Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2003), *17*, 205–220. Printed in the USA.

DOI: 10.1017/S0890060403173040

Using genetic programming and decision trees for generating structural descriptions of four bar mechanisms

ANIKÓ EKÁRT AND ANDRÁS MÁRKUS
Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary

(RECEIVED April 15, 2002; ACCEPTED January 12, 2003)

Abstract

Four bar mechanisms are basic components of many important mechanical devices. The kinematic synthesis of four bar mechanisms is a difficult design problem. A novel method that combines the genetic programming and decision tree learning methods is presented. We give a structural description for the class of mechanisms that produce desired coupler curves. Constructive induction is used to find and characterize feasible regions of the design space. Decision trees constitute the learning engine, and the new features are created by genetic programming.

Keywords: Decision Trees; Four Bar Mechanism Synthesis; Genetic Programming; Machine Learning

1. INTRODUCTION

A mechanism is defined as an arrangement of machine elements that produce a specified motion (Sandor & Erdman, 1984). The synthesis of a mechanism is the process of combining parametric elements into a mechanism that shows complex behavior. We investigate here a simple, but practically important, class of mechanisms: four bar mechanisms. The utilization of four bar mechanisms ranges from a simple device, such as a windshield-wiping mechanism or a door-closing mechanism to complicated devices, such as a rock crusher, sewing machine, round baler, or automobile suspension system (Norton, 1992; Waldron & Kinzel, 1999). Figure 1 shows the structure of the four bar mechanism.

From the kinematic point of view, four bar mechanisms can be designed for path generation, rigid body guidance, and 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 an input angle (Fig. 1) associated with each point as well. Because the path generation problem with more than five points is overconstrained (Sandor & Erdman, 1984), 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 (Sandor & Erdman, 1984) 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: for example, mechanism synthesis is considered as a nonlinear programming problem and an exact gradient method is used for the dimensional synthesis of mechanisms (Mariappan & Krishnamurty, 1996).

For the situation when more than five points are specified some variant-based methods have been developed, such as case-based reasoning (Bose, Gini, & Riley, 1997): after a multilevel 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 where the coupler curve contains no crossings or there is a curve in the case base that is close enough. In another work (Hoeltzel & Chieng, 1990) neural networks are used for learning and synthesizing mechanisms for coupler curves similar to the ones stored

Reprint requests to: Anikó Ekárt, Computer and Automation Research Institute, Hungarian Academy of Sciences, PO Box 63, 1518 Budapest, Hungary. E-mail: ekart@sztaki.hu

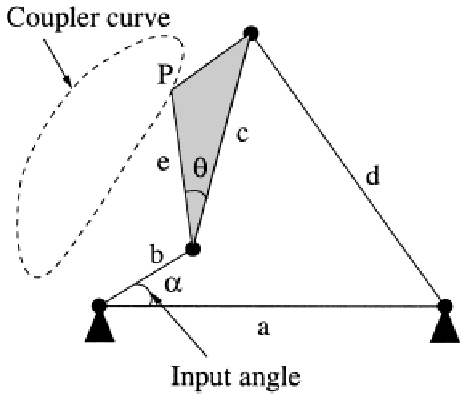


Fig. 1. The four bar mechanism layout. a, the fixed link; b, the input link; c, the coupler link; d, the follower link; P, the tracer point; α, input angle.

in the knowledge base. Because the coupler curves are stored as bitmaps, pattern matching is used for finding similar curves. Vancsay (2000) uses genetic algorithms for synthesizing a mechanism that generates a given coupler curve in a constrained environment: the fixed link must be within a given area, and the link lengths are also limited.

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. The mechanisms 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. In the present paper we explain our ideas in more detail and extend our previous results (Ekárt, 2001; Ekárt & Márkus, 1999).

2. THE DESIGN METHOD

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

$$\frac{e}{b}\cos\theta + \cos\theta > 6 \quad \text{or}$$

$$\frac{c}{b} - \frac{d}{b} + \frac{\sin\theta}{1.4} \leq -0.3.$$

A 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 catalogue of four bar mechanisms;
2. specification of input requirements;
3. classification of the elements of the catalogue;
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. The class of mechanisms that produce similar coupler curves is given by this structural description. Hoeltzel and Chieng (1990) make a classification of four bar mechanisms based on the form of coupler curve, but they provide no structural description for the members of a class.

Our catalogue of four bar mechanisms contains 7276 elements, and we created it as a computer version of the classical catalogue of Hrones and Nelson (1951). 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 catalogue are points of the mechanism space, with the structural parameters $a \in \{1.5, 2, 2.5, \ldots, 6.5\}$, $b = 1$, $c \in \{1.5, 2, 2.5, 3, 3.5, 4\}$, $d \in \{1.5, 2, 2.5, 3, 3.5, 4\}$, e, and θ taken for 50 sampled points on a rectangle attached to the coupler link of the mechanism (see Fig. 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.

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* class and the *negative* class. A

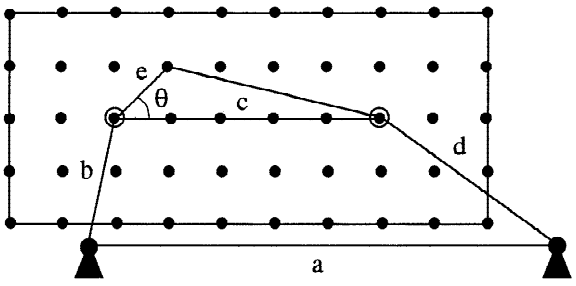


Fig. 2. The sampled points on the coupler link.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

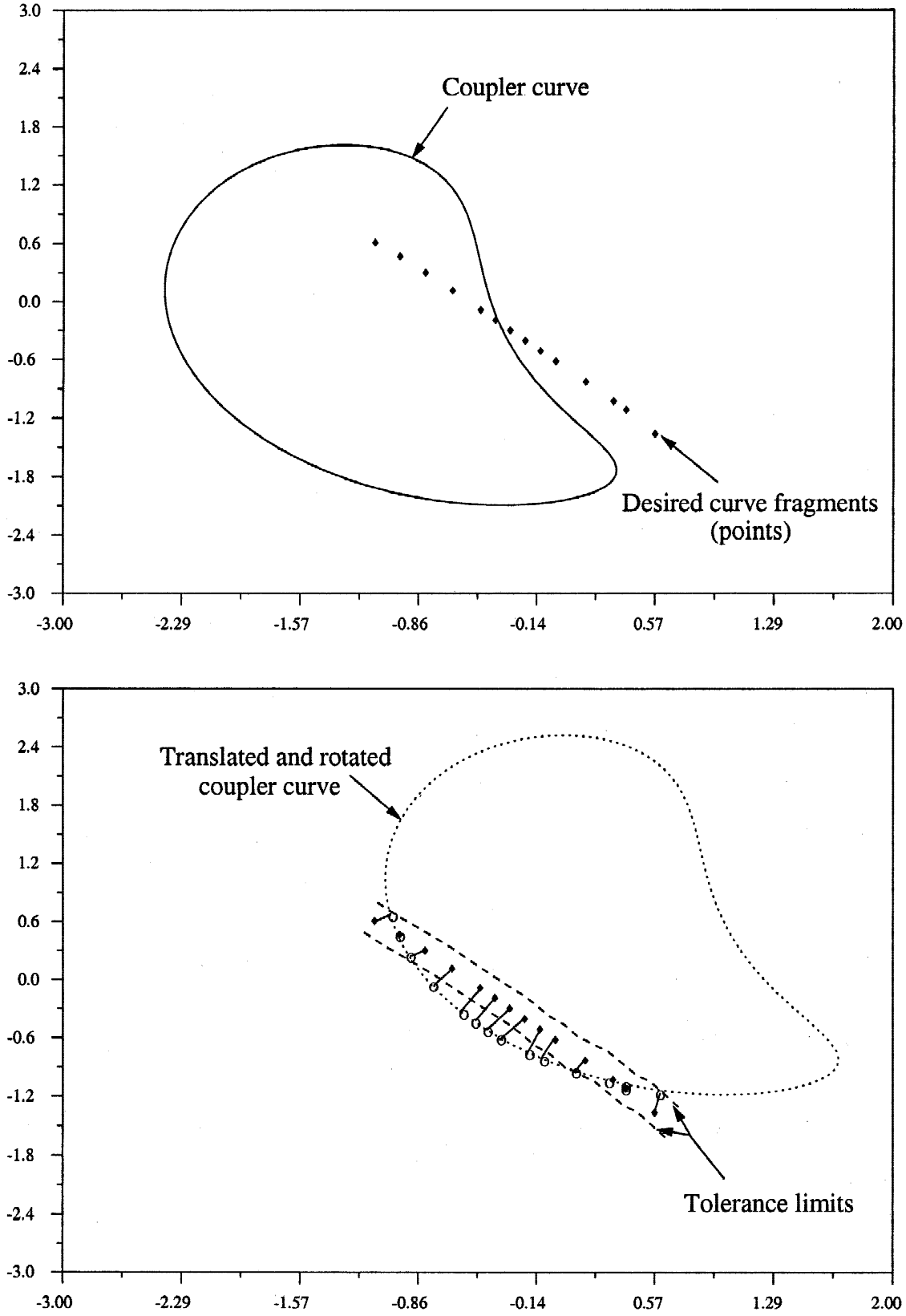


Fig. 3. Computing the similarity of the coupler curve (for the mechanism with $a = 1.5$, $b = 1$, $c = 1.5$, $d = 1.5$, $e = 1.41$, and $\theta = 2.35$) to the desired path fragments. For better intelligibility, timing is not indicated. This mechanism is classified as *negative*.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

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.* Classical methods only compare the coupler curve to the desired curve without translation and rotation, meaning that the position of the mechanism in the plane is predefined. We remove this restriction and compute the similarity of two curves after bringing them as close as possible by translating and rotating one of the curves. We compute the similarity value of a coupler curve to the desired curve (fragments) in the following way (as shown in Fig. 3):

1. To each point of the desired curve we associate a point of the coupler curve according to the input angle (timing) corresponding to that point of the desired curve, that is, we associate a point list from the coupler curve to the given point list of the desired curve. Generally, there are several possible matchings. (In Fig. 3 we show only the best matching for the given example.)
2. We calculate the similarity value for each such possible matching:
   a. We translate and rotate the point list obtained in step 1 so that the associated points of the desired curve and the coupler curve get as close as possible (this is equivalent to translating and rotating the entire coupler curve, but we consider only the point list because these points are needed for the comparison).
   b. We compute the distances of the associated point pairs.
   c. We assign the maximum of these distances as a similarity value for the given matching.
3. We select the matching with the smallest similarity value and designate this value as the similarity of the coupler curve and the desired curve. If this similarity value is within the tolerance limit, then all the points of the coupler curve selected in step 1 are close enough to the desired curve, and thus, the mechanism belongs to the positive class. Otherwise, the mechanism is classified as negative, like the example shown in Figure 3.

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 disjunctive normal form formula. Two machine learning methods have been applied: decision tree induction (by the C4.5 program; Quinlan, 1993) and genetic programming.

Decision tree induction was effective in the cases where 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 to the axes. However, it produced too many errors in the other cases. We needed a method allowing other partitions that could create descriptions corresponding to as many elements of the positive class as possible, regardless of whether they were situated in a convex region of the space. Genetic programming was our next choice. Unfortunately, in most of the cases the descriptions produced by genetic programming were not accurate enough. Two types of errors were encountered: too many members of the negative class were classified as positive and some members of the positive class were classified as negative. In this way the frontiers of the feasible regions of the mechanism space were misrepresented. Because decision tree induction is very fast and accurate when presented with the appropriate attributes and genetic programming is a plausible tool for creating new attributes, we decided to apply constructive induction (Wnek & Michalski, 1994). 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 situated in the negative region satisfies this description.

3.1. Decision tree learning

Decision tree induction is a commonly used method for concept learning (Quinlan, 1993). The input data are 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[1] 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 (Quinlan, 1993). Each example is a mechanism, having its structural parameters as attributes and the classification computed in the previous step (i.e., positive or negative). Because 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.[2] A simple decision tree is shown in Figure 4. Case studies are presented in Section 4.

A typical example of where C4.5 produced many errors is shown in Figure 5. The attributes $b = 1$, $d = 3$, $e = 1.41$, and $\theta = 2.36$ and the distribution of positive and negative

[1]The information gain is defined as the increase in information content when the set of examples is partitioned in subsets according to some criterion, such as a value of an attribute.

[2]In this case the information gain is computed for several candidate threshold values.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

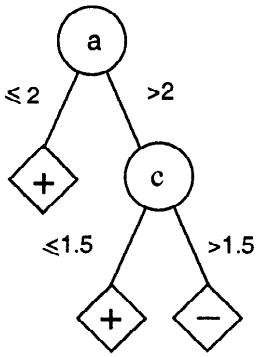


Fig. 4. Example decision tree.

examples in the mechanism space over coordinates a and c are indicated. 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 of the plane could be found. An exact description (as drawn in Fig. 5) of the positive region of the plane found by our program is

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

Overcoming this difficulty by creating new attributes is the subject of Section 3.3.

3.2. Genetic programming

As an alternative, we also generated structural descriptions of mechanism classes by using genetic programming (Cramer, 1985; Koza, 1992).

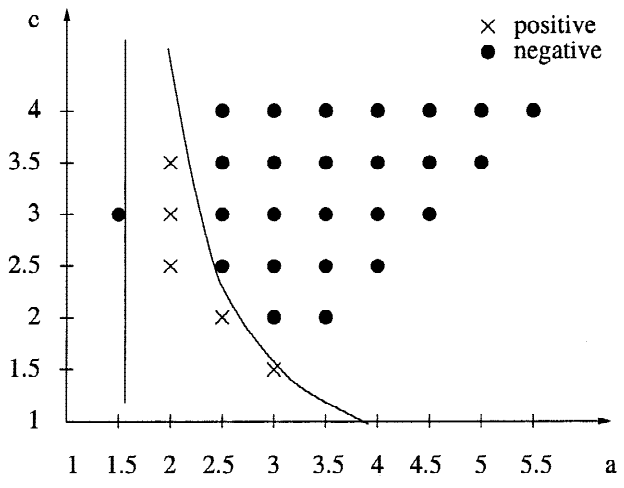


Fig. 5. The cross section of the mechanism space.

Genetic algorithms (Goldberg, 1989) transpose the notions of natural evolution to the world of computers and imitate natural evolution. In Nature new organisms adapted to their environment develop through evolution. Genetic algorithms *evolve* solutions to the given problem in a similar way. They maintain a collection of solutions, which is a *population* of individuals. The individuals are represented by chromosomes composed of *genes*. Genetic algorithms operate on the chromosomes, which represent the inheritable properties of the individuals. By analogy with Nature, through selection the fit individuals (potential solutions to the problem) live to reproduce, but the weak less-fit individuals die off. New individuals are created from one or two parents by mutation and crossover, respectively. They replace old individuals in the population, and they are usually similar to their parents. In other words, in a new generation individuals will appear who resemble the fit individuals from the previous generation. The individuals survive if they are fitted to the given environment.

Genetic programming is an extension to genetic algorithms in which the structures undergoing adaptation are not strings but are hierarchical computer programs of dynamically varying size and shape.

Genetic programming systems generally use the following algorithm:

1. Generate an initial population of individual programs consisting of random compositions of functions and terminals from a given function and terminal set.
2. Execute each program of the population on the so-called fitness cases and assign it a fitness value based on the fitness measure.
3. Create a new population of individuals by selection, transmission, and variation from the current population. Selection is based on fitness; the better performing individuals are more likely to be selected than the others. Individuals and parts of selected individuals are transmitted to the new population via reproduction and crossover, respectively. Variation is achieved through 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. We mostly employed parameter values considered typical for function regression problems. Because the number of fitness cases to be evaluated (7276) was more than two orders of magnitude larger than the usual number of fitness cases (50), we restricted the number of individuals produced (population size and maximum number of generations) in order to obtain results in reasonable time.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

Table 1. *Genetic programming parameter setting*

Objective	Evolve the structural description of the four bar mechanisms contained in the positive class
Terminal set	$a, c, d, e, \sin(\theta), \cos(\theta)$,[a] real numbers $\in [-10,10]$
Function set	$+, -, *, /$
Fitness cases	The mechanisms of the catalogue
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

[a]Because $b = 1$ for all the examples, here we use a for the ratio of the fixed link to the input link, and similarly, for any other link we use the ratio of that link to the input link. In this way, the genetic programs are valid dimensionless expressions.

The search space is a 6-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 often no linear delimiters exist between positive and negative regions, as shown in the 2-dimensional cross section of Figure 5.

Initially we defined a genetic program as an inequality on the structural parameters to be satisfied by some feasible mechanisms and possibly no infeasible mechanism. For instance, a possible inequality is $a/b \sin(\theta) - 1.2 > 0$. This condition is satisfied by all the mechanisms with $a \geq 2$, $b = 1$, and $0.79 \leq \theta \leq 2.35$; but only some of these mechanisms are feasible. In order to be feasible, a mechanism must satisfy a set of such inequalities.

We represented a genetic program as a tree with a fixed number of branches where each branch represented an inequality on the structural parameters of the mechanisms. The genetic program was then evaluated as the union of the inequalities contained in its branches. An example tree with three branches is shown in Figure 6, where a branch $f(a, c, d, e, \theta)$ represents the inequality $f(a, c, d, e, \theta) > 0$.

As we wanted to obtain comprehensible descriptions, we limited the number of inequalities so that the complexity of the results would be comparable to the complexity of decision trees. The fixed number of inequalities might seem restrictive, but in reality it allows for both larger and smaller numbers of inequalities. A larger number of inequalities occurs when some of the inequalities correspond to sets of inequalities, for example, $(a + c - 3)(c - d) > 0$ is the same as $a + c - 3 > 0$ and $c - d > 0$ or $a + c - 3 < 0$ and

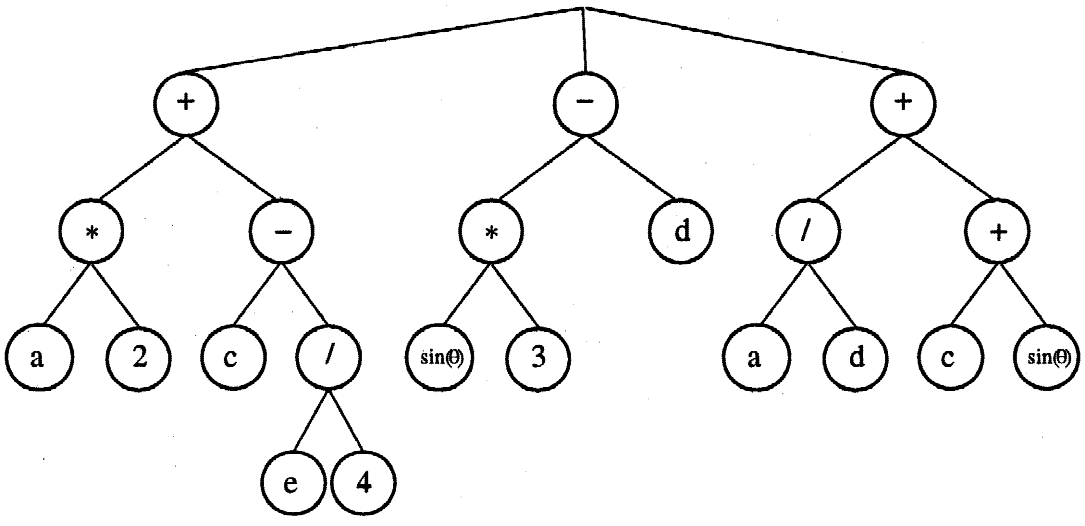


Fig. 6. Example genetic program with three branches.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

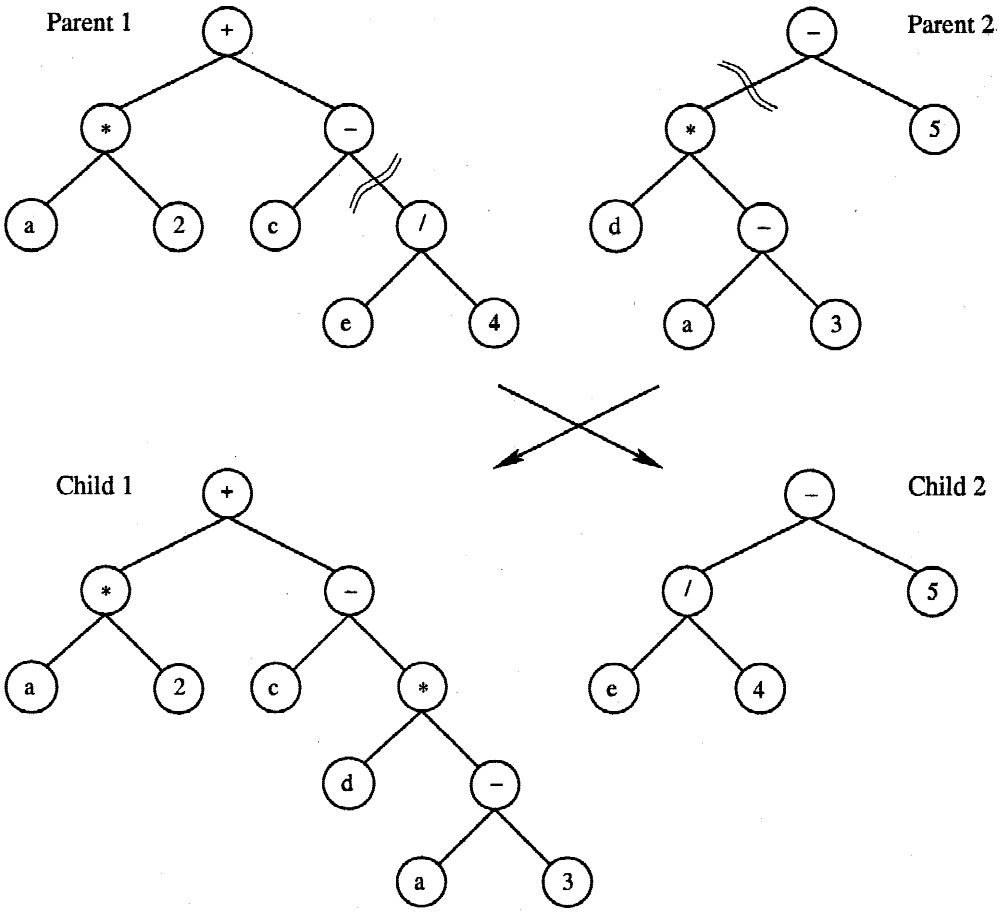


Fig. 7. Crossover of two genetic programs.

$c - d < 0$. A smaller number of inequalities occurs when some of the branches of the genetic program represent inequalities that are always true (such as $a + 1 - a > 0$).

We used both crossover and mutation as genetic operators. The offspring of the crossover were obtained from the two parents by selecting one subtree of each parent and exchanging these subtrees, as shown in Figure 7.[3] We used point mutation, that is one node of the tree was randomly selected and mutated. In the case of internal nodes, mutation consisted of randomly changing the function represented by the node (see Fig. 8a). 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 (see Fig. 8b).

The fitness cases were the mechanisms of the catalogue 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:

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

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

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

Thus, a 100% fit individual is an expression corresponding to some positive examples and no negative example. An expression satisfied by some positive and some negative examples cannot make the distinction between the positive and negative examples, and it is therefore penalized by a worse fitness value.

The descriptions found by genetic programming were less accurate than the decision trees. This is partly due to

[3] We illustrate the genetic operators on individuals corresponding to one inequality each: $f(a, c, d, e, \theta) > 0$ for expression $f(a, c, d, e, \theta)$.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

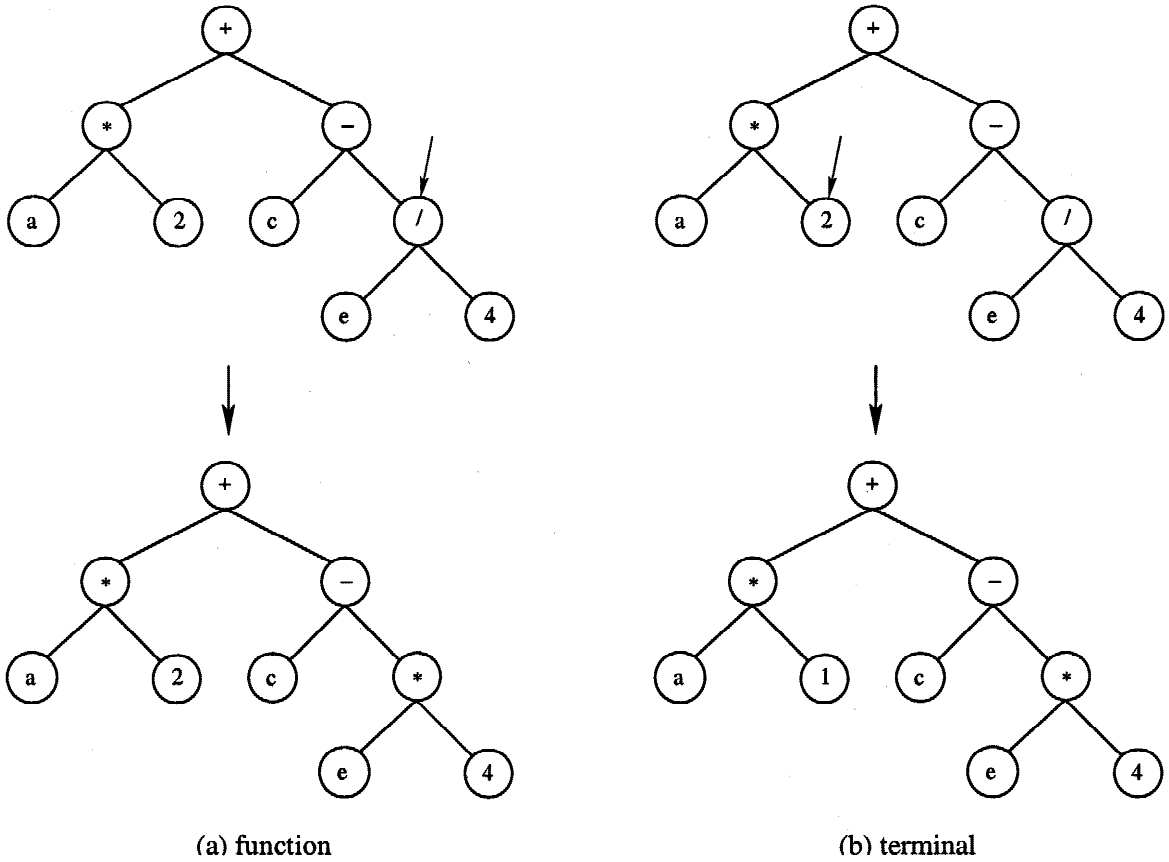


Fig. 8. Mutation of a genetic program.

the limitations of the resources (i.e., population size and generation number) and partly due to the difficulty of the problem. Recent results on problem difficulty for genetic programming suggest that the real numbers that are employed could also influence performance (Daida et al., 2001). Careful fine-tuning of the real numbers might lead to more accurate results.

3.3. Constructive induction

When both previous methods are applied alone, they had some shortcomings, hence the idea to apply them together. Genetic programming can take a long time to converge to acceptable solutions and also needs parameter fine-tuning. Decision tree learning is fast and more accurate but 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, but let us consider *incremental usefulness* from Blum and Langley (1997):

> 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 $\{x_i\} \cup A$ 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 (Matheus & Rendell, 1989; Pagallo, 1989). Because the original attributes (the structural parameters of mechanisms) are numerical values, a straightforward method is the generation of *arithmetic expressions* over the original attributes (Bloedorn & Michalski, 1998).

Having in mind the definition for usefulness, we used C4.5 as the learning engine for the constructive induction and applied genetic programming for attribute (feature) generation. The new scheme for generating the description is shown in Figure 9.

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

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

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

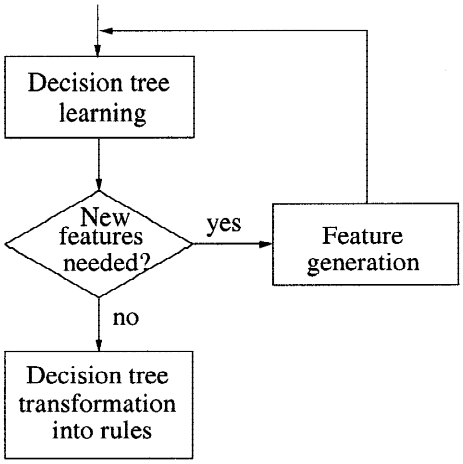


Fig. 9. Constructive induction.

We experimented with different values for n and p ($p \gg n$ or $n \gg p$), because 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 shown in Figure 9. The new features created by genetic programming tended to cover as many positive examples as possible. However, 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 to use 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 make it the information gain (as defined by Quinlan, 1993) of the feature represented by a genetic program. The decision tree induction step is modified as shown in Figure 10.

By creating the new features at the construction of nodes in the decision tree, we obtained more accurate descriptions.

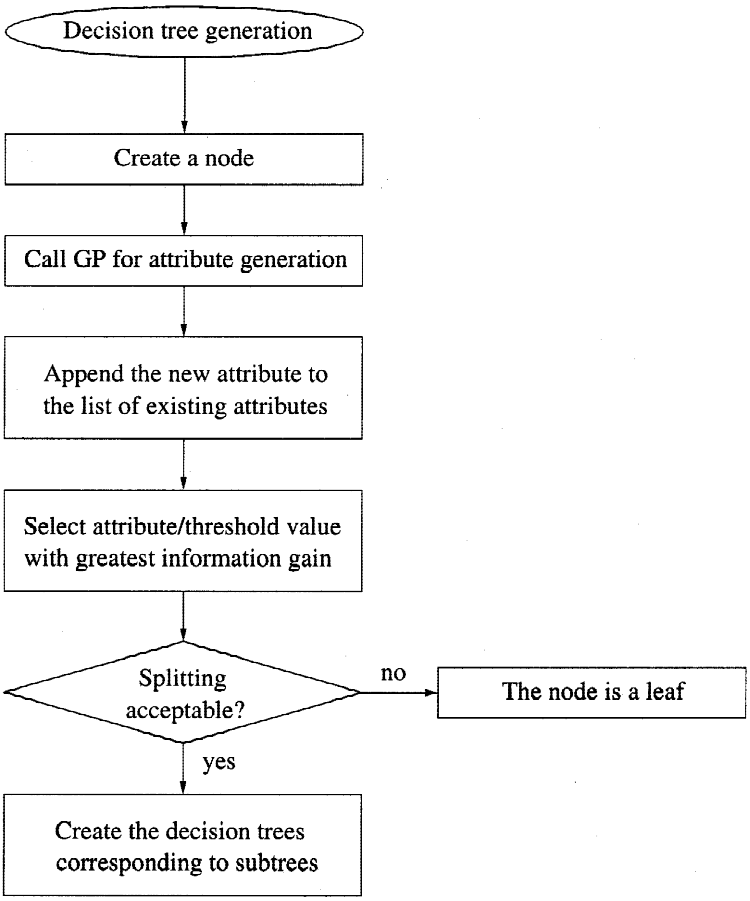


Fig. 10. Decision tree construction.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

3.4. Training data preselection

The generation of the structural description based on all available mechanisms can be very slow. The process can be made considerably faster if only a small number of examples are used instead of the whole catalogue of mechanisms. The obtained descriptions can then be tested on the whole catalogue.

In order to obtain sufficiently accurate descriptions, the *relevant* examples have to be selected for generating the description (i.e., for *training*). Because only a few mechanisms belong to the positive class and the majority belong to the negative class, we included all the positive examples in the training set. We wanted to differentiate between the two classes, so we selected the negative examples that were in the neighborhood of the positive examples (had similar values of attributes). In Figure 11 we show two possible training sets of different sizes for the example presented in Section 3.1.

The algorithm for selecting the training data is the following:

for all positive examples e
  include e in the training set
  let $a_1,\ldots,a_n$ be the attribute values for example e
  for all attributes $i \in \{1,2,\ldots,n\}$
    for all negative examples e'
      let $a'_1,\ldots,a'_n$ be the attribute values for example e'
      include e' in the training set if it has
        the same attribute values $a'_j = a_j$
          for all $j = \{1,\ldots,n\},\ j \neq i$
        and a value for attribute a'_i
          in the neighborhood of a_i: $|a'_i - a_i| < \varepsilon$

The size of the training set can be controlled through parameter ε. In our experience, a reduction of the training set to 15% of the catalogue is possible and the results are of acceptable accuracy. In addition, the descriptions generated based on the preselected examples are significantly smaller and therefore more comprehensible for humans.

4. EXPERIMENTS

In this section two design cases are discussed in detail. We chose circular and linear path fragments, because these are very important fragments of the paths needed in practical mechanism design. Thus far, approximate circle tracing mechanisms have been synthesized through laborious computation algorithms. Only the recent results of Ceccarelli and Vinciguerra (2000) allow their efficient synthesis. Approximate straight lines are required in the case of level luffing cranes used on many docks to load and unload cargo (Waldron & Kinzel, 1999). Another application of straight line generating mechanisms is in strip chart recorders. In addition, a linear path fragment followed by a circular one is required in the film-advance mechanism of any movie camera (Chironis, 1991; Norton, 1992; Sandor & Erdman, 1984), as shown in Figure 12.

For the circular path fragment (Fig. 13) we found a 100% correct description by using just decision trees, so that 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 13 and 14. 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 Fig. 13), the mechanism space was divided into positive and negative classes and then C4.5 was applied. The produced decision tree is shown in Figure 15a. At each leaf node the predicted class of the corresponding cases is given as negative or positive class.

The tree classifies all the cases correctly, so that for every example presented to it the real class is the same as the predicted class. The production rules can be derived from it

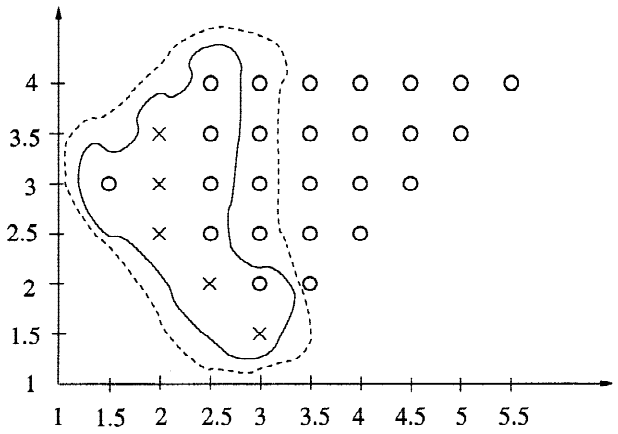


Fig. 11. The preselection of training data.

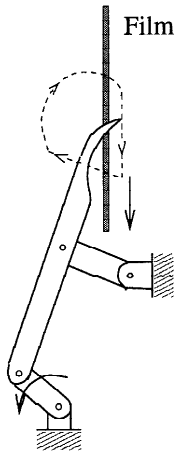


Fig. 12. The film-advance mechanism of a movie camera.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

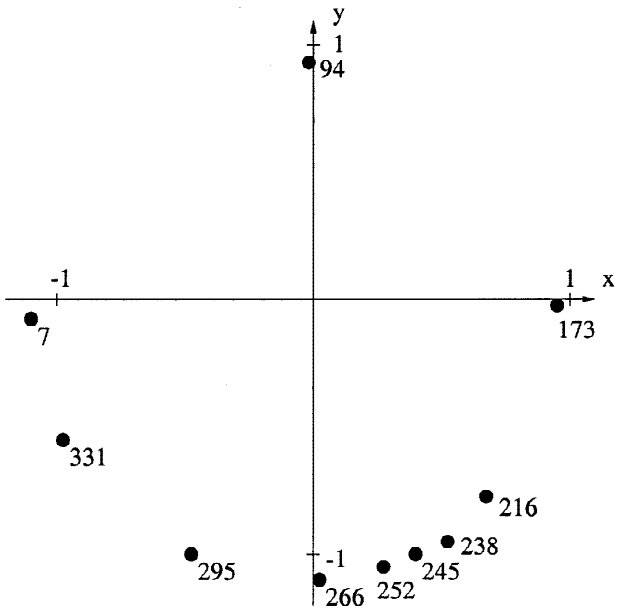


Fig. 13. The first desired path fragment.

by reading the paths from the root to the leaves. In Figure 15b we show only the rules for the positive class, because any example that satisfies neither of these rules is negative.

For the same problem, the best description (of the positive examples) produced by genetic programming is shown in Figure 16.

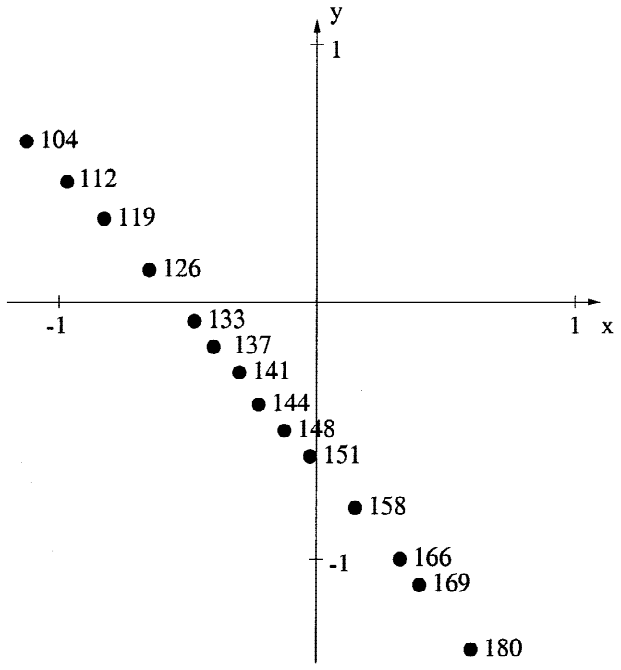


Fig. 14. The second desired path fragment.

Table 2. *The new attributes proposed by GP*

Attribute	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

We tried out different numbers of branches for the genetic trees. According to our observations, with fewer branches 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 14. 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 shown in Figure 9.

Some of the attributes proposed by genetic programming are presented in Table 2. The third column (Type) specifies whether the attribute corresponds to the positive or the negative class. The last column (G) represents the generation when the attribute emerged. The best decision tree had 41 nodes and misclassified six cases.

The next step was the introduction of the attribute generator at the level of node creation in the decision tree (see Fig. 10). Table 3 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

Table 3. *The results of 10 runs of the program*

	Decision Tree		New Features		
No.	Size	Errors	No.	Ave. Compl.	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
Ave.	39	5	9	37	55.7

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

```
e >  0.33 :  -
e <= 0.33 :
|    theta <= 1.46 : -
|    theta >  1.46 :
|    |     c <= 1.5 :  -
|    |     c >  1.5 :
|    |     |    a >  2.5 :  +
|    |     |    a <= 2.5 :
|    |     |    |    c <= 3 :  +
|    |     |    |    c >  3 :  -
```

(a) The output of C4.5.

1: $e \leq 0.33$ and $\theta > 1.46$ and $c > 1.5$ and $a > 2.5$
2: $e \leq 0.33$ and $\theta > 1.46$ and $c > 1.5$ and $a \leq 2.5$ and $c \leq 3$

(b) The rules for the feasible mechanisms.

Fig. 15. (a) The decision tree produced by C4.5 for (−) negative or (+) positive classes and (b) the rules corresponding to the positive class.

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 2 is 5). By new feature usage, we mean the percentage of decision nodes in the tree where a new feature was selected.

Up to this point, learning has been performed on the whole catalogue. In order to reduce CPU time requirements, we introduce an additional training data selection step in our algorithm presented in Section 2 (between steps 3 and 4).

Because the number of positive examples is much less than the number of negative examples and omitting some of them could result in losing important data that are not sampled elsewhere, we include all the positive examples in the training set. We select the relevant negative examples as the neighbors of positive examples in the mechanism space (as described in Section 3.4). The rest of the negative examples are put in the test data set. By this preselection we reduce the size of the training set to 983 elements and reduce the CPU time requirements by 85% at the cost of worse solution accuracy. The best tree has 21 nodes, misclassifies eight training examples, but correctly classifies all the test data. This structural description can be seen in Figure 17. A mechanism that satisfies one of the three constraint sets in Figure 17b is a feasible straight line generating mechanism. Table 4 summarizes the best results for this mechanism.

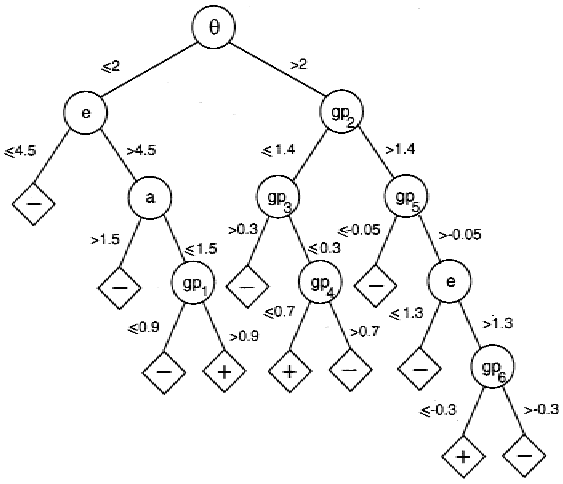
5. DISCUSSION

Let us revisit the example problem presented in Figure 14. The coupler curves of the corresponding mechanisms (that belong to the catalogue) are presented in Figure 18. 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, and 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 Figure 19. The graphics are drawn to the same scale. We represented the mechanisms (together with their coupler curve) in two positions: the start and the terminal position of the generated straight line. The input link is the emphasized link, and it is rotated in the direction indicated by the arrow.

```
( AND
  ( > ( - ( / sin(theta) 6.4 )
          ( * ( * e sin(theta) ) sin(theta) ) ) 0 )
  ( > ( - sin(theta) cos(theta) ) 0 )
  ( > ( / ( / c sin(theta) ) ( - a c ) ) 0 )
  ( > ( - ( / 0.3 sin(theta) ) ( + a e ) ) 0 ) )
```

Fig. 16. The best description created by GP.

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press



(a) The decision tree.

1: $\theta \leq 2$ and $e > 4.5$ and $a \leq 1.5$ and $gp_1 > 0.9$

2: $\theta > 2$ and $gp_2 \leq 1.4$ and $gp_3 \leq 0.3$ and $gp_4 \leq 0.7$

3: $\theta > 2$ and $gp_2 > 1.4$ and $gp_5 > -0.05$ and $e > 1.3$ and $gp_6 \leq -0.3$

(b) The structural description.

$$gp_1 = e\cos\theta + \cos\theta - 5.1$$

$$gp_2 = \frac{1.5+a}{2.8\sin\theta+0.53}$$

$$gp_3 = e\cos\theta - a + 2.9$$

$$gp_4 = \frac{c}{a(a+\sin\theta-c\,c)}$$

$$gp_5 = \frac{\sin\theta}{c} + \sin\theta - a - \frac{\sin\theta+c}{\sin\theta-c}$$

$$gp_6 = c - d + \frac{\sin\theta}{1.4}$$

(c) The GP attributes.

Fig. 17. The description of the straight line generating mechanisms.

For better distinction, instead of drawing the fixed link, we show just its joints, using the black disks.

This depiction is clear evidence of 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 very helpful for the engineer involved in mechanism design.

Table 4. *The best decision trees for the different methods*

	New Features		Decision Tree	
Method	No.	Usage (%)	Size	Errors
C4.5	0	0	47	11
Separate GP	8	60	41	6
GP at node level	10	66.7	37	4
Train data presel.	6	60	21	8

In many cases the mechanism is a part of a complex system and must 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 19a could be preferred to the mechanism of Figure 19b 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 structural description and not the enumeration of the acceptable elements of the catalogue. The catalogue consists of points of the mechanism space and therefore the found structural description applies not only to its elements but also to the mechanism space.

6. CONCLUSIONS

We combined decision tree learning with genetic programming to solve a difficult mechanism design problem. Instead of finding a single four bar mechanism that generates

https://doi.org/10.1017/S0890060403173040 Published online by Cambridge University Press

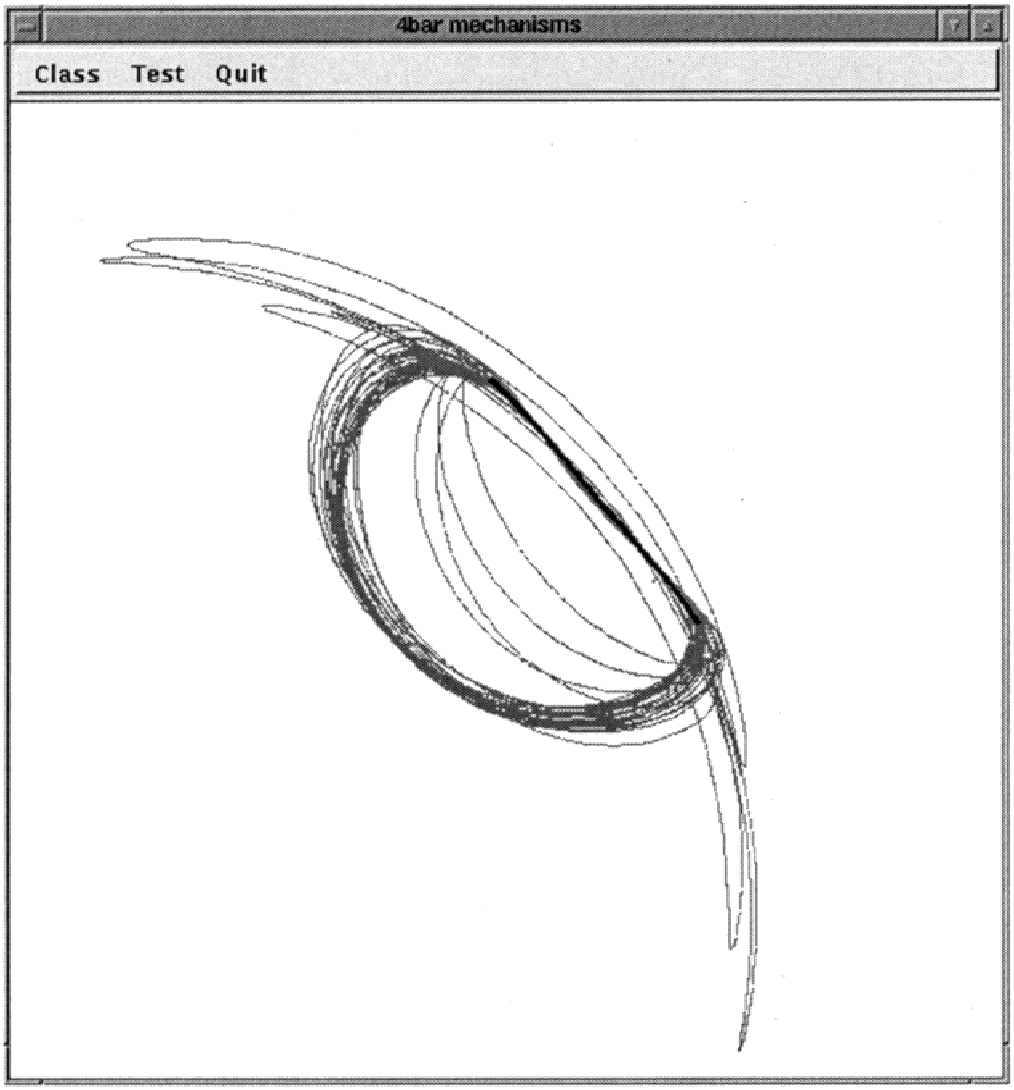
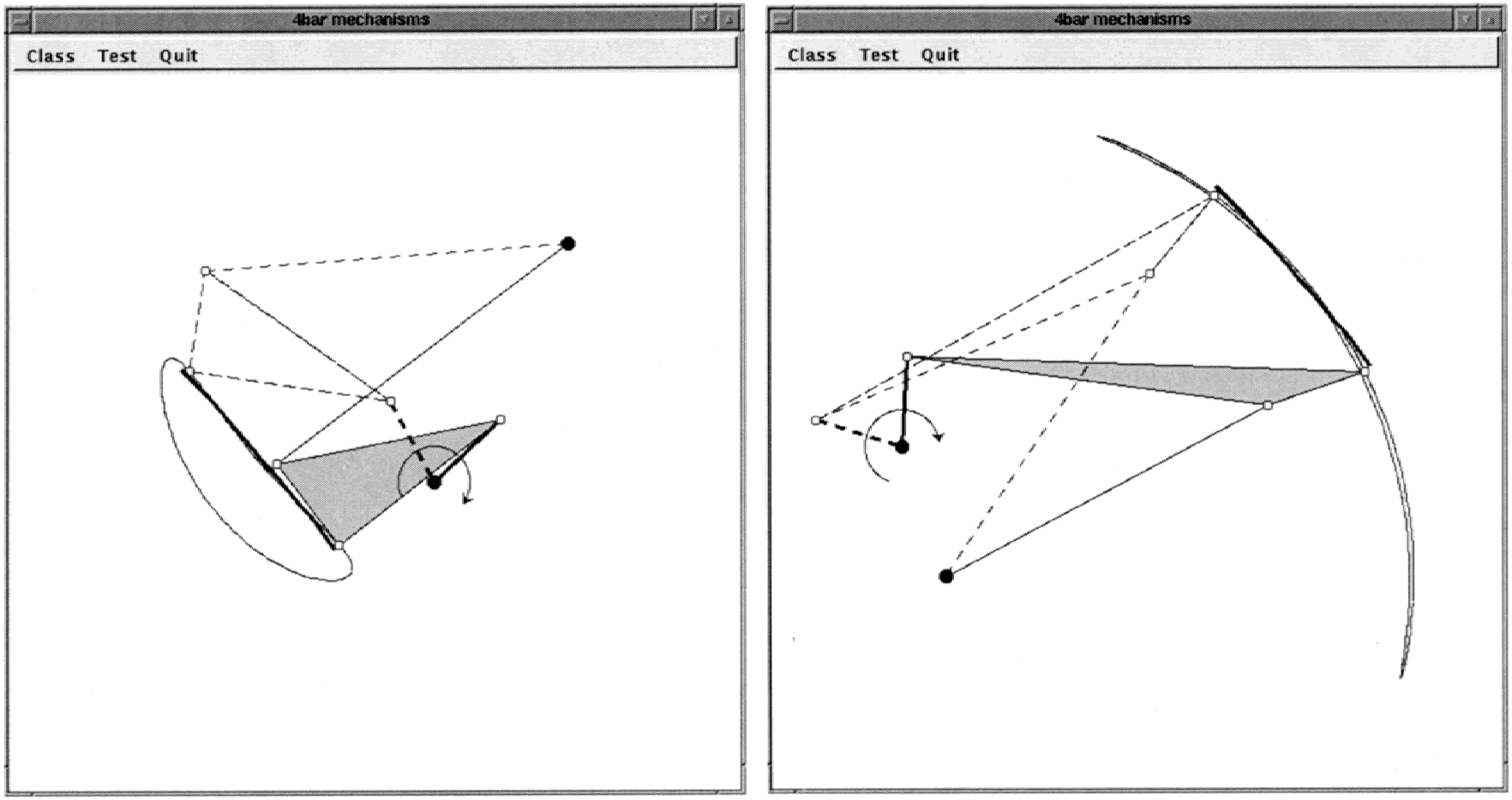


Fig. 18. The coupler curves of the straight line generating mechanisms.

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 generated by genetic programming and then presented to the decision tree learner C4.5. Second, 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. In order to reduce CPU time we introduced a training data preselection step, which also resulted in more comprehensible descriptions being generated.

ACKNOWLEDGMENTS

Part of this work was completed while the first author (A.E.) was a lecturer at the School of Computer Science, University of Birmingham, Birmingham, UK.



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

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

Fig. 19. Straight line generating mechanisms in two positions (- - -) start and (—) terminal.

REFERENCES

Bloedorn, E., & Michalski, R.S. (1998). Data-driven constructive induction. *IEEE Expert and Intelligent Systems 13(2)*, 30–37.

Blum, A.L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*, 245–271.

Bose, A., Gini, M., & Riley, D. (1997). A case-based approach to planar linkage design. *Artificial Intelligence in Engineering 11*, 107–119.

Ceccarelli, M., & Vinciguerra, A. (2000). Approximate four-bar circle-tracing mechanisms: Classical and new synthesis. *Mechanism and Machine Theory 35*, 1579–1599.

Chironis, N.P. (1991). *Mechanisms and Mechanical Devices Sourcebook*. New York: McGraw–Hill.

Cramer, N. (1985). A representation for the adaptive generation of simple sequential programs. *Proc. Int. Conf. Genetic Algorithms and Their Applications*, pp. 183–187.

Daida, J., Bertram, R., Stanhope, S., Khoo, J., Chaudhary, S., Chaudri, O., & Polito III, J. (2001). What makes a problem GP-hard? Analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines 2(2)*, 165–191.

Ekárt, A. (2001). *Genetic programming: New performance improving methods and applications*. PhD Thesis. Eötvös Loránd University, Budapest.

Ekárt, A., & Márkus, A. (1999). Decision trees and genetic programming in synthesis of four bar mechanisms. In *Life Cycle Approaches to Production Systems. Proc. Advanced Summer Institute—ASI'99*, pp. 201–208.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison–Wesley.

Hoeltzel, D.A., & Chieng, W.-H. (1990). Pattern matching synthesis as an approach to mechanism design. *ASME Journal of Mechanical Design 112*, 190–199.

Hrones, J.A., & Nelson, G.L. (1951). *Analysis of the Four Bar Linkage*. New York: Wiley.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

Mariappan, J., & Krishnamurty, S. (1996). A generalized exact gradient method for mechanism synthesis. *Mechanisms and Machine Theory 31*, 413–421.

Matheus, C.J., & Rendell, L. (1989). Constructive induction on decision trees. *Proc. IJCAI*, pp. 645–650. San Francisco, CA: Morgan Kaufmann.

Norton, R.L. (1992). *Design of Machinery*. New York: McGraw–Hill.

Pagallo, G. (1989). Learning DNF by decision trees. *Proc. IJCAI*, pp. 639–644. San Francisco, CA: Morgan Kaufmann.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann.

Sandor, G.N., & Erdman, A.G. (1984). *Advanced Mechanism Design: Analysis and Synthesis*, Vol. 2. Englewood Cliffs, NJ: Prentice–Hall.

Vancsay, G. (2000). *The application of genetic algorithms in mechanical design*. PhD Thesis. Technical University of Budapest.

Waldron, K.J., & Kinzel, G.L. (1999). *Kinematics, Dynamics and Design of Machinery*. New York: Wiley.

Wnek, J., & Michalski, R.S. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning 14*, 139–168.

Anikó Ekárt obtained a PhD in informatics at Eötvös Loránd University, Budapest. She is a Research Fellow at the Computer and Automation Research Institute, Hungarian Academy of Sciences. Her main research area is evolutionary computation, including applications of genetic algorithms and genetic programming to engineering design problems.

András Márkus has an MS in mathematics from Eötvös Loránd University, Budapest, and a PhD in mechanical engineering from Technical University, Budapest. His research interests include the application of various artificial intelligence methods to engineering and manufacturing problems. Dr. Márkus' recent efforts concentrate on the integration of constraint-based techniques and optimization methods.