Proc. of the 1992 IEEE
Int. Conf. on Tools with AI
Arlington, VA, Nov. 1992

On Reusing Linkage Designs

A. Bose and M. Gini
Computer Science

D.R. Riley and A. Esterline
Mechanical Engineering

University of Minnesota
Minneapolis MN 55455-0159

Abstract

Adapting old solutions to new problems plays an important role in design. A designer often uses solutions of previously encountered problems as a basis for solving a new design problem. This suggests using the Case-based Reasoning model of problem solving to solve design problems. The domain of design we have chosen is the design of planar linkages, in particular, closed four-bar linkages. Linkage design requires reasoning about geometry and motion in problems that are not easily decomposable into independent subproblems. The relationships between function and structure are seldom monotonic and are governed by equations of motions that often involve several of the structural parameters. The functional features are mostly quantitative and often continuous and difficult to predict without simulation.

We propose a representation for design cases and a method for indexing known cases to ease their retrieval given the specification of a new problem. Each description includes structural and performance characteristics. We devise methods to retrieve known cases, to match them to often incomplete problem specifications, and to adapt them to solve the problem.

1 Introduction

An important issue in contemporary design research is the automation of the early stages of design, and the establishment of formalisms to aid in the development of a corresponding computational framework. The work that we report here is part of a larger effort to apply results from our earlier empirical investigation of engineering practice to obtain models for knowledge acquisition and representation of early design activity, and to investigate the viability of domain-independent design problem-solving paradigms. The domain of mechanical design, especially design of four-bar linkage mechanisms, serves as a practical testbed to check the validity of our results.

Engineering design is concerned with the creation of devices that serve a purpose in the physical world in which they operate. This purpose is stated in the *problem statement*. The problem statement addresses a need and limits the final solution by specifying characteristics the artifact designed must possess and constraints the artifact must satisfy. This purpose can be represented as a collection of *tasks* or *functions*. The components of the device comprise its *structure*, obey *physical laws*, and aid in achieving the purpose

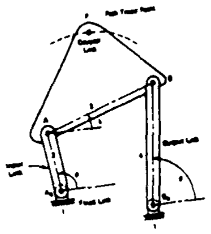

Figure 1: A Four-bar Linkage

by performing one or more of the functions. There is a causal relationship between structure and tasks, and the causality involves physical laws.

In the domain of mechanism design, the artifact is a mechanism. A mechanism is a mechanical device that has the purpose of transferring motion and/or force from a source to an output. A linkage consists of links (or bars), generally considered rigid, that are connected by joints (such as pins) to form open or closed chains (or loops). Such kinematic chains, with at least one fixed link, become mechanisms if at least two other links retain mobility, and structures if no mobility remains. Planar mechanisms exhibit motion such that all the links move in parallel planes.

The simplest closed loop linkage is the four-bar (Figure 1), which has three moving links and four pin joints and has a single degree of freedom (dof). Examples of four-bar and other linkages can be found in, for instance [3]. The complexity, and in general the versatility, of the linkage increases as the number of links increases.

Nearly every mechanical device that is conceived, designed, and fabricated contains a mechanism in one form or another [10]. Linkages make simple mechanisms and can be designed to perform complex tasks, such as nonlinear motion and force transmission. Because of their simplicity and versatility, many designers opt for linkage mechanisms whenever possible. The tasks that linkages are used for are varied and may be classified by application: in *path generation*, the path of a tracer point on the coupler link is of interest; in *motion generation*, the entire motion (path and angle) of the coupler link is of concern; in *function generation*, the relative motion or force ratio between links is of interest. Examples of the three kinds of tasks can be found in [3]. Path and motion generation can be with or without *timing*; i.e. the path or motion of the coupler may be constrained to satisfy certain relations with the input. The work that is reported here deals with path and motion problems with or without timing. However, our method is general enough to be extended to function problems as well.

The task that a linkage accomplishes depends on its topology and geometrical dimensions. Consequently, the function-structure relationship is governed by the equations of motion that can be adequately represented using vector algebra and/or complex number algebra. These equations are for displacement, velocity, and acceleration analysis, and, if dynamics is important, force analysis too [3]. In order to design a linkage for a particular application, there are several variables that are to be considered. To begin with, the type of linkage (the linkage configuration or topology) has to be ascertained (***type synthesis***). The next step requires making quantitative decisions about geometrical dimensions (***dimensional synthesis***). Finally, it is customary to simulate and analyze the motion of the linkage to ensure that no constraints are violated and to determine various performance parameters, such as velocity and acceleration.

One way in which design knowledge can be captured is by storing particulars of previously designed artifacts as cases. These particulars may be stored as models which can be refined or modified to suit a particular application. In this form, Case-based Design (CBD) becomes an augmentation of Model-based Reasoning. Most designers are accustomed to looking up standard artifacts and/or design solutions in design catalogs or handbooks published for this purpose. For instance, a comprehensive list of four-bar linkages and the curves traced by points on their couplers are enumerated in [11]. This catalog consists of hundreds of four-bar linkages. A catalog lookup on its own, however, is usually not sufficient since the catalog seldom contains the exact the solution to a given problem specification [4]. Case-based Design can be looked on as an enhancement to a straightforward catalog lookup. It goes one step further by adapting what is in the catalog to suit the needs of the problem. It reflects the propose-verify-and-redesign paradigm [2], which is popular among researchers in CBD.

Our research focuses on CBD for preliminary linkage design to illustrate how the approach may be used in other domains. The objective is to ascertain the topology of the linkage mechanism, and, to determine the locations of ground pivots and the link lengths. Cases in this domain have attributes distinguishing them from most cases reported in the CBD literature: they are not easily decomposed, function-structure relationships are seldom monotonic, and component dimensions are important even before a detailed design is attempted. Since little work has been done in CBD that involves these attributes, achievements in this domain will be a true test of this methodology.

2 A Case-based Approach to Linkage Design

The paradigm that we use for CBD is illustrated in Figure 2. The problem instance is entered as a set of design requirements. The *Retriever* uses pre-defined similarity metrics to retrieve from case memory candidate designs that have functional properties similar to the problem specifications. In so doing, it eliminates most of the search space of design solutions. Not all cases that are retrieved have the promise of becoming the pre-cursor to a solution for the problem instance. The *Potential Evaluator*, to evaluate the potential for a case to become a solution, studies the nature of functional differences between each retrieved case and the problem and checks the availabililty of adaptation rules. Cases that do not have sufficient potential are rejected. The remaining designs are adapted to the problem instance in the *Adaptor*. The adapted designs are then simulated in the *Simulator* under the constraints and requirements of the problem instance. A successful design becomes a candidate for permanent storage in case memory. The successful designs are also ranked in order of suitability for the problem instance. If the simulation of a particular design fails, the case is fed back to the *Potential Evaluator* where functional differences are evaluated against the availability of rules that deal with them. The cycle continues with the case being re-fed into the *Adaptor*. If a dead-end is reached or if the number of iterations through the loop exceeds a threshold, backtracking is performed, and alternative rules are tried in the adaptor. The adapted case is tried again in the *Simulator*. Hence this is a depth-first search in the design space with backtracking on the applicable adaptation rules.

In the context of preliminary linkage design, the problem instance consists of performance requirements (e.g., part of a coupler curve), constraints on

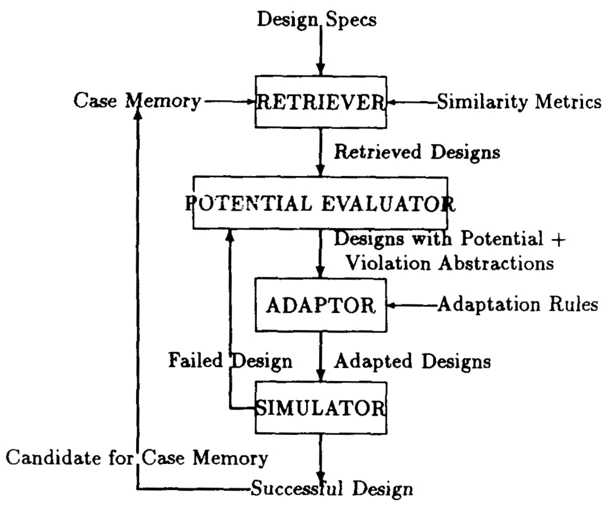

Figure 2: A Schematic of the Case-based Designing Paradigm

input and output (e.g. crank-crank, crank-rocker), dynamic considerations (e.g., velocity at specified point(s) on the coupler path), and desired mechanical characteristics (e.g., minimum transmission angle). A case has all the relevant structural and functional characteristics of a linkage stored within it. Structural characteristics include link lengths and ground pivot locations. Functional characteristics include the path traced by the coupler path tracer point, the crank angles at different crank locations, etc. Cases are grouped into clusters depending on the type of problem they solve and various abstractions of performance characteristics. For example, the coupler curve is segmented on the basis of curvature and stored as a closed string. Cases with identical strings are in the same cluster. Abstraction and clustering enable efficient indexing and retrieval of a case that is similar to the problem instance. Case retrieval is dependent on there being a sufficiently large intersection of information in a case and the given problem instance. A case is represented at several levels of abstraction to facilitate the pruning of the search tree at the successive levels for retrieving similar cases. Adaptation rules use strategies that are for structural alteration (e.g. changing link lengths of the base linkage). The case with the best ranking is selected for adaptation (or modification), in light of information about requirements that are violated from the matching process, to the problem requirements. Once a case is modified, it is simulated[1]. If a simulation fails (i.e. one or more constraints are violated), the functional differences that caused the failure are evaluated and abstracted to yield symbolic information on the violation(s). An attempt is then made to further adapt the design. If success is not forthcoming after several evaluate-adapt-simulate cycles, the current candidate is abandoned in favor of the next best case as ranked by the matching process, and the simulation- and-repair cycle repeats. If no candidate remains, failure is reported.

3 Storing and Indexing Cases

Linkage designs that are unique in their functional characteristics are stored as cases for future use in problems that require similar characteristics. Since problems consist mainly of functional (task) specifications, it is deemed sufficient to index the cases on the basis of function only. Several different indices are used to capture the functional features of a case. Since geometric reasoning is an intrinsic part of linkage design, our representation techniques for cases include structures to store the important geometric information. Most representation schemes currently in use[2] are not ideally suited to our problem, especially considering the tight memory requirements arising from the need to store many different cases at multiple levels in the system.

The basic functional characteristics of a four-bar linkage include the path described by a reference point on the coupler (coupler point) of the linkage, and the corresponding crank and coupler rotations. This information is discretized and stored as an attributed curve, which we shall call the ***attributed coupler curve*** or *ACC*. The *ACC* can vary greatly depending on the structural parameters of the linkage.

[1]The simulation algorithm is based on the LINCAGES-N software package developed at the Productivity Center, University of Minnesota.

[2]For example, see [14] for a comprehensive description of some of these schemes.

$$ArcIndex = sense * ceiling(log_b(\rho))$$

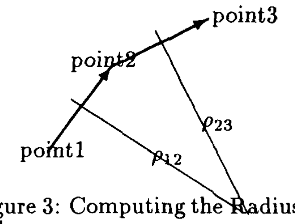

Figure 3: Computing the Radius of Curvature and Arc Index

All coupler curves are closed since the crank is always rotated 360 degrees. We generate an *ACC* by recording N positions of the coupler point corresponding to N positions of the crank. Each crank position is obtained by rotating the crank $360/N$ degrees from the previous position in the anti-clockwise direction. The initial crank position corresponds to a zero degree crank angle. The *ACC* at the lowest level can be viewed as a list of N records with each record containing information on the position of the coupler point in 2-D space, and the corresponding crank and coupler rotations. Since the value of N we use is fairly large, we may assume that the actual curve can be approximated by straight line segments joining consecutive points in 2-D space.

To abstract information about the shape of the curve, we break the curve into *segments*. A segment can belong to one of two categories: an *arc segment*, or a *distinguishing point segment*. For an arc, the segmentation is done on the basis of curvature, using a measure called the *arc index*.

As shown in Figure 3, the point of intersection of the perpendicular bisectors of the straight lines joining three consecutive points on the curve is first computed. The distance from this point of intersection to any of the three points is the approximate radius of curvature of the portion of the curve represented by the three points. Next, the arc-index for this portion of the curve is computed. A logarithmic transformation is used to collapse the radius of curvature space to the arc-index space for two reasons. First, this serves as a convenient abstraction, where information about the curvature is retained, and yet the feasible curvature space is shrunk to facilitate matching between curves. Second, the arc-index is more sensitive to changes in smaller curvatures than to larger ones. This more accurately abstracts the shape of the curve and aids in retrieving *similar* curves. In order to distinguish between concave and convex arcs, convex arcs are represented by negative arc indices, hence the presence of *sense* in the formula. This process is repeated for all consecutive three point sets. Adjacent three point sets with the same arc-index are merged to form segments.

Distinguishing points on the curves are also detected and recorded. There are two kinds of distinguishing points: *cusps* and *crossings*. A cusp is defined as a point on the curve where the included angle between the straight lines representing the curve is greater than or less than certain thresholds (270 and 90 degrees respectively). A crossing is a point at which the curve crosses itself. These two kinds of points aid in characterizing the curve as belonging to a particular category, and thus aid in retrieving the right kind of curve.

In summary, the coupler curve at the top level is represented as a *segment sequence* of arc-indices and distinguishing points. At a lower level, each arc segment has associated with it a length index , which is a measure of the length of the arc that makes up the segment. This is to distinguish between segments that may have similar curvature but vary considerably in length. Also included at this level are the crank rotation and the coupler rotation angles for each arc segment. At the lowest level, the ACC is represented as a list of N records as described earlier. At this level, we also record the structural characteristics of the linkage which include the link lengths, the position of the coupler point relative to the coupler pivots, and ground pivot positions.

4 Specifying a Problem

A problem consists of the functional and structural properties desired in the linkage to be designed. A problem specification is often incomplete. This may result in satisfactory matches with several cases.

The functional information in a problem at the very least consists of *curve fragments* that the coupler point trace should contain. Each curve fragment consists of a list of points, with neighboring points connected by straight lines (to make it continuous). Each curve fragment is hence a fully connected bi-partite graph. There may be more than one curve fragment specified, which would imply that the specification contained multiple unconnected bipartite graphs. A tolerance is specified in conjunction with the curve fragments, thus making the constraints on the coupler point trace 'tubular' in the sense that the trace has to lie within the two dimensional 'tube' formed by considering the tolerance above and below a straight line.

Besides this basic coupler point trace information, additional requirements about specific points that need to be traversed by the coupler point may be specified. These points are called *mandatory points* and the tolerances for these points are much tighter. Whereas the designer uses curve fragments to constrain the general shape of the coupler point curve, these mandatory points have additional importance in the eyes of the designer and have to be contained (within the specified tolerance) in the coupler point curve.

Additional information on coupler rotation requirements between given points may be specified. These are useful in describing *motion* problems (see Section 1). *Timing* information in the form of crank rotations between given coupler point positions or coupler rotations may also be specified. Tolerances may also be associated with these requirements since, in general, these requirements are seldom exact.

The only structural information that is required in a problem is the location of the ground pivot points. Since our domain consists of planar four-bar linkages, the points are in two-dimensional space and are two in number. Additional structural requirements that may be specified include upper and lower bounds on link length ratios, velocity of the coupler point, and linear and angular acceleration of the coupler link. If velocity and acceleration requirements are specified, the angular velocity of the crank has to be specified also.

5 Retrieving Similar Cases and Evaluating Their Potential

Cases are retrieved based on a match between the problem specifications and the cases in case memory. Successive matches are performed hierarchically at different levels of abstraction.

5.1 The Mechanics of Case Retrieval

Information provided in the problem specification is transformed to a normal form by performing the operations of translation, rotation, and scaling. It is then abstracted to provide the basis for retrieving suitable cases. The abstraction is done at the same three levels used for representing cases. At the top level, segmentation is performed on the curve fragments and a partial string sequence is computed. The string sequence is partial because of the gaps between each individual fragment. *Don't cares* (meaning one or more segments of any type or curvature can be inserted in their place) are inserted at appropriate places in the string sequence to provide a regular expression[3]. Features of this regular expression are used to index into the case-base. These features include the minimum and maximum values of the arc indices in the curve, and the number of arc segments.

Initial matching between a case and a problem is performed by constructing a finite state automaton based on the regular expression derived from the curve fragments of the problem. The string sequence corresponding to each candidate case supplies input to the automaton. If an accepting state is reached, a match between the problem and the case has been found at the first level (in other words the string sequence derived from the curve fragments of the problem is present as a pattern in the string sequence of the case). Details of this matching algorithm can be found in [1].

Once a match with one or more cases is found at the first level, a more detailed match is attempted at the next level of each case. Depending on the information available from the problem specification, the length index and coupler and crank rotation information is used for matching at this level.

[3]A *regular expression* is a language accepted by a finite state automaton. A *finite state automaton* consists of a collection of states, an alphabet of input signals, and a function which for every possible combination of current state and input, determines a new state. A finite state automaton is said to *accept* any sequence of symbols which paces it in its final state.

Cases that survive the pruning at these levels are passed on by the *Retriever*. The potential of the retrieved cases to become a solution for the given problem is evaluated in the *Potential Evaluator*. This is done at the next (most detailed) level. The given specifications are matched against the information in the case to ensure that no specifications are seriously violated. Tolerance information provided in the problem is used to perform inexact matching. The functional differences that cause the violations are categorized for reference by the adaptation rules. If there are adaptation rules for the particular category of functional violations, a case survives the pruning at this level, and it is included in the list of candidate solutions passed on to the *Adaptor*.

5.2 Discussion

There are several important considerations in devising a scheme for case retrieval in a domain such as planar linkage design. One of these is the existence of non-monotonic continuous functions over time. We employ functional indexing as the primary means of retrieving cases because of the need to match the functional information in a case to that in a problem. As described in Section 3, a case has a three-level representation. At the top level, cases are indexed on the basis of segment sequence derived from the curve traced by the coupler point. The segment sequence gives a sense of the nature of the closed curves and reveals the existence of features, such as points of crossing and cusps. Associated with each segment at the next level is a measure of the length of the segment (length index). These provide information on the differences between curves with similar segment sequences.

A danger in abstracting information, particularly from continuous functions, and then using the abstracted information as the basis for indexing and retrieval is the possibility of excluding promising cases from consideration. We overcome this by abstracting in a way so as to encompass a sizeable chunk of the design space. For instance, an arc index, by its very definition, may be the same for several curve segments which are quite distinct from each other and yet are related to each other curvature-wise. Hence, our method of abstraction ensures that no cases with potential to become solutions are ruled out prematurely. The price we pay for ensuring this is that we must initially consider some unpromising cases.

Since a problem specification is seldom expected to contain complete information to enable a comprehensive match with all functional aspects of a case, methods have to be devised for detecting partial matches. Converting functional information in the cases to string sequences and treating the incomplete information in the problem specifications as regular expressions enables matching cases with problems. This is because the regular expressions permit the insertion of one or more segments into the "gaps" denoted by the *don't cares*. This enables all segment sequences (denoting cases) that have the curve fragment segments (from the problem) embedded in them to be detected.

```
RULE Shrink_Curve
  Antecedents
    fit_tube
    closed_curve
    crossings_absent
    (shrinkage_percent < 10%)
  Consequents
    (couplerPoint_radius := 1.1*couplerPoint_radius)
    (couplerPoint_angle := 1.1*couplerPoint_angle)
```

Figure 4: An Example of an Adaptation Rule

6 Adapting Retrieved Cases

When indexing and retrieval are based on functional features, it is inevitable that the cases that are retrieved are the ones closest to the problem specifications. However, not every retrieved case has the potential to become a solution for the given problem. This is because the matching process is sensitive to functional differences which are by themselves seldom sufficient to indicate the potential of a candidate case to become a solution. The causal relationships between structure and function are seldom monotonic. So just because a case is functionally close to a problem does not always imply that the structure of the artifact in the case can be modified or adapted to fulfill the problem specifications.

Adaptation rules play an important part in evaluating potential candidates for a solution. They provide a mapping from functional to structural differences. In order for adaptation rules to be effective, the functional differences have to be qualitative in nature or easily categorizable into groups through abstraction. Easily categorizable quantitative differences can be converted to qualitative ones, but functions that are continuous in nature normally require that limits or thresholds be used, which can never be completely accurate. Within preliminary mechanical design in general and mechanism design in particular, the functional information is often continuous and available in quantitative form. Abstraction of differences in continuous valued functions precedes adaptation. This information may represent spatial properties; so spatial reasoning is an integral part of adaptation.

The inclusion of tolerances in the problem specification affects adaptation also. The threshold for functional differences allowed between retrieved cases and the problem specifications depends on the tolerances. If a larger tolerance is allowed, more cases are retrieved. Thus in the process of adaptation, the mapping is between larger functional and design spaces.

All adaptation rules have antecedents that contain qualitative categorizations of the functional differences between a case and the problem specifications. Also present in the antecedents are the contexts in which the rules are relevant. The consequents of these rules cause alterations in structure. These alterations consist of changes in link lengths or the location of the tracer point on the coupler.

Let us consider an example of an adaptation rule

Regular Expression : d.9.-9.-10.d.7.6.5

Curve Fragment 1

Arc Ind.	*Len. Ind.*	*Seg. Type*
9	26	Bound.
-9	79	Int.
-10	23	Bound.

Curve Fragment 2

Arc Ind.	*Len. Ind.*	*Seg. Type*
7	10	Bound.
6	17	Int.
5	20	Bound.

Figure 5: Segmentation of the Problem Curve Fragments

Seg. Seq. : 9.-9.-10.9.8.7.6.5.6.7.8.7.6.5.6.7.8

Arc Index	*Length Index*
9	26
-9	86
-10	24
9	20
8	16
.	.
.	.

Variable	*Value*
Fixed Link	10.0 units
Crank	4.0 units
Coupler	6.5 units
Follower	10.0 units
C. Point Radius	6.0 units
C. Point Angle	45 deg

Figure 6: Description of a Candidate Case

that is listed in Figure 4. The purpose of rule *Shrink_Curve* is to cause a contraction in the coupler curve by altering the location of the tracer point with respect to the pivot points of the coupler link. The context for this rule is **fit_tube**. In order for this rule to fire, the curve has to be closed, there must be no crossings present, and the fraction by which the curve is to be shrunk is less than ten percent. The consequents update the values of the two variables that determine the location of the tracer point on the coupler link.

7 An Example

We demonstrate CBD in linkage design with the help of a simple example. The problem specification includes two curve fragments. The ground pivot locations and the tolerance of the curve fit are also specified. No angular constraints are provided. Hence, this is a path generation problem. The solution for this problem will be a four-bar linkage whose fixed link can fit between the specified ground pivot locations and whose coupler point curve trace includes the curve fragments within the tolerance limits specified. In other words, the two conditions that are to be satisfied are **fit_pivot** for each ground pivot and **fit_tube** for each curve fragment.

Before case retrieval can commence, the problem specifications have to be transformed to a normal form to facilitate comparison with cases. This entails the three elementary transformations of scaling, translation, and rotation on the basis of the ground pivot locations. This automatically takes care of the **fit_pivot** conditions. In the next step, the abstraction of the problem is done to yield features that may be used for case retrieval. The curve fragments are segmented. A summary of the segment information is given in Figure 5. Features of the regular expression are used to

First Alignment : 9.-9.-10.–d–7.6.5.d

Segment Sequence : 9.-9.-10.9.8.7.6.5.6.7.8.7.6.5.6.7.8

Second Alignment : 9.-9.-10.————d————7.6.$\underset{8}{5}$.d

$$\begin{tikzcd}
1 \arrow[r, "9"] & 2 \arrow[r, "-9"] & 3 \arrow[r, "-10"] & 4 \arrow[r, "9"] & 5 \arrow[r, "8"] & 6 \arrow[r, "7"] & 7 \arrow[l, bend right, "8"'] \arrow[d, "6"] \\
 & & & & \boxed{10} \arrow[urr, bend left, "7"] & 9 \arrow[l, "6"] & 8 \arrow[l, "5"]
\end{tikzcd}$$

The accepting state (10) is reached twice corresponding to the two alignments shown above.

Figure 7: The Finite State Automaton Constructed from the Regular Expression of the Problem and the Two Alignments of the Case Segment Sequence that Match the Regular Expression

index into the case base.

One of the cases that is retrieved is summarised in Figure 6. Only the information that is relevant to this problem is presented. Each segment also contains the points that are included within it. Needless to say, every case has additional information that is organised as described in Section 3.

A finite state automaton is constructed from the regular expression of the problem fragments[4]. When the segment sequence of the candidate case is run through the automaton as shown in Figure 7, an accepting state is reached twice. This implies that there are two possible alignments between the regular expression and the segment sequence. These are also shown in Figure 7. Each of these alignments is now inspected in closer detail.

For both alignments, the length indices of the interior segments derived from the curve fragments of the problem are compared to the corresponding length indices of the case segments. In the case of the first alignment, the length indices are close enough, but for the second alignment there is a large difference between the length indices of one interior segment (in fragment 2) and so the second alignment is rejected. The case, along with the first alignment is now passed to the Potential Evaluator.

For the first alignment, condition **fit_tube** is true for fragment 2, but not for fragment 1. The actual fit is illustrated in Figure 7. A displacement vector metric is used to compute the approximate distance and the direction the relevant portion of the case curve needs to be tailored in to fit into the tube for fragment 1. The value, 4 percent, and the direction, inwards, causes the Potential Evaluator to accept the case because of the existence of rule *Shrink_Curve* (see Figure 4). In the Adaptor, the rule *Shrink_Curve* is activated. This rule alters the location of the tracer point with respect to the coupler. In the Simulator, the modified coupler point curve is obtained. It now fits in the "tubes" and

[4]The *d*'s are for "don't cares", i.e. any integer(s) is(are) acceptable at that position in the sequence.

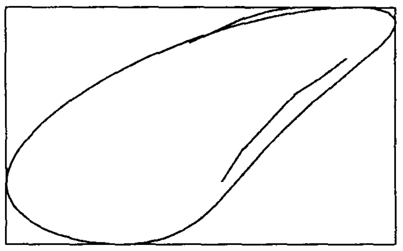

Figure 8: The Coupler Curve Trace from the *Retrieved* Case Compared to the Curve Fragments of the Problem

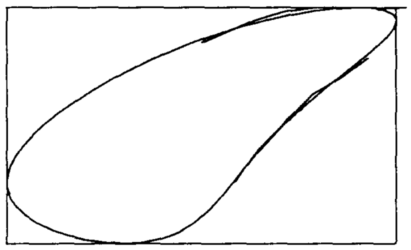

Figure 9: The Coupler Curve Trace from the *Adapted* Case Compared to the Curve Fragments of the Problem

fit_tube is now true for both fragments (Figure 9). Hence, one solution has been found.

8 Related Work

The paradigm for CBD that we use is similar to the one for Case-based Planning in CHEF described by Hammond in [7], but there are several important differences. The domain of CHEF is Szechwan cooking in which clear causal relationships exist between structure (the ingredients of a dish) and their function (the taste, texture etc. of the dish). This enables the use of a failure predictor that can predict what might go wrong in a new dish that is prepared by adapting ingredients and/or steps from a retrieved case. Also, the search space is finite and the variables are discrete. This not only yields simpler failure detection algorithms, but also more effective adaptation and repair strategies.

In [5], Goel and Chandrasekaran describe a scheme for storing cases according to a device's functions, its structure, and its behavior. The causality between the structure and the functions of the model is captured by the behavior which is based on physical laws. When a new device with similar functions is to be designed, the structure can be modified with the help of information on the behavior of the components of the device. This scheme has been implemented in the domains of Acid Coolers, and Reaction Wheel Assemblies. In both cases, the causal relationships are captured as qualitative relations that are montonically increasing or decreasing. The designs are conceptual in nature, and do not deal with dimensional synthesis.

Sycara and Navinchandra in [17] take a broader view of CBD. Apart from the indirect use of models in the form of qualitative relations describing the functions of a device, they also use analogical reasoning to retrieve examples from diverse domains, where the thread of commonality runs only at a very abstract level. It is unlikely that reasoning at such an abstract level can be useful for routine component design, but this approach may prove to be very useful for innovative design both at the system and component level, where computer-aided tools are lacking. Their examples are taken from the domain of water taps and faucets. The causal relationships are again monotonically increasing or decreasing. The designs are conceptual in nature and dimensional synthesis is not undertaken.

There has been some recent work in integrating case-based reasoning with other forms of resoning to improve overall performance. Golding and Rosenbloom in [6] describe an architecture where the rules are first applied to a target problem to get a first approximation to the answer; but if the problem is found to be similar to a known exception of the rule, then the exception is used to model that aspect of the solution. The cases thus record the exceptions to the rules. In [13], Rajamoney and Lee describe prototype-based reasoning to be an integration of Model-based and Case-based Reasoning. They claim that this form of reasoning is useful for solving large problems. They decompose a large problem into sub-problems and use cases to solve each subproblem. The individual solutions are then combined and solved using Model-based Reasoning.

The approach to case retrieval we take borrows heavily from work done in diverse fields. Since we identify the main aspect of case retrieval to be the matching of attributed coupler curves, the work done in the field of inexact matching of attributed shapes is relevant to us. This work mostly pertains to the recognition of three-dimensional objects in vision systems. Shapiro et al., in [16], make use of a relational paradigm to match stored models of 3-D objects against unknown 3-D objects. In order to characterize the similarity and differences between 3-D objects, they define a measure of relational similarity, and a measure of feature similarity. This leads to the concept of inexact matching, where matches that are not necessarily perfect, but merely good enough, are sought.

Methods from AI have been used by design researchers to store linkage examples. Hoeltzel and Chieng[9] developed Pattern Matching Synthesis (PMS) for matching curves of four-bar linkages (although they do not explicitly refer to CBD). They determine the dimensions of the path generator mechanism and then the dimensions of the timing compensator mechanism by associating curves to clusters.

From a set of parametrically generated coupler curves, their system extracts a set of clusters, used as patterns for training a neural network. The neural network can recognize a new instance of a stored pattern and may function as an associative memory, retrieving the entire curve (with the associated linkage dimensions) that best matches a given curve. This method cannot deal with incomplete information.

Kota et al.[12] describe a system called MINN-DWELL for high speed interactive kinematic synthesis and analysis of multi-link dwell mechanisms. An atlas of four-bar straight-line linkages containing about 350 different linkages is developed. The designer selects the linkage from the atlas based on the high level features that are desired. The system then computes the linkage parameters.

9 Conclusion

We have described a model for performing case-based preliminary linkage design. First, we have demonstrated the suitability of the CBD paradigm for a class of design problems characterized by the fact that they are not readily decomposable, and the function-structure relationships are seldom monotonic, and often quantitative, continuous and unpredicatble. Adaptation and/or repair of candidate cases is non-trivial because of these function-structure relationship properties; a change in a structural dimension can cause unpredictable changes in function and so the task of failure checking is complex.

Second, we have devised efficient and extensible representation techniques for storing cases and representing problems, both of which have a significant amount of qualitative and quantitative information within them. Since geometric reasoning is an intrinsic part of linkage design, our representation techniques for cases include structures to store the important geometry information. Efficient matching techniques for geometric figures, which are of vital importance from the viewpoint of case retrieval in the domain under study, are also devised.

A prototypical system has been implemeted using a combination of C++, C, and CLP($\Re$). Preliminary parameteric and validation studies have provided encouraging results.

Acknowledgements

This work was supported in part by NSF Grant Number DDM 8717731-01.

References

[1] A.V. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science - Vol A*, chapter 5, pages 255–300. Elsevier Science Publishers, 1990.

[2] B. Chandrasekaran. Design: An information processing level analysis. In D. Brown and B. Chandrasekaran, editors, *Design Problem Solving: Knowledge Structures and Control Strategies*, pages 23–30. Pitman, 1988.

[3] A.G. Erdman and G.N. Sandor. *Mechanism Design: Analysis and Synthesis, Vol 1*. Prentice-Hall, Inc, 1984.

[4] A. Esterline, D.R. Riley, and A.G. Erdman. Design theory and ai implementations of design methodology. In *Preprints of the NSF Engineering Design Research Conference*, 1989.

[5] A. Goel and B. Chandrasekaran. Functional representation of designs and redesign problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1989.

[6] A.R. Golding and P.S. Rosenbloom. Improving rule-based systems through case-based reasoning. In *Proceedings Ninth National Conference on Artificial Intelligence*, pages 22–27, 1991.

[7] K.J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press Inc, 1989.

[8] N. Heintze, J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. *The CLPR Programmer's manual, Version 1.1*, Nov 1991.

[9] D.A. Hoeltzel and W-H Chieng. Pattern matching synthesis as an automated approach to mechanism design. *ASME J. Mechanisms, Transmissions and Automation in Design*, 1989.

[10] D.A. Hoeltzel and W-H Chieng. Knowledge-based approaches for the creative synthesis of mechanisms. *Computer-Aided Design*, 22(1):57–67, Jan-Feb 1990.

[11] J.A. Hrones and G.J. Nelson. *Analysis of the FOUR-BAR linkage: Its Application to the Synthesis of Mechanisms*. The MIT Press, 1951.

[12] S. Kota, A.G. Erdman, and D.R. Riley. Minn-dwell - computer aided design and analysis of linkage-type dwell mechanisms. *Mechanisms and Machine Theory*, 4(1):1–8, 1988.

[13] S.A. Rajamoney and H-Y. Lee. Prototype-based reasoning: An integrated approach to solving large novel problems. In *Proceedings Ninth National Conference on Artificial Intelligence*, pages 34–39, 1991.

[14] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Inc, 1990.

[15] L.G. Shapiro and R.M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(5):504–519, 1981.

[16] L.G. Shapiro, J.D. Moriarty, R.M. Haralick, and P.G. Mulgaonkar. Matching three-dimensional objects using a relational paradigm. *Pattern Recognition*, 17(4):385–405, 1984.

[17] K.P. Sycara and D. Navinchandra. Integrating case-based reasoning and qualitative reasoning in engineering design. In J. Gero, editor, *AI in Engineering Design*. 1989.