Case-Based Algebraic Constraint System for Engineering Design

Ruowen Rong[1], Carlos Saldanha[2], David Lowther[3]

GTE Laboratories Inc., Waltham, MA, USA[1]
Centre de Recherche Informatique de Montreal, Montreal, PQ, Canada[2]
Dept. of Electrical Engineering, McGill University, Montreal, PQ, Canada[3]

Abstract

In this paper, we present an approach to solve design problems using different reasoning techniques. First, case-based reasoning was used in the design problem structuring phase to get the initial design of the device by retrieving similar existing designs. Then we adapting the initial design by using algebraic constraint propagation techniques, qualitative methods and interval arithmetic to solve the design problem.

Introduction

Generally, a design process can be divided into two phases, i.e., the problem structuring phase and the problem solving phase. The problem structuring phase begins with a synthesis process to determine the topology of a device and it results in the construction of an initial design space. In the domain of device design, the design space contains the initial specification, the goal and the related parameters and the knowledge of constraints in the problem. Then, in the problem solving phase, the value of parameters of the device are calculated and verified to make sure that all the constraints are satisfied, some analysis and optimization are performed, and by using feedback from the analysis, the parameters of the device are further refined. Such design process can be illustrate in Figure 1.

Two forms of design knowledge are commonly available to the designer, i.e. the collection of previously designed devices and the set of physical laws. Often the designer creates a new device by adapting existing similar designs according to the appropriate physical laws. The modified design is then tested and refined until all the requirements are satisfied. *Case-Based Reasoning* (CBR) (Kolodner 1993) is a paradigm that emulates such a process. How an existing design should be adapted to achieve the desired function is an important issue in the case-based approach to design. Such adaptation requires an understanding of how device parameters behave in accomplishing the target functionality. This knowledge can often be represented by the parametric model of a device. Such model is usually described as a set of mathematical equations relating the structural, functional and performance parameters of the device. These equations can be organized into a declarative constraint network which can then be manipulated symbolically by an Algebraic Constraint System (ACS). The result is an interactive approach for design space exploration by propagating parameter values and constraint solving. As such, the ACS is a useful tool for use in the adaptation phase of CBR. Designers can investigate modifications on previous device specifications in order to meet new requirements. The aim of this paper is to illustrate the role of the ACS in the CBR approach to engineering problem-solving. A prototype of Case-Based Algebraic Constraint System (CBACS) (Saldanha and Lowther 1987) (Rong and Lowther 1994) has been developed to demonstrate such ideas. The flowchart of CBACS is shown in Figure 2.

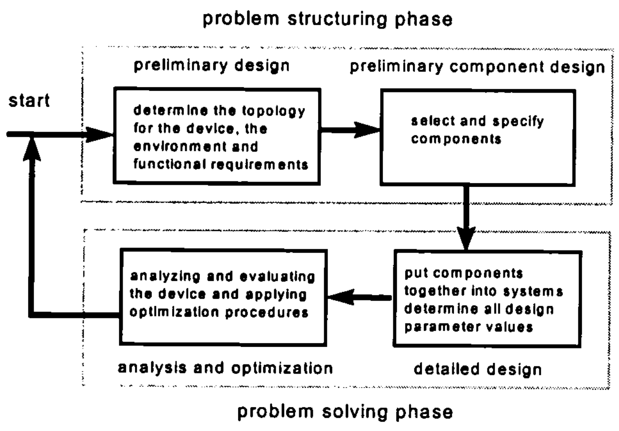
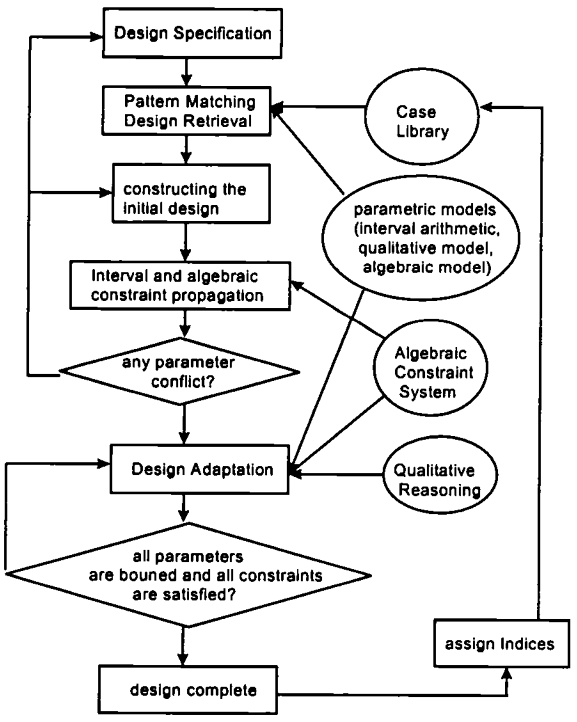

problem structuring phase
preliminary design
preliminary component design
start
determine the topology for the device, the environment and functional requirements
select and specify components
analyzing and evaluating the device and applying optimization procedures
put components together into systems determine all design parameter values
analysis and optimization
detailed design
problem solving phase

Figure 1 The Process of Design

Knowledge Representation

In general, people use a combination of reasoning methods in problem-solving (Gick and Holyoak 1980). In the context of design, past episodes can quickly lead a problem solver into the neighbourhood of a target solution to a new requirement. Other reasoning methods can effectively guide designers in exploring this neighbourhood with the aim of locating the target solution(s). Device models need to be integrated with design cases in order to make use of the adaptation-through-interactive-exploration facilities in CBACS.



Design Specification
Pattern Matching Design Retrieval
Case Library
constructing the initial design
parametric models (interval arithmetic, qualitative model, algebraic model)
Interval and algebraic constraint propagation
any parameter conflict?
Algebraic Constraint System
Design Adaptation
Qualitative Reasoning
all parameters are bouned and all constraints are satisfied?
assign Indices
design complete

Figure 2 The Flowchart of CBACS

Hierarchical Organization of Design Cases Through Models

A *base model* of devices can be described as having certain common properties, i.e. they are all built out of physical materials, they are all subject to the basic physical laws, etc. A hierarchy of models can be constructed from this base model. This way, a solution in an abstract level can be used as a guide for the search of the solution of a less abstract one. The abstraction of knowledge divides the problem space into levels, each level holding a different view of the problem. Existing design cases can be classified as physical instantiations of specific device models, thus associating all model knowledge to design instances. The case organisation scheme being described readily lends itself to implementation using object-oriented programming techniques where the models are depicted as objects defining a class hierarchy.

Conceptually, cases and their features form a semantic network whose nodes represent case classes, cases, device components and their attributes; while network links depict the relations amongst them. Figure 3 illustrates a simple semantic network describing the basic objects of an actuator. This network provides background domain knowledge which is not normally available in a case-base of design episodes. Coupled with inference capabilities, that advantage provided in CBACS is the ability to perform the "deep-reasoning" that is normally associated with the adaptation phase of CBR. Finding similar episodes in the case-base to a current problem is achieved by matching abstract features rather than design details. This corresponds to using domain knowledge to enable matching of episodes that are semantically similar.

The Algebraic Model of a Device

Generally devices can be represented by the algebraic models (equations) relating the structural and functional parameters of the device. An algebraic model of a device can be represented as a constraint network (Mackworth and Freuder 1985). In CBACS, nodes in the network are data structures that represent equations while arcs are data structures representing parameters. When equations are entered into the system, parameters are extracted from the equations and the network is automatically constructed. Equations are identified by a name given when they are created. When an equation is entered into to the system, the algebraic expression is parsed into executable Lisp code by the *Algebraic Symbolic Solver* (ASS). The equation structure contains an expression for each variable in the equation. These expressions are solved by ASS and are used by the inference mechanisms to invoke the equation in either direction. Such kind network is generated and interpreted by the *Algebraic Constraint System* (ACS) module in CBACS.

The equation network is created incrementally as the expert enters individual equations. For each equation entered, the program: adds a new node in the equation network; updates existing links for those parameters already defined; creates a new structure for any parameters which do not already exist in the system; and finally, new links are created to connect the constraint to the newly constructed parameter objects. We can inform the system about the analytical model of a device by simply entering the set of equations defined in the model. For example, to teach CBACS to understand Ohm's law, the equations: $V = I * R$ and $P = I * V$ are entered. CBACS will transform the two equations into a knowledge structure. This process is illustrated in Figure 4. As shown in the simple example,

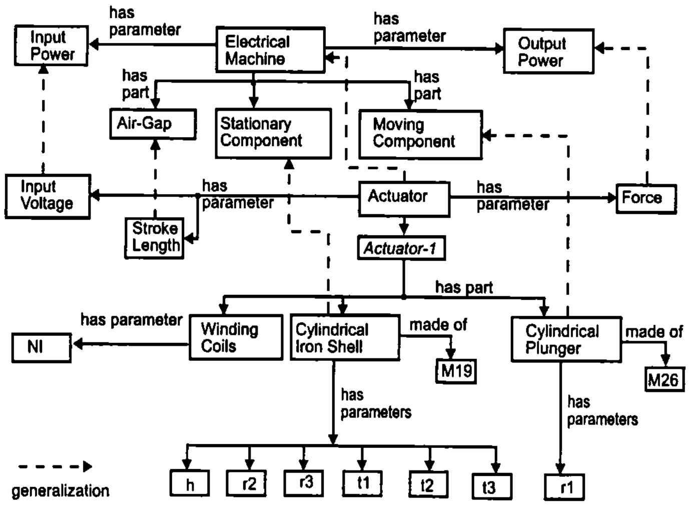


Figure 3 The Semantic Network of an Actuator

```
Input equation: E1: V = I * R
                E2: P = I * V
The computer returns:

{E1 IS-A:     EQUATION  HAS-VARIABLES: V I R
    HAS-CONSTANTS: nil RULE:    (*EQUAL* V (TIMES I R)
    CLAUSES:   ((V (TIMES I R)
                (I (TIMES V (EXPT R -1))
                (R (TIMES V (EXPT I -1))) }
{E2 IS-A:  EQUATION  HAS-VARIABLES: P I V
    HAS-CONSTANTS: nil    RULE:    (*EQUAL* P (TIMES I V)
    CLAUSES:   ((P (TIMES I V)
                (I (TIMES P (EXPT V -1))
                (V (TIMES P (EXPT I -1))) }

Propagating constraints, please wait ... ...
Generating new equations ...

{NE1   IS-A:  EQUATION  HAS-VARIABLES: P I R
    HAS-CONSTANTS: nil
    RULE:   (*EQUAL* P (TIMES (EXPT I 2) R))
    CLAUSES:   ((P (TIMES (EXPT I 2) R))
                (I (EXPT (TIMES P (EXPT R -1)) 1/2))
                (R (TIMES P (EXPT I -2)))) }
{NE2   IS-A:  EQUATION  HAS-VARIABLES: P V R
    HAS-CONSTANTS: nil
    RULE:   (*EQUAL* P (TIMES (EXPT V 2) (EXPT R -1)))
    CLAUSES:   ((P (TIMES (EXPT V 2) (EXPT R -1)))
                (V (EXPT (TIMES P R) 1/2))
                (R (TIMES (EXPT V 2) (EXPT P -1)))) }

New equations have been added to the system.
```

Figure 4 The Ohm's Law in CBACS

the system not only built the structures of the input equations but also discovered and constructed two new equations, i.e. $P = V^2 / R$ and $P = I^2 * R$. This illustrates the system's expertise in algebraic manipulation which is imperative to engineering problem-solving. These new equations can be used to compute I or V directly given the value R and P.

The Qualitative Model

Qualitative reasoning forms a substantial part of our everyday experience. If we consider an equation connecting two variables, $Y = f(X)$, we can determine that Y will increase as X increases without knowing the exact functional form of the relation, much less the numerical parameter values. Even if we could find out the parameter values, we might want to reason about the equation generally, over a range of situations in which Y remains some monotonically increasing function of X.

Qualitative process theory (Forbus 1984) provides a formalism for encoding knowledge about the physical world and some methods for reasoning with that knowledge. Qualitative reasoning can also determine dynamic behaviour, i.e., the state changes of a dynamic system, and this information is very useful in the design process.

The qualitative model in CBACS is composed of nodes, which represent parameters, and links, which represent their relations. As in the qualitative process, we label links as either *M+*, *M-*, *I+*, *I-*. The link *(X M+ Y)* denotes that Y varies monotonically with X (i.e. if X increases then so does Y), while the link *(X I+ Y)* denotes that Y's rate of change dY/dt varies monotonically with X. Similarly, the *M-* and *I-* links denote inverse monotonic relationships.

The qualitative model used in CBACS is only a primitive model with respect to the models specified in qualitative process theory. Currently we have only implemented the monotonic relation between two variables, which can show how a change in one attribute would affect the other attributes. As a result, our model on its own, cannot be used for simulation or prediction. Its role in the system is to work with the other models (algebraic models, in particular) to predict the directional change of certain parameters. Figure 5 illustrates the qualitative relations of some parameters of an actuator and the constraints of these parameters represented by the equation network.

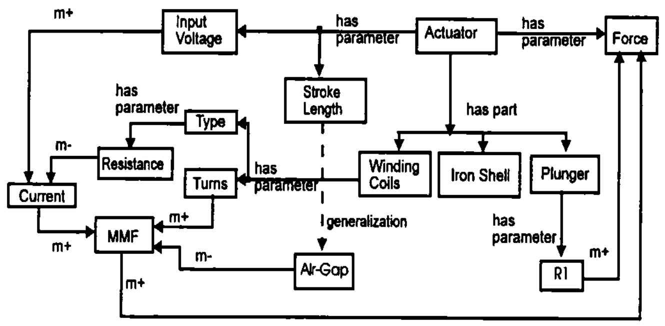


Figure 5 The Qualitative Model of an Actuator

Algebraic Constraint System and Design Adaptation

As we have discussed earlier, in the case-based approach to design, when a similar case is retrieved, the new design will take the associated device model which includes the generic equations and parameters (some of which have default values) associated with a model of the specific device type. The next step in the CBR approach is to adapt the old design to meet the new requirement specification. First, the differences between the problem specifications and those of the retrieved case are analyzed to identify changes that must be made in an old solution. From the qualitative model of the device CBACS can determine how changes in one parameter can result in changes in the other parameters. This is accomplished as follows:

- Variables can be related to each other through simple qualitative mathematical relations. For example, in the domain of electromagnetic design the relation: *(winding-coil-turns m+ mmf)* expresses the fact that the number of turns in the winding coil has a positive monotonic relation to the magnetomotive force, i.e., the higher the number of the turns the bigger the magnetomotive force.
- When looking for a feature-to-feature or feature-to-class explanation, CBACS traverses these qualitative mathematical relations and propagates a "direction of change" for each variable if it is supplied as a second

argument. For example, *(pressure inc)* means that the pressure is increasing *(inc = increasing, dec = decreasing, and cons = constant).*

With the qualitative model shown in Figure 5, CBACS can "discover" that *force* monotonically increases with the magnetomotive force (*mmf*). It also finds that *mmf* is related to the *current*, *turns* and *air-gap* which, in turn, are dependent on the *input voltage*, *winding* and *stroke length* respectively. Thus CBACS will determine from the qualitative relations that increasing the *input voltage* and the number of *turns* and decreasing the *stroke length* will increase the *force*. This qualitative mathematical reasoning provides a fast way of determining appropriate changes in the attributes of a device to meet the design requirements.

When the value of a parameter is changed, CBACS will propagate such modifications throughout network node using a constraint propagation algorithm. Various constraint satisfaction algorithms have been developed (Davis 1987) (De Kleere and Brown 1986). A variant of constraint propagation called *local propagation* has been selected in CBACS as the algorithm for constraint propagation because of its simplicity and resemblance to manual design methods. *Local propagation* incrementally attempts to propagate after each event and only the constraints associated with the subject of the event are tested. In the local propagation algorithm used in CBACS, no linear search is needed in order to locate the relevant constraints after an event due to the data structures used. Constraint Propagation places fewer restrictions on the types of equations that can be used, and is efficient even when the equations are extremely sparse. The method is well suited to solve systems of equations that are under constrained, and to situations where the end-user plays a role in reducing the solution space by assigning values to variables or other constraints.

Local propagation is executed by checking the equation and, if only one variable is unbounded, the program will find the clause associated with the variable and apply the equation to solve that variable. The new value will then be assigned to the value slot of the parameter frame. If all the variables are bounded in an equation, a test is made to see if the left-hand side of the equation is numerically equal, within a predefined threshold, to the right-hand side of the equation. If this is so, the parameter values are consistent with the model, otherwise a data inconsistency exists. If such a conflict occurs, backtracking to a previous design decision and selecting another propagation path may solve the problem. However, if a conflict still exists after all the alternatives have been attempted, there must be some inconsistency in the input parameters or the actual model is invalid. Assuming the coherence of the knowledge-base, the input requirement have to be modified in order to complete the design. On the other hand, if no new variable can be solved for, the operation will return a list of parameters that still need to be determined in other to achieve a design solution.

Whenever a new parameter value is assigned in the equation network, the constraint propagation is invoked. Those equations graphically linked to the parameter are checked and, if the equation is already satisfied or there is insufficient data to apply the equation, no further propagation will be done. However, if a data inconsistency is found, the propagation is aborted and the user is notified about the conflict. On the other hand, if a value for another parameter in one of the equations can be obtained, the value assignment is made and recorded in the history stack. This triggers a recursive call to local propagation. The procedure terminates after all the equations have been checked at each level of recursion and no more deduction can be made.

The ACS can also work on interval constraint propagation (Brett, Saldanha and Lowther 1990) (Rong and Lowther 1995). An interval is a set of numbers of the form: $[a , b] = \{ x \mid a \leq x \leq b\}$. In engineering design, many parameters are subject to value bounds. An interval is a good way of representing such information. The benefits that interval mathematics provide to the expert are not restricted to determining the valid ranges of parameter values. Intervals may be combined with void variables so that inequalities can be represented. For example, the inequality $X \leq Y * Z + 100$ can be reformulated as $X - Y * Z \leq 100$. A void variable *VD1* is then assigned to $X - Y * Z$ and the inequality is incorporated into the equation network by adding *VD1 = [-infinite, 100]*. Note that the equation structure does not have to be modified to provide for inequality relations.

When interval values propagate through the equation network, new intervals are computed for unknown parameters. There is an extra level of complexity involved with interval propagation compared to single-valued propagation. In the latter case, once a value is computed for a variable, it does not re-compute again (unless an inconsistency occurs later on, then the value may be retracted upon backtracking). Interval value propagation, in contrast, computes an interval in the equation network with interval values for each of the arguments involved in the expression (Unknown parameters are assigned the indefinite interval [*-inf* , *inf*] if they are not subject to design specifications or user inputs when constraint propagation is initiated). Next, it intersects the result with the previous value of the variable in question and returns this value as the new interval value for the parameter. This process continues until the interval labels for all the parameters in the network fail to decrease in width. Hence, the process incrementally reduces the solution space for valid designs, shortening the search towards the end goal. Such refinement process described above has been described as Waltz algorithm (Davis 1987).

The ability to use such general types of intervals greatly simplifies the description of an analytical model in CBACS. For example, making sure dimensional information is strictly non-negative is carried out by assigning *[0 , inf]* to the necessary variables. In effect, the interval represents a vast number of parameter values that could have been assigned during regular (single-valued) propagation. This means the amount of backtracking the system must perform when it encounters conflicts is significantly reduced.

All events during the constraint propagation are recorded in a history stack. An event is a record of the variable bindings made as a result of propagation. Each event is labelled chronologically, and is represented by a list containing the value assignments that have occurred. Each event is a triple identifying the source of the assignment, the variable, and the value. By recording all events in the history stack of CBACS, a trace of the reasoning process leading to the current design state is maintained and makes backtracking possible if needed.

Example: Designing an Actuator

To illustrate the ideas in the paper, a simple example is provided in the context of actuator. An actuator is a fundamental electromagnetic device designed to produce mechanical motion and force. It consists of a coil to carry current and generate ampere turns, an iron shell or case to provide a magnetic circuit, and a movable plunger to deliver force. The structure of an actuator is shown in Figure 6. Despite the physical simplicity of the actuator,

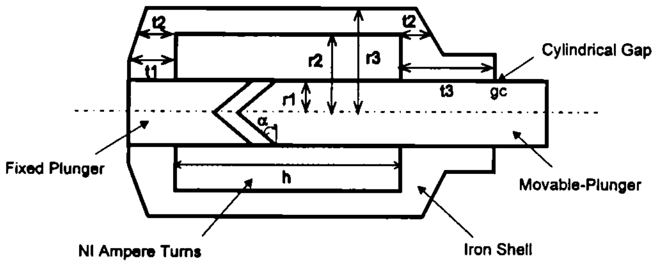


Figure 6 The Structure of an Actuator

designing these devices is a difficult, timing-consuming process because there are a lot of interacting details that must be considered. For example, one of the problems is the temperature rise which makes the coil less efficient because it reduces the ampere turns and hence reduces the flux density and the output force. Another problem is the saturation of the iron path, which prevents the increase in flux density, and hence the increase of output forces. The latter might be solved be changing the iron path area, the pole shape, or the material of the iron shell and plunger. The solution to these problems is usually implied in previous design cases.

The design equations which represent the analytical model of the actuator are entered into CBACS in the format shown in Figure 7. The variables *h, r1, r2, r3, t1, t2, t3, HR, gc* and *alpha* define the geometry of the plunger and the iron shell. The variable *stroke-len* is the stroke length and *F* is the required force. The variable *NI* stands for the Ampere turns. *Mu0* is the permeability of free space and *copper* is the conductivity of the copper. The variables *meff, space_f, t_rise, temp_f, leak_f* and *duty_cycle* represent the magnetic circuit efficiency, space factor, temperature rise, temperature factor, flux leakage

```
(equation 'e1
   :rule '(force = (pi * B^2 * r1^2 * cos (alpha) ^2 / (2.0 * mu0))
   :constants '(mu0) )
(equation 'e2
   :rule '(NI = (B * Stroke_len * cos (alpha)^2) / (meff*mu0))
   :constants '(meff mu0) )
(equation 'e3
   :rule '(t_rise = (duty_cycle * NI^2) / (2.0 * temp_f * space_f
                  * copper_cnd * (r2-r1) * h^2))
   :constants '(t_rise duty_cycle temp_f space_f copper_cnd) )
(equation 'e4  :rule '(HR = h / (r2 - r1)) )
(equation 'e5  :rule '(r1^2 = r3^2 - r2^2)) )
(equation 'e6  :rule '(t1 = (r3^2 - r2^2) / (2.0*r1)) )
(equation 'e7  :rule '(t2 = (r3^2 - r2^2) / (2.0*r2)) )
(equation 'e8  :rule '((leak_f * B * r1) / (0.1 * NI) = 2 * t3 * mu0 / gc)
               :constants '(mu0 leak_f ) )
(equation 'e9  :rule '(ieds1 = r2 - r1) )
(equation 'e10 :rule '(ieds2 = r3 - r2) )
(ival 'ieds1 '[0+  inf])
(ival 'ieds2 '[0+  inf])
```

Figure 7 Actuator Design Equations

coefficient and the duty cycle of the actuator respectively. The constraint network corresponding to these equations is illustrated in Figure 8.

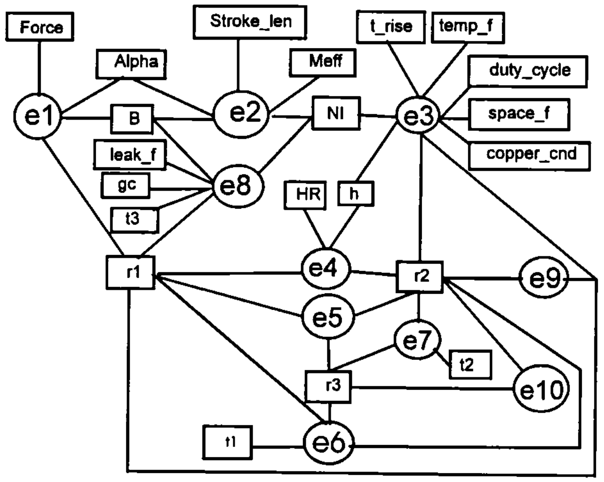


Figure 8 The Constraint Network of an Actuator

Suppose the user wants to design an actuator with force of 300.0 Newton and stroke length of 4.0 mm and these are the only specifications that are given at the beginning of the design. Obviously, the input specification is incomplete, the design is under constrained and more input parameters are needed to start the design process. When CBACS receives this input specification, it first starts the constraint propagation to test if there are any more parameters that can be inferred from the input parameters. It also checks if there is any conflict in the input parameters at the same time. If there is any new parameter that can be inferred from the input parameters, it will be added in the input specification. In this example, no new parameter can be inferred and no conflict has been detected.

Next, CBACS searches the case library to find similar existing design cases that match the input specifications. Since, in the case of an actuator, the most important parameters are *force* and *stroke length*. Therefore even with only two input parameters, it is possible to find the similar cases. As we have mentioned before, the new design case will share the same structure of those retrieved cases but may have different parameter values. Hence, the constraints of the retrieved case, such as the parameters and the constants in the corresponding equations are assigned to the new case.

At this point, a lot of parameters of the new case are unbound. CBACS now invokes the interval constraint propagation process with a constraint on the flux density, *B*, in the interval of [0.80, 1.5] trying to narrow the search space for the new design. After the interval propagation, we obtain the following results:

Force = [300.0 , 300]	stroke = [4.0 , 4.0]
NI = [3638.0 , 6821.0]	r1 = [10.33 , 19.36]
r2 = [19.43 , 33.21]	r3 = [22.01 , 38.44]
h = [54.62 , 83.06]	t1 = [0.0 , 53.26]
t2 = [0.0 , 28.31]	t3 = [1.93 , 12.71]
B = [0.8 , 1.5]	meff = 0.70
space_f = 0.5	copper = 5.8e7
temp_f = 12.0	leak_f = 0.8
gc = 0.1 mm	duty_cycle = 0.1
t_rise = 70.0	mu0 = 1.2566e-6

The design is still not complete. CBACS now has to guess a value of an unbound parameter. From the analysis of the input specification and the parameters in the retrieved case, it discovers that the input *force* is less than the *force* of the retrieved case. Tracing the qualitative relation links, the system knows that *r1* is the major parameter that influences the *force*, so it assigns the value of *r1* in the new case using interpolation of its the nearest neighbours. When the new value of *r1* is added, it invokes the algebra constraint propagation process. In this example, all the parameters are bound after the constraint propagation. The complete design has the following parameters:

Force = 300.0 Newton	Stroke = 4.0 mm
NI = 4096 AmpTurns	r1 = 17.2 mm
r2 = 27.1 mm	r3 = 32.0 mm
h = 59.1 mm	t1 = 8.6 mm
t2 = 5.5 mm	t3 = 6.0 mm

The new design is finally stored in the case library with the same index as its neighbours but with different parameter values.

Conclusion

Case-based reasoning can be viewed as a paradigm that emulates the way human designers solve problems. However, CBR in engineering design usually means case-based retrieval only. The remaining phases of the paradigm are left to the designers. CBACS has been developed to facilitate the adaptation phase of CBR. This has been achieved by using algebraic constraint propagation techniques, qualitative methods and interval arithmetic. The additional knowledge required to apply these methods involves the acquisition of declarative models of engineering devices which in turn provides the basis for organising the case library. The paper describes the design case representation scheme, how generic device knowledge is modelled in the system, and details the integration and role of the various reasoning methods from the CBR perspective. Finally, the paper illustrates the key component of the design adaptation process - how the system can propagate changes of design parameter values in the analytical models of devices.

References

Brett, C.S., Saldanha, C.M. and Lowther, D.A., 1990. Interval Mathematics for Knowledge-Based computer Aided Design in Magnetics. *IEEE Transactions on Magnetics*, Vol. 26, No.2 March, pp. 803-806.

Davis, E., 1987. Constraint Propagation with Interval Labels. *Artificial Intelligence*, Vol. 32, pp. 281-331.

De Kleere , J., and Brown, J.S., 1986. Theories of Causal Ordering. *Artificial Intelligence*, Vol. 29, pp. 33-61.

Forbus, K.D., 1984. Qualitative Process Theory. *Artificial Intelligence*, Vol. 24, pp. 85-168.

Gick, M and Holyoak, K.J. 1980. Analogical Problem Solving. *Cognitive Psychology*, Vol. 12, pp. 306-355.

Kolodner J. L. 1993. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA.

Kuipers, B. J. 1986. Qualitative Simulation. *Artificial Intelligence*, Vol. 29, pp. 289-338.

Mackworth, A.K. and Freuder, E.C., 1985. The Complexity of Some Polynomial Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, Vol. 25, pp. 65-74.

Rong, R. and Lowther, D.A., 1994. Storage and Retrieval of solutions in the design of electromagnetic devices. *IEEE Transactions on Magnetics*, Vol. 30, No. 5 pp. 3648-3651, Sept. 1994.

Rong, R., Lowther, D.A., Brett, C.S., 1995. "Determine the Range of Design Parameters to Support Design Optimization." International Symposium on Non-Linear Electromagnetic Systems, Cardiff, Wales, UK, September 1995.

Saldanha, C.M. and Lowther, D.A., 1987. Devices Modelling in An Electromagnetic Design System. *IEEE Transactions on Magnetics*, Vol. 23, No. 5, Sept., pp. 2644-2646.