Decision Trees and Genetic Programming in Synthesis of Four Bar Mechanisms

Anikó Ekárt and **András Márkus**
Computer and Automation Research Institute
Hungarian Academy of Sciences
1518 Budapest, POB. 63, Hungary
E-mail: {ekart;markus}@sztaki.hu

Abstract

Kinematic synthesis of four bar mechanisms is a design problem that is difficult to solve by generative methods. The present approach is a variant based method that combines the genetic programming and decision tree learning methods. The aim of the research is to give a structural description for the class of mechanisms that produce desired coupler curves. For finding and characterizing feasible regions of the design space constructive induction is used. The new features are created by genetic programming.

1 Introduction

A mechanism is defined as an arrangement of machine elements that produce a specified motion [10]. The synthesis of a mechanism is the process of combining parametric elements into a mechanism that shows complicated behavior. In the paper a simple, but practically important class of mechanisms is investigated: four bar mechanisms. Figure 1 shows the structure of the four bar mechanism, where a represents the fixed link, b the input link, c the coupler link, d the follower link, and point P the tracer point.

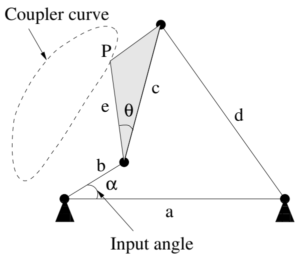


Figure 1: The four bar mechanism layout.

From the kinematic point of view four bar mechanisms can be designed for: (1) path generation, (2) rigid body guidance, and (3) function generation. The current application is related to the path generation problem: given certain path fragments, find the mechanism (i.e., its structural parameters) whose coupler curve contains these fragments. A path fragment is given as an ordered set of points. In the case of path generation with prescribed timing, there is associated with each point an input angle (α in Figure 1) as well. Since for points more than five the problem is overconstrained, the acceptable tolerance between the input path fragments and the coupler curve is also specified.

For path generation with prescribed timing, the classical analytical approach [10] is limited to the case of five specified points. Otherwise, there is no general recipe for choosing the structural parameters for the mechanism that approximates a given path. Recent work has been done for finding numerical methods for the general case: e.g., mechanism synthesis is considered as a nonlinear programming problem and an exact gradient method is used for dimensional synthesis [7].

For the situation when more than five points are specified there have been developed some variant based methods, such as case-based reasoning [3]: after a multi-level case retrieval process, adaptation is accomplished by simple transformation rules (increasing or decreasing the length of one link by a small percent). However, the method is applicable only in cases when the coupler curve contains no crossings or there is a close-enough curve in the case-base. In another work [4] neural networks are used for learning and synthesizing mechanisms for coupler curves similar to the ones stored in the knowledge base. Since the coupler curves are stored as bitmaps, pattern matching is used for finding similar curves.

Unlike the previous approaches, we do not generate a single mechanism, but give structural constraints for mechanisms whose coupler curves contain the desired path fragments. Any mechanism satisfying these constraints will meet the input requirements. If there are further restrictions for the acceptance of a mechanism (such as dimensional limits of the links), the structural description should be refined accordingly.

2 The Design Method

The goal of this work is to provide such a description of the feasible mechanisms, that can be exploited by the designers of mechanisms.

Particularly, we create the structural description of the four bar mechanisms that satisfy the input requirements. That is, given some curve fragments and a corresponding tolerance, the coupler curve generated by any acceptable mechanism must be within the given tolerance limit from these curve fragments. The admissible mechanisms are given by constraints on their structural parameters. Any mechanism satisfying these constraints can be designated as a solution for the given family of path generation problems.

The method consists of the following steps (where steps 2-4 represent the definition of the actual design problem): (1) creation of a catalog of four bar mechanisms; (2) specification of input requirements; (3) classification of the elements of the catalog; (4) generation of the structural description.

The main result is that the design space can be described by constraints on the structural parameters (a,b,c,d,e,θ) of the mechanisms. For any desired path fragment, we give the structural description of the mechanisms that produce coupler curves similar to it. By this structural description the class of mechanisms that produce similar coupler curves is given. In [4] there is made a classification of four bar mechanisms based on the form of coupler curve, but no structural description for the members of a class is provided.

Our catalog of four bar mechanisms contains 7276 elements and was created as a computer version of [5]. Each element consists of the structural and functional description of a mechanism. The structural description contains the parameters of the mechanism, as shown in Figure 1. The functional description is the coupler curve generated by the mechanism and it is recorded as an ordered list of points. We call mechanism space the universe of four bar mechanisms. The dimensions of this space are the structural parameters of mechanisms. The elements of the catalog are points of the mechanism space, with the structural parameters $a \in \{1.5, 2, 2.5, ..., 6.5\}$, $b = 1$, $c \in \{1.5, 2, 2.5, 3, 3.5, 4\}$, $d \in \{1.5, 2, 2.5, 3, 3.5, 4\}$, e and θ taken for 50 sampled points on a rectangle attached to the coupler link of the mechanism (see Figure 2). This rectangle extends the coupler link c in directions parallel and perpendicular to it with a distance equal to the length of the input link b.

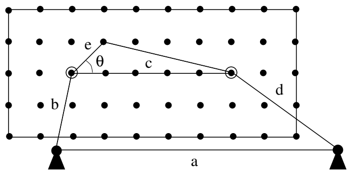


Figure 2: The sampled points on the coupler link.

In the second step the desired curve fragments are specified as a list of points and corresponding input angles, (i.e., our problem is path generation with prescribed timing) and the tolerance is also given.

In the classification step the mechanisms are grouped into two classes: the **positive** and the **negative** class. A mechanism is positive, if its coupler curve fulfills the input requirements. The mechanism is negative in the opposite case. A coupler curve corresponds to the input requirements when it can be moved (translated and rotated) so that its similarity to the desired curve is within the tolerance limit. The similarity of a coupler curve to a curve fragment is computed in the following way: (1) to each point of the curve fragment the closest point of the coupler curve is associated (without breaking the order of points); (2) the distances of these point pairs are computed; and (3) the maximum of the distances is designated as similarity value (i.e., each such distance must be within the tolerance limit for a positive mechanism).

The key issue is the generation of the structural description of the positive class. The structural description consists of a set of constraints for the structural parameters. The description can be written as a DNF (disjunctive normal form) formula. Two machine learning methods have been applied: decision tree induction and genetic programming.

Decision tree induction was effective in the cases when the elements of the positive class were situated in a convex region of the mechanism space and the mechanism space could be partitioned by planes parallel with the axes. However, it produced too many errors in the other cases. We needed a method that could create descriptions corresponding to as many elements of the positive class as possible, regardless whether they were situated in a convex region of the space and allowing other partitions. Genetic programming was our next choice. Unfortunately, in most of the cases the descriptions produced by genetic programming were not accurate enough. Since decision tree induction is very fast and accurate, when presented with the appropriate attributes, and, on the other hand, genetic programming is a plausible tool for creating new attributes, we decided to apply constructive induction [11]. This step is discussed in detail in the next section.

3 Creating the Structural Description

Generating the description of one class can be seen as partitioning the space into two regions: positive and negative. The generated description corresponds to the positive region, and no element of the negative region satisfies this description.

3.1 Decision Tree Learning

Decision tree induction is a commonly used method for concept learning [9]. The input data is a set of examples of the concept to be learned. An example consists of a set of attributes and belongs to a class. When building up the tree and selecting the next attribute to be tested at a certain node, the information gain is computed for each candidate attribute. Then, the attribute with the best gain is selected.

For studying the power of decision trees in our problem, we used the C4.5 system [9]. Each example is a mechanism, having its structural parameters as attributes and the class rating computed in the previous step. Since the attributes have continuous numeric values, at each decision node the cases are separated into two groups according to whether the value of the tested attribute is less or greater than a threshold value. Case studies are presented in Section 4.

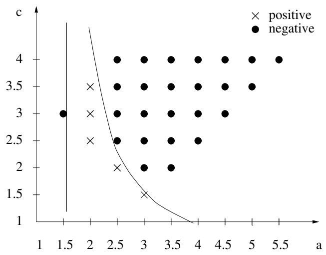


Figure 3: The cross-section of the mechanism space.

A typical example, where C4.5 produced many errors, is shown in Figure 3. Attributes $b = 1$, $d = 3$, $e = 1.41$, $\theta = 2.36$ and the distribution of positive and negative examples in the mechanism space over coordinates a and c is drawn. For this cross-section, the decision tree produced by C4.5 misclassifies 3 of the 30 examples. The explanation is that this plane could not be partitioned by straight lines parallel to the axes of the coordinate system. Thus, using the original attributes C4.5 could not produce an exact classification. If the attributes were transformed, a better partition could be found. An exact description (see Figure 3) of the positive region of the plane found by our program is

$$(a - 1.55) * (2.31/c + 1.56 - a) > 0.$$

3.2 Genetic Programming

As an alternative, we also generated structural descriptions of mechanism classes by using genetic programming [6]. Genetic programming is an extension of genetic algorithms, where the structures undergoing adaptation are not strings but hierarchical computer programs of dynamically varying size and shape.

The algorithm of genetic programming systems is the following:

1. Generate an initial population of individuals (computer programs consisting of random compositions of functions and terminals from the function and terminal set).
2. Execute each program of the population on the fitness cases and assign it a fitness value, using the fitness measure.
3. Create a new population of individuals by reproduction, crossover and mutation.
4. Iterate through steps 2-3 until the termination criterion is satisfied. The fittest individual that appeared in any generation is designated as the result of genetic programming.

We used the genetic programming paradigm with the setting shown in Table 1. A genetic program was a tree with a fixed number of branches (i.e., automatically defined functions or ADFs). Each branch represented a condition on the structural parameters of the mechanisms. Any genetic program was evaluated to the conjunction of the conditions contained in its ADFs. The following is an example of possible condition:

$$a \,/\, b * sin(\theta) - 1.2 > 0.$$

Table 1: Genetic programming parameter setting.

Objective	Evolve the structural description of the four bar mechanisms contained in the positive class
Terminal set	a, b, c, d, e, $sin(\theta)$, $cos(\theta)$, real numbers $\in [-10, 10]$
Function set	$+$, $-$, $*$, $/$
Fitness cases	The mechanisms of the catalog
Population size	50
Crossover probability	90%
Mutation probability	10%
Selection method	Tournament selection, size 10
Termination criterion	none
Maximum number of generations	50
Maximum depth of tree after crossover	20
Initialization method	Ramped half and half

The search space is a six-dimensional cube (corresponding to the parameters of the four bar mechanisms). Here, the goal of genetic programming is to find descriptions for the positive region of the space. Usually, the positive examples are not situated in a convex region of the search space and the delimiters between positive and negative regions are often non-linear, as shown in the two-dimensional cross-section of Figure 3.

We used both crossover and mutation as recombination operators. The offspring of crossover were obtained from the two parents by selecting one subtree of each parent and exchanging these subtrees. We used point mutation, i.e., one node of the tree was randomly selected and mutated. In the case of internal nodes, mutation consisted in randomly changing the function represented by the node. In the case of leaf nodes, mutation was different for terminals representing the structural parameters of mechanisms and real constants. A structural parameter was mutated to another structural parameter, and a constant was mutated by changing its value by 0.5.

The fitness cases were the mechanisms of the catalog given by their structural parameters and class rating (positive or negative). The genetic program was run on the fitness cases, and its fitness value was computed as follows:

$$err(i) = \begin{cases} 1 & \text{if } class(i) = 0 \text{ and } gp\_class(i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$fitness = \begin{cases} 0 & \text{if } \forall i \;\; gp\_class(i) = 0 \\ 1 - \frac{1}{N} * \sum_{i=1}^{N} err(i) & \text{otherwise,} \end{cases}$$

where $err(i)$ = the error of the genetic program for fitness case i, $class(i)$ = the class rating of the mechanism contained in the fitness case (0 for negative and 1 for positive), $gp\_class(i)$ = the class rating given by the genetic program, and N = the number of fitness cases.

Thus, a 100 % fit program is an expression corresponding to some positive examples and no negative example. A program with a lower fitness value is an expression additionally satisfied by some negative examples and therefore cannot distinguish between the positive and negative examples.

We used a fixed number of conditions keeping the representation simple. When fewer conditions were needed, some ADFs of the genetic program represented conditions that were always true.

3.3 Constructive Induction

Both previous methods when applied alone had some shortcomings, hence the idea to apply them together. Decision tree learning is not always able to separate the two classes because the structural attributes presented to it are not the most relevant ones. There are several notions of relevance, let us consider **incremental usefulness** of [2]:

> Given a sample of data S, a learning algorithm L, and a feature set A, feature x_i is incrementally useful to L with respect to A if the accuracy of the hypothesis that L produces using the feature set $\{x_i\}\ U\ A$ is better than the accuracy achieved using just the feature set A.

In other words, decision trees (or any learning algorithm) can perform better, if presented with more useful attributes. New attributes could be generated as $logical\ expressions$ over the original attributes [8]. Since the original

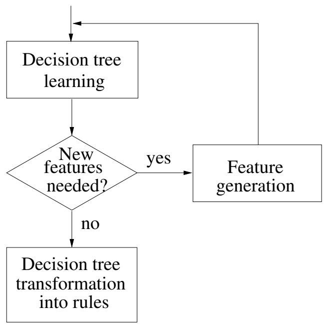


Figure 4: Constructive induction.

attributes (the structural parameters of mechanisms) are numerical values, a straightforward method is the generation of $arithmetic\ expressions$ over the original attributes [1].

Having in mind the definition for usefulness, we used C4.5 as the learning engine for the constructive induction and we applied genetic programming for attribute (feature) generation (as shown in Figure 4).

Thus, genetic programming proposes new attributes, and C4.5 uses them in addition to the original ones. The genetic programs are arithmetic expressions on the original attributes. The fitness measure is modified in order to include both kinds of errors:(1) misclassification of positive cases and (2) misclassification of negative cases. The error of the genetic program was computed as:

$$err(i) = \begin{cases} n & \text{if } class(i) = 0 \text{ and } gp\_class(i) = 1 \\ p & \text{if } class(i) = 1 \text{ and } gp\_class(i) = 0 \\ 0 & \text{otherwise} \end{cases}$$

We experimented with different values for n and p ($p \gg n$ or $n \gg p$), since we wanted to find good descriptions for both the positive and the negative examples. By obtaining a good description of the negative examples and presenting the attribute to C4.5, we expected C4.5 to eliminate many negative examples at the beginning. The rest of parameter settings for genetic programming are the same as described in Section 3.2.

Initially, we generated the new features separately from decision tree construction, as it is shown in Figure 4. The new features created by genetic programming tended to cover as many positive examples as possible. But when constructing the decision tree, at each node it is sufficient that the chosen feature describe the examples to be classified at that node. There is no need for a general feature. If we had found such a general feature, we would not have needed decision tree induction at all. Thus, in order to get new features corresponding to each node of the decision tree, we introduced the attribute generator at the level of node creation in the decision tree builder. We modified the fitness measure for the genetic programs to be the information gain (as defined in [9]) of the feature represented by a genetic program. By creating the new features at the construction of nodes in the decision tree, we obtained more accurate descriptions.

4 Experimentation

In this section two design cases are discussed in detail. We chose a circular and a linear path fragment, because these are very important fragments of the paths needed in practical mechanism design. A linear path fragment followed by a circular one is required in the film-advance mechanism of any movie camera [10], as shown in Figure 5(a).

For the first case a 100% correct description was found by using just decision trees, no other method could perform better. In the second case the description given by decision trees had errors, thus constructive induction was needed.

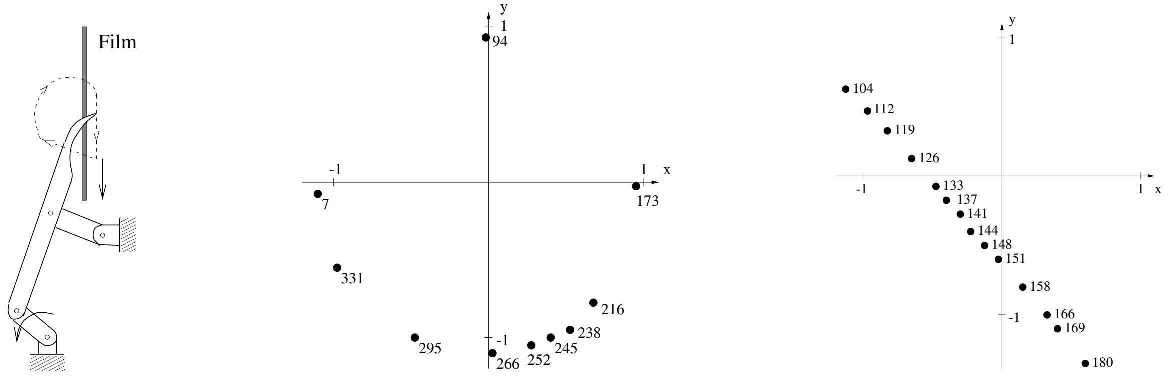
The desired curve fragments for the two case studies are given in Figures 5(b) and 5(c). The path fragment is given as an ordered list of points (the ordering is reflected in the numbering of points).

In the first case (see Figure 5(b)), the mechanism space was divided to the positive and negative classes and then C4.5 was applied. The produced decision tree classifies all the cases correctly, so that for every example presented to it the real class is the same as the predicted class.

With genetic programming, we tried out different numbers of ADFs. According to our observations, with fewer ADFs more accurate descriptions were found in fewer generations. Hence the idea of applying genetic programming for attribute generation in constructive induction.

The second desired path is a straight line approximated by the ordered point list of Figure 5(c). In this case, C4.5 produced a decision tree of size 47 containing 11 errors (i.e. misclassified cases). Our next step was the separate generation of new attributes by genetic programming, as it is shown in Figure 4.

Some of the attributes proposed by genetic programming are presented in Table 3(a). The third column (Type) specifies whether the attribute corresponds to the positive or the negative class. The last column (G) represents the generation when



(a) (b) Circular path fragment (c) Linear path fragment

Figure 5: The film-advance mechanism of a movie camera and the two corresponding desired path fragments.

the attribute emerged. Genetic programming suggests threshold value 0 for the attributes, but C4.5 may decide in favor of some other threshold value. The best decision tree had 41 nodes and misclassified 6 cases.

The next step was the introduction of the attribute generator at the level of node creation in the decision tree. Table 3(b) summarizes the results of 10 runs. The size of a decision tree is given as the number of nodes, and the errors are the misclassified cases. The number of new features actually used and their average complexity are shown. The complexity of a feature is the number of nodes in its tree representation (E.g. the complexity of the second attribute from Table 3(a) is 5). By new feature usage we mean the percentage of decision nodes in the tree where a new feature was selected.

For the straight line generating mechanisms, C4.5 alone created a decision tree of size 47 with 11 errors. When using constructive induction with independent feature generation, the best decision tree size was 41, there were just 6 errors at 60 % new feature usage. By introducing the feature generator at node creation, the best tree size became 37, with 4 errors at 66.7 % new feature usage.

Table 2: Experimental results.

Att.	Expression	Type	Fitness	G
1	$sin\,\theta\, -\, 0.7$	p	87.2	6
2	$e\, -\, a\, +\, 1$	p	61.3	1
3	$d\, /\, cos\,\theta$	n	76.4	2
4	$c\, -\, a\, -\, e\, -$ $(1\, +\, e)\, *\, cos\,\theta$	p	89.0	5
5	$a\, /\, 2.3\, -\, 5.2\, /\, sin\,\theta$	n	97.0	2
6	$8.7\, *\, sin\,\theta\, -\, 4.48$	n	84.7	1
7	$cos\,\theta$	n	76.4	9
8	$e\, -\, a$	n	80.5	3

(a) The new attributes proposed by GP.

	Decision tree size	Decision tree errors	New features nr.	New features avg. compl.	New features usage [%]
1	37	4	10	28	66.7
2	39	4	9	58	78.9
3	41	5	9	37	45
4	41	5	10	39	55
5	45	6	6	86	34.8
6	33	5	8	45	68.75
7	41	6	10	33	50
8	31	6	4	11	33.3
9	33	5	11	36	75
10	45	7	9	21	50
Avg.	39	5	9	37	55.7

(b) The results of 10 runs of the program.

5 Discussion

Let us see again the example problem presented in Figure 5(c). The coupler curves of the corresponding mechanisms (that belong to the catalog) are presented in Figure 6. It can be seen, that there are several different types of curves that all contain the straight line (within the given tolerance). As a matter of fact, the mechanisms that generate these curves differ from each other in many ways: the length of links, the position and orientation of the fixed link, the input angle corresponding to any selected point of the straight line. For example, let us consider two distinct curves and the corresponding mechanisms that generate these curves, as shown in Figures 7(a) and 7(b). The graphics are drawn at the same scale. We represented the mechanisms (together with their coupler curve) in two positions: the start (dashed lines) and the terminal position (solid lines) of the generated straight line. The input link is the emphasized link and it is rotated in the direction indicated by the arrow. For sake of distinctness, instead of drawing the fixed link, we show just its joints

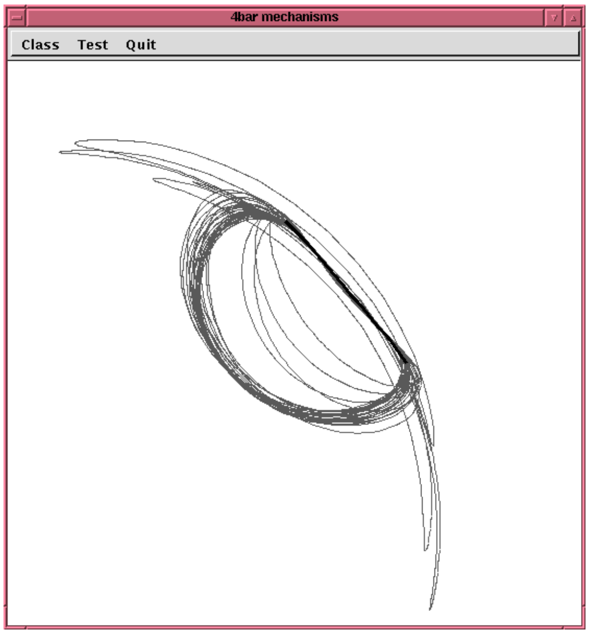
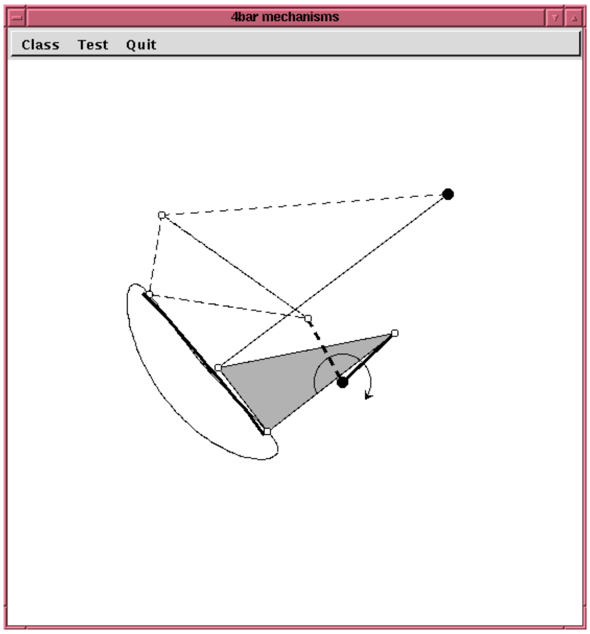


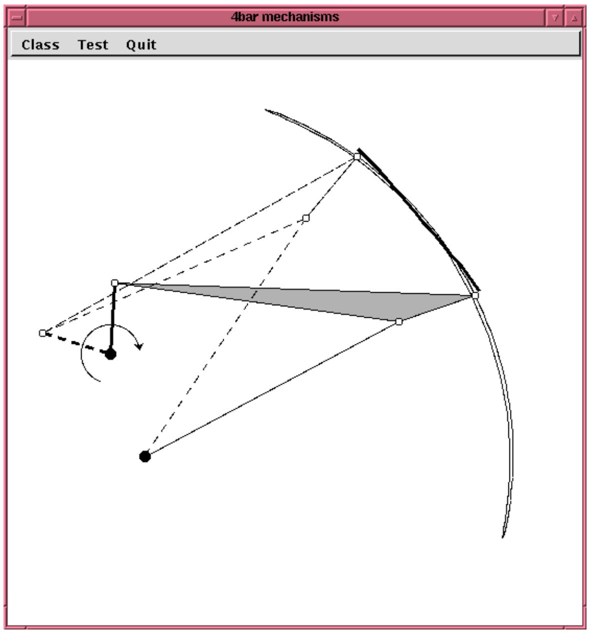
Figure 6: The coupler curves of the straight line generating mechanisms.

by the black discs.

This represents clear evidence for the fact, that the common features of such different mechanisms (belonging to the same class) are not obvious, and, therefore, discovering these common features is not an easy task. Then again, the knowledge of such features (the structural description in our terminology) could be of much help for the engineer involved in mechanism design.



(a) $a = 3,\ b = 1,\ c = 2.5,\ d = 4,\ e = 2.24,\ \theta = 0.46$



(b) $a = 1.5,\ b = 1,\ c = 4,\ d = 4,\ e = 5.02,\ \theta = 0.1$

Figure 7: Different straight line generating mechanisms.

In many cases the mechanism is a part of a complex system and it has to fit in a predefined area. That is, the mechanism must remain within the predefined area during path generation. In addition, the position and orientation of the fixed link might also be constrained. The mechanism of Figure 7(a) could be preferred to the mechanism of Figure 7(b) from the point of view of fitting in a predefined area. The choice of one or another mechanism from a class is left to the designer.

The class is defined by the found structural description and not the enumeration of the acceptable elements of the catalog. The catalog consists of points of the mechanism space and therefore, the found structural description applies not only to its elements, but to the mechanism space.

In the current implementation of the system, learning is performed on the whole catalog. In order to reduce CPU time requirements, we plan to introduce an additional training data selection step in our algorithm presented in Section 2 (between steps 3 and 4). Since the number of positive examples is much less than the number of negative examples, and omitting some of them could result in loosing important data that are not sampled elsewhere, we think that all the positive examples should be included in the training set. For the selection of negative examples to be included in the training set, two methods are to be considered: (1) random selection of a predefined number of negative examples; and (2) intelligent selection of the relevant negative examples.

We anticipate that the second method will outperform the first one and its CPU requirements will be much less than those of the already implemented method.

6 Conclusions

In this paper decision tree learning was combined with genetic programming in solving a difficult real world design problem. Instead of finding one four bar mechanism that generates a given path, we developed the structural description of the feasible regions of the design space. In simpler tasks, decision trees performed well, but in most cases constructive induction was needed. First, the new attributes were independently generated by genetic programming. Then we introduced the attribute generator at the level of decision nodes in the tree, so that for each node the attribute with the highest information gain could be created and selected properly.

Acknowledgements

This work has been supported by grant No. T023305 and T25471 of the National Research Foundation of Hungary. The authors are grateful to J. Váncza for many helpful discussions.

References

[1] Eric Bloedorn and Ryszard S. Michalski, Data-driven constructive induction, *IEEE Intelligent Systems*, 30–37, (March/April 1998).

[2] Avrim L. Blum and Pat Langley, Selection of relevant features and examples in machine learning, *Artificial Intelligence*, **97**, 245–271, (1997).

[3] Ashim Bose, Maria Gini, and Donald Riley, A case-based approach to planar linkage design, *Artificial Intelligence in Engineering*, **11**, 107–119, (1997).

[4] D. A. Hoeltzel and Wei-Hua Chieng, Pattern matching synthesis as an approach to mechanism design, *ASME Journal of Mechanical Design*, **112**, 190–199, (1990).

[5] John A. Hrones and George L. Nelson, *Analysis of the four bar linkage*, Wiley, 1951.

[6] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, Cambridge, Massachusetts, 1992.

[7] Jawaharlal Mariappan and Sundar Krishnamurty, A generalized exact gradient method for mechanism synthesis, *Mechanisms and Machine Theory*, **31**, 413–421, (1996).

[8] Christopher J. Matheus and Larry Rendell, Constructive induction on decision trees, in *Proceedings of the IJCAI*, ed., Morgan Kaufmann, pp. 645–650, (1989).

[9] J. Ross Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann, 1993.

[10] George N. Sandor and Arthur G. Erdman, *Advanced mechanism design: Analysis and synthesis*, volume 2, Prentice Hall, 1984.

[11] Janusz Wnek and Ryszard S. Michalski, Hypotesis-driven constructive induction in AQ17-HCI: a method and experiments, *Machine Learning*, **14**, 139–168, (1994).