A case-based approach to planar linkage design[*]

Ashim Bose[†] and Maria Gini
Department of Computer Science
University of Minnesota, Minneapolis, MN 55455

Donald Riley
Department of Mechanical Engineering
University of Minnesota, Minneapolis, MN 55455

Abstract

We describe a method to store and retrieve design cases of four-bar linkages. The method abstracts relevant features from the planar curve that characterizes each linkage design, and records these features at several levels of abstraction. This allows to use a fast multi-level retrieval procedure, that successively eliminates inappropriate cases at each level. The retrieval incorporates tolerance measures and allows for incomplete problem specifications. Adaptation methods are then used to transform the cases retrieved to match the new design problem. Results of experimental studies to evaluate the efficacy of our method are presented.

Keywords: case-based design, planar linkages, mechanical linkage design, mechanical linkage synthesis.

1 Introduction

One way in which design knowledge can be captured is by storing particulars of previously designed artifacts as *cases*. When a new design problem is encountered, cases that most closely match the new requirements are *retrieved* and *adapted* to suit these requirement [Adelson, 1989, Gero, 1990, Maher, 1990]. Case-based Design [Riesbeck and Schank, 1989, Sycara and Navinchandra, 1989] reflects a commonly used design method where designers look up standard artifacts and/or design solutions in design catalogs or handbooks published for this purpose.

[*]This work was funded in part by the NSF under grant DDM 8717731-01
[†]Current affiliation: Space Telescope Science Institute, Baltimore, MD 21218

The objective of this paper is to describe our work in storing design cases of four-bar planar linkages, and retrieving the most similar ones using incomplete specifications. The context of this work is planar linkage design, though the methods used can be extended to cover other domains that are characterized by planar geometric information.

Our main contribution is in extending the applicability of case-based methods to problems that are characterized predominantly by their geometric properties. Other Artificial Intelligence based methods, such as expert systems [Bertini, 1993], have been used for design mechanisms. When the problems are geometric, like in the case of linkage design, it is easier for the designer to sketch the desired path or fragments of it that to describe it by words. More details on related work are given later in Section 9.

A brief discussion of the domain of planar linkage design is in order [Erdman and Sandor, 1984].

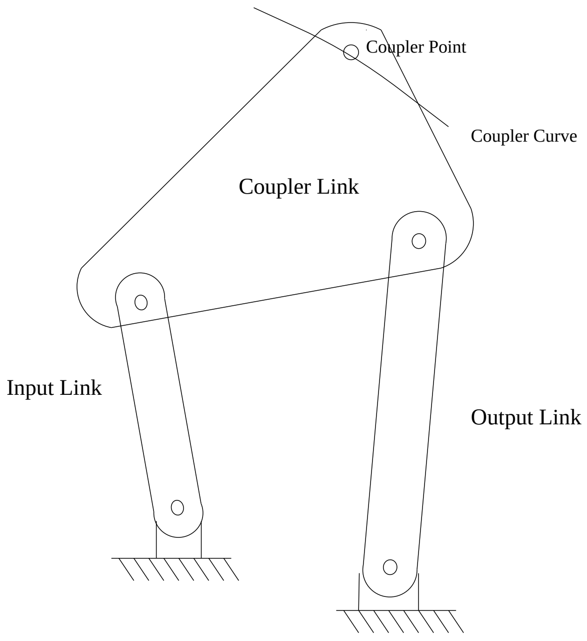


Figure 1: A four-bar linkage

A linkage consists of rigid links, which are connected by joints to form open or closed chains (or loops). All the links of a planar linkage move in parallel planes. The simplest closed loop planar linkage is the four-bar, which has three moving links, four pin joints, and a single degree of freedom. The link used to impart the motion is called *input link* or *crank*. As the crank rotates, the *coupler link* rotates and translates, so that a *coupler point* on it traces a planar curve, called the *coupler curve*. This curve, with the corresponding crank and coupler rotations, constitutes the basic functional characteristics of the linkage. A four-bar linkage is shown in Figure 1. In the rest of the paper we will often use the term linkage to refer to a four-bar planar linkage whenever it is clear from the context what we mean.

The shape of the coupler curve depends on the dimensions of the links and on the position of the coupler point in the coupler link. This is what is called the structure of the linkage, The relationship between function and structure is governed by equations of motion that can be represented using vector algebra and/or complex number algebra. Given the structure of a linkage it is simple enough to obtain its functional features. However, the converse is seldom true. Linkage design entails finding the structural characteristics of a linkage, given some desirable functional features that the linkage should possess. Since the shape of the coupler curve is described by highly non-linear equations, it is very difficult, given a desired coupler curve, to design a linkage that will produce it. As a service to designers, [Hrones and Nelson, 1951] have produced a catalogue of several thousand coupler curves.

In the case-based paradigm that we use, we store the functional and structural features of several linkage cases, retrieve the cases that are closest to the desired functional features, and then adapt them. What makes linkage design challenging for Case-Based Design is the fact that the design of a linkage cannot be easily decomposed into independent subproblems, the function-structure relationships are seldom monotonic, and the component dimensions are important even in preliminary design stages [Bose *et al.*, 1991].

2 Storing and Indexing Cases

Linkage designs that are unique in their functional characteristics are stored as cases. Since design problems consist mainly of functional specifications, we index the cases on the basis of their functional characteristics. As mentioned earlier, the functional characteristics of a four-bar linkage include the coupler curve (i.e. the path described by the coupler point), and the corresponding crank and coupler rotations.

The coupler curve can vary greatly depending on the structural parameters of the linkage. If there are no bounds on the length of the links, the number of linkage designs is infinite. However, physical realizability and physical constraints on the relative lengths of the links provide limits to the actual number of linkages. To make the design space finite we limit the link lengths (in our implementation we use 1-5 length units for the crank, 5-20 for the coupler, 5-20 for the follower; the coupler point angle ranges from $-150°$ to $150°$ inclusive). We also normalize the cases by keeping the distance between ground pivots constant. We use a value of 10 units for the distance between ground pivots. This value is arbitrary and could be easily changed. The horizontal lines shown in the examples in Figures 2 and case2 represent the plane of the ground pivots with the end points indicating the ground pivot locations.

We will now describe how we represent the functional characteristics of a linkage and motivate our representational choices. This requires introducing a few definitions.

Attributed Coupler Curve. We discretize the coupler curve and store it as an attributed curve, which we call the *Attributed Coupler Curve.* We generate an Attributed Coupler Curve by recording N_{cc} positions of the coupler point corresponding to N_{cc} positions of the crank. Each crank position is obtained by rotating the crank $360/N_{cc}^{\circ}$ from the previous position in the counterclockwise direction. All coupler curves are closed since the crank is always rotated 360°. The initial crank position corresponds to a 0° crank angle. This discretization corresponds to approximate the actual curve by straight line segments joining adjacent points in 2D space. In our experiments we use a value of 50 for N_{cc}, which corresponds to increments of the crank of 7.2°. We have found that this value of N_{cc} gives a sufficiently accurate representation of the coupler curve without creating an excessive number of points.

The coupler curve, at this level, can be viewed as a list of N_{cc} records with each record containing information on the position of the coupler point in 2-D space, and the corresponding crank and coupler rotations.

Unfortunately, this representation does not capture in an abstract way the shape of the curve, and so we need to represent the coupler curve at a higher level of abstraction. For this purpose we introduce the notion of *segments.* A segment is a component of a curve. A segment can be either an *arc segment* or a *point segment*, a point segment can be either a *cusp* or a *crossing.*

Arc segments. Arc segments are constructed using a measure of curvature called the *Arc Index.* The Arc Index is defined as $ArcIndex = sense * ceiling(log_b(\rho + 1))$, where b is a constant, ρ is the approximate radius of curvature, and *sense* is used to distinguish between concave and convex arcs.

The radius of curvature ρ is computed using the points in the Attribute Coupler Curve. We start with the first three points, and compute the intersection of the perpendicular bisectors of the two segments joining adjacent points. The distance from this intersection point to any of the three points is the approximate radius of curvature. The process is repeated for all adjacent three point sets. Adjacent arcs with the same Arc Index are then merged to form segments.

We have experimented with different values of b. The value we used for all examples reported here is 5, which is half the distance between the ground pivots. The logarithmic transformation is used to collapse the radius of the curvature space to the arc-index space. This reduction of the feasible curvature space facilitates matching between curves during case retrieval. Also, the Arc Index is more sensitive to changes in smaller curvatures than to larger ones, and this more accurately abstracts the shape of the curve.

Point segments. There are two kinds of point segments: *cusps* and *crossings*. A cusp is defined as a point on the curve where the angle included between the straight lines representing the curve is greater than or less than certain thresholds (270° and 90° respectively). A crossing is a point at which the curve crosses itself. These two kinds of points aid in characterizing the curve, and thus aid in retrieving the right kind of curve.

Now we are ready to summarize what we have described above.

Functional Information. The functional information about the coupler curve is represented at three levels as follows:

Level 1 A case C is a closed string of segments

$$C = s_1.s_2 \ldots s_{CS}$$

where CS is the number of segments in the case. Each segment s_i can be one of the following:

$$s_i = \begin{cases} a_i & \text{the Arc Index (used for arc segments),} \\ x & \text{a crossing (a type of point segment),} \\ c & \text{a cusp (a type of point segment).} \end{cases}$$

The string is *closed* in the sense that there is in effect no first or last segment (the first segment follows the last one in the sequence).

Level 2 For each arc segment with signature a_i in C, the following attributes are recorded in an attribute-value table $A/V = \{(a, v)|\ a$ is an attribute and v is its value$\}$:

$$\begin{aligned} l_i &= \text{length of the arc segment,} \\ \theta_i &= \text{angle swept by crank thru } i^{th} \text{ segment,} \\ \alpha_i &= \text{rotation of coupler thru } i^{th} \text{ segment.} \end{aligned}$$

Level 3 The coupler curve in C is represented as an ordered set of N_{cc} points

$$C = \{p_1.p_2 \ldots p_N\}.$$

where each point is the position of the coupler point corresponding to regularly spaced positions of the crank (obtained, as described earlier, by rotating the crank $360/N_{cc}^{\circ}$). The points are ordered by following the coupler curve in the counterclockwise direction.

Each segment s_i contains the points (in order) it is made up of. In the ordering along the curve

$$\begin{aligned} s_i &= \{p_a, \ldots, p_b\}, a \leq b, \\ s_{i+1} &= \{p_b, \ldots, p_c\}, b \leq c, \\ s_{CS} &= \{p_N, \ldots, p_1\}. \end{aligned}$$

Structural Information. The structural information in a case consists of the structural dimensions that define the four-bar planar linkage. These are the lengths of all the links, and the position of the coupler point relative to the coupler link. Angles are in radians, all other dimensions in length units. The structural information is stored with the functional information at Level 3. All cases are normalized by making the distance between ground pivots equal to 10 units. These are recorded in an attribute-value table $A/V = \{(a, v) | a \text{ is an attribute and } v \text{ is its value}\}$. The attributes are:

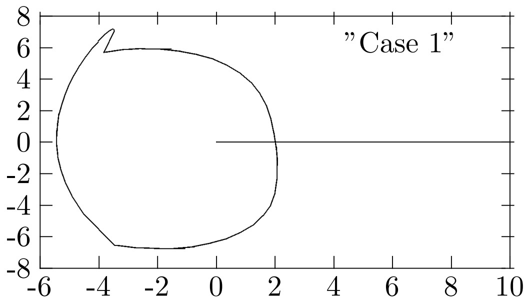
- length of the crank link,
- length of the coupler link,
- length of the follower link,
- length of the fixed link, or distance between the ground pivots. In the implementation we keep this constant and equal to 10 units,
- distance of the coupler point from the crank end of the coupler link,
- angle locating the coupler point with respect to the coupler link.

.

In summary, the coupler curve at the top level (Level 1) is represented as a *segment sequence* of arc indices and point segments. At a lower level (Level 2), each arc segment has associated 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 (Level 3), the coupler curve is represented as a list of N_{cc} records as described earlier.

One example of this representation scheme is shown in Figure 2. Case 1 has two point segments, both of which are cusps, and ten arc segments.

Another example is shown in Figure 3. Case 2 includes one crossing segment and one cusp segment, along with nineteen arc segments. Note that the segment sequence



Level 1:
4.3.c.2.c.2.3.4.2.3.4.5

Level 2 (partial description):

Arc Index	*Length of Arc Segment*
4	33
3	126
c	0
2	30
c	0
2	9
3	8
4	263
2	13
3	145
4	18
5	19

Level 3 (only the structural information is shown here):

Linkage	*Dimensions*
Length of Crank Link	5.0
Length of Coupler Link	3.0
Length of Follower Link	15.0
Length of Fixed Link	10.0
Coupler Point Radius	3.0
Coupler Point Angle	0.15708

Figure 2: Description of Case 1

has two x's indicating the location of the same crossing point in relation to the other segments as the curve is traced segment by segment in the same order as the coupler curve is traced when the crank rotates.

3 Specifying a Problem

A problem consists of the *functional* and *structural* properties desired in the four-bar planar linkage to be designed. Problem specification are often incomplete and there might be multiple designs the designer has to choose from.

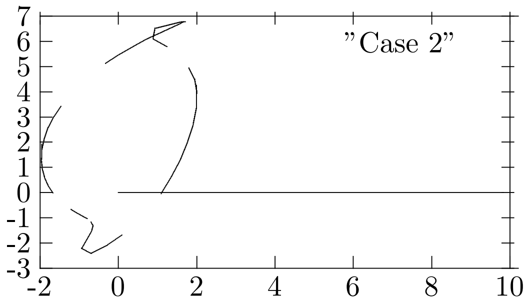
The *functional information* in a problem at the very least consists of *curve fragments* that the coupler curve should contain. Each curve fragment is specified, at the lowest level of abstraction, as a list of points. There may be more than one curve fragment specified. A tolerance is specified in conjunction with the curve fragments, thus making the constraints on the coupler curve "tubular" in the sense that the trace has to lie within the two dimensional "tube" formed by considering the tolerance above and below the straight line connecting the points. This is illustrated in Figure 4. Default values of tolerances (of $\pm 10\%$) are assumed if none are specified.

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. Additional information on coupler rotation requirements between given points may also be specified. These are useful in describing *motion* problems where the rotation of the coupler in relation to its translation is important. *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.

The only *structural information* that is required in a problem is the location of the two ground pivot points. Additional structural requirements that may be specified include upper and lower bounds on link length ratios, velocity of the coupler point, 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.

The problem specification is first transformed to a normal form, by performing translation, rotation, and scaling. It is then abstracted at the same three levels used for representing cases. The only major difference is due to the presence of "gaps" between the curve fragments. Segments adjacent to a gap are called *boundary* segments, all other are called *internal*. This distinction is used later in the matching process.

Segmentation of the curve fragments yields a partial string sequence (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)



Level 1:

3.2.-1.x.1.2.c.2.-7.-4.x.4.3.2.3.-6.-2.-1.-2.1.3.4

Level 2 (partial description):

Arc Index	*Length of Arc Segment*
3	33
2	27
-1	22
x	0
1	0
2	1
c	0
2	7
-7	7
-4	8
x	0
4	47
...	...

Level 3 (only the structural information is shown here):

Linkage	*Dimensions*
Length of Crank Link	4.0
Length of Coupler Link	6.0
Length of Follower Link	15.0
Length of Fixed Link	10.0
Coupler Point Radius	3.0
Coupler Point Angle	-0.733

Figure 3: Description of Case 2

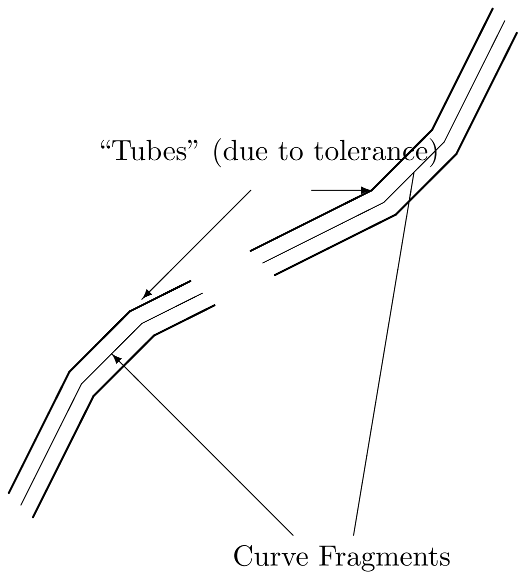


Figure 4: Effect of tolerance on curve fragments specified in a problem

are inserted at appropriate places in the string sequence to provide a regular expression[1].

Formally, a problem instance is described as follows:

Level 1 A problem P is a string of segments

$$P = \{s_{11}.s_{12} \ldots s_{1PS}.d.s_{21} \ldots s_{2PS}.d \ldots s_{F1} \ldots s_{FPS}\},$$

where:

$$\begin{aligned} F &= \text{number of curve fragments in the problem,} \\ iPS &= \text{number of segments in the } i^{th} \text{ fragment of the problem,} \\ d &= \text{dont'care; any combination of 1 or more arc segments and 0 or} \\ &\quad\ \text{more point segments.} \end{aligned}$$

Each segment s_{ik} can be one of the following:

$$s_{ik} = \begin{cases} a_{ik} & \text{the Arc Index,} \\ x & \text{a crossing,} \\ c & \text{a cusp.} \end{cases}$$

[1]A *regular expression* is a language accepted by a finite state automaton. See [Hopcroft and Ullman, 1979] for a detailed introduction.

Level 2 Each arc segment s_{ik} of the k segment in P has attributes recorded in an attribute-value table $A/V = \{(a, v)|a$ is an attribute and v is its value$\}$. The attributes are:

$$\begin{aligned} l_{ik} &= \text{length of the arc segment,} \\ t_{ik} &= \text{type of segment (boundary or internal),} \\ \theta_{ik} &= \text{angle swept by crank thru } i^{th} \text{ segment,} \\ \alpha_{ik} &= \text{rotation of coupler thru } i^{th} \text{ segment.} \end{aligned}$$

Level 3 The input curve in P is represented as an ordered set of PP points

$$P = \{p_1, p_2, \ldots, p_{PP}\}$$

The structural information, that must include at least the location of the two ground pivots, is also stored at this level.

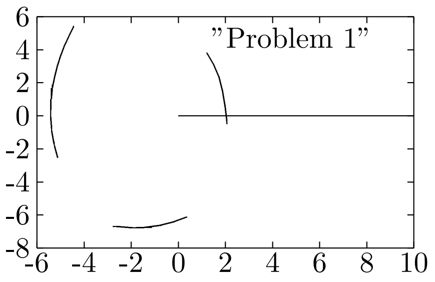
We illustrate this by showing an example of a problem specification. The problem description consist of ground pivot locations and curve-fragment sketches shown in Figure 5. The horizontal lines represent the plane of the ground pivots with the end points indicating the ground pivot locations in relation to the curve fragments. The curve fragments constitute what we call the *Problem Curve*. Each Problem Curve is made of a small number of Non-Empty Sectors (N_{s+}). The curve shown in Figure 5 has three Non-Empty Sectors.

4 The Mechanics of Case Retrieval

Cases are retrieved by matching the problem specifications to the cases available in case memory. To reduce processing, successive matches are performed hierarchically at the different levels of the representation, that correspond to different levels of abstraction. Before the process of retrieval begins all cases that have no potential are eliminated through preprocessing.

4.1 Preprocessing

Before a match between a case and the problem can be attempted, all cases that do not satisfy the global attributes of the problem are eliminated from further consideration. The global attributes are obtained from the Level 1 segment sequences and consist of the maximum Arc Index, minimum Arc Index, and the number of segments. The elimination of unsuitable cases is done by traversing a modified version of the *k-d tree*.



Level 1:

d.5.4.3.d.4.d.3

Level 2 (partial description):

Curve Fragment	*Arc Index*	*Length*	*Segment Type*
1	5	38	boundary
1	4	33	internal
1	3	14	boundary
2	4	156	boundary
3	3	62	boundary

Figure 5: Sketches of curve fragments and representation for Problem 1

The *k-d tree* is a binary tree in which each node represents a subset of the cases and a partitioning of those cases. The root of the tree represents the entire set of available cases. Each nonterminal node has two successors, that represent the two subsets of cases defined by the partitioning. The terminal or leaf nodes represent mutually exclusive small subsets of cases, which collectively form a partition of the space. Each case has k global attributes, any one of which can serve as the discriminator for partitioning. Hence the discriminating key number[2] can range from 1 to k.

We use a variation of the *optimized k-d tree* reported in [Friedman *et al.*, 1977] that minimizes the expected number of cases examined.

- The discriminator at every non-terminal node N is the one with the greatest spread

[2]In the original k-d tree proposed by Bentley in [Bentley, 1975], the discriminator for each node is chosen on the basis of its level in the tree; the discriminator for each level is obtained by cycling through the keys in order, i.e. $D = L \, mod \, k + 1$, where D is the discriminating key number for level L and the root node is defined to be at level zero. The partition values are chosen to be random key values.

Size of Case-base	*Cases Eliminated*	*Percentage of Nodes Visited*
100	65	39
200	134	36
300	198	36

Figure 6: Results of preprocessing for Problem 1

in values for the leaves of the sub-tree with root N. The spread is measured by the variance normalized by the square of the mean. The discriminators we use are the global attributes described earlier, i.e. the maximum Arc Index, minimum Arc Index, and the number of segments.

- The partition value for node N is the median value for the discriminator of all the leaves of the sub-tree with root N.

The results of preprocessing before case retrieval for Problem 1 are presented in Figure 6. The preprocessor eliminates at least 60 percent of the cases.

4.2 Matching

A match between a problem P and a case C at Level 1 occurs if the partial segment sequence of the problem is embedded in the segment sequence of the case. These embeddings are detected by first constructing a finite state automaton from the partial segment sequence, and then running the segment sequences through this automaton. If an accepting state is reached, a match has been found and the case is added to the set S_1 of candidate cases for matching at Level 2.

In order to construct a finite state automaton, the partial segment sequence of the problem P has to be first converted to a regular expression. The difference between the partial segment sequence and the regular expression is that the latter has more information on the possible values for the d's i.e. *don't cares.* Note that since d can be *any* possible combination of integers, the positive closures that replace the d's could in theory be infinitely long. However, the actual range of the arc indices we have in our case-base is -10 to 11 inclusive, and so the length of the positive closure is considerably constrained. Besides, because of the continuity constraints on smooth curves the differences between adjacent arc indices is seldom very large, and this property can be used to make the positive closure sets even smaller. If a designer desires to place constraints on the curvatures in the "gaps", these constraints can be used to further shorten the set of a particular positive closure. Details of this matching algorithm can be found in [Aho, 1990].

Match 1:

```
5.4.3.-4.-6.5.4.3.4.3.-10.3.2.5.7.6
| | |         |     |
5.4.3.   d   .4. d. 3.      d
```

Match 2:

```
5.4.3.c.2.c.2.3.4.2.3.4
| | |           |   |
5.4.3.    d    .4.d.3.d
```

Figure 7: Two of the matches at Level 1 for Problem 1

Possible alignments of the segment sequences for two of the matches for Problem 1 are illustrated in Figure 7.

Once a match is found at Level 1, a more detailed match is attempted at Level 2. Depending on the information in the problem specification, the match algorithm uses the corresponding information in the case at Level 2 (the length, the coupler and crank rotation for each arc). The tolerance provided in the problem is also used at this level.

For each case in S_1, matching at Level 2 is done using the following procedure. Each non-null value v_{pi} for the attributes in each segment s of the problem P is compared to the value v_{ci} of the same attribute of the corresponding segment in the case C. Assuming that t_i is the percent tolerance for the i^{th} attribute, a match for s occurs if

s is an internal segment and $v_{pi}.(1 - t_i) \leq v_{ci} \leq v_{pi}.(1 + t_i)$

or

s is a boundary segment and $v_{pi}.(1 - t_i) \leq v_{ci}$

The solution set S_2 at the end of matching at Level 2 is then used for matching at Level 3. For each point pp_i in P that belongs to segment sp_k, find the closest point pc_j in C that belongs to segment sc_l where sp_k has been aligned with sc_l during the match at Level 1. A match is obtained if, assuming dt_i is the threshold distance for the i^{th} problem point, for each pair of pp_i and pc_j,

$distance(pp_i, pc_j) \leq dt_i$

For Problem 1, out of a total of 36 cases that survived the preprocessing, five cases provided a match at Level 1. At Level 2, four of the cases were eliminated. The remaining case survived the matching at Level 3. The surviving case that is a solution to the problem has been described earlier as Case 1 in Figure 2.

5 Case Adaptation

Adaptation is necessary only when the differences between the problem and the functional properties of a case exceed the specified tolerances. Adaptation, in our context, entails making small changes to the structural parameters so that the functional differences become within the tolerances. The task of adaptation in general is difficult because of the non-monotonic relationships that exist between the structural and functional parameters of a linkage. There is no certainty that changing a parameter in one direction by increasing or decreasing it, will bring about a corresponding change in one direction (increase or decrease) in any functional feature. This makes it particularly difficult to qualitatively reason about the functional differences.

In our system, 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. 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.

A forward-chaining architecture is used. If more than one rule is applicable in a situation, a rule is randomly selected and then the rules fire in order of recency. This is to minimize the chance that a rule undoes what had been accomplished by a previously fired rule. The search is depth-first with a depth-bound arbitrarily selected to be 15. Once this bound is reached, the system backtracks and tries alternate paths. There is also an upper bound on the total number of trials (50). If this bound is exceeded, or no other rules are applicable, failure is reported.

Let us consider an example of an adaptation rule that is listed in Figure 8. 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,

```
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 8: An adaptation rule

there must be no crossings present, and the fraction by which the curve is to be shrunk is less than 10%. The consequents update the values of the two variables that determine the location of the tracer point on the coupler link.

In addition to the adaptation rules, we have developed another method for adapting the retrieved cases to the problem specification, based on algorithmic constraint satisfaction (Modifier Matrix Method). The Modifier Matrix Method is an algorithmic method that transforms a retrieved case to suit the given problem by computing an approximate mapping between the structural changes and the functional changes of a four-bar planar linkage. More details on both adaptation methods and a complete listing of the adaptation rules are given in [Bose, 1992].

6 A complete example

Since retrieval and adaptation are intimately related in our paradigm, we now present a rather complete example that includes a little of all that has been discussed till now.

The problem specification, shown later in Figure 12 superimposed on the solution found, includes two curve fragments. The ground pivot locations and the tolerance of the curve fit are specified, but no angular constraints are provided. Hence, this is what in linkage design is called 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 are normalized. Normalization entails the three elementary transformations of scaling, translation, and ro-

Level 1:

d.9.-9.-10.d.7.6.5

Level 2 (partial description):

Curve Fragment	*Arc Index*	*Length*	*Segment Type*
1	9	26	boundary
1	-9	79	internal
1	-10	23	boundary
2	7	10	boundary
2	6	17	internal
2	5	20	boundary

Figure 9: Representation for Problem 2

tation on the basis of the ground pivot locations. This automatically takes care of the *fit_pivot* conditions. In the next step, the curve fragments are segmented and the multi-level representation is constructed. A summary of the segment information is given in Figure 9. Features of the regular expression obtained from the Level 1 representation are used to eliminate unsuitable cases from the case-base during preprocessing.

One of the cases that is retrieved is shown in Figure 10. Only the information that is relevant to this problem is presented. Each segment also shows the number of points that are included within it. Needless to say, every case has additional information that is organized as described earlier in Section 2.

A finite state automaton is constructed from the regular expression of the problem fragments[3]. When the segment sequence of the candidate case is run through the automaton, as shown in Figure 11, 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 11. 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, has then to be evaluated to see if it can be adapted to

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

Level 1:

9.-9.-10.9.8.7.6.5.6.7.8.7.6.5.6.7.8

Level 2 (partial description):

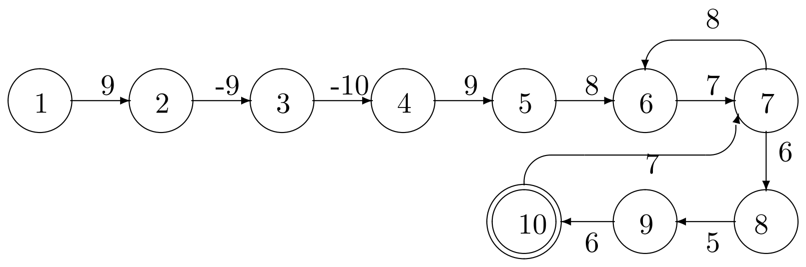
Arc Index	*Length of Arc Segment*	*Number of Points*
9	26	3
-9	86	6
-10	24	3
9	20	3
8	16	3
7	12	3
6	16	4
5	19	5
6	18	4
7	64	7
8	163	13
7	82	9
6	33	6
5	14	4
6	34	5
7	18	3
8	22	3

Level 3 (only the structural information is shown here):

Linkage	*Dimensions*
Length of Crank Link	4.0
Length of Coupler Link	6.5
Length of Follower Link	10.0
Length of Fixed Link	10.0
Coupler Point Radius	6.0
Coupler Point Angle	45°

Figure 10: Description of a candidate case for Problem 2

```
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       .7.6.5. d.
```



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

Figure 11: 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

solve the problem.

For the first alignment, the condition *fit_tube* is true for fragment 2, but not for fragment 1. The actual fit is illustrated in Figure 12. A displacement vector metric is used to compute the approximate distance and the direction the relevant portion of the case curve needs to be moved in to fit into the tube for fragment 1. The value, 4 %, and the direction, inwards, causes the system to accept the case because of the existence of rule *Shrink_Curve* (shown earlier in Figure 8). The rule *Shrink_Curve* is activated, and, as a result, the location of the tracer point with respect to the coupler is changed. Before accepting the solution, the modified coupler point curve is computed and checked against the problem specifications. The new curve fits in the “tubes” and *fit_tube* is now true for both fragments, as illustrated in Figure 13. Hence, one solution has been found.

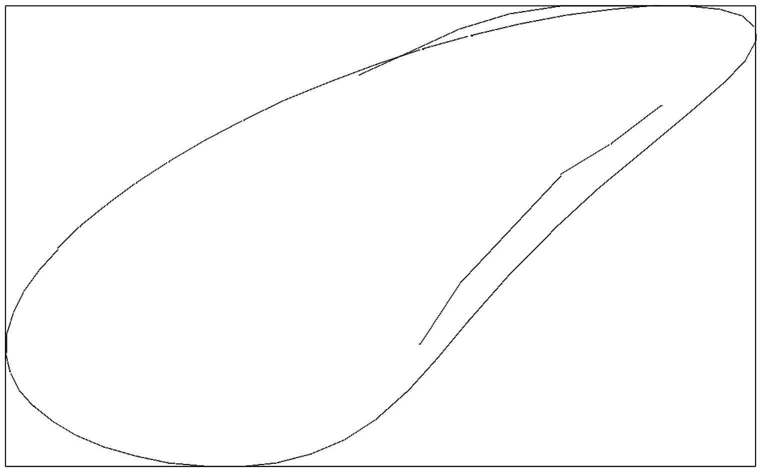


Figure 12: The coupler curve from the *retrieved* case compared to the curve fragments of the problem

7 Experimental Results

The methods we described for storing cases, retrieving the relevant ones, and adapting them based on their differences with the problem at hand, were tested through extensive experimentation.

For our experiments, we randomly generated 300 cases of crank-rocker mechanisms, which are four-bar linkages where the crank rotates 360° but the follower link only oscillates. In [Hrones and Nelson, 1951] approximatively 7,000 such mechanisms are listed. The main reason for random generation is that we want to cover the search space without introducing any bias in the selection of the mechanisms.

The distribution of the segment indices for all the cases we have generated is shown in Figure 14. The distribution has two peaks, the largest corresponding to a positive Arc Index, the smallest to a negative Arc Index. The fact that the frequencies for the positive arc indices are much higher than the corresponding negative indices, indicates that the coupler curves are predominantly concave. With a few simplifying assumptions, we have estimated the theoretical bounds for the magnitude of the arc indices to be 1 and ($\approx$)45. As can be seen from the distribution, the upper bound is never achieved, indicating that the coupler curve never approaches the limiting case of a straight line.

The problems used to test the ability of the system to retrieve and adapt cases were

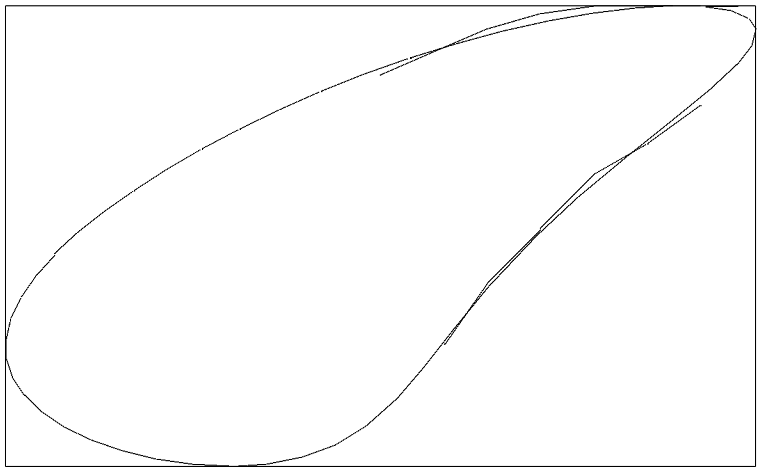


Figure 13: The coupler curve from the *adapted* case compared to the curve fragments of the problem

created by selecting specific cases from the case-base (called *reference cases*). Out of the case-base of 300 cases, we selected 30 reference cases with "interesting" features while keeping in mind that all features were represented. Some features that were considered were crossings, cusps, "balloon curves", "banana curves", "pear curves", etc. The coupler curves of the 30 reference cases we used are shown in [Bose, 1992].

Once a reference case was selected, five problems were generated from each reference case by making specific alterations to a specific part of the coupler curve. These alterations were one or more of the following types: deletion, translation, rotation.

For purposes of alteration, the coupler curve was divided into five parts, and alterations were done to one or more parts, depending on the number of Non-Empty Sectors (N_{s+}) in the problem curve. Since each case is stored at Level 3 as a list of 50 points, we found it convenient to split it into five sectors with ten points in each sector. We always start from the lower right portion of the curve because that is the general area where the curve begins to be traced as the crank angle increases from zero in the counterclockwise direction.

As we already mentioned, each problem was derived from a reference case. We generated problems in this manner to ensure that the problems were not so different from the reference cases to make it impossible to retrieve any useful case. The search space is very large (more than 7,000 mechanisms) but only a small portion of it (300

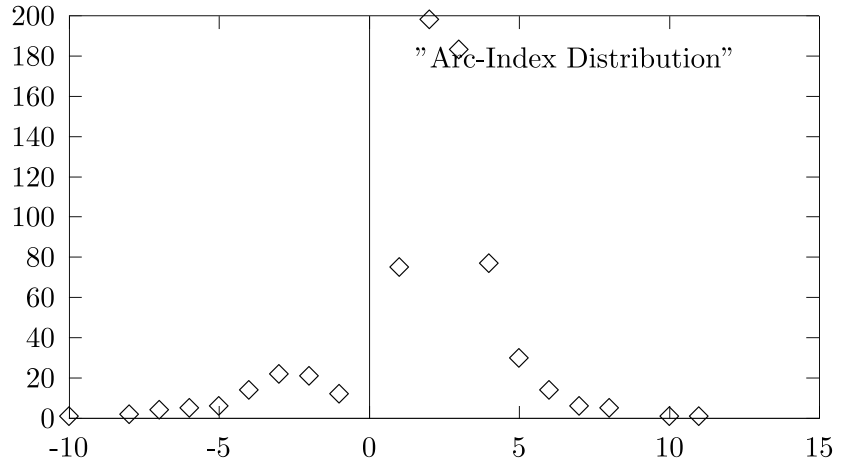


Figure 14: Frequency distribution of the arc indices for a case-base size of 300. In the case-base the number of crossings is 6, the number of cusps is 16

cases) is in the case-base. Arbitrarily sketching curves would not provide a reliable indicator to the efficacy of our methods.

The experiments we describe here were designed to study the efficacy of our methods for a different number of Non-Empty Sectors (N_{s+}) in the problem curve. For each problem derived from a reference case, N_{s+} is varied from one to five. So, we considered five instances of each problem with a number of non-empty sectors ranging from N_{s+} = 1 to N_{s+} = 5. The first instance of the problem has a fragment in the SouthEast sector, and for each increment in N_{s+} for the problem, we add another fragment as we move through the sectors in the counterclockwise direction.

The design space in this case is real-valued both along the function and structure dimensions, and in order for our paradigm to be effective, each case along with its adaptation strategies, should cover as large of the real-valued space around it as possible. So, in effect, our testing is geared towards finding out the efficacy of our retrieval and adaptation methods for problems with solutions in the space around a reference case, and also the extent of this space.

The system is implemented in C++ and CLP(R) on a Unix workstation.

The average number of cases eliminated by preprocessing for the five values of N_{s+} are plotted in Figure 15.

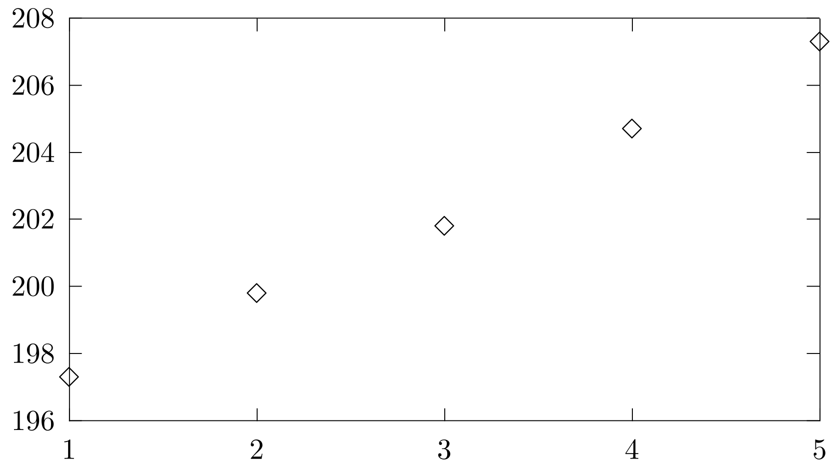


Figure 15: The average number of cases eliminated by preprocessing is shown as a function of the number of Non-Empty Sectors N_{s+} in the problem curve. The number increases with the number of Non-Empty Sectors N_{s+}.

The results of Level 1 retrieval indicate the number of cases retrieved at level 1 for each instance of the problem. Problems where the arc-index ranges lie close to the "twin-peaks" of the arc-index distribution of the case-base (Figure 14) cause a greater number of cases to be retrieved.

Level 2 retrieval results indicate that the cases that survive have parts of their case curves with the same approximate shape as those of the problem curve. Hence there is a big drop in the number of cases retrieved at Level 2 for each instance of the problem. Cases with similar shapes are retrieved even when the location of the case curves in relation to the ground pivots are quite different. This is because the representation and retrieval until Level 2 is based on the shape of the curves and not their locations along the coordinate axes.

The number of cases retrieved at Level 3 indicate that, in most instances, a case is uniquely identified by the time $N_{s+} = 3$. This is because the curves of the cases in the case-base are unique and different from each other, if not in shape then at least in location with respect to the ground pivots, and by the time $N_{s+} \geq 3$, a case is, in most instances, uniquely identified.

The results of retrieval at the three levels are plotted in Figure 16. As expected, an increase in N_{s+} causes a smaller number of cases to be retrieved at all three levels,

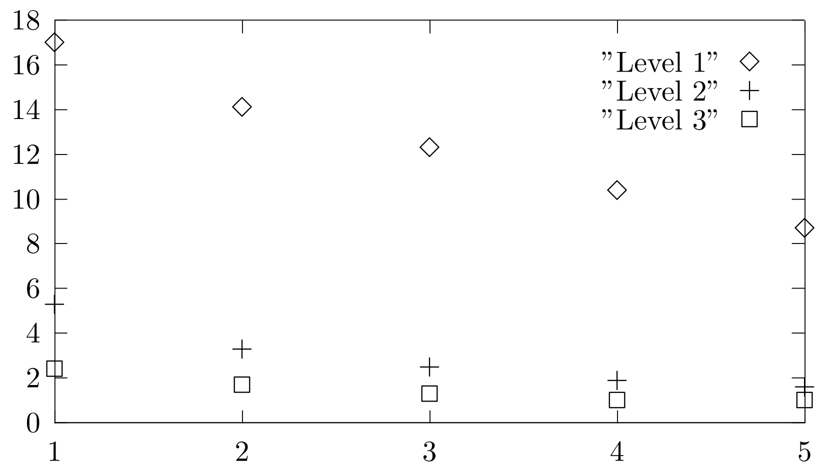


Figure 16: Number of cases retrieved at the 3 levels for different numbers of Non-Empty Sectors N_{s+}.

though the effect becomes less pronounced at the later levels.

Once retrieval at Level 3 is complete, we use adaptation rules to change linkage dimensions to tailor the retrieved case to the specifications of the problem curve for each problem.

The results for 30 problems with different numbers of non-empty sectors are presented in Figure 17.

As it can be seen from the results, some problems were harder to solve than others, because of the alterations we performed on the coupler curves of the reference cases to construct the problems. For example, if a crossing was present in the reference case, it added complexity to the retrieval and adaptation, particularly if the crossing was left out of the problem.

8 Computational Complexity

The segmenting algorithms for both the problems and cases have a space and time complexity of $O(N_{cc})$ where N_{cc} is the number of points used to define the attributed coupler curve. The preprocessing algorithm is based on the optimized *k-d tree* described in [Friedman *et al.*, 1977]. The number of nodes in the tree, assuming a case-base of N cases, is $2N-1$ (N terminal leaves + $N-1$ nonterminal nodes). Hence the tree requires

Problem Number	Solution found for $N_{s+}=$				
	1	2	3	4	5
1	yes	yes	yes	yes	yes
2	yes	yes	no	no	no
3	yes	yes	no	no	no
4	yes	yes	yes	yes	no
5	yes	yes	no	no	no
6	yes	no	no	no	no
7	yes	yes	yes	yes	yes
8	yes	yes	yes	yes	yes
9	yes	yes	yes	yes	yes
10	yes	yes	no	no	no
11	yes	yes	yes	no	no
12	yes	yes	yes	yes	yes
13	yes	yes	yes	yes	yes
14	yes	yes	no	no	no
15	yes	yes	no	no	no
16	yes	yes	yes	yes	yes
17	yes	yes	no	no	no
18	yes	yes	yes	no	no
19	yes	yes	yes	yes	yes
20	yes	yes	yes	no	no
21	no	no	no	no	no
22	yes	yes	yes	yes	no
23	yes	yes	no	no	no
24	yes	yes	yes	yes	no
25	yes	yes	yes	no	no
26	yes	no	no	no	no
27	yes	yes	yes	yes	no
28	yes	yes	yes	no	no
29	yes	yes	yes	yes	no
30	yes	yes	yes	no	no
$\sum$yes	29	27	19	13	8

Figure 17: Number of problems solved (out of 30) for different N_{s+}.

$O(N)$ space. Since all k attributes of all cases have to be inspected at each level of the tree, and the tree has depth $logN$, the construction of the tree requires $O(kNlogN)$ time. The expected search time to find the promising cases is $O(logN)$.

We use a deterministic finite automaton for the matching at Level 1. The algorithm we use is described in [Adelson, 1989]. This algorithm runs in $O(2^m + n)$ time and $O(2^m)$ space where m is the number of segments in the problem and n is the number of segments in a case. The matching at Level 2 runs is $O(m)$ time with constant space. At Level 3 where a detailed match occurs, the time is $O(N_{cc})$.

9 Related Work

A large amount of work has been done in the design and simulation of linkages and mechanisms using ideas and methods from Artificial Intelligence [Gero, 1992]. Among the approaches most used are geometric constraints [Kramer, 1992], qualitative methods [Faltings, 1990, Weld and de Kleer, 1990], expert systems [Bertini, 1993], or mixed methods [Joskowicz and Sacks, 1991].

In the case of four-bar planar linkages, Hoeltzel and Chieng [Hoeltzel and Chieng, 1989] developed a system called Pattern Matching Synthesis (PMS). From a set of parametrically generated coupler curves, their system extracts a set of clusters, that are then used as patterns for training a neural network. The neural network can retrieve the entire curve (with the associated linkage dimensions) that best matches a given curve. This method cannot deal with incomplete information. Hoskins and Kramer [Hoskins and Kramer, 1993] describe how to construct and train a neural network to compute the parameters of the four-bar linkage that best approximates a user-specified curve. The resulting design is then tuned using optimization techniques. Also this method requires a completely specified curve and so it won't work with partial problem specifications.

Kota et al.[Kota *et al.*, 1988a] describe a system called MINN-DWELL for interactive kinematic synthesis and analysis of multi-link dwell mechanisms. The designer selects a linkage from an atlas of four-bar linkages containing approximatively 350 different linkages. The system then derives the design parameters and computes the four-bar linkage parameters. An expert system that helps with the initial selection of models for dwell linkages is reported in [Kota *et al.*, 1988b]. Both systems operate at a more abstract level than ours and do not allow the user to specify geometrically the desired coupler curve.

Our approach to four-bar linkage design uses Case-Based Design, a relatively new computational paradigm. Case-Based Design has been used for other synthesis applications, such as the design of faucets in the system CADET [Navinchandra *et al.*, 1991]. CADET focuses on functional behavior and qualitative modeling, and does not deal with geometric aspects like we do. As we discussed earlier in the paper, designers often use past cases to help them in their design, and so case-based design appears an obvious choice. Our contribution has been in extending the applicability of case-based methods to problems that are characterized predominantly by their geometry.

Most work in segmenting and indexing curve segments has been performed in Computer Vision [Shapiro and Haralick, 1981]. String matching algorithms are sometimes used to detect matches between a line drawing and a template. Fourier transformations have also been used in matching line drawings. Reference [Dougherty and Giardina, 1988] contains an overview of these techniques. Our matching method allows for incomplete information (the *don't care* elements in the string) and allows matching strings of different length.

Our method for representing the coupler curve is similar to geometric hashing methods [Lamdan *et al.*, 1990, Lamdan and Wolfson, 1988]. Geometric hashing relies on computing invariant geometric features for the models and storing them in a hash-table. Given an object to be recognized, its features are matched against the model features at the appropriate place in the hash-table. A voting scheme is used to select the best candidates for full matching. Curve geometric hashing [Wolfson, 1990] uses shape signatures, such as the curvature of a curve, to index curve segments. A curve is then converted into a shape signature string. String matching algorithms are used to find the longest matching subcurve that appears in two curves.

Instead of a hash-table, we use the hierarchical representation we described earlier. In addition to the geometric specification of the coupler curve there are other attributes needed to specify a four-bar linkage. These other attributes, such as the location of the ground pivots, are not used for retrieval at the Level 1 but start playing an important role at Level 2. The arc indexes we use to represent curves are different from the curvature representation commonly used in geometric hashing. Our representation reduces the curvature space and so facilitates matching during case retrieval.

Our representation is particularly appropriate for hierarchical matching. Cases that survive the first level match are then matched more accurately at the second level, and finally at the third level. Reducing the number of candidates as soon as possible without eliminating any of the correct cases decreases the overall processing time. We use optimized k-d trees for preprocessing, which reduces retrieval time. Considering that the set of cases could be extremely large, savings in retrieval time are very important for the success of any case-based approach. We also allow the user to specify different tolerances for different points on the coupler curve. Tolerances are used in the matching process, and by having the user specifying what is acceptable we give him/her finer control over the retrieval and modification of similar cases.

10 Concluding remarks

We have presented a method to store and retrieve design artifacts based on functional features that can be defined in terms of planar curves. Retrieval is inexact and can be done using incomplete specifications. This method can be extended to cover other domains where cases can be described in terms of planar curves, with a few changes in the representation and abstraction schemes to suit the particular domain under consideration. Experimental validation studies show the viability of our methods.

Since no system is perfect, we have found a few weaknesses in our method of representation and retrieval. The most obvious one is the effect of non-smooth descriptions of the problem curve on the derived partial segment sequence. Any unnecessary undulations in the sketches will yield additional members in the segment sequence that could prevent matches with cases. One way to overcome this problem is to "smooth" the partial segment sequence using a smoothness operator like the averaging neighbors used in [Wolfson, 1990].

Another problem has its roots in the implicit assumption that a designer, when sketching the problem curves will specify fragments that are "complete" in the sense that a solution could not possibly have just *one* segment that would match with part of one fragment and part of another. This assumption manifests itself in matching using regular expressions, where the same input string member (a segment in a case) cannot match with more than one member in the pattern (the problem partial sequence). This assumption can be eliminated by using algorithms for string "warping" where multiple matches between one member of one string and two or more adjacent members of the other string are possible, at a greater computational expense.

There is room for expanding the scope of this work also. In the adaptation of the cases retrieved, we have concentrated on path problems, but all our representation schemes and matching algorithms are general enough to handle motion and function problems. The rule-base can be extended to cover these problems. In generating the test cases, we have limited our scope to crank-rocker mechanisms and did not consider more than one branch of a linkage, but our retrieval and adaptation methods are general enough to handle other mechanisms and multiple branches of a linkage. Attention could also be paid to higher order linkages, especially the six and eight-bar linkages.

References

[Adelson, 1989] B. Adelson. Cognitive research: Uncovering how designers design; cognitive modeling: Explaining and predicting how designers design. *Research in Engineering Design*, 1(1):35–42, Jan-Mar 1989.

[Aho, 1990] 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.

[Bentley, 1975] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[Bertini, 1993] Leonardo Bertini. A prototype expert system for embodiment design of mechanisms and articulated systems. *AI in Engineering*, 8:57–65, 1993.

[Bose *et al.*, 1991] A. Bose, A. Esterline, D.R. Riley, and A.G. Erdman. Case-based preliminary linkage design. In *Proceedings of The Design Productivity Conference, Hawaii*, 1991.

[Bose, 1992] A. Bose. *Case-based Design of Planar Linkages*. PhD thesis, University of Minnesota, November 1992.

[Dougherty and Giardina, 1988] E.R. Dougherty and C.R. Giardina. *Mathematical Methods for Artificial Intelligence and Autonomous Systems*. Prentice-Hall, Inc, 1988.

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

[Faltings, 1990] Boi Faltings. Qualitative kinematics in mechanisms. *Artificial Intelligence*, 44:89–119, 1990.

[Friedman *et al.*, 1977] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[Gero, 1990] J.S. Gero. Design prototypes: A knowledge representation schema for design. *AI Magazine*, 11(4), 1990.

[Gero, 1992] J.S. Gero, editor. *Artificial Intelligence in Design*. Kluwer Academic Publ, 1992.

[Hoeltzel and Chieng, 1989] 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.

[Hopcroft and Ullman, 1979] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Inc, 1979.

[Hoskins and Kramer, 1993] J.C. Hoskins and G. A. Kramer. Synthesis of mechanical linkages using artificial neural networks and optimization. In *Proceedings of 1993 IEEE International Conference on Neural Networks (ICNN '93)*, volume 2, pages 822J–822N, 1993.

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

[Joskowicz and Sacks, 1991] L. Joskowicz and E. P. Sacks. Computational kinematics. *Artificial Intelligence*, 51:381–416, 1991.

[Kota *et al.*, 1988a] 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.

[Kota *et al.*, 1988b] S. Kota, A.G. Erdman, D.R. Riley, A. Esterline, and J.R. Slagle. A network based expert system for intelligent design of mechanisms. *AI in Engineering Design and Manufacturing*, 2(1):17–32, 1988.

[Kramer, 1992] G. A. Kramer. A geometric constraint engine. *Artificial Intelligence*, 58:327–360, 1992.

[Lamdan and Wolfson, 1988] Y. Lamdan and H.J. Wolfson. Geometric hashing: a general and efficient model-based recognition scheme. In *Second International Conference on Computer Vision*, pages 238–249, 1988.

[Lamdan *et al.*, 1990] Y. Lamdan, J.T. Schwartz, and H.J. Wolfson. Affine invariant model-based object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(5):578–598, 1990.

[Maher, 1990] M. L. Maher. Process models for design synthesis. *AI Magazine*, 11(4):49–58, 1990.

[Navinchandra *et al.*, 1991] D. Navinchandra, K. Sycara, and S. Narasimhan. A transformational approach to case-based synthesis. *AI EDAM*, 5(1):31–45, 1991.

[Riesbeck and Schank, 1989] C.K. Riesbeck and R.C. Schank. *Inside Case-based Reasoning*. Lawrence Erlbaum Associates, Publishers, 1989.

[Shapiro and Haralick, 1981] 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.

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

[Weld and de Kleer, 1990] D. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufman, 1990.

[Wolfson, 1990] Haim J. Wolfson. On curve matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):483–489, 1990.