Interval Approaches For Concurrent Evaluation of Design Constraints

by

D. Navinchandra, J. R. Rinderle

EDRC 24-19-90 C , J

INTERVAL APPROACHES FOR CONCURRENT EVALUATION OF DESIGN CONSTRAINTS

D. Navinchandra
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

J. R. Rfnderte
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, Pennsylvania

Abstract

By concurrent design we mean, in pan, concurrent consideration of a broad range of life-cycle constraints concerning, for example manufacturing and maintenance. The multitude of constraints arising from these considerations make it difficult to identify satisfactory designs. An alternative to explicitly considering all constraints is to determine which of the constraints are relevant, redundant or inconsistent and to consider only those which impact design decisions.

The proposed approach is based on two simple ideas: (1) Constraints provide a uniform representation for a variety of life-cycle* concerns, and (2) Interval methods applied to constraints can be used to identify critical constraints, eliminate redundant constraints and to narrow the space of design alternatives.

The application of the *necessary* and *sufficient* intervals of constraints and constraint propagation techniques are used to classify constraints in this way and to focus design activity.

Introduction: Concurrent Design

The practice of design is frequently sequential in nature. In the design of a jet engine turbine disk, for example, the aerodynamic shape of the blade might first be determined, later modified to satisfy structural constraints, and then modified to satisfy manufacturability and maintenance considerations.

It is not surprising that such a situation exists, since there are few individuals capable of bringing this vast range of considerations to bear during design. However, the fact that manufacturing and maintenance considerations are introduced only on an ad hoc basis during preliminary design gives rise to fundamental design deflciencies. It is the purpose of concurrent engineering design to include a broad range of functional and life-cycle concerns during preliminary design phases. While it is possible to obtain an appearance of concurrence by rapidly iterating through the basic sequential design process, we seek a greater degree of concurrency by attempting to identify critical life cycle concerns early and to use those concerns to direct design decisions.

Representing Life-Cycle Concerns

Life-cycle concerns impose required relationships among features of the design to effect functionality, manufacturability. reliability. and servicibilty. In the context of engineering design, these required relationships can be thought of as constraints among design features. Constraints may embody a design objective (eg. weight), a physical law (e.g. $F = ma$), geometric compatibility (e.g. mating of pans), production requirements (e.g. no blind holes), or any other design requirement We express constraints as algebraic relations among feature parameters (eg. hole diameter, wall thickness, stress level). Collectively, the constraints define what will be an acceptable design. Constraint based representations provide a uniform representation for a variety of life-cycle design considerations including function. geometry and production. As the designer makes decisions, the originally acceptable intervals on the parameters are refined to smaller intervals or to specifIc values. As changes occur, the intervals are propagated through the constraints to other pans of the design until quiescence is obtained or until violations are encountered. Because there is a single, uniform representation for all constraints there is no differentiation between functional. geometrical, manufacturing, and other, so called, life-cycle

[1]Acknowledgments - This wort has been sponsored, *in* part, by Defense Advanced Research Projects Agency (DARPA). under contract No. MDA972-88-C-0047 as pan of the DARPA Initiative in Concurrent Engineering (DICE). The authors also acknowledge the support of the Engineering Design Research Center at Carnegie Mellon University (NSF Grant CDR-85-22616).

constraints. Methods used to refine the design by processing constraints are applied uniformly to all life-cycle constraints. Using this approach makes it equally likely that violations are detected among functional constraints and those which have traditionally been considered "down-stream" of the design activity. It is for this very reason that our approach achieves concurrency.

Although constraints are a general mechanism to represent design considerations, it is not possible to identify all design constraints at the time the design problem is first proposed. This is because the set of relevant constraints depends on the design context. If the geometry of the designed artifact is such that casting is an appropriate manufacturing method, then casting are required. Alternatively a set of machining constraints is necessary if the part is to be machined. Similarly, there are constraints that are dependent on material, assembly methods, and a host of other considerations. The relevant constraints depend on the current design features. The features themselves may be completely defined aspects of the design such as a fully dimensioned ventilation hole or may be partially specified features. Because constraints are required relationships among feature parameters they may be retracted, augmented or refined as the design evolves. The design can be thought of as being complete when the set of constraints stabilize and when all the constraints have been satisfied. If we assume for the remainder of this article that constraints can be expressed as algebraic relationships among feature parameters, then we can say that a design is complete when all variables have been assigned values and when all the constraints have been satisfied simultaneously.

Automation in Constraint Based Design

The design of certain small mechanical components can often be effectively accomplished by executing some design procedure. Automating such procedures is a common approach to automating the design, however, as the complexity of the component itself, or the scope of considerations increase, it is difficult to identify practical design procedures.

An alternative to coding comprehensive design procedures is to identify isolated procedures and to determine sequences for executing the procedures until all design variables are established. Certain rule based design systems and some *constraint propagation* systems follow this paradigm. Each rule or constraint can be thought of as a procedure indicating how some specific design decision or variable can be determined once other design parameters are known. The rules, or constraints, can then be executed, in sequence, until the design is complete. This is an effective method for certain classes of designs, most notably when the degree of component interaction is small and when die design is fundamentally serial in nature. It is *not* effective when the description is fundamentally *what needs to be achieved* rather than *means for achieving it.* Such a descriptive (rather than prescriptive) approach to design specification is advantageous because the designer does not need to consider how a specific constraint will be satisfied, thereby facilitating the inclusion of constraints which cannot readily be interpreted as procedures. Furthermore, descriptive specification does not impose a causality to the constraints. For example, if a simple disk was constrained to weigh a specific amount, the constraint itself could be interpreted as a procedure to determine diameter, or thickness, or density given the other two. In this way the constraints imposed on a design serve as both a specification of the design and as a set of procedures for determining design parameters. The design task, in affect, is reduced to that of satisfying the given constraints.

Constraint satisfaction, however, is not easy. Design constraints are usually numerous, complex and highly non-linear. Satisfying a large set of arbitrarily complex equality and inequality constraints is, in essence, the non-linear programming problem and, in general, is not solvable. Although the general problem cannot be solved, much can be done to assist the designer. It is possible to provide the designer with insights about critical interactions among features, redundant requirements and inconsistencies. Such information is useful to the designer even if the constraints are ultimately solved numerically'because a purely numerical solution does not facilitate understanding of the design task.

A large body of research exists on solving constraint problems. The SKETCHPAD [Sutherland 83] system was an early effort on solving constraints by propagation and relaxation. Mackworth [Mackworth 77] introduced algorithms for maintaining consistency in a network of constraint relations. The ThingLab research effort [Borning 79] lead to ideas on propagating constraints across part-whole hierarchies of objects. A constraint representation formalism was introduced by Sussman and Steeie [Sussman 80]. Recently, Gosling [Gosling 83] presented a planning technique which, coupled with propagation, helps solve algebraic constraints. Other relevant work on solving sets of algebraic equations has come from Popplestone [Popplestone 80] and Serrano [Serrano 87]. These research efforts provide a core of solution techniques for handling and propagacng variables with exact values. Unfortunately, many if not most of the engineering design constraints are expressed as incquaiiacs. The very nature of constraints is such that they often do not prescribe specific values for design parameters but rather prescribe ranges for the values. The ideas presented in this paper are based on treating design parameters as intervals. The nouon of interval arithmetic was developed by Moore (Moore 66, Moore 79]. The value of interval based methods for design has also been recognized by Ward [Ward 89]. By adopting this approach we are, in essence, treating equalities as inequalities. Instead of equating a variable to a number we assign a range of values (an interval) to the variable. This generalizes the *notion* of equality assignment and adds flexibility to the representation of parameters, making it possible to capture incompleteness and uncertainty in a design.

A design which is not yet complete may have some parameters which have not been assigned exact values and there may be

tome uncertainty about the final design characteristics. Intervals may be used to express upper and lower bounds on parameter values, making it possible to assess some properties of the artifact before exact values are assigned. For example* in a motor design problem **one** might not **know the exact shaft** size, but might be able to estimate the **general range** of values based on prior experience. This information **can** sometimes be used to guide the designer early in the design process. This is similar to the engineers' *back of the envelope* calculations. There are several levels of specificity which may be used to represent a parameter value: exact assignments, ranges, defaults, orders of magnitude, and unbounded intervals. Each of these levels of specificity may be expressed as intervals. At **any** point during a designing process the parameters may have values at any of the above levels of specificity. By representing all levels of specificity as intervals and using a uniform technique for propagating the intervals through the constraints, we are able to evaluate the constraints on the design and provide the designer valuable feedback about potential constraint violations.

Constraint Propagation in Design

By propagating design decisions through constraints it is possible to determine how the various design parameters affect one another. In the process, redundant constraints are identified and eliminated.

Consider, for example, a *DC* motor. The torque (T in-oz) is related to speed (co rad/sec) as shown in Figure 1 and as given by the constraint:

$$T = 100 - \frac{\omega}{18}$$

Assume that the torque must be at least 30 in-oz (.21 N-ra) and must not exceed 75 in-oz (.53 N-m) and that the speed may assume **any** value between 150 and 300 rad/sec. The given interval, [30, 75 in-oz],[2] in conjunction with the motor characteristics imposes upper and lower bounds on speed of 90 and 252 rad/sec as shown in Figure 1. Intersecting this interval with the original interval we obtain a refined interval on speed, [150, 252 rad/sec]. This new interval is propagated through the constraint, once again, to find upper and lower bounds on torque, [30,58.3 rad/sec]. This interval on torque and the corresponding interval on speed indicate that the original specifications requiring torque to be less than 75 in-oz and speed to be less than 300 rad/sec were not necessary. By propagating intervals it was possible to identify redundancies and therefore simplify the design task without making specific commitments about any of the design parameters.

The process of propagating intervals through constraints can be continued through long chains of constraints. The process provides a means for determining bounds on design variables

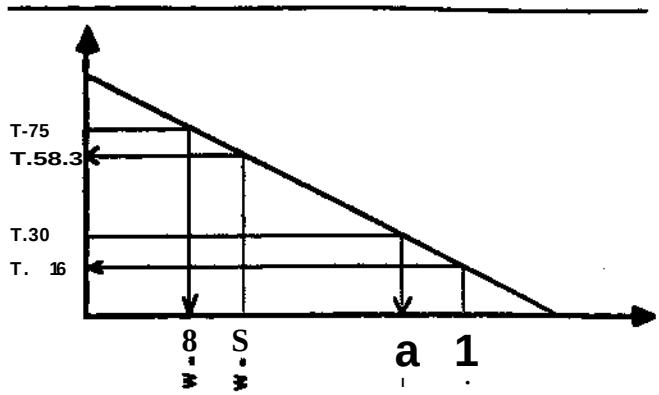

Figure 1: Torque - Speed characteristics of DC motor.

thereby delimiting a feasible space for the final design. Propagation can be done through chains of constraints resulting in a successive narrowing of parameter intervals. Continuing our example, assume the power of the motor (given by *Power* »co7) is required to be less than or equal to 8500 in-oz/second (60 W), that is, in the interval [0, 8500]. Holding the interval on Power and propagating *T* and co through the two constraints yields the interval [222, 252] for the speed and the interval [30, 38.2] for torque. This narrowing requires about 20 iterations. By propagating intervals successively, any variable can affect any other variable as long as there is a chain of constraints connecting them. Propagation can occur in any direction; it is not the case that one variable in a constraint must be selected, a priori, as being dependent while all others are regarded as independent. As constraints are propagated and as intervals narrow, specifications may be found to be inconsistent with other constraints thereby identifying violations[3] and redundancies before specific design decisions are made. Interval propagation makes it possible to gain insight about a design without having to choose specific values for the design parameters.

Working with intervals, in this way, allows one to simultaneously consider a wide range of alternatives and to examine interactions among design parameters before the design is completed. The example also shows how it is possible to narrow design choices without actually committing to any single operating point We believe that the ability to draw important inferences about a design problem, early in the process is important in concurrent engineering. Later in the paper, we show how interval based propagation methods can be used to provide a designer feedback about the likely violation of life-cycle constraints, even when the design is incomplete. We believe this is a viable way of achieving concurrency in design.

[2]Thc SJ. units are not generally repealed in the interval notation to avoid confusion.

[3]That is, when an interval is reduced to a null set.

Interval Approaches

Because intervals are a convenient mechanism to describe aspects of incomplete designs and because intervals can be used to draw inferences about other aspects of the design it is valuable to consider the mathematics of intervals. In the DC motor example, discussed previously, it was possible to determine an interval on torque given an interval on speed simply by evaluating the torque-speed function at the end points of intervals. This was a direct result of the fact that the torque speed function used was strictly monotonic. Indeed this is the case for all functions y »/(xj,X2,...) which are strictly monotonic in each of the arguments, i.e. . $dfldx_t > 0$ *or* $df/dx_t < 0$ for all x_t. When monotonicides are known it is possible to determine, a priori, at which point over the intervals of the arguments that the functions will be maximal and minimal and therefore, the interval on y can be determined simply by evaluating the function at the two critical "comers" of the space deflned by the intervals on the x_t. When the function is known to be monotonic in all of its variables, but the monotoniciry (i.e. increasing or decreasing) is not known a priori, it is possible to determine the interval of the function by evaluating the function at all combinations of argument interval end points, i.e. at *all* corners of the space. Therefore, if the function has n arguments then 2^n function evaluations are required. It is this characteristic of the four principal arithmetic operators that make it possible to define an interval arithmetic.

Interval Arithmetic. Interval arithmetic is used as the basis for evaluating algebraic relations containing interval variables. An interval number is an ordered pair of real numbers, [a, b], with $a \leq b$. Intervals may be treated as sets and can be operated on by set theoretic operators such as intersection, union and subset

Interval arithmetic operators are defined on the upper and lower bounds of the operands. For example, the expression (x^2 +>>) for the intervals x »[1,4] and y «[5, 10] is bounded by the interval [6,26]. This is determined by expanding the square operator and applying the following interval arithmetic formulae [Moore 66]:

$$[a,b] + [c,d] = [a + c, b + d) \quad (1)$$

$$[a_t b] - [c_t d] = [a - d, b - c] \quad (2)$$

$$[a_9 b] \times [c_9 d] = [min(ac, ad_t bc_t bd)_9 \ max(ac, ad, be bd)) \quad (3)$$

$$[a_9 b] \ I \ [c_9 d] = [a,b] \times [1/d, 1/c)^4 \quad (4)$$

The four basic arithmetic operators are monotonic with respect to each of their arguments. Furthermore, the sum and difference operators have known monotonicides and therefore it is possible to determine a priori which combinations of arguments must be evaluated to determine the maximum and minimum interval of the function. The multiplication operator however, may have a strictly increasing or decreasing monotonicity depending on whether or not the arguments are positive or negative. In this case we must evaluate the multiplication function at each combination of interval end points and select the minimum and maximum to determine the interval. Nevertheless, these basic operators compute intervals which are *precisely* the inteival that occurs.

Although the four basic arithmetic operators produce exact intervals, the representation of higher level functions in terms of these basic arithmetic operators introduces some difficulty. Consider the function of y «$(x-2)^2$ over the interval x »[0,10]. The function itself is not monotonic over that interval. Since subdivision into monotonic intervals would require in the general case difficult solution procedures we prefer to express the function in terms of the monotonic arithmetic operators, i.e., y»(r-2)(x-2). In this case the square operator has been replaced with multiplication, however, implicit in the deflnition of interval multiplication is that the two arguments may vary independently. This is clearly not the case for the square function which we are discussing, however, since the independence of the arguments is less restrictive, the result of applying the conventional interval arithmetic will result in an[1] interval on y which is conservative in the sense that the actual interval on the function will lie within the computed interval. In this case the interval computed using interval arithmetic is [-16, 64] which includes the actual interval of [0,64]. The conservative interval calculation destroys the one to one correspondence between intervals on arguments and intervals on functions. This is important in the context of design because it is often necessary to determine what range of arguments will satisfy a range on the function itself. The extent to which the computed interval deviates from the actual interval is critical to the degree to which strong inferences can be made regarding intervals on variables.

There are some specific techniques intended to mediate against the expansion of intervals. One such approach is the centered form of functions based on a fourier expansion of the intervals and is described in [Moore 79]. Other heuristics, for example, to deal with even exponents are also useful. There are several ad-hoc methods to obtain less conservative intervals, often exact intervals. Since the computation of intervals is not the focus of this paper, it will not be discussed at greater length here.

Interval Propagation

In this section we re-examine constraint propagation in terms of interval algebra. Intervals may be propagated in a network of constraints. Propagation refines the intervals on variables in the constraints.

Constraints are evaluated using the basic interval anthroeac operations. For example, let Vj be an interval calculated from the equation V, *op* V_2 * V_3 . Where, *op* is one of the four basic interval arithmetic operators. This operation guarantees that for any value in the intervals V, and V_2 the result of applying *op* will be in V_y In other words, the result is *necessarily* in the interval V_3.

[1]The division operator is not valid when the interval of the divisor includes zero.

After a constraint expression is evaluated the new interval is propagated Fdr example, when t new interval is determined for a variable, that variable's current interval is updated by intersecting it with the calculated necessary interval If the intersection is null, then the original interval or die constraint is said to be inconsistent This kind of propagation can be carried through complex equations using the interval arithmetic operators. The process guarantees the result will necessarily be in the calculated interval

In the DC-motor example, we have the equation *Power m* a>7\ Assume we know the interval on Power [8000, 25000] (inch-oz/scc) and Torque [30, 75] (inch-oz). We seek an interval on co such that, for any values of power and torque (within their intervals) the speed, co, falls within the interval. In other words, we seek an interval in which CD has to *necessarily* be in. As shown in Figure 2, the speed must fall between *a* and *b*. The interval may be computed using the basic interval arithmetic to evaluate the constraint, expressed terms of the variable in question: o « ^ « ^ t t T "[106,8331 •

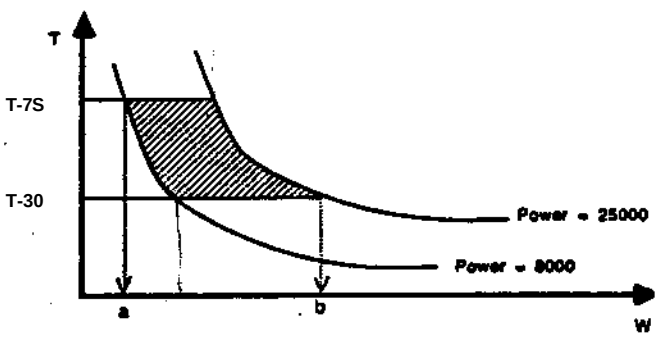

Figure 2: Calculating the necessary interval

Necessary and Sufficient Intervals

The faa that a variable falls within a necessary interval does not guarantee that all constraints can be satisfied, i.e. *necessary* doe* not imply *sufficient.* Consider for example the situation depicted in Figure 2. Although o>[106, 833 rad/sec] is necessary, an arbitrary value in this interval is not sufficient to satisfy the power requirement for arbitrary values of allowable torque since the rectangle of valid torques and speeds extends beyond the bounding power curves. Even when we know that both torque and speed fall within the necessary intervals it is still necessary to check that the power constraint is satisfied. There may, however, be an interval on speed which guarantees that the power requirement will be satisfied whenever torque requirements are met. We say that such an interval is *sufficient* to satisfy the constraint For example, the interval on speed *sufficient* to satisfy the power requirement is shown in Figure 3. For any value of torque and speed in their respective intervals, the power will always be between 8000 and 25000 in-oz/sec.

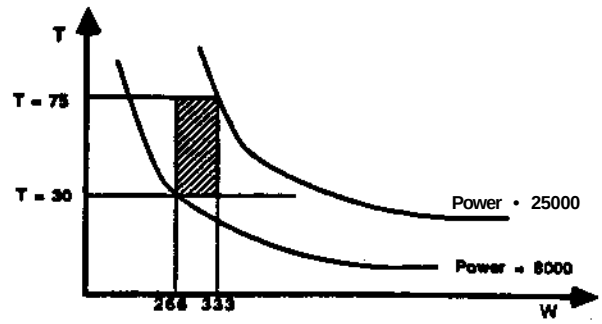

Figure 3: Sufficiency based propagation.

The concept of necessary and sufficient intervals can be very useful to the designer. If two constraints each have associated with them a necessary interval on the same argument and those necessary intervals do not overlap it is not possible to simultaneously satisfy both constraints. The ability to identify a constraint contradiction of this son early in the design cycle makes it possible for the designer to determine appropriate relaxations of these constraints.

The interval of some variable, sufficient to satisfy a constraint insures that a constraint will be satisfied whenever all of the other variables fall within their necessary intervals. Therefore, if the necessary interval of one constraint falls completely within the sufficient interval of a second constraint, then that second constraint will be unconditionally satisfied whenever the first is and therefore the second constraint does not need to be considered explicitly. Identifying a redundant constraint of this sort is similarly useful to the design since only those constraints which are truly binding need be considered.

The relation between the necessary and sufficient intervals can be made more clear with a simple example. Consider the torque-speed characteristics shown in Figure 4 which is typical of an AC induction motor. We are interested in determining at what speed the motor will run given that the operating torque is in the nnge $[T_{min} > T_{max})$. From the figure we can see that when the speed falls in the interval [co,, coj or [0)3,©4] then the torque will fall within the required interval. Either one of the intervals is sufficient to guarantee that the torque requirements will be met However, because this is indeed two intervals rather than one it may be more convenient to deal with an interval representing what is necessary to satisfy that constraint rather than what is sufficient Reading directly from the graph, the necessary interval is [CO], 0)4].

The necessary and sufficient intervals are of course not unique.

- Any sub-interval of a sufficient interval is also a sufficient interval.
- A super-interval of any necessary interval is also a necessary interval.
- The union of all sufficient intervals are contained in any necessary interval.

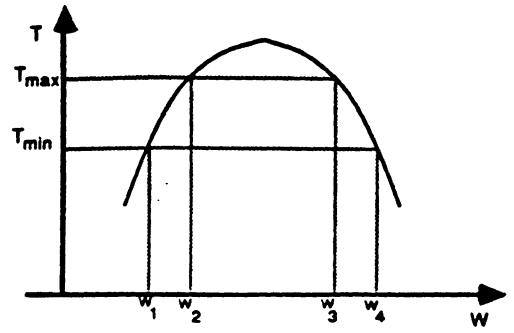

Figure 4: AC-Motor Characteristic

Calculation using the Sufficiency Condition

In this section we present a method to evaluate the intervals on a variable based on a sufficient condition.

Our goal is as follows: Given the constraint V_1 *op* V_2 = V_3, to determine V_2 such that for any value in given V_1 and V_3,the application of op yields a result in V_3. In other words, what should V_2 be so that for any value in V_1 , V_1 *op* V_2 lies in the interval V_3.

To obtain the unknown interval V_2, we express the relation V_1 *op* V_2 = V_3 in terms of the interval we want to be within : in this case V_3. Assume V_2 = [*v2l*, *v2u*], where *v2l* and *v2u* are the values we seek. Now consider the relation V_1 *op* V_2 = V_3 . The left hand side can be evaluated in terms of the unknowns *v2l* and *v2u* by applying the interval operators. These two unknowns can be found by solving the interval equation, as demonstrated by the following example : Consider the D.C motor case in which Power is desired to be in the interval [8000, 25000] and the interval on Torque is given to be [30, 75]. We are to find an interval on ω such that *Power* = ω *T* is satisfied for all ω and *T*. Following the above procedure: Let ω = [ω*l*, ω*u*]. Applying the equation *Power* = ω*T*; substitute the intervals for Power, Torque and ω we get [8000, 25000] = [ωl, ωu] [30, 75] from which it follows that [8000, 25000] = [30 ωl,75 ωu], since torque and speed are known to be positive definite. Equating the upper and lower limits of the interval equation, 8000= ωl 30 gives ωl= 266 and 25000 = ωu 75 gives ωu= 333 . These limits on speed give the sufficient interval.

Now consider the dual case, that of finding an interval on speed sufficient to satisfy the torque requirement of [30,75] given that power falls within the stated interval of [8000, 25000]. Now, V_3 is Torque and V_1 is Power. Expressing the equation in terms of Torque, Torque = Power/ω; we get [30, 75] = [8000, 25000]/ [ωl, ωu], from where it follows that 30 = 8000/ωu and 75 = 25000/ωl. The limits on speed are hence: ωu = 266 and ωl = 333 , which is a nonsense interval and so for the given intervals on Torque and Power we cannot find an interval for w such that for all values of Power, we are within the interval of Torque. There is no interval on speed sufficient to guarantee that there is some valid torque for any power in the stated interval.

This example not only demonstrates the simple methodology to evaluate sufficient intervals but also illustates the asymmetric nature of sufficiency intervals and their conditional existence. Conditions for existence of sufficient intervals and refinements, resolutions and retractions needed for their existence are topics of current research.

Sufficiency on Ranges. We now introduce a new kind of variable called a range variable. If a constraint has a range variable, it means that the constraint has to satisfiable over *all* values in the interval of the range variable. For example, if we are designing a clutch for the motor, we might set up constraints which include friction as one of the variables. Assume it is desired that the clutch operate under both wet and dry conditions. This means that the coefficient of friction is liable to change. It follows that the design should be such that it satisfies the constraints for all values of friction between its wet and dry limits. In this context, friction is a range variable which has upper and lower bounds [Gagne 53, Hindhede, Et.al. 83]. A constraint containing a range variable should be satisfied for *all* values of the range variable between its upper and lower bounds, as opposed to just *any* value as in the case for interval variables. The notion of range was identified by Ward [Ward 89], which he calls the *Sufficient-points* criterion. Range variables can be handled by using the Sufficiency criterion. For example, in Figure 3 the interval on ω satisfies the constraints on Power for all values of Torque in the range <30, 75> .

The propagation techniques presented in this section showed how it is possible to evaluate constraints even if many variables are underspecified and the corresponding design is incomplete or preliminary. The design example presented in the next section of the paper shows how constraints relating to life-cycle concerns such as manufacturing can be considered early in the design process.

Design Example: Two Bar Truss

Consider the design of a simple two bar truss designed to carry a vertical load as shown in Figure 5. The truss is to be made by pinning together two solid round bars of aluminum at the base and at the apex. The truss must span 20 ft. (6.1 m) at the base and must hold the load at least 8 ft. (2.4 m) above the base. Stresses in the truss must not exceed the yield point of the aluminum, 35,000 psi (240 MPa), nor may they exceed the critical buckling stress. Furthermore, the truss must not deflect more than one-quarter of an inch under the 25,000 pound (111,000 N) load.

Also consider some life-cycle constraints on delivery and assembly. To accommodate delivery on a truck assume that the round bars should not exceed 26 ft. (7.9 m) in length and to allow for assembly by an individual the pieces must not weigh more than 50 pounds (23 kg.) each.

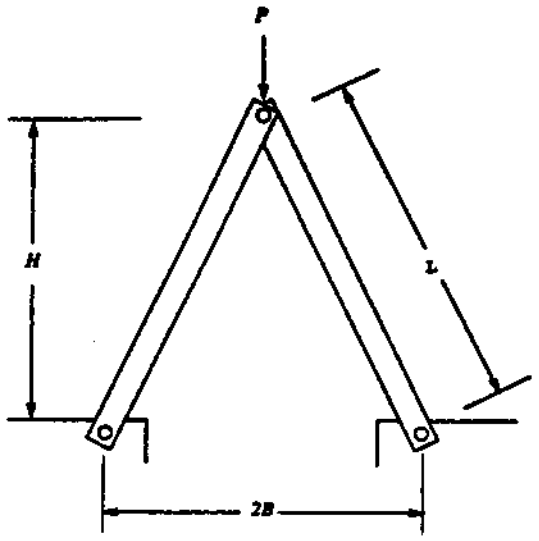

Figure 5: Two Bar Truss Configuration

It is possible to express all design considerations in terms of a set of constraints involving intervals. Intervals on stress, deflection, critical buckling stress, a_{te}, height, //, length, I, and weight, *W*, are all given explicitly as:

a	I 0, o^jj		(5)
5	I 0. 5^]		(6)
σ_{bc}	I a. H		(7)
W	I 0, 50 *Ib.]*	[0. 23 *kg.]*	(8)
H	I W- -I	[2.4 m. H	(9)
L	I 0, 2#rJ	[0, 7.9 *m]*	(10)

We also have geometric compatibility constraints:

$B^2 + H^2 - L^2$	(11)
sinO • *HIL*	(12)
cosO » *BIL*	(13)
tan6 « *H/B*	(14)

Constraints which relate stress, deflection and critical buckling stress to the geometric parameters are:

$$\sigma = \frac{p}{24\sin(6)} \quad \text{Yielding} \quad (15)$$

$$\delta = \frac{PB}{24£\sin^2 9\cos 9} \quad \text{Deflection} \quad (16)$$

$$\sigma_{bc} = \frac{nEA\ \cos^2(9)}{2B^2} \quad \text{Rucldinff} \quad (17)$$

$$W \text{ « } ALp \quad \text{Weight} \quad (18)$$

The intervals on height and tube length are updated by propagating the given intervals through the geometric constraints. The Pythagorean relationship (Equation 11) between half-span, height and length allows us to conclude that the necessary interval on length is LJ12.8 ft, H as a result of the given interval on height, H[8 ft, —). The intersection of this new interval on length with the initially acceptable length interval narrows the length interval to U12.8 ft, 26 ft]. Propagating this new length interval back through the very same constraint allows us to infer that height must fall in the interval H[8 ft, 24 ft]. Now applying the geometric constraints which relate //, *B, L* and 6 allows us to conclude that 0 must necessarily fall in the interval 8[38.7°, 67.4°].

Let us now examine how the buckling constraint may be used to find an interval for the cross-sectional area of the bar. The critical budding stress must be greater than the actual stress. Because we don't have the actual value of or, we find, by direct substitution that the interval on o^ is *[P/QA* ^in(9)), ••]. Using the interval on 6 it is then possible* to compute the necessary interval on tube cross section. A, [5.5 in^2, H from the buckling constraint This interval is further propagated through the weight constraint to identify an interval on weight, W[84 Ib., ••]. We note immediately that the computed necessary interval on weight does not intersect the allowable interval which is W[0, 50 Ib.]. We have therefore determined that it is not possible to satisfy all of the given constraints. The design problem as posed has no valid solution. We have identified a violation of a life cycle constraint before selecting actual dimensions of the structure. The designer may, at this stage, elect to ignore one or more of the constraints, for example the weight constraint, or may consider some alternative configuration, for example a more complex truss arrangement or a two bar truss formed from tubes rather than from solid bars.

Let us consider a tubular truss and defer consideration of the buckling constraint The stress and deflection constraints can now be used to determine both what is necessary and sufficient on cross-sectional area. A, to satisfy those constraints given the known interval on 6. We find that to satisfy the deflection constraint it is necessary that A[1.58 in^2, -] and sufficient that A [4.62 in^2, H^5- Further, using basic interval arithmetic the necessary interval on A to satisfy the stress constraint given the interval on 6 is A[.39 in^2, H and the sufficient interval is A(.57 in^2, H- Since the interval on A necessary by virtue of the deflection constraint is contained in the interval on A sufficient in light of the yielding constraint we know, a priori, that sansfying the deflection requirement will result in the satisfaction of the yielding requirement In that sense the yielding requirement is redundant and need not be considered explicitly. It is now

[9]Because the tenn *{Sin² 8 CosQ] in* the deflectioo constrain a not roonotonic in the interval 0 (38.7°, *67A%* the intervals are calculated usng special case operators.

possible to select a truss angle, 0, and tube cross sectional area, A, to satisfy the deflection requirement in light of some objective criteria, perhaps weight Having taken that last step we simply determine the diameter and tube thickness necessary to satisfy the area requirement and the buckling constraint

In this example we have shown how it is possible to consider simultaneously both equality and inequality constraints arising from transportation, assembly, performance, geometry, and solid mechanics in an integrated, homogeneous representational framework based on intervals. We have also seen how successive application of the constraints - even the same constraint more than once - can significantly refine the intervals, narrowing the range of alternatives. We have shown that by the simple propagation of intervals it is possible to identify a priori inconsistencies among the constraints, alerting the designer to the need to reformulate the problem specifications or the candidate design configurations. We have also shown how the computation of necessary and sufficient intervals on design parameters can simplify design decision making by identifying certain constraints which are redundant and in so doing simplify both computation and enhance the designer's understanding of the design scenario. In a particular case the elimination of the stress constraint made it possible to identify a noniterative design problem solving strategy.

Planning Constraint Evaluations

In the above example, we saw how constraint propagation can be used to help the designer gain insight into a design problem. The example did not address the issue of planning. We know how to propagate intervals but how do we know which constraint to propagate intervals through? One cannot try to propagate values through all constraints, every time a change occurs. Value and interval propagation in constraint networks has to be properly planned.

We currently rely principally on dependence ordering constraint propagation techniques such as those used by [Wang 73] and [Serrano 87]. These methods rely on bi-partite graph matching techniques to determine which constraint *is* used to evaluate each of the design parameters. We also use a combination of strong component and topological sort algorithms to identify a solution strategy. These methods are being extended to accommodate the particular characteristics of interval approaches where, a parameter interval may be computed by more than one constraint and that each constraint may be used more than once.

Conclusions

Designers can be assisted in determining the relevancy of design constraints by the determination of design parameter intervals which are sufficient to guarantee constraint satisfaction or axe necessary to make possible constraint satisfaction. Redundant constraints can be removed from current consideration and relevant constraints can be used to propagate design information. As a result, the computational complexity of the concurrent design problem is diminished and understanding of critical design constraints is enhanced.

Technical Acknowledgments

The authors would like **to** thank Mr. V. Krishnan (Graduate **Research** Assistant) **for his valuable** comments and ideas about interval methods.

References

[Boming 79]
Boming, A., ThingLab- A Constraint Oriented Simulation Laboratory." Technical report, Xerox Palo Alto Research Center, 1979.

[Gagne53]
Gagne Jr., AJF., Torque Capacity and Design of Cone and Disk Clutches ,N *Product Engineering,* December 1953.

[Gosling 83]
Gosling, J., *Algebraic Constraints,* PhD dissertation. Department of Computer Science, Carnegie-Mellon University, 1983.

[Hindhede.Et.aL 83]
Hindhede, U., J.R. Zimmerman, R.B. Hopkins, R.J. Erisxnaa, W. C. Hull, J.D. Lang, *Machine Design Fundamentals: A practical approach.* John Wiley & Sons, 1983.

[Mackworth 77]
Mackworm A.K., "Consistency in Networks of Relations," *Artificial Intelligence,* VoL 8.1977, pp. 99-118.

[Moore 66]
Moore, RJS., *Interval Analysis.* Prentice-Hall, Inc. 1966.

[Moore 79]
Moore, RJE.. *Methods and Applications of Interval Analysis.* Si am. Philadelphia, 1979.

[Popplestone 80]
Popplestone, R. J., Ambler, A. P. and Bellos, L, "An Interpreter for a Language for Describing Assemblies," *Artificial Intelligence.* VoL 14, No. 1,1980, pp. 79-107.

[Serrano 87]
Serrano, D., *Constraint Management in Conceptual Design.* PhD dissertation, Dept of Mechanical Engineering, M.I.T.. 1987.

[Sussman 80]
Sussman, G. J. and Steele Jr. G. L.. "CONSTRAINTS • A Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence.* Vol. 14,1980. pp. 1-39.

[Sutherland 83]
Sutherland, I.E., "Sketchpad - A Man-Machine Graphical Communication System," Technical report TechRepon #296. MIT Lincoln Lab. Cambridge, Massachusetts, 1983.

[Wang 73]
Wang, R.T.R., *Bandwidth Minimization. Reducibility Decomposition, and Triangularization of Sparse Matrices.* PhD dissertation. Computer and Info. Science Research Center, Ohio State Universtiy. 1973.

[Ward 89]
Ward, A.C.. *A Theory of Quantitative Inference Applied to a Mechanical Design Compiler.* PhD dissertation, M.LT., 1989.