpretation using the consequent and find the satisfiability value of the consequent under the extended interpretation.
       - 1.2.3.1 If the consequent becomes *valid*, record each resulting local variable value and record the conditional constraint as being *valid*.
       - 1.2.3.2 If the consequent is *satisfiable*, record each resulting local variable value and record the conditional constraint as being *satisfiable*.
       - 1.2.3.3 If the consequent becomes *unsatisfiable*, record the conditional constraint as being *valid*.
     - 1.3 For a conditional constraint with a *satisfiable* antecedent, find the satisfiability of its consequent
       - 1.3.1 If the consequent is *valid*, extend the current interpretation using the antecedent and find the satisfiability of the antecedent under the extended interpretation.
         - 1.3.1.1 If the antecedent becomes *valid*, record each resulting local variable value and record the conditional constraint as being *valid*.
         - 1.3.1.2 If the antecedent is *satisfiable*, record each resulting local variable value and record the conditional constraint as being *satisfiable*.
         - 1.3.1.3 If the antecedent becomes *unsatisfiable*, record the conditional constraint as being *valid*.
       - 1.3.2 If the consequent is *unsatisfiable*, record the conditional constraint as being *valid*.
       - 1.3.3 If the consequent is *satisfiable*, extend the current interpretation using the antecedent and consequent and find the satisfiability of the antecedent and the satisfiability of the consequent under the extended interpretation.
         - 1.3.3.1 If both the antecedent and the consequent become *valid*, record each resulting local variable value and record the conditional constraint as being *valid*.
         - 1.3.3.2 If none of them becomes *unsatisfiable* and one of them is still *satisfiable*, record each resulting local variable value and record the conditional constraint as being *satisfiable*.
         - 1.3.3.3 If any of them becoms *unsatisfiable*, record the conditional constraint as being *valid*.
2. For each variable, set the value of the global variable to the union of the local variable intervals, which result from step 1. If any of global variable is ϕ, then report that a violation is detected.

Figure 2. An Algorithm for Extending Interpretation using Grouped Conditional Constraints

bound is the minimum of the lower bounds of two intervals and the upper bound is the maximum of the upper bounds of two intervals.

5. An Example

An example, as shown Figure 3, determining the lead access hole space in the printer circuit board is used to illustrate the application of the proposed algorithm shown in Figure 2. Assume that the followings are given:

component lead diameter(LDDIA) is somewhere in the interval[0.03, 0.05],
lead extension(LDEXT) is somewhere in the interval[0.02, 0.06],
maximum component length(COMPL) is somewhere in the interval[0.5, 0.7].

If conditional constraint is not allowed to be propagated as other interval-based approached, then the final resulting interval will be the same as the above given intervals. However, a group of conditional constraints, C1, C2, and C3 has a *sufficient* antecedent space because every value in the interval [0.03, 0.05] for component lead diameter(LDDIA) satisfies one of three antecedents. Therefore the algorithm shown in Figure 2 can be applied to the group of conditional constraints. This results in setting of minimum bend radius(MNBER) to [0.045, 0.1]. Then, since all variables except one variable in C4 have values, C4 can be propagated. This results in setting of calculated lead spacing(CALDS) to [0.66, 1.07]. Now a group of conditional constraints, C5 through C9, has a *sufficient* antecedent space. Therefore the algorithm shown in Figure 2 can be applied to the group of conditional constraints. This results in setting of standard lead spacing (STDLS) to [0.7, 1.1]. The final resulting intervals are summarized in Table 2.

6. Conclusion

In this paper we have discussed propagation of conditional constraints in interval-based approaches. Literature review has shown that there has been little research that addresses conditional constraints in interval-based approaches. Two algorithms have been proposed: 1) a propagation algorithm for an interval-based conditional constraint, which is similar to one for an exact-value conditional constraint; 2) a propagation algorithm for interval-based conditional constraints which satisfy some conditions. The former algorithm can be applied to any conditional constraint. However, with the former algorithm, conditional constraints are usually categorized into the case that they cannot be propagated. After investigating two methods, whose concepts have been well known, in which such conditional constraints can be propagated, we have proposed a new algorithm under a certain condition so that such conditional constraints can be propagated. An example has shown that applications of the latter algorithm results in smaller

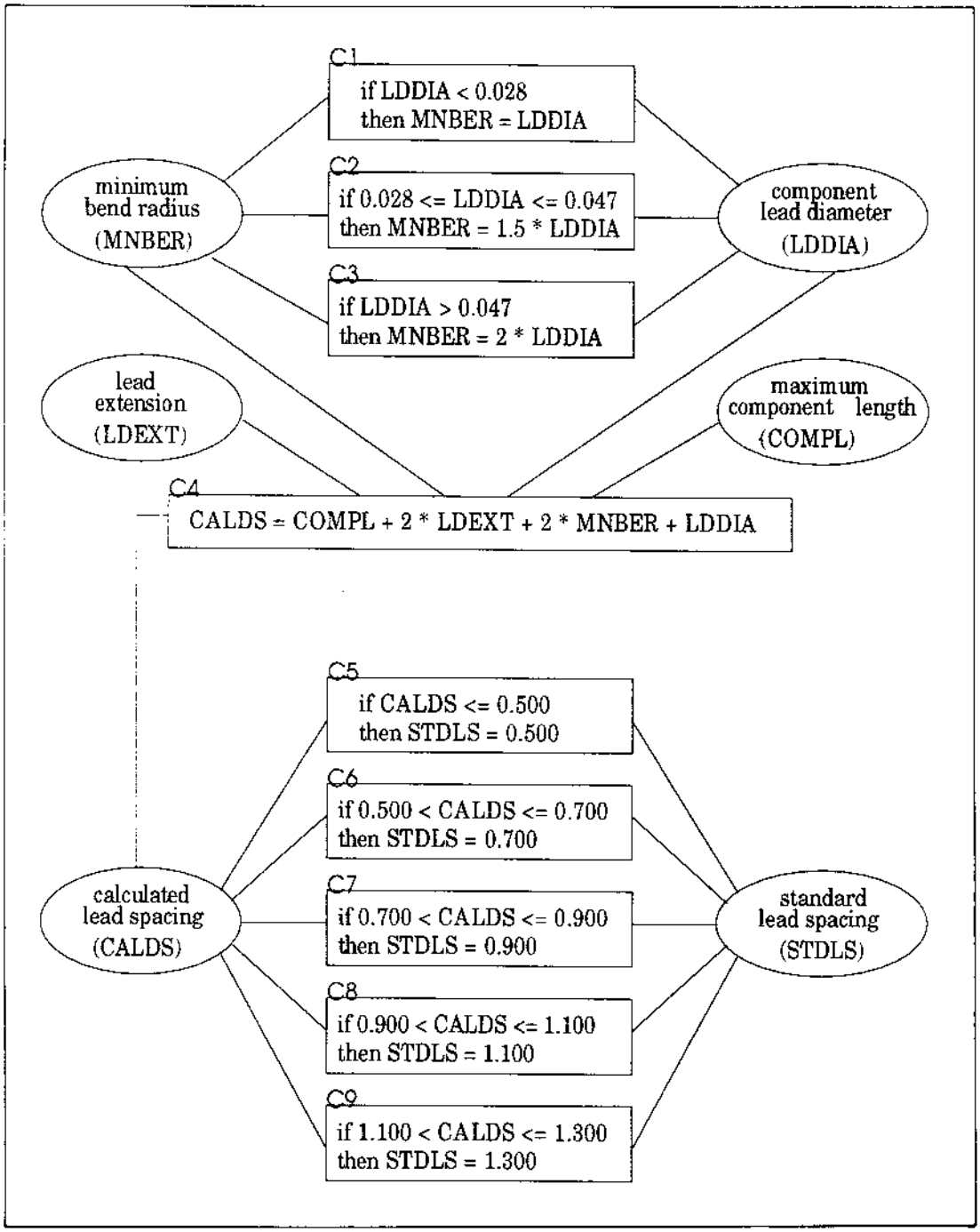

Figure 3. Constraint Networks Determining Lead Access Hole Spacing

Table 2. Final Resulting Intervals

variable	input interval	result when conditional constrains cannot be propagated	result when grouped conditional constrains can be propagated
LDDIA	[0.03, 0.05]	[0.03, 0.05]	[0.03, 0.05]
LDEXT	[0.02, 0.06]	[0.02, 0.06]	[0.02, 0.06]
COMPL	[0.5, 0.7]	[0.5, 0.7]	[0.5, 0.7]
MNBER	unknown	unknown	[0.045, 0.1]
CALDS	unknown	unknown	[0.66, 1.07]
STDLS	unknown	unknown	[0.7, 1.1]

resulting design space comparing to the applications of the former algorithm.

7. References

[1] Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., and Berthier, F., "The Constraint Logic Programming Language CHIP," *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, 1988.

[2] Dincbas, M., Simonis, H., and Van Hentenryck, P., "Solving a Cutting-Stock Problem in Constraint Logic Programming Language,"*Proceedings of the Fifth International Conference and Symposium on Logic Programming*, (ed.) R.A. Kowalski and K.A. Bowen, 1988.

[3] Van Hentenryck, P., *Constraint Satisfaction in Logic Programming*, The MIT Press, Cambridge, Massachusetts, 1989.

[4] Lauriere, Jean-Louis, "A Language and a Program for Stating and Solving Combinatorial Problems," *Artificial Intelligence*, Vol. 10, pp.29-127, 1978.

[5] Hyvönen, E., "Constraint Reasoning Based On Interval Arithmetic," *IJCAI*, pp.1193-1198, 1989.

[6] Ward, A.C., and Seering, W.P., "Quantitative Inference in a Mechanical Design Compiler," *Proceedings of the 1989 ASME Design Theory and Methodology Conference*, 1989.

[7] Habib, W., and Ward, A.C., "Proving the Label Interval Calculus for Inferences on Catalog," *Design Theory and Methodology-DTM'90*, (ed.) J. Rinderle, pp.63-68, 1990.

[8] Ward, A.C., A Theory of Quantitative Inference Applied to a Mechanical Design Compiler, *Ph.D. Dissertation*, MIT, 1990.

[9] Ward, A.C., Lozano-Perez, T., and Seering, W.P., "Extending the constraint propagation in intervals," *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, 4(1), pp.47-54, 1990.

[10] Navinchandra, D., and Rinderle, J.R., "Interval Approaches for Concurrent Evaluation of Design Constraints," *Concurrent Product and Process Design,* (ed.) N.H. Chao, and S.C-Y. Lu, pp. 101-108, 1989.

[11] Rinderle, J.R., and Krishnan, V., "Constraint Reasoning in Concurrent Design," Design Theory and Methodology-DTM '90, (ed.) J. Rinderle, ASME, New York, pp.53-62, 1990.

[12] Lu. S.C-Y., and Wilhelm, R.G., "Automating Tolerance Synthesis: A Framework and Tools," *Journal of Manufacturing Systems*, 10(4), pp.279-296, 1991.

[13] Sussman, G.J., and Steele, G.L., "CONSTRAINTS-A Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence,* Vol. 14, pp.1-39, 1980.

[14] Dubois, D., and Prade, H., "Conditioning, Non-Monotonic Logic and Standard Uncertainty Models," *Conditional Logic in Expert Systems*, (eds.) I.R. Goodman, M.M. Gupta, H.T. Nguyen and G.S. Rogers, Elsevier Science Publishers B.V., pp.115-158, 1991.

[15] Kim, K., An Interval-based Approach for Concurrent Engineering Under Imprecision, *Ph.D. Dissertation*, Dept. Industrial Engineering, North Carolina State University, 1993.