Constraint Reasoning and Planning in Concurrent Design

by

V. Krishnan, D. Navinchandra, P. Rane, J. R. Rinderle

EDRC 24-36-90 c. 2

Constraint Reasoning and Planning in Concurrent Design

V. Krishnan *
D. Navinchandra +
P.Rane*
J. R. Rinderle *

CMU-RI-TR-90-03

Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

28 February 1990

+ **Robotics Institute, CMU**

* **Department Mechanical Engineering, CMU**

Table of Contents

Constraint Reasoning and Planning in Concurrent Design

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 unifonn 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. Regional monotinicity properties are used to identify critical constraints.

A related aspect of concurrent design problems is the large number of complex constraints which have to be satisfied to complete a design task. As it is impossible to guarantee the simultaneous solution of a large set of design constraints, we have investigated algorithms for planning and simplifying such constraint problems.

Chapter 1

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 further modified to satisfy manufacturability and maintenance considerations.

It is not surprising that such a situation exists, since there are few individuals capable of bringing a full range of life-cycle concerns to bear during design. Nevertheless, the fact that manufacturing and maintenance considerations are introduced only on an ad hoc basis during preliminary design gives rise to fundamental design deficiencies. It is the purpose of concurrent engineering design to include % 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.

LI 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 (e.g. weight), a physical law (e.g. *F » ma)*, geometric compatibility (e.g. mating of parts), production requirements (e.g. no blind holes), or any other design requirement We express constraints as algebraic relations among feature parameters (e.g. 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 design considerations including function, geometry. production and disposal Because there is a single, uniform representation for all constraints there is no differentiation between functional, geometrical, manufacturing, and other, so called, life-cycle constraints. Methods used to refine the design by processing constraints are applied uniformly to all life-cycle constraints: All constraints, whether they be behavioral, geometrical or those which have traditionally been considered *down-stream,* have equal impact on design decision making. 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 artifaa is such that casting is an appropriate manufacturing method, then casting constraints are required. Alternatively, a set of machining constraints is necessary if the part is to be machined. Similarly, there are constraints that

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 [Boming 79] lead to ideas on propagating constraints across part-whole hierarchies of objects. A constraint representation formalism was introduced by Sussman and Steele [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 propagating variables with exact values. Unfortunately, many if not most of the engineering design constraints are expressed as inequalities. 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. To accomodate inequality constraints while maintaining uniformity of representation, we choose to represent all feature parameters as interval values. Conventional parameter assignment, e.g. $L=\backslash 5iru_f$ can be expressed as assignment to a narrow interval, e.g. L=[1.495,1.505], explicitely representing the scale of indifference or perhaps manufacturing tolerance. Inequality constraints, e.g. L £ 10/h, may b e expressed as an open interval, e.g. L» [10, H[1]. making explicit jhe (as yet) unbounded range for a parameter value. Interval specification is also convenient for expressing constraints which are left implicit such as the positive value requirement, e.g. L £ 10/h -» L= [0,10]. Relationships among feature parameter values are also conveniently expressed as interval assignments, e.g. L_x £ *2 ->Lj-l?*![0,0]*].

The ideas presented in this paper are based on treating design parameters as intervals. The notion 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]. The interval representation of values generalizes the notion of equality assignment, provides a mechanism to deal with tolerances and adds flexibility, 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 some uncertainty about the final design characteristics. Inteivals may be used to express upper and lower bounds on parameter values, maicing it possible to estimate some properties of the artifact before exact values are assigned. This information can sometimes be used to guide preliminary design, augmenting the *rules of thumb* or *back of the envelope calculations* commonly employed by designers. Furthermore, interval based representation is a convenient framework for implementing and bounding *order of magnitude* analysis and default sizing. In this way many levels of specificity may be used simultaneously at any point in the design process. 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.

[1]This is an open interval and should correctly be written as *ly,* H . Because the only open intervals which we consider are unbounded and because the apparent mismatch in open interval delimiters is often confusing we have chosen to ignore this fine distinction.

1.3 Context and Definitions

Consider a stage of design when the concept and the configurations to meet the design requirements have already been synthesized and studied, resulting in a set of constraints.[2] This set of constraints is referred to as a model. A model in which all variables are restricted to physically realizable values is said to be *bounded*.[3]

The goal is to obtain a satisfactory design that optimally satisfies the design objective. The following definitions are in order.

Let x be a vector = $(x_1, x_2, x_3, \ldots x_{k-1}, x_k, x_{k+1}, \ldots x_n)$ where $x_1, x_2, \ldots x_n$ are the design variables.
A function $f(x_1, x_2, \ldots x_n)$ is monotonically increasing with respect to x_i, if an increase in x_i does not result in a decrease in f.

A monotonic variable is one that is monotonic in *every* function in the model.

An *active* constraint is one whose presence influences the solution of the model. An active constraint is a relevant constraint but need not be tight i.e satisfied as an equality at the design solution.

Constraint 1 *dominates* Constraint 2 if the feasible region of constraint 1 is a subset of the feasible region of constraint 2. Satisfaction of constraint 1 implies satisfaction of constraint 2.

An active constraint satisfied as an equality constraint at the design solution is called a critical constraint. A critical constraint is an active constraint, but an active constraint need not be critical.[4]

The following example clarifies these definitions.

---

[2]By constraints, we mean, a required relationship among design objectives and variables. We limit our discussion to algebraic constraints.

[3]In particular, design variables should not reach the values 0 or ∞.

[4]Not all tight constraints are critical. Consider Minimizing $f=(x-5)^2$, subject to $x^2=25$; The minimum is at 5 but is not changed when the constraint is removed. So the equality constraint is not active and therefore not critical.

objective should be bounded by at least one active constrainL[5]

Monotonidty Principle2: Every monotonic nonobjective variable in a well-bounded problem is either

1. Irrelevant and can be deleted from the problem together with all constraints in which it occurs, or
2. Relevant and bounded by two critical constraints,[6] one from above and one from below.

The utility of the Monotonidty Prindples is in proving criticality and irrelevancy of constraints. This can result in the deletion of constraints and reduction in size and complexity of the model if the variables are globally monotonic. The Monotonidty Prindples are used to solve the hydraulic cylinder problem given below. [Papalambros and Wilde 88].

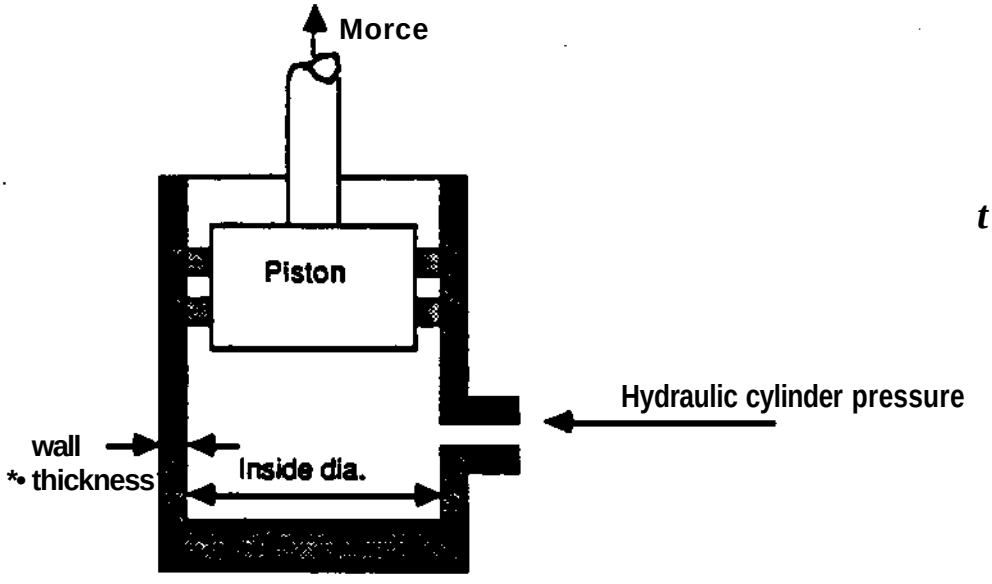

Figure 1-2: Hydraulic Cylinder

Notations: d_o » outside dia.; d_i » inside dia.; s » hoop stress; t =* thickness;

Goal: To design the hydraulic cylinder so as to meet the *following functional specifications*

- F=s Load Handling Capadty 2 22 pound wt(10 kgs)
- *P*= Pressure £ 3.5 psi.

Minimize $d_o = d_x$ 4- *It* , subject to the Constraint Set

1.-F+22S0

[5] If a set of constraints bind the monotonic objective variable, the dominant constraint in this set is a critical constraint.

[6] [Papalambros and Wilde 88] use the term "active" to mean what we are calling "critical".

Since </o»<f,-+2r we have:

$$\frac{s}{p} = \frac{(.dj+2t)^2+d_i^2}{(d_i+2t)^2-d_i^2}$$

If we use Al aUoy A96061 with S^OCXXpsi) and p=4000 (psi) then *thes£S*$_{yp}$ yields,

$$P \quad P$$

and therefore

$$\wedge \wedge \frac{\{d_{\&}2t?-+d_f\%}{(d_i+2t)^2-d_i^2}$$

which becomes $4t? + 4d_it-2d_i^{2}'Z0.Q$. This resultant constraint is not globally monotonic with respea to

Reasoning using Monotonicity Principles will not work because d_i is not a globally monotonic variable.

Although the newly introduced constraint was not globally monotonic it still is monotonic in trie regions $t < d_i$ and t £ d_t. If t £ d_t is unreasonable in the domain of application, then the solution can be obtained by solving the problem in one region. Else, the problem is solved separately in the two regions, and the solution assembled to obtain the global solution. Most design objectives are too complicated to be globally monotonic, but do vary monotonically over regions. Similarly in real design problems, difTerent constraints may become active and dominant in different regions; hence great leverage can be obtained by exploring regional information. We need means for representing, abstracting and manipulating regional information. The need is met by the application of Interval methods.

1.5 Interval Methods

Interval Methods provide a convenient framework to characterize regional properties of objectives and constraints. An interval is a set [a, *b*] such that all real numbers between *a* and *b* are included in the set Intervals can be operated on by set theoretic operators such as intersection, union and subset An interval of a function provides upper and lower bounds for the range of the function, when its arguments span an interval. For eg. the interval of the function $(x^2 + y)$ for the interval x »[1,4] y =* [5,10] is [6,26]. This implies that all the values taken by the function (x^*+y) for the given range of arguments are above 6 and below 26.

Interval arithmetic is used as the basis for evaluating algebraic relations containing interval variables. yielding interval results. Interval arithmetic operators are defined on the upper and lower bounds of the operands. The interval on $(x^* + y)$ in the above example, was determined by expanding the square operator and applying the following interval arithmetic formulae. [Moore 66]

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

$$[a,b] - [c_9d] = [a - d,\ b - c] \quad (2)$$

$$[a_fb] \times [c,d] = [min(ac,ad,bc_tbd),\ max(ac,ad_fbcbd)] \quad (3)$$

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 dfteni" « 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 our research, it will not be discussed at greater length here.

1.7 Constraint Propagation in Design

Intervals can be effective for representing and reasoning about design parameter values. It is also possible to propagate interval values through a set of constraints and detect potential constraint violations. 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. The intervals of the various parameters are also refined in this process.

Consider, for example, a DC motor. The torque *(T* in-oz) is related to speed (co rad/sec) as shown in Figure 3 and as given by the constraint:

$$T * 100 - jco$$

Assume that the torque must be at least 30 in-oz (.21 N-m) and must not exceed 75 in-oz (.53 N-m) and that the speed may assume any value between 150 and 400 rad/sec. The given interval, [30,75 in-oz],[8] in conjunction with the motor characteristics imposes upper and lower bounds on speed of 125 and 350 rad/sec as shown in Figure 3. Intersecting this interval with the original interval we obtain a refined interval on speed, [150, 350 rad/sec]. This new interval is propagated through the constraint, once again. to find upper and lower bounds on torque, [30,70 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 400 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 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* = coT) is required to be less than or equal to 8500 in-oz/second (60 W), that is, in the

[8]The SX units are not generally repeated in the interval notation to avoid confusion.

1.8 Interval propagation

In this section we delve in more detail on the propagation of intervals through a set of constraints and the evaluation of intervals through necessary and sufficient conditions. Consider the evaluation of intervals using the basic interval arithmetic operations. For example* let V_3 be an interval calculated from the equation V_x *op* V_2 * V3. Where, *op* is one of the four basic interval arithmetic operators. This operation guarantees that for any value in the intervals V_x and V_2 the result of applying *op* will be in V_v In other words, the result is *necessarily* in the interval V_v

After a constraint expression is evaluated the new interval is propagated. For example, when a 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 the 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* * ooT. Assume we know the interval on Power [8000, 25000] (inch-oz/sec) 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 oth$r 6ords, we seek an interval in which co has to *necessarily* be in. As shown in Figure 4 , 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: co = £ £ £ = I80^!?4S001 = [106,833].

1.9 Necessary and Sufficient Intervals

The fact that a variable falls within a necessary interval does not guarantee that all constraints can be satisfied, Le, *necessary* does not imply *sufficient.* Consider for example the situation depicted in Figure 4. Although co{lO6, 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 inteival on speed which guarantees that the power requirement will be satisfied whenever torque requirements are met We say that such an inteival is *sufficient* to satisfy the constraint For example, the interval on speed *sufficient* to satisfy the power requirement is shown in Figure 4. For any value of torque and speed in their respective intervals, the power will always be between 8000 and 25000 in-os/sec.

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 sort 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

1.10 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,the application of op yields a result in specified 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 specified 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 = [v_{2l}, v_{2u}], where v_{2l} and v_{2u} 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 v_{2l} and v_{2u} 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 required to be in the interval [8000, 25000] and the interval on Torque is given to be [30, 75]. Our goal is to find an interval on ω such that *Power* = ωT is satisfied for all ω and T in their respective intervals. Following the above procedure: Let $\omega = [\omega_l, \omega_u]$. Applying the equation *Power* = ωT; substitute the intervals for Power, Torque and ω we get [8000, 25000] = $[\omega_l, \omega_u]$ [30, 75] from which it follows that [8000, 25000] = [30 ω_l,75 ω_u], since torque and speed are known to be positive definite. Equating the upper and lower limits of the interval equation, 8000= ω_l 30 gives ω_l= 266 and 25000 = ω_u 75 gives ω_u= 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]/ $[\omega_l, \omega_u]$, from where it follows that 30 = 8000/ω_u and 75 = 25000/ω_l. The limits on speed are hence: ω_u = 266 and ω_l = 333 , which is a nonsense interval. 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. The existence of a sufficient intervals has an impact on the design decision making. For example, if we are designing a d.c motor, the existence of a sufficient interval means the ability to accomodate any motor in the given torque range or the need to use a particular motor in the given torque range. Conditions for existence of sufficient intervals, resolutions and retractions needed for their existence are topics of current research.

1.11 Interval Criticality, Dominance, Activity

Constraints in design may not be globally monotonic, globally active, dominant or critical, but certainly are regionally. The concepts of constraint criticality, dominance and activity, defined over regions, are therefore, more effective in identifying the critical constraints and pruning the insignificant ones. Interval Methods, with which we represented and manipulated regional information in the previous sections, can again be used to characterize dominant, critical and active constraints regionally.

Interval Dominance Constraint $c_1 < 0$ dominates constraint $c_2 < 0$ in the interval I, if the interval $(c_2 - c_1)$

monotonically decreasing; the new constraint cannot bind d_t from below. The monotonic objective variable is again bound from below by the relationship between force and $\&_{i\%}$ and so ^72(mm)$_t$F»44$_t$000(/hf) p-4000 *(psi).*

The ConstraintC} becomes 4i* + 288*-1036820.0;

$$C2\text{-}c_x = 10365 - 287t - 4t^2;$$

interval! $c_2 - c_x$] for H3t tj

$$-[10365 - 287f_M - 4r_M{}^2{}_f \quad 10365 - 287\,r, - 4t_l^2]$$

which is necessarily > 0 for r_M £ 27mm.

So in the interval [3, 27] constraint 1 dominates constraint 2.

At r s 27(mm), constraint c_1 becomes tight and dominates c_2-

The solution is r=27 (mm), ^ = 72 *(mm) and* d_o = 126 (wn), for this is the smallest value of *t* for which both constraints are satisfied.

Thus, despite the absence of global monotonicities, the design solution was achieved by combining regional information. This suggests that one could use interval methods not only for constraint reasoning but also for optimizing the objectives. In fact interval methods guarantee that the resultant interval obtained is an inclusion, i.e. the result includes and binds all values of the function in this region. Interval Methods based algorithms use this assuredness property of intervals to obtain the global optimum. In the next few sections we will investigate the application of interval based methods to

- Obtaining the globally optimum solution and
- Combining constraint reasoning with global optimization.

1.12 Global Optimization

Global Optimization of a nonlinear, nonconvex objective subject to nonlinear constraints is yet an unsolved problem. There is no *single best* method to accomplish this goal of attaining the global optimum. The problem with most traditional nonlinear programming techniques is that they are local methods. They can get stuck in local valleys, and there is no guarantee that the solution is globally optimum. Under strong assumptions about the function involved, like convexity etc. the solution can be assured to be globally optimum.

Interval methods have been used to solve the global optimization problem [Ratschek and Rokne 88]. The rationale behind these approaches for unconstrained optimization is as follows:

- Use interval methods to represent regional information.
- Exploit the bounds provided by the interval method as a part of a branch and bound search strategy.
- Combine with a subdivisioning procedure, that helps accelerate the search, by yielding tighter bounds.

To solve the constrained optimization problem, these methods subdivide the constrained design space into halves, until they get a part of the space which satisfies all the constraints. Due to the the extreme

L13 Interval Variables Approach

Unlike conventional interval algorithms which keep sub-dividing the design space until all constraints are satisfied, the Interval variables approach simplifies the model regionally by reasoning from a variable point of view. The interval dominancy, criticality and activity conditions, as defined in an earlier section, are used to reason from the variable point of view. The Interval Variable approach, by itself, may not be able to solve all the problems completely. In such cases it may be used as a pre-piocessor that simplifies themodeL

Interval Variables approach

1. Form the adjacency matrix for the model.[12]
2. Consider nonobjective variables first, giving preference to those variables that figure in the least number of constraints.
   - If the variable occurs in only one constraint, and is regionally monotonic in that constraint, eliminate the constraint and the variable.
   - If the variable occurs in several constraints, and is regionally monotonic with respect to each of the constraints, then call these constraints, nonobjective conditionally criticaUexactly one constraint in this group is critical). Apply Interval Dominance conditions to all pairs of constraints in this conditionally critical set to identify the critical constraint While considering the various pairs, dominance relations are propagatedj.e. If A dominates B, and B dominates C, then A dominates C; The pair A,C need not be tested for dominance.
   - If the nonobjective variable is not monotonic, subdivide the design space further, to obtain desired monotonicities.

Consider monotonic objective variables, starting first with variables that occur in the least number of constraints. For each variable,

- Partition design space to obtain desired monotonicities.
- List all the constraints with the desired monotonicities under an objective conditionally critical set
- Test for interval dominancy of constraints.
- Delete the dominated constraints. Apply Monotonicity Principle 1 to obtain the constraint which is regionally critical.

In a larger sense, the interval variable approach is similar to active set strategies, as it tries to solve the problem by finding out the critical constraints. However instead of expecting a few constraints to be critical throughout the design space, the interval variables approach looks for regional criticality. In highly nonlinear situations, such as design constraints and objectives, we believe this results in signifficant benefits.

[12]Adjacency matrix lists the numbers of the constraints and the variables that occur in each of the constraints. It is described m the next chapter.

Stait with the following intervals:-

h m [0.1,1]

''-[1.3]

b = [3, 10]

I»[3.3O]

a = *bt* = [3, 30]

Step 1 Select die only nonobjective variable, *t* , for consideration. The constraints in which it appears are as follows:-

gl m -*at* + 16.8 £ 0.0

g3 = -a + *ht* £ 0.0

g4 m -ti<^{2} + 9.08 £0.0

$5 • -1+0.02776* + (0.094 r^2/*^{3}) £ 0.0 and the set constraints

*21.0, *t* £ 3.0

On gxouping the constraints into those that bind the variable *t* from above and those that bind from below, we obtain

Set A(binds t from above)

g3 = -a + /ir S 0.0

S = -1+0.02776 + (0.094 *file?)* £ 0.0 and

r-3.0 £ 0.0

Set B(constraints providing lower bounds)

gl a -a* + 16.8 £ 0.0

g4 = - a r ^ 9.08 ^0.0

-r+lSO.O

Because *t* is a monotonic nonobjective variable, it has to be bounded by two critical constraints, one from above and one from below. So exactly one constraint from each set must be criticaL

By Interval Dominance we can show that the set constraint r-3.0 £ 0.0 dominates the other two constraints in set A. So it is critical and r = 3.0;

This makes sure that -r+1 2 0.0 is not criticaL Using r=3.0 and the interval dominance condition we can show that

gl = -ar+16.8 £ 0.0 dominates

g4 = -<z*^{2} + 9.08 £0.0 over the specified interval of a.

g4 can be deleted and g2 is critical resulting in a=(16.8/3>=5.6;

h is bounded from below by two constraints. The set constraint *h* £0.1 and

the constraint *g6* = -*h* + 0.125 £0.0;

Obviously g6 dominates the set constraint and so h= 0.125;

Thus the solution in this region of the design space is *h* = 0.125, *a* = 5.6, *t* = 3.0, *b* = 1.9.

Chapter 2

Planning Constraint Solution Strategies

In concurrent design problems we often have large numbers of complex constraints which have to be satisfied to complete a design task. As it is impossible to guarantee the simultaneous solution of a large set of design constraints, we have investigated algorithms for planning and simplifying such constraint problems.

Satisfying a large number of constraints does not imply that all the constraints be solved simultaneously. Often, some parts of the design tend to be more coupled than others. This chapter presents algorithms for finding the coupled constraints and for developing a solution strategy which minimizes simultenaety.

The simplest type of constraint sets are those which do not need any simultaneous solution of constraints. Such constraint sets are said to be *Serially Decomposable*. The constraints can be solved serially, yielding the value of one new variable for each constraint evaluation. We present algorithms to detect serial decomposability and for ordering the solution sequence of such constraint sets. When a constraint set is not serially decomposable, the constraints have to be solved for simultaneously. Instead of trying to solve the entire constraint set simultanecusly, we would like to isolate and identify subsets of the entire constraint set which necessarily have to be solved simultaneously. This chapter presents algorithms for achieving this.

One of the assumptions we make in ordering algebraic constraints is that they are invertible. That is, for any function $F(X)$, one can find the value of any variable x_i in X if the values of all the other variables are known. Not all constraints are explicit and not all constraints are invertible. For example, a Finite Element package, unlike an algebraic relation, takes inputs and produces outputs. One cannot determine the inputs from the outputs. Such constraints have to be handled in a special way. We present an algorithm for ordering constraint sets which contain both reversible and irreversible constraints.

2.16 A Design Example

Before embarking on discussions about graph theory and algorithms, let us examine the major ideas of this chapter with the aid of a simple, but illustrative example. We will be using this example throughout this chapter.

Consider the design of a friction-type disc clutch shown in Figure 2-7. This example is taken from [Hindhede, Et.al. 83]. The problem is to find the size of the clutch plate (D_{out}), and the inner diameter of the lining (Din), to safely transmit 32 HP at 3000 rpm.

There are several design equations relating the known and unknown clutch parameters. We treat these

$$S_{hoop} = \frac{\rho D^2_{out} \omega^2}{4} \qquad (11)$$

The design task is to find values for the unknown variables such that the above constraints are all satisfied simultaneously.

2.16.1 Ordering the constraints

Instead of trying to solve all the above equations simultaneously, one can try to identify a reasonable strategy to solve the equations. The algorithm presented in this paper produces the following strategy:

Step 1: Calculate for $T_{mombial}$ from Equation (9) by expressing the equation in terms of $T_{nomiBai}$ and substituting for power and angular speed.

Step 2: Calculate for $T^{\wedge\wedge}$ from Equation (8) by substituting the just determined value of $^{\wedge}nominal'$

Step 3: Calculate for D_{oyi}, D_{M}, D_g and F_a simultaneously from equations (5), (6), (7) and (lGpi r

Step 4: Calculate for S_{hoop} from Equation (11)

The above strategy shows three aspects of solution planning: (1) Constraints may be treated as procedures, where any variable in the constraint can be determined if the values of all the other variables are known. (2) Some variables can be determined only after other variables are determined. This produces a chain (or ordering) of constraint evaluations, and (3) Variables which depend on one another have to be solved for simultaneously. The actual numerical method used to solve the constraints is outside the scope of this report. It is our aim, to find a viable solution strategy which identifies and isolates only those constraints which have to be solved simultaneously. This is done to avoid treating the entire constraint set simultaneously.

2.17 Planning Algorithm for Serially Decomposable Constraint Sets

Some constraint sets do not need any simultaneous solution of constraints. We call such constraint sets *Serially Decomposable.* This is because the constraints can be solved serially, that is, there is no simultenaety among the constraints. For example, consider a reformulation of the clutch problem:

$$D_e^3 = \frac{6\,T_{design}}{\pi \mu} \qquad (12)$$

$$Power = {}^{\wedge}T_{ncmiikMl} \qquad (13)$$

$$S_{hoop} = \frac{9\ p\ D\}\ co^2}{25} \qquad (14)$$

$$D_{out} = 1.2D, \qquad (15)$$

$$D_{in} = 0.8\,D_{\epsilon} \qquad (16)$$

$$T_{design} = {}^{\wedge}nominal\ *, \qquad ™$$

Step 3. For all the **tows** with only **one** T in it:
a. read off the corresponding column (variable name)
and push it on the stack ORDER
b. remove the row from the matrix
a remove the column with the T in it

Step 4. Go to Step 1.

A problem with using the above algorithm, is that it fails ungracefully when the constraint set is not serially decomposed The algorithm has no way of telling whether a given constraint set is serially decomposable or not, it starts generating an ordering and fails only when it reaches an impasse. It would be better if we could determine, up front, whether a constraint set is orderable or not The next section presents an algorithm aimed at quickly determining whether a constriant set is serially orderable or not. The algorithm works without actually trying to order the constraints, and thus runs very fast

2.18 Special Treatment of Serially Decomposable Constraint Sets

The constraint sets encountered in design practice are usually extremely complex. The number of constraints and the number of variables is large. An interesting property of the constraint sets is that they are usually sparse. Each of the individual equations have a small number of variables. A direct simultaneous solution or optimization of the set of constraints is computationally very complex. It is undesirable as it does not take advantage of the sparseness of the constraint set The sparse set of constraints is broken down in to different groups which can be solved separately and progressively one after another.

A serially decomposable set can be ordered and solved directly without taking recourse to simultaneous solution methods. A constraint set, which is not serially decomposable on the whole, may have parts which are serially decomposable. This property can be used effectively to partition the constraint set into smaller and more managable subsets.

Detection of serially decomposable sets is valuable. At present, the constraint sets are reordered using various algorithms, to partition the system of equations. It would be good to detect before resorting to reordering, whether the constraint set is serially decomposable or not

An analytical method has been developed to test the serial decomposability of a constraint set This method is based on an adjacency matrix representation of the constraint set and a set of boolean properties of the set

Adjacency matrix representation

A constraint set can be represented as a bi-partite graph. The nodes of the graph are the equations and the variables. The edges are between the equations and the variables present in the equations.

The adjacency matrix of the constraint set is matrix fonned by presenting the equations along the rows and variables along the columns. An element of a row is 1 if the variable corresponding to the column is

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n}1 \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & & \cdots & \\ \cdots & & \cdots & \\ \cdots & & \cdots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

where elements ay are boolean.

If A is 2 x 2 then,

det(A)» (a_n AND a^) XOR (a_{12} AND a_{21}).

If Aisnxnthen

det(A) = (z_{l2} AND minor ($z^{\}$)) XOR (a_{12} AND minor (a_{12})) XOR... XOR (a_{1n} AND minor(a_{1n})).

a nary-XOR ((a_n AND minor (a_n)),(a_{12} AND minor (a_{12})),... (a_{1n} AND minor(a ^))).

T. '

The XOR expansion is defined as n-ary XOR; i.e. before evaluating any of the boolean expressions, the whole expression should be expanded. After the complete expansion, the AND operations should be evaluated. Then the n-ary XOR should be applied to the whole expression. If there is *exactly* one '1' in the expression, the result will be 1, else the expression will evaluate to 0.

The determinant can be expanded by any row or column. Consequently, the value of the detenninant does not change on rearrangement of rows and columns.

Criterion for Serial Decomposability

The detenninant of the adjacency matrix indicates the presence of loops.

If A is an adjacency matrix, then

if det(A) s l, the constraints can be serially decomposed,

= 0, the constraints can not be serially decomposed.

Consider the first example. The adjacency matrix on rearrangement:

$$A\,s = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{matrix} (5) \\ (4) \\ (1) \\ (2) \\ (3) \\ (6) \\ (7) \end{matrix}$$

where the columns are :($r_{nominal}$.$r_{de\$ign}$^$_a$X>$_e$i>ouf$^{£>}$in'^{5}hoop)

Expanding the determinant of A by first row.

A serially decomposable matrix has a path through the all the 1 elements, which is a tree. In a matrix, which is not serially decomposable, such a path does not exist. It contains a path which is a graph with loops rather than a tree. The number of these loops and their inter-relations are an important consideration in a solution of the constraint management problem.

The present work does not include non-square constraint sets and directed constraints. It can be extended to cover these cases. The theoretical approach seems to have a potential to give rise to an analytical formulation for optimal partitioning of a constraint set.

2.19 Ordering a Non-Decomposable Constraint Sets

When a constraint set is not serially decomposable, the constraints have to be solved simultaneously. Instead of trying to solve the entire constraint set simultaneously, we would like to isolate and identify subsets of the entire constraint set which necessarily have to be solved simultaneously. This section presents an ordering algorithm which helps identify such subsets.

Let us return to the original clutch example. The original problem, as we noted earlier, is not serially decomposable. This can be seen with the aid of the adjacency matrix representation. Figure 2-9 (a) shows the matrix for the given equations. By carefully reordering the rows and columns, one can find a solution plan which is better than trying to solve all the variables and constraints simultaneously. Figure 2-9 (b) shows the ordered matrix: after calculating $T_{nominal}$, it is possible to calculate T_{design} from equation 8. The next four variables have to solved as a block (simultaneously).

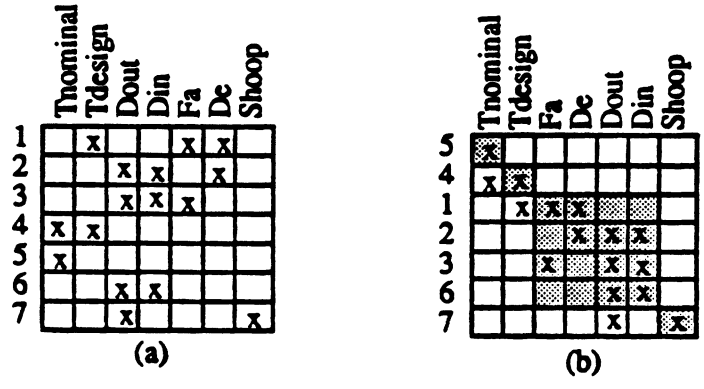

Figure 2-9: Adjacency Matrix Representation of the Ordering Task

2.19.1 Intuitive Explanation

The algorithm consists of two stages: *Matching* and *Component Finding*. The first stage matches variables to equations. This is done because we know that each equation can be used to solve for only one variable. Every variable will be calculated from one constraint. It is for this reason that we start by matching constraints to variables. It is important to try and find as many matchings as possible. For example, in the two equations $F1(x,y)$ and $F2(x)$, there are two variables x and y which have to be matched. The matching problem is shown in Figure 2-10 (a). The variables are listed on the left and the constraints are listed on the right. The lines indicate which variable is involved in which constraint. This

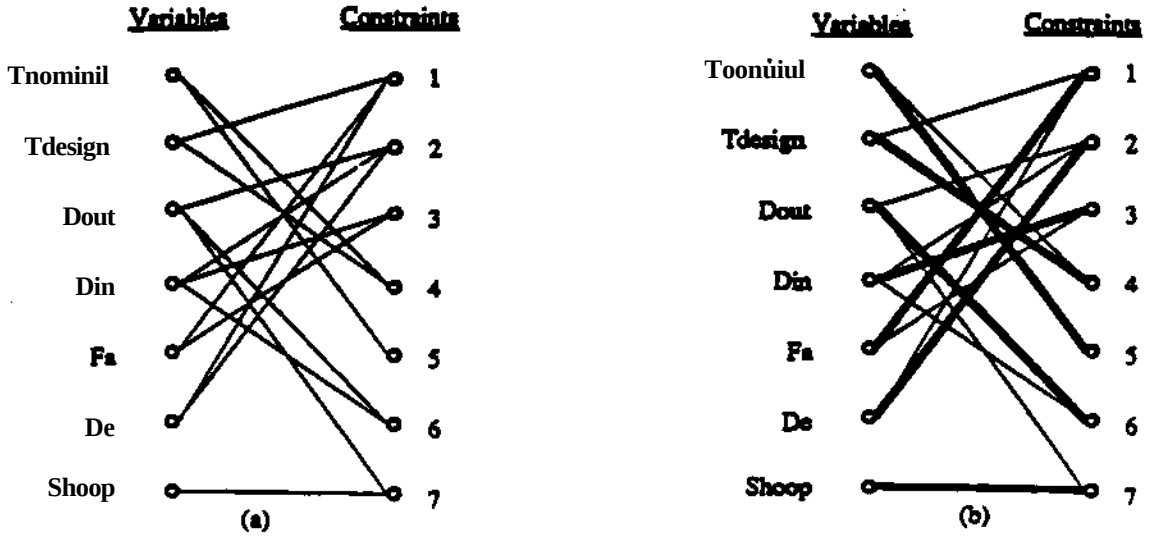

Figure 2-11: Bipartite Matching

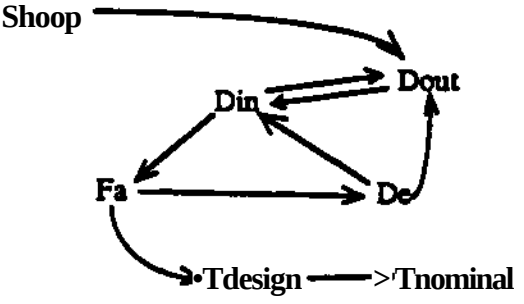

Figure 2-12: Dependency graph among variables

In the digraph, $T_{m)miMi}$ does not depend on any unknown quantity and hence can be immediately determined from the equation it was matched to. Once $T_{nomil\%al}$ is known, T_{duiim} can be calculated. The rest of the variables, however, cannot be chained as the first two. The variables D_{0lg9} $D^{\wedge}$ D_4 and F_{ϵ} are in a cycle of dependencies. That is they depend on one another and have to be solved simultaneously. The variable S_{hoop} is not in the cycle. Cycles are identified using a standard graph theoretic algorithm called the Strong Component Algorithm.

A strong component of a digraph is a maximal set of nodes in which there is a path from any node (variable) in the set to any other node in the set A depth-first search based technique is used to determine strong components efficiently (Aho, Hopcroft & Ullman [V]84]. The strong component algorithm consists of the following steps:

---

1. Perform a depth-first search of the digraph (G) starting at any node *N.* Make a note of all the nodes visited in the list LI. The depth first search procedure is as shown below:

   DFS(G, Current-Node)
   1. Add Current-Node to globally defined list: VISITED
   2. Get the dependents (D) of the Current-Node which are not in VISITED
   3. IF there are no such dependents return NULL
      ELSE each dependent (d) do DFS(G, d)

2.19*2 The Complete Planning Algorithm

In summary, the steps arc as follows.

Step L As the evaluation of a constraint yields the value of only one variable at a time, we have to first decide which variables will be ralnilatrri from which constraint As we would like to evaluate as many variables as possible, the matching of variables to constraints is done using a bi-partite graph matching technique.

Step 2. A directed graph of dependencies among variables is generated. For example, if one is going to calculate for variable *a* from the constraint^ *h c)*$_9$ then *a* is said to depend on *b* and c.

Step 3. Cycles in the above di-graph indicate simultaneity among variables. Using an algorithm to find strongly components in the di-graph, the smallest cycles are found and isolated.

Step 4. After all cycles are isolated, the rest of the di-graph becomes a tree. A reverse topological sort yields the steps which can be taken to find the values of the variables. The algorithm (RTS) is as follows:

RTS(Tree)

Step A. Initialize a stack called ORDER y '

Step B. If there are no nodes in the Tree, return ORDER

Step C Find all nodes that have no children (depend on no other variable)
If there are no such nodes, then the input graph is not a tree.

Step D. For each node found in Step C, do the following:
i. push the node onto the stack ORDER
ii. remove the node from the digraph

Step E. Go to Step B.

The algorithm is based on three standard graph theoretic algorithms: Bipartite-matching, Strong Components and Reverse Topological Sort. These algorithms are all described in introductory graph theory texts. Please see [Aho, Hopcroft & UUman '83].

2.20 Breaking the Strong Components

Strong components can sometimes be broken or simplified by picking the value of one of the variables in the strong component The process is analogous to untying knots in a string. Untying a large knot might either reveal smaller knots or might eliminate the knot altogether. The idea behind breaking a strong component is to perform a single-degree-of-freedom search on one variable instead of solving all the variables simultaneously. Consider, for example, a coupled constraint set with *n* variables and *n* constraints. Assume that all simultenaety is eliminated if one variable *x* is guessed. After guessing *x* the values of all the remaining *n* -1 unknowns can be easily determined from *n* -1 constraints. The remaining constraint can be used to calculate a new value for x The new value is compared to the guessed value. If there is some error, a new value for *x* is guessed and the process is repeated. Iterations are carried out untill the error is within acceptable limits.

is added in the die second stage. The heuristic will wrongly consider F_a andD, as possible candidates.

Most-Dependents Heuristic

We have found that the best variables to pick are often the ones which are most "coupled." Simply put, **we son the nodes (variables) in the strong component by the total number** of **dependent** variables in the strong component. In **die clutch** digraph, applying this heuristic **places D_M and D_{oyi} as** the **best** choices **(Figure** 2-**15). The numbers in the figure indicate the number of dependents at** each node.

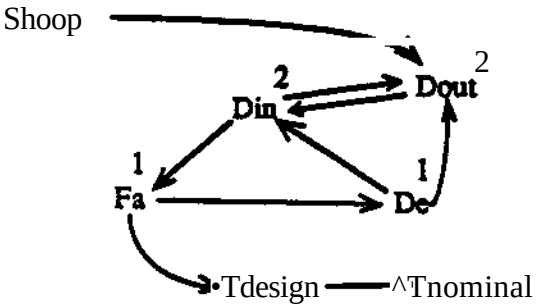

Figure 2-15: Counting the number of dependents at each node.

The number of dependents of a variables is a heuristic measure of how critical it is to know the value of a variable, before other parts of the design can be determined. It is this property that, we believe, makes the heuristic work. We have conducted experiments to verify this hypothesis.

2.20.1 Experiments with the Most-Dependent Heuristic

We **are** currently conductive extensive experiments to assess the efficacy of heuristic approaches to selecting variables which can break a given strong component **The** experiments are being run on hundereds of randomly generated constraint sets.

Preliminary results show, that if one uses the Most-Dependents heuristic it takes (on the average) two tries to find a variable which breaks the strong component If no heuristic is used it takes about five or six tries before the appropriate variable is found. These experiments yielded results only for small components. Larger components were rarely eliminated by choosing a single variable. Detailed experimental results will be reported in a later version of this document

2.21 Handling Uni-Directional Constraints

One of the assumptions made in the above ordering algorithm is that all constraints are invertible. That is. for any function $F(X\backslash$ one can find the value of any variable x_i in X if the values of all the other variables are known. Not all constraints are explicit and not all constraints are invemible. For example, a Finite Element Method (FEM) based tool takes some inputs and produces outputs. One cannot determine the inputs from the output: The constraint is a Black-Box. Algebraic constraints can also be implicit For example, it may not be possible to calculate for all the variables in a very complex transcendental function. For such constraints only a subset of the involved variables can be solved for, thereby making the rest of the variables serve merely as inputs.

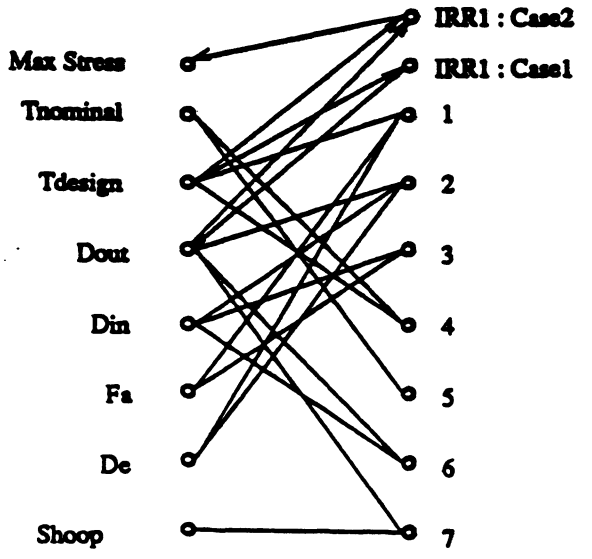

Figure 2-16: A Directed Bipartite Graph representation

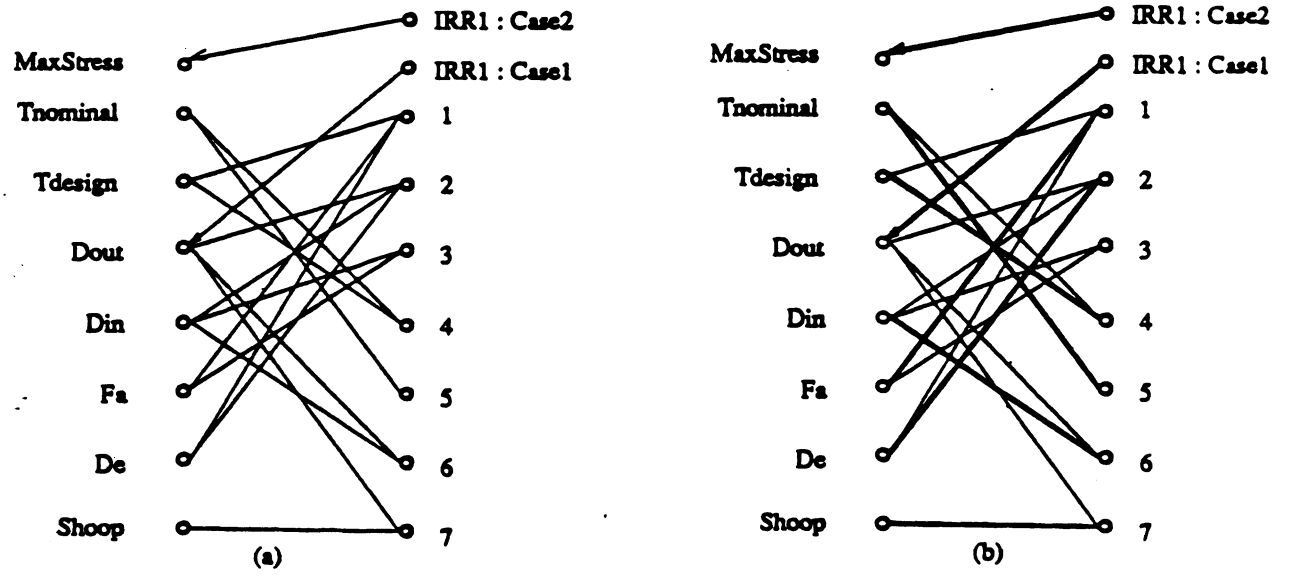

Figure 2-17: A Directed Bipartite Graph representation

case of an irreversible constraint, the matched variable is only dependent on the inputs to the constraint. In other words, one need consider only those uni-directional arcs which point from variables to constraints. In the clutch example, the matchings are as follows:

$MaxStress$	$IRR1:Case2(T_{design}, D_{out})$
$T_{nominal}$	$F5(T_{nominal})$
T_{design}	$F4(T_{nominal}, T_{design})$
D_{out}	$IRR1{\cdot}Case1(T_{design})$
D_{in}	$F6(D_{out}, D_{in})$
F_a	$F1(Td, F_a, D_e)$
D_e	$F2(D_{out}, D_{in}, D_e)$
S_{hoop}	$F7(D_{out}, D_{in}, F_a, S_{hoop})$

2.22 Related Work

The notion of using bipartite matching and the strong components algorithm together was originally suggested by Wang (Wang 73). The algorithms were originally used to solve Gaussian matrices for solving sets of equations using Newton-Raphson like methods. Serrano applied a similar algorithm for finding strong components in sets of constraints (Serrano 87). The aim of this work was to concentrate solution on components and to avoid having to solve the entire constraint set simultaneously. Both these efforts are aimed at bi-directional constraints. We have extended the algorithms to uni-directional constraints. We have also developed the notion of breaking strong components using heuristic approaches.

Recently, Eppinger & Whitney have described a coordination problem in complex design projects [Eppinger & Whitney '89]. A design project is viewed as being composed of several tasks, each of which needs some input data and produces (as output) some data for other tasks. The dependencies among the tasks can be expressed in an adjacency matrix. The paper presents a heuristic approach to ordering the tasks. A comparison study of our approach to ordering uni-directional constraints and the proposed heuristic approach is in order.

[Sutherland 83) Sutherland, *IE.* ,
Sketchpad^AMan-MachineGraphicalCommunicationSystem.
Technical Report TechRepoit #296, MTT Lincoln Lab. Cambridge, Massachusetts, 1983.

[Waid89] Ward.A.G
A Theory ofQuantitative Inference Applied to a Mechanical Design Compiler.
PhD thesis, MiT., 1989.

[Navinchandra & Rinderle '89]
Navinchandra D., J. Rinderle, Interval approaches for Concurrent Evaluation of Design Constraints, In proceedings of the Symposium on Concurrent Product and Process Design, held at the American Society of Mechanical Engineers Winter Annual Meeting, San Francisco, December, 1989

[Eppinger & Whitney *89]
Eppinger, S.D. , D. Whitney, Coordinating Tasks in Complex design projects, In Proceedings of the MTT-JSME (Japan Society of Mechanical Engineers) joint workshop on Concurrent Engineering. Boston, Nov, 1989.

[Aho,Hopcroft& Ullman f83] y; '
Aho, A.V., J.E. Hopcroft, J.D. Ullma, Data structures and Algorithms, Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading, MA 1983

[Serrano 87]
Serrano, D., Constraint Management in Conceptual Design, PhD dissertation, Dept of Mechanical Engineering, MTT, 1987

[Sriram etal '89]
Sriram, D., G. Stephanopoulos, R.D. Logcher, D. Gossard, N. Groleacu, D. Serrano, D. Navinchandra, "Knowledge-Based System Applications in Engineering Design: Research at MTT\ AI Magazine, Fall 1989

[Wang 73]
Wang, R.TH., Bandwidth Minimization, Reducibility Decomposition, and Triangularization of Sparse Matrices, PhD dissertation. Computer and Info. Science Research Ctenter, Ohio State University, 1973

$\overline{p}$
p * 3r
z « *b-la*
P » 5

The run is as follows:

```
CLisp> (load "loop") ... ; load all the files

CL±sp> (setq equations
             ' ((X - Y + Z ** 2) (Y - X * Z) (B - C ** 3)
               (A - (B * 10) / P)
               (Z - B - 2 * A) (P - 5) (P - 3 * R)))

CLisp> (loop::order eqns2 :verbose t) /verbose switch is on

Step 1: Solve for P from constraint:
       (P - 5)
Step 2: Solve for R from constraint:
       (P - 3 * R)

Under Constrained by 1 degrees of freedom ; some stats     F.
Collapsing ((X Y))                        ; trace information

Step 1: Solve for A from constraint
 NIL
Step 2: Solve for B from constraint
  (A - B * 10 / P)
Step 3: Solve for Z from constraint
  (Z - B - 2 * A)
Step 4: Solve the following variables simultaneously:
        (X Y)
        from the constraints:
        ((Y - X * Z) (X - Y + Z ** 2))
Step 5: Solve for C from constraint
  (B - C ** 3)

(((P (P - 5)) (R (P - 3 * R))) (A B Z (X Y) C)) /returned list
```

The function used is *order* which is called from the *loop* package. The ordering is shown in two parts. The first pan indicates the part that is directly decomposed. The second pan is where components are found. In this case, the problem is under constrained. This means that there is an extra variables during bipartite matching. The system decides that variable *A* be determined from constraint *NIL*. This means that the value of *A* has to be guessed, as it is an extra degree of freedom. The rest of the setps are self explanatory.

The function returns the results as a list The list also two parts. The first pan is a list of lists. Each sublist contains two elements. The first is the name of the variable and the second is the equation from which it should be calculated. The second list in the result corresponds to the second part of the output The list shows the order in which the variables have to be solved in. Variables in parenthesis have to be solved simultaneously. In the ouput above, we can solve for *A*, *B* and Z_f solve *X* and *Y* simultaneously and finally solve for C.