Genetic Programming for Expert Systems

Konrad Sickel, *Member, IEEE* and Joachim Hornegger, *Member, IEEE*

***Abstract*— Genetic programming is the usage of the paradigm of survival of the fittest in scientific computing. It is applied to evolve solutions to problems where dependencies between multiple input factors are unknown. In this paper we propose and evaluate the application of a specifically adapted genetic programming framework to optimize the rule base of an expert system. The expert system controls a computer-aided-design software and targets the automation of a manufacturing process.**

The used steady state genetic programming framework introduces some variations on the selection and evolution operators normally used in genetic programming. In particular: size enforcing mutation, dynamic fitness calculation and size constraint ranking. The genetic programming system is evaluated with real data and led to an improved expert system performance of about 22 percent.

I. Introduction

Genetic programming (GP) was popularized by Koza [1] as a method for using natural selection for the creation of new computer programs.

GP has already been applied as an optimization technique on a wide variety of problems and applications [2], [3]. Examples vary from the analysis of high energy physics data [4] over recognition of handwritten digits on mail [5], improvements in robot control [6] to solving of the ocean coloring problem [7]. Furthermore GP itself was topic of high research interest. Especially the influence and performance of selection, crossover and mutation operators are discussed and compared [8], [9], [10], [5], [11].

GP has the advantage that it yields programs or in our case rules, which can be analyzed and interpreted afterwards. This is one of the reasons why we consider GP for our optimization problem. The problem we are dealing with is an expert system for computer-aided design (CAD) manufacturing. The expert system framework is similar to the one described in [12]. The framework consists of an expert system controlling a CAD software supported by a feature detection unit. The expert system designs a customized medical prosthesis constraint by the input data and on the fly detected features. The framework performs well on several of the steps to design the prosthesis and unfortunately, mediocre on others. Aim of this work is to evaluate GP as a way to improve the weak points of the framework.

Konrad Sickel is with the Pattern Recognition Lab, Department of Computer Sciences, Friedrich-Alexander-University Erlangen-Nuremberg, Martensstr. 3, 91058 Erlangen, Germany (phone: +49 9131 85 27826; email: konrad.sickel@informatik.uni-erlangen.de).

Joachim Hornegger is head of the Pattern Recognition Lab, Department of Computer Science, Friedrich-Alexander-University Erlangen-Nuremberg and with the Erlangen Graduate School in Advanced Optical Technologies (SAOT) Martensstr. 3, 91058 Erlangen, Germany (email: joachim.hornegger@informatik.uni-erlangen.de).

Since we are dealing with a rule based system, we assume that we know how certain rules should work and be applied. Nevertheless, we might be wrong. One of the excellent properties of GP is that it can be applied, when the interrelationships among the relevant variables are unknown [2]. In our case, we assume that some of the detected features together define a certain design step. In other words, we assume a certain relationship of these features. GP provides us with an objective method to verify these assumptions.

Rule definition with help of GP has already been successfully applied, for instance by [13], [14] and [15]. These approaches focus on developing rules for classification purposes, like optical-character-recognition (OCR) or data mining. Applying GP in the field of medical prosthesis was already applied by Collet et al. [16]. They addressed the problem of fitting the various parameters of a Cochlear Implant for a patient. They evaluated an interactive evolutionary algorithm approach with a micro-population and compared the results with an expert. Tsakonas et al. [17] proposed an evolving rule-based system for medical decision support. They focused on generating short and simple rules for diagnosis of aphasia's subtypes and the classification of pap-smear examinations.

In contrary to these works, we want to apply GP as a tool to optimize the definition of a plane in 3-D for the design of customized hearing aids. The generated rules include complex matrix and vector based computations using anatomical features. We evaluated our GP framework with the given data and could extract new knowledge for our rule base. The resulting programs provided us with several new dependencies between the input features as well as new ideas.

Our main contribution is to apply GP in the field of rule optimization for expert systems with complex rule syntax. In this work we explain the used GP framework, the encountered problems and adaptations developed to solve the optimization problem.

This paper is structured as follows: Section 2 gives a brief overview of the targeted automation framework. Section 3 describes our GP framework in detail. In Section 4, the example data is introduced and the experiments and results are presented. Finally, some conclusions are drawn in Section 5.

II. Expert System Framework

The expert system for our particular optimization problem is described in detail in [12]. It consists of five major parts:

- rule base,
- feature detection unit,
- modeling software,
- rule interpreter and

978-1-4244-8126-2/10/$26.00

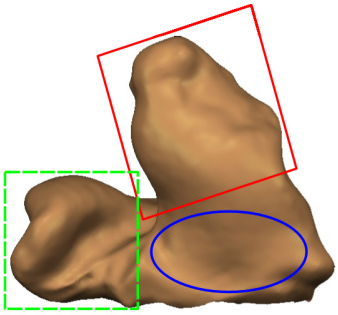

Fig. 1. Unprocessed impression of a right ear. The impression is acquired by scanning the surface of moldable material, which was allowed to settle in the ear yielding a negative of the ear. The ear canal is indicated by the upper right (red) rectangle. It continues downward into the tragus area indicated by the ellipse (blue). The major structure on the left is called helix indicated by the dashed rectangle (green).

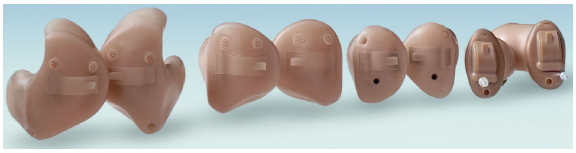

Fig. 2. Samples of different customized hearing aids in different styles.

- performance data acquisition unit.

The rule base for the considered application represents the knowledge about modeling a customized in-the-ear hearing aid (HA). Modeling with a specialized CAD software like [18] consists mainly of cutting, rounding, smoothing and component placement actions. Fig. 1 shows an example of the input data and Fig. 2 some samples of customized HAs. The developed rule base consists of 44 guide rules correlating to design steps in the workflow. To design a customized HA approximately 20 of these rules are applied. The applicability depends on the various input constraints for a customized HA (size type, amplification needed, feedback, etc.) as well as the input shape. All rules are encoded in *if-then-else* rules. The approach is quite common for rule based systems. Such procedural knowledge representation gives us, in contrast to a declarative representation, the possibility of defining rules that match the manual HA design process and can be easily interpreted by the process experts. Despite its simplicity the representation is sufficiently powerful to encode the knowledge for the surface shaping process.[19] To transcribe the rules we have developed a simple script language with a context free grammar similar to PASCAL.

The developed language supports the standard data types, like Booleans, integers, floats, strings and arrays of all types. In addition 3-D points, planes and matrices are added as special data types. For each data type the standard calculation and comparison operators are available, which allow vector and matrix based computations. The script language supports several control structures, such as `if-then-else` blocks as well as `for`, `while` and `repeat-until` loops. Furthermore,the definition of functions and procedures is possible.

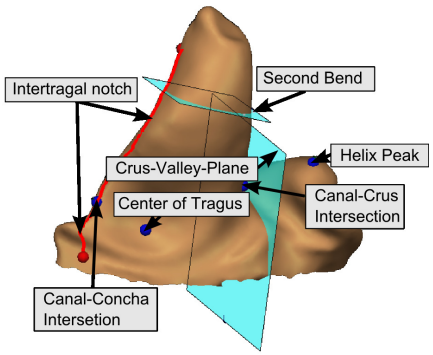

Fig. 3. The figure shows a subset of features detected on the ear impression. Features can be points (helix peak), planes (second bend), ridges (intertragal notch) and areas.

The script parser and interpreter were implemented with the tools bison and flex [20]. In addition to the knowledge description, the script language provides the interface to the modeling software and the feature detection. A simplified example of a cutting rule is given in the following. The currently used rule in the knowledge base consists of approximately 150 lines.

```
if Option(1) == "ITE" then
  OpenGuideStep(Shaping, "Initial Cut")
    plane cutting_plane = Plane(AntiTragusPoint,\\
    TragusPoint, AntiHelixPoint )
    cutting_plane = MovePlane(cutting_plane,\\
      CanalTipPoint, -2.0)
    point ref_point = TragusPoint
  CloseGuideStep(OpenCut, cutting_plane)
endif
```

The given example is enclosed by a simple `if` condition, which decides based on an option if the rule is applicable for a certain device type. The `OpenGuideStep` and `CloseGuideStep` define the interface to the modeling software. This includes setting up the environment (`Shaping`), a rule name displayed to the user, the tool applied (`OpenCut`) and the parameters used `cutting_plane`. The rule body consists of a simple plane definition using three feature points. This is followed by a movement of the just defined plane about 2 mm along its normal. The movement direction is made non-ambiguous using another feature point.

The feature detection adapted to ear impression is able to identify the typical anatomical features as well as additional features defined by process experts. Overall the feature vector consists of 44 elements. Features are either points, planes, area or ridges (paths). Fig. 3 shows a selection of the detected features. A detailed description of the features and used algorithms is given in [21].

The modeling software is a CAD software which provides the tools to cut and round 3-D meshes [18], [22]. It contains tools to integrate additional mesh structures like a ventilation tube [23] or to compute an inner wall. Furthermore it is possible to virtually place electric components like the

receiver or a battery, which enables the user to verify if the chosen component will fit in the device.

The rule interpreter connects rule base, feature detection and modeling software. It executes every active rule, retrieves the needed features and applies the operations to the given mesh.

The automation framework is currently used by a HA manufacturer in a semi-automatic version. Semi-automatic in this context means that an operator has to confirm the rule result before it is applied to the mesh.

To analyze the *performance of the system* and to be able to improve it we store the result of these checks. Performance in this case is defined as the number of rules applied without user interaction. For example, if the user modifies a plane generated by the expert system, both planes are stored. This allows the identification of weak rules in the knowledge base.

III. Genetic Programming Framework

We implemented our genetic programming framework on basis of the field guide by Poli et al. [2]. The implementation is done in C++, uses the boost libraries [24], a tree implementation by Gottschlich [25] and OpenMP [26] for parallelization. The implemented framework follows the common scheme depicted in Fig. 4.

A. Terminals, operators and functions

Each genome is represented by a syntax tree, which is made up of nodes. Each node has zero or more inputs. A special node is a terminal, these can be either zero-argument functions or constants like the input features. A node with one or more inputs is either an operator or a function.

The available functions, operations and terminals are a subset of the script language introduced in [12]. The functionality allows the definition and modification of Boolean, integer, float, 3-D point and plane terminals. Arithmetic operators for standard data types as well as 3-D vectors (points) are available. It is possible to compare values if they are equal ($==$) or greater ($>$) and use boolean combinations with OR and AND. Finally the definition of `while` and `for` loops as well as `if` conditions is possible, see Tab. I.

B. Population members

A population member (genome) is represented by a syntax tree, which itself is a rule. Each member can make use of all input features, but is not required to do so.

C. Methods

For each generation of our population the following steps are applied:

1) Generate - Create new members from the existing population by applying GP operators.
2) Evaluate - Assign a fitness value and rank to each member of the population.
3) Select - Select from all individuals those to keep and those to discard.

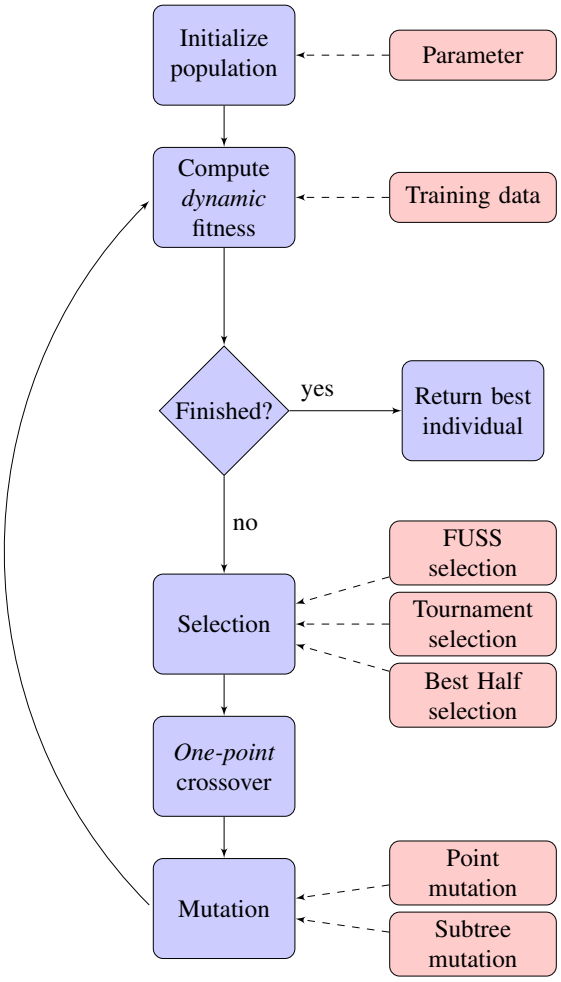

Fig. 4. Scheme of our genetic programming framework. It supports three different selection strategies and two different mutation operators. Note: Only one of the selection strategies or mutation operators is used at a time.

1) Selection strategies: The selection strategy includes the select and the generate step mentioned above. It defines, which members of the population reproduce and how the reproduction is achieved. The selection method defines the evolutionary pressure applied by the system. If the pressure is too strong, the population diversity drops. If it is too low, the system has no pressure to evolve. Based on a literature research we implemented tournament selection [1], [2], fitness uniform selection (FUSS) [9] as well as a simple greedy algorithm we named *best half* selection.

Tournament selection is the classical selection strategy described in [1], [2]. It involves running several tournaments among N individuals chosen at random. The winner of each tournament is selected for crossover. Selection pressure is adjusted by tunning N.

The FUSS scheme is a relatively new approach to preserve diversity in a population [9]. FUSS biases selection toward sparsely populated fitness values instead of biasing it toward higher fitness. The scheme first computes the interval of the

TABLE I
TERMINAL AND FUNCTION SET OF THE GENETIC PROGRAMMING FRAMEWORK.

Terminal set

Kind of primitive	*Example(s)*
Feature point	TragusPoint, AntiTragusPoint, ...
Feature plane	FirstBendPlane, SecondBendPlane, ...
Variables	bool_var_1, point_var_3, int_var_2, ...

Function set

Kind of primitive	*Example(s)*
Arithmetic	+, -, *, /
Boolean	AND, OR, ==, >
Conditional	if-then
Looping	while, for
Vector functions	dot product, cross product, Normalize, ...
Plane functions	MovePlane, getNormal, RotatePlane, ...

fitness values in the current generation. Randomly a value in this interval is chosen. The genome with the closest fitness value is taken as parent. If this fitness value is shared by more than one genome a tree size based ranking is used. It was demonstrated that this approach works well for complex learning tasks

The best half scheme simply ranks the individuals by the fitness values and selects the fittest 50 percent.

2) Reproduction: In our GP-framework reproduction is implemented similar to a pipeline. In the first step a genome is picked following one of the selection strategies. Afterwards, depending on the used crossover probability another genome may be picked and an offspring may be produced. Thereupon the original genome or the offspring will be mutated. The mutation may produce the input without modifications. Consequently, a member of the next generation may be a simple copy, an offspring, a mutation or a mutated offspring of a given genome. An exception is the usage of *elitism* in which case the best genome will be copied without modification into the next generation. *Elitism* can have positive and negative effects. On the one hand it ensures that the best genome will be kept in the population. On the other hand if only a limited number of new genomes is created it causes some kind of unfairness in the selection of parents.

Our homologous *crossover* operator is type-constrained. Crossover points are always chosen between similar nodes. Similarity of nodes is defined by their *return type*. Similar return types, for example have a terminal node for a 3-D point and a function node computing a 3-D point. Furthermore the crossover point selection is done following Koza's [1] suggestion to select crossover points not uniformly. We select terminals 10 percent of the time and function or operation nodes 90 percent of the time.

For sampling the solution space adequately it is important to keep the population diverse. This allows computing better solutions while maintaining the performance of the so far best solutions. In [8] it was shown that using mutation, especially allowing root node mutation, is crucial for keeping a good population diversity without using explicit diversity operations. Our GP-framework supports *subtree* as well as *point mutation*. In subtree mutation one node of the tree is chosen randomly and replaced by a randomly generated subtree with fitting return type. Subtree mutation will be applied only once per genome. Point mutation (sometimes referred as node mutation) is more gentle to the tree and can be interpreted as a rough equivalent of the bit-flip mutation used in genetic algorithms. If point mutation is used every node in the tree may be target of mutation. If a node is chosen it will be flipped for example by replacing a terminal with another terminal or an operation with another operation while keeping the inputs of the operation.

3) Tree size and bloat: For runtime considerations and later analysis of the created rules it is crucial to limit the tree growth. Restricting the tree size can be achieved in multiple ways. One way is to border the tree depth. We agree with Parkins [5] that this is a poor choice, because it limits the power of crossover and mutation operators and forces us to select a suitable depth. A more suitable way is to fix the number of allowed nodes in the tree. This preserves the strength of the genetic operators, while stopping the tree to grow infinitively. The constraint on the maximum number of nodes is enforced as a special type of *size enforcing mutation*. Our idea is inspired by the work of Crawford-Marks [27]. In contrast to them we incorporate the size constraint in the mutation operator. If a tree exceeds the maximum number of nodes, instead of standard mutation randomly selected branches of the tree are deleted.

4) Fitness computation and rank: The goal of our GP optimization is to improve several of the cutting plane rules. Therefore we compare a plane defined by a genome with a cutting plane set by an expert user. The similarity measure includes orientation and localization of the planes. The orientation measure d_O compares the normals $\vec{n}_A$ and $\vec{n}_B$ of plane A respectively plane B.

$$d_O(A,B) = 1 - |<\vec{n}_A, \vec{n}_B>| \tag{1}$$

In eq. (1), $<,>$ denotes the inner product. The localization measure d_L compares the distance of plane A to the origin d_A respectively d_B.

$$d_L(A,B) = |d_A - d_B| \tag{2}$$

The fitness function combines both measure with a weighting factor α ($0 \leq \alpha \leq 1$).

$$d_W(A,B) = \alpha \cdot d_O(A,B) + (1-\alpha) \cdot d_L(A,B) \tag{3}$$

The genomes with the smallest fitness value d_W will be ranked best. If there are genomes with the same fitness then a *size constrained ranking* is used which incorporates the tree size of the genomes as a second criterion. If this still does not solve the ranking uniquely, a random selection is done.

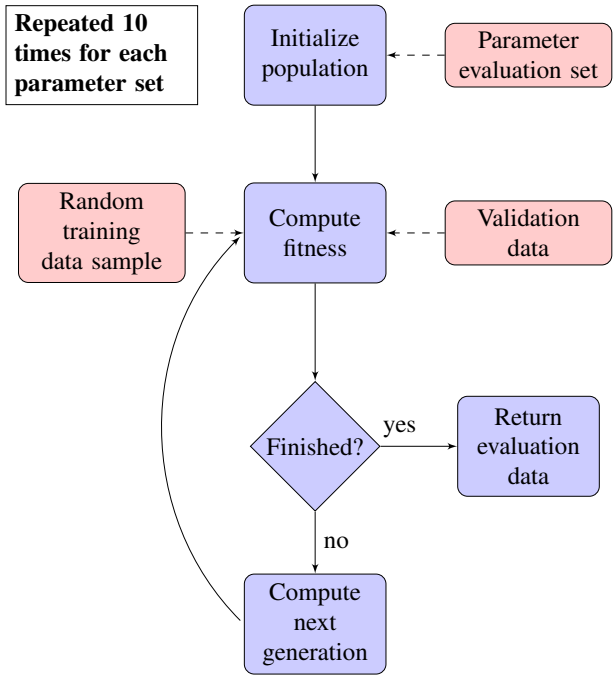

Fig. 5. Scheme of evaluation done with the genetic framework. A training sample is chosen randomly from the training set. Validation is done with all samples in the validation set. For statistically significant results the whole scheme is repeated 10 times for each parameter set.

IV. Results

A. Data set

The data set consists of 105 samples. Each sample contains the ear impression as a triangulated mesh, the result of the feature detection as well as the expert defined target plane. The detected and used features are composed of nine feature points and five feature planes, a subset is shown in Fig. 3. One interesting fact about the data is that the coordinate system is different for each sample. The samples are rotated and translated due to the acquisition procedure [22]. One claim about GP is, that there is no need to preprocess the data [2]. This might not be true for any GP-problem, but in our case the data preprocessing would only involve rotation and translation correction. We assume, that our GP-framework can handle this and consequently did no preprocessing.

To simplify the data we only chose samples of left ears, which reduced the sample set to 65. 35 samples were used as training samples and 30 for validation. Fig. 5 shows a scheme of the evaluation procedure applied.

B. Parameter selection

Since GP has many parameters to tweak we did some preliminary experiments and used parameters settings published in the literature as a starting point for the following experiments (Tab. II). In our experiments we focused on the influence of the selection strategies, crossover probability, mutation method, mutation probability as well as population and tree size.

TABLE II
Default parameters values for the GP-framework.

Parameter	*Value*
Population size	200
Max number of generations	100
Stopping threshold	10^{-6}
Fitness weighting factor	0.9 (see eq. 3)
Initial tree size	350
Max tree size	1000
Selection strategy	Tournament 2
Crossover probability	0.5
Crossover non-uniform selection	0.1 / 0.9
Mutation method	Subtree mutation
Mutation probability	0.5
Elitism	on

C. Experiments

1) Fitness evaluation: Our genome programs target is to compute a plane while using a certain syntax. Every root branch of the genome tree represents a line of the program (rule). Consequently, each line may compute a plane, which may be used an modified in the following lines. In other words a genome consists of several trees, which have a certain execution order. To avoid special restrictions on the genetic operators we developed a special performance evaluation scheme: *dynamic fitness*. The scheme will take all planes defined by the tree into account. The final fitness value of a genome is equal to the fitness value of the fittest plane defined in its syntax tree. *Note:* In the following fitness evaluation graphs the fitness value is very good from the beginning. The reason for this is that one of the input feature planes is a good starting point for the plane we want to optimize.

2) Population size and number of generations: Before starting the detailed evaluation we did some experiments with different values for maximal number of generations and population size. We could verify the statement by Poli et al. [2] that usually the best solutions are found in early generations. In our case the best solutions were generated between the 16 and 253 generation. Most of the time the best rule was generated in the first 100 generations. Consequently we did the detailed evaluation with 100 as parameter for the maximum number of generations.

For the population size our experiments indicated that a good trade-off between execution speed and performance is to use a population size of about 200.

3) Selection strategies: To find the best selection strategy for our system we compared the simple best half (b2), tournament 2 (t2), 5 (t5) and the FUSS strategy. The evaluation results are given in Fig. 6 and Tab. III.

The performance graph rules out FUSS as a suitable selection strategy. In spite of the very good best fitness value the strategy does not converge very well. The other three strategies behave very similar. In the end we decided to go on with t2, because the development of the program size showed a better diversity. Diversity is defined as the standard deviation of the tree size in the population. The selection

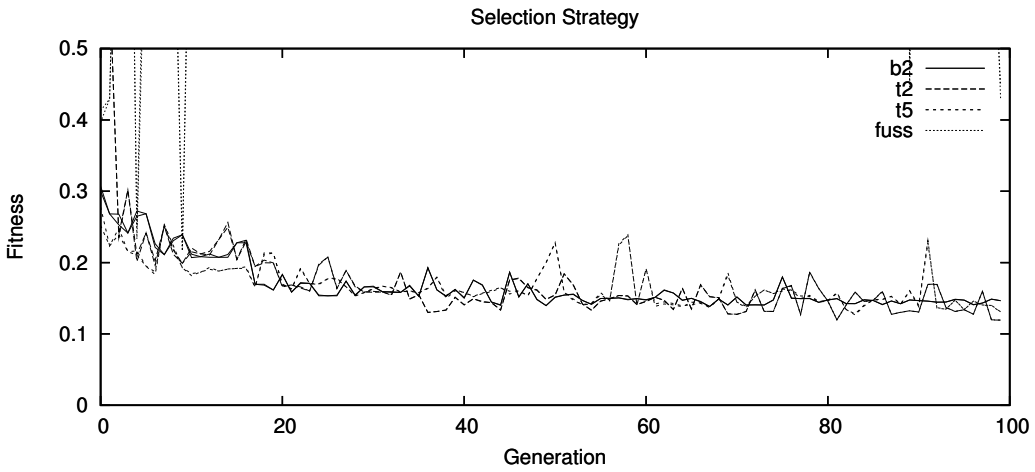

Fig. 6. Graph showing the fitness development for different crossover strategies. *Note:* The graph shows the averaged fitness of the validation set in contrast to Tab. III, which contains the training values.

pressure for b2 and t5 is very similar comparing the mean and standard deviation of the tree size. For t5 and b2 the mean was more or less constant after 10 generations. The standard deviation of the program size dropped for t5 and b2 after 10 generations below 10. In contrast to that, the mean size for t2 varied between 250 and 400 nodes. The standard variation dropped below 10 after approximately 30 generations. Hence, t2 allows a higher population diversity with similar performance compared to t5 and b2 (see, Tab. III).

TABLE III
RESULTS OF THE SELECTION STRATEGY EVALUATION, SHOWING THE AVERAGE RESULTS OF ALL EVALUATION RUNS.

Selection strategy	*Best fitness value*	*Best generation*
Best half	0.01392	35
Tournament selection 2	0.01592	21
Tournament selection 5	0.01275	18
FUSS	0.00600	54

4) Crossover: After fixing the selection strategy on tournament 2 we evaluated the influence of the crossover probability. We started with 0.5, which was inspired by the parameter default setting described in [2].

For evaluation we varied the crossover probability between 0 and 1 with step size 0.25. The results in Tab. IV show that it is reasonable to use higher probability rates. This is supported by the fact that the diversity of the population is higher in these cases. In addition, an analysis of the evolved programs showed that more branches of the syntax tree had influence on the final outcome with higher crossover probability.

We selected 0.75 as default value for the next experiments, because it achieved the lowest fitness value and had similar diversity properties compared with 1.0.

5) Mutation: Having the crossover probability set we focused on the mutation parameters. We started with comparing the two implemented mutation methods. The results

TABLE IV
RESULTS OF THE CROSSOVER PROBABILITY EVALUATION SHOWING THE AVERAGE OF ALL RUNS. THE MUTATION PROBABILITY WAS SET TO 0.

Probability	*Best fitness value*	*Best generation*
0.0	0.02307	5
0.25	0.01176	59
0.5	0.00671	42
0.75	0.00407	31
1.0	0.00445	42

are shown in Fig. 7 and Tab. V.

The graph indicates to favor subtree mutation for the remaining experiments. The point mutation did not converge very well in our experiments unlike the subtree mutation which reached its optimum in generation ≈ 46 and afterwards remained at this level. Concerning population diversity both methods performed quite similar. Point as well as subtree mutation induce a high standard variation for the tree size in all generations.

TABLE V
COMPARISON OF SUBTREE AND POINT MUTATION WITH MUTATION PROBABILITY 0.5 SHOWING THE AVERAGED RESULTS OF ALL RUNS.

Mutation method	*Best fitness value*	*Best generation*
Subtree mutation	0.00376	65
Point mutation	0.00408	60

After choosing subtree mutation as mutation method we evaluated the mutation probability similar to the crossover probability by evaluating the range [0,1] with step size 0.25. The results are shown in Tab. VI. The results indicate that high mutation probabilities should be used. This is supported by the fact that the evolved rules were using more complex rules – cooperation of several tree branches, while lower

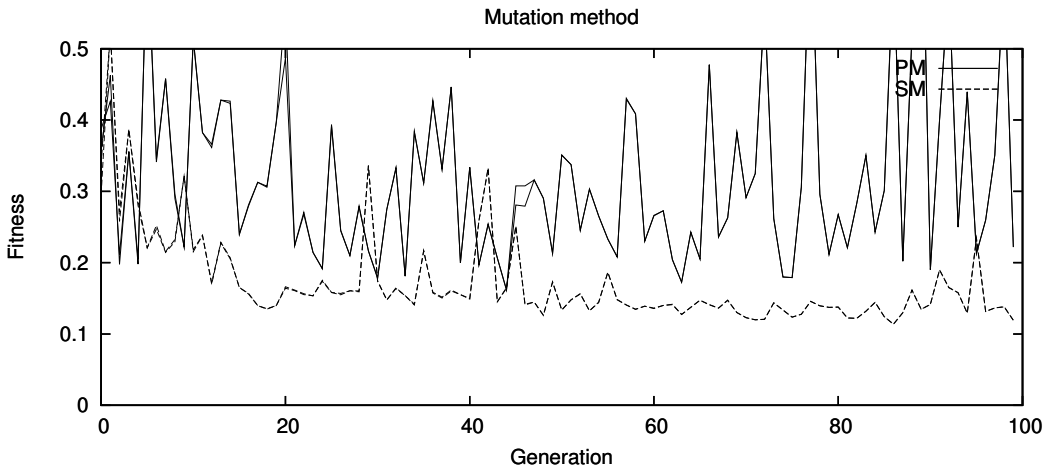

Fig. 7. Graph showing the fitness development of subtree (SM) and point (PM) mutation. *Note:* The graph shows the averaged fitness of the validation set in contrast to Tab. V, which contains the training values.

values evolved rules which rely only slightly modified input features. In the end we decided to use 0.75 for the final experiments. It did not only compute the best fitness values and converged faster than 1.0, but also created a good population diversity with a standard deviation in tree size of about 150 nodes in each generation.

TABLE VI
COMPARISON OF DIFFERENT MUTATION PROBABILITIES USING SUBTREE MUTATION.

Probability	*Best fitness value*	*Best generation*
0.0	0.02347	4
0.25	0.00637	66
0.5	0.00376	65
0.75	0.00275	54
1.0	0.00282	70

6) Initial genome size: The last parameter we evaluated was the initial genome size. We used as default value 350 which approximately is the number of nodes needed to encode the current rule in the expert system. *Note:* For each genome the initial size is modified by a random factor. To verify this estimated value we run experiments with initial size equal to 50, 100, 200, 350 and 500. Our experiments showed that the initial starting size did not have high impact on the mean and standard deviation of the genome size. In all cases the mean converged to approximately 700 nodes and the standard deviation varied between 100 and 200 nodes. Comparing the fitness value in Tab. VII we decided to use 200 nodes as initial genome size. This setting outperformed the others in respect to the fitness value as well as the convergence speed.

D. Resulting rule

For final evaluation we used the settings defined in Section IV-C. We manually analyzed the rules defined by the best genomes of each run. On average approximately 70 percent of the program does not influence the computed plane. The two major findings are:

1) an indication to update the target plane with a simple shift along the normal of a feature plane and
2) a rotation of the target plane defined by feature points not taken into account so far.

To verify these findings we compared the performance of the expert system before and after including the new rules into the knowledge base. Tab. VIII shows that the fitness improved about 22 percent on the left samples and 35 percent on the right samples. The latter surprised us positively since we only used samples of the left side during our evaluation.

However, since we could rely only on a small data set, the result should be interpreted with care. In addition, the

TABLE VII
COMPARISON OF DIFFERENT INITIAL VALUES FOR THE GENOME SIZE SHOWING THE AVERAGED RESULTS OF ALL RUNS.

Initial size	*Best fitness value*	*Best generation*
50	0.01183	63
100	0.00512	60
200	0.00492	27
300	0.00548	68
350	0.01849	25
400	0.03488	32
500	0.01143	33

TABLE VIII
FITNESS EVALUATION WITH AND WITHOUT RULE UPDATE. μ_0 INDICATES THE MEAN FITNESS VALUE FOR THE OLD RULE AND σ_1 THE STANDARD DEVIATION OF THE NEW RULE.

Side	$\mu_0(\pm\sigma_0)$	$\mu_1(\pm\sigma_1)$	*Improvement*
left	0.333 (0.235)	0.259 (0.266)	$\approx$ 22 %
right	0.246 (0.078)	0.158 (0.067)	$\approx$ 35 %

labeling was done by one expert only and the result may differ, if we acquire a second data labeled by a different expert.

V. Conclusions

In this paper, we introduced the idea of utilizing genetic programming to enhance the rule base of an expert systems. Therefore we used a general genetic programming framework with syntax trees as genomes. The tree nodes, terminals, operations and functions follow the syntax in which the rule base is defined. To adapt our general genetic programming framework to the optimization task we introduced size enforcing mutation, dynamic fitness calculation and size constraint ranking.

In an extensive evaluation using real data, we identified the GP-parameters such as crossover and mutation rate, that suited the problem at hand best. The manual analysis of rules generated by the GP-framework, on one hand offered new knowledge about dependencies between several input features and yielded to a significantly improved rule in the expert system (22 percent). On the other hand it verified several assumptions about feature dependencies used in the rule based system.

Our next steps will be to acquire a larger sample set and extend our evaluation on the other cutting and rounding planes in the rule base. To improve the evaluation speed and for simpler integration of other evolutionary computation techniques we plan to employ to utilize the Open BEAGLE framework.

References

[1] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[2] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published by `http://lulu.com`, 2008, available at `http://www.gp-field-guide.org.uk`.

[3] M. Willis, H. Hiden, P. Marenbach, B. McKay, and G. A. Montague, "Genetic Programming: An Introduction and Survey of Applications," in *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, A. Zalzala, Ed. University of Strathclyde, Glasgow, UK: Institution of Electrical Engineers, 1-4 Sep. 1997, pp. 314–319.

[4] J. Link, P. Yager, and J. Anjos, "Application of genetic programming to high energy physics event selection," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 551, no. 2-3, pp. 504 – 527, 2005.

[5] A. Parkins and A. Nandi, "Genetic programming techniques for hand written digit recognition," *Signal Processing*, vol. 84, no. 12, pp. 2345 – 2365, 2004.

[6] P. Nordin, W. Banzhaf, and M. Brameier, "Evolution of a world model for a miniature robot using genetic programming," *Robotics and Autonomous Systems*, vol. 25, no. 1-2, pp. 105 – 116, 1998.

[7] C. Fonlupt, "Solving the ocean color problem using a genetic programming approach," *Applied Soft Computing*, vol. 1, no. 1, pp. 63 – 72, 2001.

[8] K. Badran and P. I. Rockett, "The influence of mutation on population dynamics in multiobjective genetic programming," *Genetic Programming and Evolvable Machines*, 2009, online First.

[9] M. Hutter, "Fitness Uniform Selection to Preserve Genetic Diversity," in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC-2002)*. IEEE, 2001, pp. 783–788.

[10] P. Kouchakpour, A. Zaknich, and T. Bräunl, "Population variation in genetic programming," *Information Sciences*, vol. 177, no. 17, pp. 3438–3452, 2007.

[11] R. Poli and W. B. Langdon, "Genetic Programming with One-Point Crossover and Point Mutation," in *Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag, 1997, pp. 180–189.

[12] K. Sickel, S. Baloch, V. Bubnik, R. Melkisetoglu, S. Azernikov, T. Fang, and J. Hornegger, "Semi-automatic manufacturing of customized hearing aids using a feature driven rule-based framework," in *Proceedings of the Vision, Modeling, and Visualization Workshop in Braunschweig*, 2009.

[13] D. Andre, "Learning and Upgrading Rules for an OCR System Using Genetic Programming," in *In Proceedings of the 1994 IEEE World Congress on Computational Intelligence*. IEEE Press, 1994, pp. 27–29.

[14] R. R. F. Mendes, F. de B. Voznika, A. A. Freitas, and J. C. Nievola, "Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution," in *Principles of Data Mining and Knowledge Discovery, Lecture Notes in Artificial Intelligence*. Springer Verlag, 2001, pp. 314–325.

[15] J.-Y. Potvin, P. Soriano, and M. Valle, "Generating trading rules on the stock markets with genetic programming," *Computers & Operations Research*, vol. 31, no. 7, pp. 1033 – 1047, 2004.

[16] P. Legrand, C. Bourgeois-Republique, V. Péan, E. Harboun-Cohen, J. Levy-Vehel, B. Frachet, E. Lutton, and P. Collet, "Interactive evolution for cochlear implants fitting," *Genetic Programming and Evolvable Machines*, vol. 8, no. 4, pp. 319–354, 2007.

[17] A. Tsakonas, G. Dounias, J. Jantzen, H. Axer, B. Bjerregaard, and D. G. von Keyserlingk, "Evolving rule-based systems in two medical domains using genetic programming," *Artif. Intell. Med.*, vol. 32, no. 3, pp. 195–216, 2004.

[18] 3Shape A/S, "3Shape Hearing Aids, ShellDesigner," `http://www.3shape.com/our-products/hearing-instruments/shelldesigner.aspx`, June 2009.

[19] J. C. Giarratano and G. D. Riley, *Expert Systems: Principles and Programming*, 4th ed. Course Technology, 2004.

[20] C. Donnelly and R. M. Stallman, *The Bison Manual, Using the YACC Compatible Parser Generator*, 8th ed. Free Software Foundation, 2003.

[21] S. Baloch, V. Bubnik, R. Melkisetoglu, S. Azernikov, and T. Fang, "Automatic Detection of Anatomical Features on 3D Ear Impressions for Canonical Representation," in *Proceddings of Geometric Modeling and Processing (GMP)*, 2010, (under review).

[22] T. Gao and S. S. Jarng, "Design and realization of hearing aids based 3d rapid shell molding cadcam," in *IEEE International Symposiumon Industrial Electronics, 2009. ISIE 2009.*, july 2009, pp. 1488 –1492.

[23] K. Sickel, "Shortest Path Search with Constraints on Surface Models of In-ear Hearing Aids," in *52. IWK, Internationales Wissenschaftliches Kolloquium*, P. Scharff, Ed., vol. 2, Ilmenau, 2007, pp. 221–226.

[24] boost.org, "Boost C++ Libraries," `http://www.boost.org/`, November 2009.

[25] J. Gottschlich, "C++ Trees Part 1 and 2," `http://www.gamedev.net/reference/programming/features/coretree2/`, November 2009.

[26] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel programming in OpenMP*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[27] L. Spector, R. Crawford-Marks, and R. Crawford-marks, "Size control via size fair genetic operators in the pushgp genetic programming system," in *In*. Morgan Kaufmann Publishers, 2002, pp. 733–739.