Incorporating Linkage Learning into the GeLog Framework*

Tim Führer† and Gabriella Kókai†

Abstract

This article introduces modifications that have been applied to *GeLog*, a genetic logic programming framework, in order to improve its performance. The main emphasis of this work is the structure processing of genetic algorithms. As studies have shown, the linkage of genes plays an important role in the performance of genetic algorithms. Thus, different approaches that take linkage learning into account have been reviewed and the most promising has been implemented and tested with *GeLog*. It is demonstrated that the modified program solves problems that proved hard for the original system.

1 Introduction

The *GeLog* program combines two approaches, inductive logic programming (ILP) and artificial evolution [1]. This work aims at improving the *GeLog* framework by incorporating methods that help the evolutionary algorithm to maintain a rugged search behavior without losing the ability to quickly find (local) optima. Both requirements are most relevant to noisy search spaces, which are often characteristic in inductive logic programming. This article introduces the modifications that were applied to the *GeLog* framework and presents the results of two experiments, which demonstrate that the program has been drastically improved.

The following section briefly introduces the *GeLog* framework. Section 3 explains the term linkage and introduces related approaches. The modifications that have been applied to *GeLog* are depicted in Section 4. In Section 5 some test results are presented. Finally, Section 6 concludes this article and provides a short outlook on future investigations and improvements.

2 Brief Introduction into GeLog

The *GeLog* framework is a genetic logic programming framework, an inductive logic programming system combined with an evolutionary search algorithm [1]. In-

*This work is supported by the grants of Bayerischer Habilitationsförderpreis 1999.

†Department of Computer Science II, University of Erlangen-Nuremberg, Martensstr. 3, 91058 Erlangen, Germany, e-mail: fuehner@iis-b.fhg.de, kokai@informatik.uni-erlangen.de

ductive logic programming is a machine learning approach, in which correlations of objects are ascertained by induction. Hypotheses are searched for and evaluated by comparing their classification results with a sufficiently large number of instances for which it is known whether their objects are correlated or not [2]. It is thus assumed that hypotheses classifying these training instances correctly will also approximate the target function well over any other set of instances. The learned hypotheses can be interpreted as PROLOG programs, since they consist of set of rules, that is, first order Horn clauses.

GeLog's data representation resembles more to that of genetic programming: the individual solutions consist of PROLOG program parts, which encode the hypotheses' rules. Thus, an individual comprises the target predicate as its left hand side and a number of disjunctions (right hand sides), all of which are conjunctions of literals. The following example demonstrates how individuals are represented in the the original *GeLog* implementation:

```
daughter(X0, X1) :- female(X0), parents(X1, X0, X1).

                    parents(X0, X1, X1).

                    female(X1), female(X0), parents(X1, X1, X0).
```

The depicted individual consists of three disjunctions (right hand sides); each disjunction contains a number of conjuncted literals and is terminated by a dot.

The pay-off of one hypothesis results from the number of correctly classified instances. Different selection operators have been implemented: Roulette Wheel Selection, Rank Selection, and Elitism (for further explanation of these operators see [3] and [4]).

Due to the non-standard data representation special recombination and mutation operators had to be implemented:

- Two recombination operators; (1) two individuals exchange entire disjunctions by single- or multi-point crossover, (2) two individuals exchange predicates by performing single- or multi-point crossover at disjunction level.
- Mutation operators; (1) insertion and deletion of literals, (2) insertion and deletion of entire disjunctions, (3) insertion of new variables, and (4) substitution of variables.

3 Linkage Learning and Related Work

The first complete theory of the dynamics and processing units of genetic algorithm was developed by Holland [5]. In his schema theorem he suggested that genetic algorithms process the search space implicitly parallel. A specific individual is also a representative of a class of individuals that have certain gene values (alleles) in common. For example, individual 100101 represents the class of individuals with a leading '1' (denoted as 1*****); but it also represents individuals that contain two '0' alleles on second and third position (*00***), etc. Thus, by selecting individual solutions the (schema) classes which are represented by the individual gain influence. For example, if *00*** exhibits a relative high fitness, i.e., individuals that contain the specified '0' alleles are on average fitter than others are, this schema is represented more often than other schemata. A higher fitness is achieved if those parts of solutions are recombined that caused the former individuals to exhibit a higher fitness than other individuals. In other words, by combining fit schemata even fitter schemata are generated.

Based on the insights attained by the schema theory Goldberg formulated what he called the building block hypothesis [3]. He concluded that the central processing units of genetic algorithms are "short, low-order, and highly fit schemata". These entities he called building blocks. Goldberg also found that some problems are hard to solve for genetic algorithms because of difficulties in processing building blocks. Consider four building blocks: H_1 = 1*****, H_2 = *****1, H_3 = 0*****, and H_4 = *****0. Let the fitness of H_1 and H_2 be remarkably greater than the fitness of H_3 and H_4, also let the fitness of a recombination of H_1 and H_2 (1****1) be smaller than 0****0 (the combination of H_3 and H_4). As the two recombined schemata exhibit a relatively high order, chances are high that they are disrupted quickly, resembling schemata $H_1 - H_4$. Since the selection probabilities for schemata H_3 and H_4 are low it is difficult for the genetic algorithm to recombine them both yielding the highly fit schema 0****0 again.

The situation changes if the defining genes of the schemata are linked more tightly, since the probability of disruption decreases drastically. On the one hand that increases the chances of preserving the fit recombined schema, on the other hand it ensures that the unfit schema is discarded and not split into the two fit sub-schemata which lead to the deception. This is obviously a simplification of the dynamics of genetic algorithms and has been criticized for that reason (cf. [6, 7]). However, it could be shown that for many problems improving the linkage situation of building blocks also improved the performance of the genetic algorithm. It is therefore worthwhile to develop techniques that lead to tighter linkage of building blocks.

It was long assumed that individuals in genetic algorithms would eventually evolve towards tighter linkage. However, early efforts that used inversion operators to achieve tight linkage proved that selection is too powerful and thus counteracting linkage learning [8].

3.1 Messy GA

One of the early approaches that took this observation into account was the so-called *messy genetic algorithm* [9]. In addition to a "messy coding" which allowed for a reordering of the chromosome, linkage learning and selection were separated into two phases such that selection is prevented from vitiating linkage learning. The two phases are repeated alternately increasing the order of building blocks that are processed. In the first phase all possible building blocks of the current order are generated. This explicit enumeration is very expensive ($O(2^k\ell^k)$, where ℓ is the chromosome length and k is the highest order of building blocks, i.e., the number of genes, that define a building block). After this enumeration the threshold operator tries to select individuals such that only those compete that define the same class of schemata. For example, `00*` and `11*`, but not `0*1` and `*01`. The second phase resembles to a simple genetic algorithm. A variation of this approach replaces the expensive enumeration of all building blocks of a specific order by a probabilistic technique [10]. Instead of generating all order-k schemata explicitly, this technique makes use of the fact that one bit string may contain multiple schemata at the same time, since the bit string is normally longer than the order of the schema. Thus, only a fraction of the former $O(2^k\ell^k)$ individuals had to be created. The threshold selection operator must then decrease the string lengths, such that only fit schemata remain. However, the threshold selection operator has proven quite unfit in this task [11].

3.2 Gene Expression Messy GA

Another messy genetic algorithm was developed by Kargupta [12]. The process of gene expression as observed in nature inspired his approach. Consequently this type of algorithm is called *gene expression messy genetic algorithm (GEMGA)*. The linkage learning is done by induction; the genes that improve the solution's pay-off are assumed to correlate. In a *first transcription phase* the contribution of a gene is to the fitness of the individual is determined. This is done by flipping each gene to its opposite value if the fitness increases, the original value does not contribute to the fitness, otherwise it does and is marked such that it cannot be changed in the future. In the *second transcription phase* all genes in a chromosome that have been marked as unchangeable are collected and compared with the same unchangeable genes of another randomly chosen chromosome. The intersection of the genes is saved (linkage set) and either is added to a list of the former chromosome or, if the set is already present, its weight is increased. After some iterations a matrix is build, which contains the probabilities of the presence of a gene under the condition that a specific gene is in the linkage set.

Afterwards, the schemata that have been identified as good are manifolded using *class selection*: two chromosomes are randomly picked, the fitter of both is marked, the genes in the linkage set of the marked chromosome are copied to the other chromosome, provided that the destroy genes exhibit less linkage than the genes by which they are replaced. Additionally tournament selection is applied.

Recombination is done by randomly picking an individual and selecting its maximum weighted linkage set, another individual is selected, and the corresponding genes are exchanged if the disrupted linkage sets in the latter chromosome have a smaller weight than the maximum weight of the former.

3.3 Linkage Learning GA

A completely different approach was taken by Harik [13]. Harik showed that a specific recombination operator, the so-called *exchange crossover operator*, could under certain conditions improve the linkage of genes. The chromosomes in Harik's linkage learning genetic algorithm (LLGA) are declared as rings. Each gene is described by its allele (value) and a locus, i.e., the interpretation position of this gene. By introducing so-called introns, genes that are not interpreted at all, the relative distance of two genes can be adjusted. In Figure 1 an example of a chromosome containing three genes is given. One can see that by inserting non-coding genes (introns) between the coding genes (exons) the distances (y_1, y_2, and y_3) can be varied.

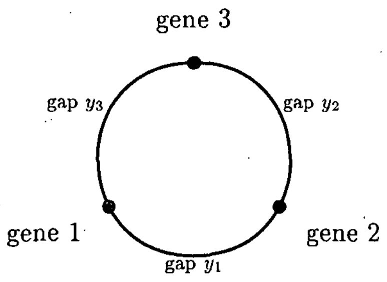


Figure 1: Chromosome in linkage learning genetic algorithm.

In contrast to most common implementations the *exchange crossover* operation is directional, that is, one individual serves as donor, the other one is the recipient. First some exchange material is randomly chosen from the donor chromosome, then a random graft point is declared at the recipient. The exchange material is then inserted within the graft point of the recipient. As one can see in Figure 2 the crossover leaves an over-determined chromosome, that is, some of the genes appear twice. Therefore an expression step is appended: A starting point and an interpretation direction are defined. Beginning from the starting point each gene that has been previously defined on the circle is simply removed, yielding a valid chromosome.

Harik proved that by applying this operator the individuals evolve towards tighter linkage. He assumed that the population will eventually consist mostly of both optimal building blocks and deceptive building blocks (as described earlier). This assumption can be made as the genetic algorithm eventually rules out all apparently unfit building blocks. Harik observed two effects:

(1) *Linkage Skew:* tightly linked building blocks in the donor chromosome have a higher survival probability than loose linked building blocks. This mechanism is comparable to fitness-proportional selection.

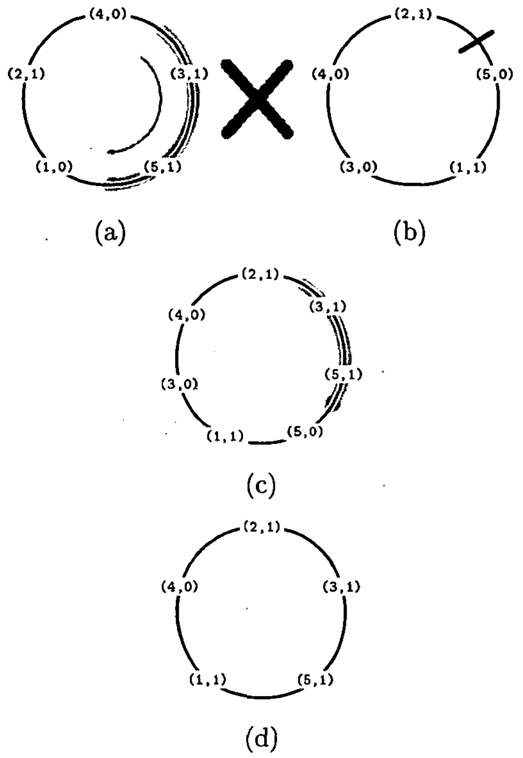


Figure 2: Exchange Crossover Operator: (a) donor, (b) recipient, (c) offspring before expression, and (d) offspring after expression.

(2) *Linkage Shift:* if exchange material from the donor is copied onto the recipient—which contains an optimal building block—the building block is either disrupted or its linkage is increased.

However, in order for linkage learning to work selection must be slowed down, since it counteracts the evolution towards tighter linkage (as shown by Harik). Harik suggests two different methods to slow down selection:

(1) *Restricted tournament selection*: In contrast to conventional selection operators where each individual is competing against one another, with tournament selection individuals only replace solutions which have a similar bit-string [13].

    Thus, this selection operator is not only well suited for multi-modal optimization tasks, but will also maintain a high level of diversity within the population.

    The main program's pseudo code listing in Section 4.5 comprises a detailed description of restricted tournament selection.

(2) Probabilistic expression: This approach refers to an alternative way of chromosome encoding in which all genes appear twice, exhibiting the actual allele and its opposite. The starting point of the chromosome interpretation is randomly changed, resulting in a change of the genes' alleles with some probability. Thus, even if an allele has leveled out in the population, it might be revived.

3.4 Summary

The *messy genetic algorithm* and its variant the *fast messy genetic algorithm* can be considered early approaches. They have proven to work in limited settings, but haven proven infeasible for real-world applications. A very promising approach has been suggested by Kargupta [12]. However, the number of additional fitness evaluation (in the transcription phase) and the large administration effort, which is necessary in order to store linkage information, seem to be a remarkable drawback. Harik's LLGA, on the other hand, proved to work well on exponentially scaled problems, that is, problems where parts of the genes contribute differently to the fitness value. As the hard problems for *GeLog* can be assumed to belong to this kind of problem class, this approach seems well suited for *GeLog*.

4 Improving the GeLog Framework

This section introduces the modifications that were applied to the data structures and the operators, which were implemented in order to achieve linkage learning. We have chosen the Linkage Learning GA (LLGA) approach to achieve this goal, since it offers a relatively good scalability and the genotypic representation is appropriate for *GeLog*. Moreover, the apparently reasonable theory of the LLGA and the promising results suggested an application to the *GeLog* framework.

The probabilistic expression (PE) as suggested by Harik [13] is not incorporated for the maintenance of diversity. Instead, tournament selection and restricted tournament selection are used. While tournament selection is a standard selection scheme in genetic algorithms, restricted tournament selection is commonly used for multi-objective optimization problems [13].

4.1 Chromosome

The genotypes in *GeLog* are represented by a so-called object graph [1], which allows for a direct transformation into the data structures used in logic programming. However, this representation is not ideal for the processing of building blocks in genetic algorithms. Not only is there evidence [5] that short alphabets have a positive impact on the implicit parallelism, but also for the linkage learning a chromosome of fixed length seems more appropriate. It is important that the entire search space is explicitly represented in one individual.

Except for the necessary changes in the genotype representation, the new version tried to stay as close to the original representation as possible. As in the original work, each individual contains a number of disjunctions. However, in contrast to the previous implementation the number of disjunctions is fixed, i.e., each individual consists of the same number of disjunction. A gene, or bit, in the chromosome stands for one conjunction. A conjunction represents one variable assignment corresponding to the respective predicate.

For example, let the background knowledge, i.e., the pool of valid predicates which may be used at the right hand side of the individual, be:

```
female/1 parents/3
```

with the target predicate: `daughter(X0, X1)`
In the original work an individual could look as follows:

```
daughter(X0, X1) :- female(X0), parents(X1, X0, X1).

                    parents(X0, X1, X1).

                    female(X1), female(X0), parents(X1, X1, X0).
```

The new representation consists of a fixed number of fixed length bit strings. The individual must hence be transformed into something like this:

```
1000000100    (1. disjunction)
0000000010    (2. disjunction)
1100010000    (3. disjunction)
```

How do we achieve an appropriate representation?

First of all, we have to determine the length of the chromosome, since it will be fixed throughout the entire process. Thus, the chromosome's length must allow for encoding all valid predicates with all possible assignments:

$$l = \sum_{p \in B} arity(t)^{arity(p)},$$

where B is the background knowledge, p is one of the background knowledge's predicates, and t is the target predicate.

For the former example the length would be:

$$\begin{aligned} l &= arity(\texttt{daughter})^{arity(\texttt{female})} + arity(\texttt{daughter})^{arity(\texttt{parents})} \\ &= 2^1 + 2^3 = 10 \end{aligned}$$

If we introduce a number of additional, unbound variables (v) that the literals may take as arguments this can be transformed into

$$l = \sum_{p \in B} (arity(t) + v)^{arity(p)}.$$

In our example we allow for additional two variables, with the target predicate's original two variables we obtain variables `X0..X3`:

$$\begin{aligned} l &= (\mathit{arity}(\texttt{daughter}) + v)^{\mathit{arity}(\texttt{female})} + (\mathit{arity}(\texttt{daughter}) + v)^{\mathit{arity}(\texttt{parents})} \\ &= 4^1 + 4^3 = 68 \end{aligned}$$

Let $p_n(x)$ be the nth predicate of the background knowledge with allocation $x = (x_0, \ldots, x_m)$; where $m = \mathit{arity}(p_n) - 1$. All variables x_i (with $0 \leq i \leq m$) must be one of the target predicate's or of additional variables: $0 \leq x_i < (\mathit{arity}(t) + v)$. The locus of this mapping can be calculated as follows:

$$\mathit{locus}(p_n(x)) = \sum_{j=0}^{n-1} (\mathit{arity}(t) + v)^{\mathit{arity}(p_j)} + \sum_{i=0}^{\mathit{arity}(p_n)-1} x_i \cdot (\mathit{arity}(t) + v)^i.$$

Let us calculate the locus of the `parent/3` predicate of the former example using allocation `parents(X3, X1, X1)`.

$$\begin{aligned} \mathit{locus}(\texttt{parents(X3, X1, X1)}) &= (\mathit{arity}(\texttt{daughter}) + v)^{\mathit{arity}(\texttt{female})} \\ &+ 3 \cdot (\mathit{arity}(\texttt{daughter}) + v)^0 + 1 \cdot (\mathit{arity}(\texttt{daughter}) + v)^1 \\ &+ 1 \cdot (\mathit{arity}(\texttt{daughter}) + v)^2 = 4^1 + 3 + 1 \cdot 4 + 1 \cdot 16 = 27 \end{aligned}$$

Thus, gene number 27 indicates whether the predicate `parents(X3,X1,X1)` is present or not. Its allele (value) is either 1 or 0.

The coding of the genes is messy, that is, their position in the bit string is not fixed but they may float around. A gene's predicate allocation is not determined by the gene's position in the bit string but by its locus, which is in general different from the position.

It is quite obvious that the length of the chromosomes is increasing exponentially with the arity of the predicates of background knowledge and the arity of the target predicate. This is problematic since the genes have messy coding, which means every single gene contains a number as large as the chromosome's length. This is necessary since the locus—the position of the genes within the chromosome—has to be stored. For example, for 100 literals in the background knowledge, an average arity of 10, and a target predicate's arity of 10 the chromosome length is 10^{12}, a number that must be stored in all 10^{12} genes of the chromosome.

4.2 Mutation

With the new representation the change of one single bit deletes or adds one mapping of a predicate. Thus, a single allocation, a predicate, or even a disjunction may be erased altogether by the change of one bit. The following example illustrates this: Let the background knowledge be the same as in the previous example. The three rows represent the variable numbers within the literal. The columns denote the subscript of the `X` variables. There are four variables (`X0`, `X1` are arguments of the target literal daughter; X2, X3 are unbound variables), and we need to be able to place any of these four variables on any position within each literal. Therefore, we only need four positions for the female literal, as it takes only one argument (i.e., X0, X1, X2, or X3). For the second literal, parents, we need 4^3 positions, since it requires three arguments.

```
1. Variable  0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123 0123
2. Variable       0000 1111 2222 3333 0000 1111 2222 3333 0000 1111 2222 3333 0000 1111 2222 3333
3. Variable       0000 0000 0000 0000 1111 1111 1111 1111 2222 2222 2222 2222 3333 3333 3333 3333

             0010|0000|0100|0000|0100|0000|0000|0000|0100|0000|0000|0010|0000|0000|0000|0000|0000
```

is equivalent to the structure:

```
female(X2), parents(X1, X1, X0), parents(X1, X3, X0),
parents(X1, X3, X1), parents(X2, X2, X2).
```

Two mutations are performed:

```
                    ϟ                                            ϟ
   0010|0000|0100|0000|0100|0000|0000|0000|0100|0000|0000|0010|0000|0000|0000|0000|0000
⇒  0010|0000|0100|0010|0100|0000|0000|0000|0100|0000|0000|0000|0000|0000|0000|0000|0000
```

This is equivalent to inserting a new predicate allocation (parents(X2, X2, X0)) and deleting one (parents(X2, X2, X2)):

```
female(X2), parents(X1, X1, X0), parents(X2, X2, X0),
parents(X1, X3, X0), parents(X1, X3, X1).
```

Since the modified version of *GeLog* aims at maintaining a high level of diversity, it does not depend on mutation. Compared to the original work, the mutation rates have therefore been decreased drastically.

4.3 Diversity

In order to make linkage learning work, it is necessary to maintain a large diversity in the population. It might therefore be desirable to keep solutions in different parts of the search space and optimize these solutions individually. A better solution replaces another solution only if both are similar to another, i.e., if their distance is small.

In order to evaluate the distance of two individuals, their chromosomes have to be compared. Since the chromosomes are bit strings, the Hamming distance (i.e., the sum of all difference bits) is an appropriate distance metric. As individuals in *GeLog* do not only consist of one but a number of chromosomes, the Hamming distances for all chromosomes have to evaluated. One approach is to calculate the Hamming distance for each pair of chromosomes of the two individuals (x and y), after which the sum of these chromosome-wise distances yields the distances of the individuals:

$$\sum_{c=0}^{m}\sum_{i=0}^{n}|x_i^c - y_i^c|,$$

where m is the number of chromosomes and n is the number of genes. In contrast to a single chromosome where a gene at a specific locus always encodes the same trait, it cannot be pre-termined what kind of disjunction one chromosome will be coding for. Therefore, the distance between two individuals is not as obvious as in a single chromosome case. One extreme case is, two individuals containing the same chromosomes yet in a different order.

One solution to this problem is to form the sum of the distances of all chromosomes in one solution with all the chromosomes in the other solution:

$$\sum_{c=0}^{m}\sum_{d=0}^{m}\sum_{i=0}^{n}|x_i^c - y_i^d|,$$

where m is the number of chromosomes and n is the number of genes. This distance metric is referred to as `sum/sum`.

This approach, however, does not aim at finding corresponding chromosomes in the compared individuals. For example, although the sum of all distances might be large, the distance between certain chromosomes is possibly small. The `sum/min` approach takes this into account by identifying matching slots. By finding the permutation p of disjunctions that maximizes

$$\sum_{c=1}^{m} \frac{\min\limits_{\substack{1\geq j\geq m;\\ j\neq c}} \left\{ \sum_{i=0}^{n} |x_i^c - y_i^{p_j}| \right\}}{\sum_{i=1}^{n} |x_i^c - y_i^{p_c}|}$$

those disjunctions are identified that exhibit a much smaller distance than the disjunctions with the second smallest distance.

```
for all $c_1$ := chromosomes in individual1 do
  find $c_2$ := unmarked chromosomes in individual2 with minimum distance to $c_1$
  if two chromosomes have the same minimum distance then
    choose one randomly
  end if
  store distance
  mark chromosome $c_2$
end for
sum up all stored distances
```

Figure 3: Pseudo code of the sum/min algorithm

The algorithm's complexity is $O(n \cdot n!)$ in the number of chromosomes. Since distance comparisons are needed very frequently, *GeLog* uses a variation of this procedure. Instead, all distances between all chromosomes are evaluated and those chromosomes are assumed to match that have the smallest distance. If any chromosome has the same distance to more than one chromosome in the other individual, one of these chromosomes is chosen randomly. Figure 3 demonstrates the algorithm.

4.4 Recombination

The different recombination operators of the original *GeLog* version, as explained in Section 2, have been replaced by a single operator. Selection yields two individuals, a donor and a recipient. First one chromosome is selected in the donor individual and a corresponding chromosome in the recipient is chosen. After crossover has been performed a slot for the new recombined chromosome has to be chosen. This process of choosing chromosomes, recombining them and storing them is repeated for all chromosomes.

Since it is not obvious how to select chromosomes for recombination, several strategies have been developed:

(1) `ordered`: the chromosomes are selected in the order they appear in the individual,

(2) `shuffled`: the chromosomes are randomly shuffled and selected in this new order, or

(3) `fitness`: the chromosomes are selected fitness proportionally, i.e., by roulette wheel selection. This scheme is very expensive, as a huge number of fitness evaluations have to be performed.

The new individual (offspring) is now created by the recombined chromosomes. However, each chromosome has to go into a different slot in the offspring. Therefore, a selection strategy is also required for storing:

(1) `ordered`: the chromosome is stored in the same slot as the recipient's chromosome was selected from,
(2) `shuffled`: a randomly shuffled list of all chromosomes slots is generated, the chromosomes are stored in that new order, or
(3) `similarity`: the chromosome is placed into the slot that has the shortest distance, i.e., the number of the recipients slot which is most similar to the recombined chromosome.

The most important of the nine possible combinations are explained in the following:

(1) `ordered/ordered`: Each chromosome has its fixed slot, for the whole evolutionary process.

(2) `shuffled/similarity`: The parents are chosen randomly, the offspring, however, replaces the chromosome, which it is most similar to. This selection/storing scheme induces a similar distribution on the single individual as the restricted tournament selection did on the entire population.

(3) `shuffled/shuffled`: The parents are selected randomly, the offspring is stored at a random position. This scheme is suitable if restricted tournament selection is used and all clauses should be intermixed.

(4) `fitness/similarity`: As `shuffled/similarity`, but holds the danger that the inner-individual diversity gets lost too fast since only well performing chromosomes are selected.

As introduced by Harik [13] an *exchange crossover operator* was used to recombine two chromosomes. When combined with a harsh control of selection, this operator induces a tighter linkage on the building blocks.

4.5 Flow of GeLog

In this section, pseudo-code is presented for the fitness evaluation routine and for the main program.

The fitness evaluation consists of two parts: first the genome of the individual has to be decoded into a hypothesis, and in a second step it is checked, whether the hypothesis correctly classifies all training instances. The hypothesis must classify negative instances as false and positive instances as true. The fitness value corresponds to the percentage of correctly classified training instances.

Fitness Evaluation

```
input individual $A$
hypothesis $H$ := decode $A$
for all positive training instances $EP[j]$ ($j$ := 0..number of pos. instances) do
  if $H$ accepts $EP[j]$ then
    increase fitness of $A$
  else
    decrease fitness of $A$
  end if
end for
for all negative training instances $EN[j]$ ($j$ := 0..number of neg. instances) do
  if $H$ rejects $EN[j]$ then
    increase fitness of $A$
  else
    decrease fitness of $A$
  end if
end for
```

The following flow demonstrates that the GA flow differs substantially depending on the selection operator. With conventional selection operators the fitness of individuals is evaluated after the new population has been created, whereas with restricted tournament selection more evaluation steps have to be performed.

The program has been implemented using the programming language C++; all experimental runs have been conducted on Intel/AMD processor based computers running the Linux operating system. As PROLOG interpreting system the *SICStus* framework version 3.8.5 was used, which can be easily linked to C/C++ programs. However, calling the external PROLOG process is expensive and decidedly contributes to the time required by fitness evaluations. This circumstance is especially problematic with restricted tournament selection, for which a number of additional fitness evaluations have to be performed.

Main Program

```
initialize prolog interpreter
load background knowledge
randomly initialize start population
for all individuals $A[i]$ ($i := 0..$population size) do
  evaluate $A[i]$
end for
while not (termination criterion reached or max. number of generations) do
  if selection = restricted tournament then
    while not new population complete do
      $A$ := select randomly
      $B$ := select randomly
      $A'$ := exchange crossover $A$, $B$
      $B'$ := exchange crossover $B$, $A$
      mutate $A'$,$B'$
      $W[]$ := select randomly $w$ individuals
      $A''$ := $W[j]$, where distance($W[j]$, $A'$) = $\min_{i:=1..w}$(distance($W[i]$, $A'$))
      evaluate $A'$
      if fitness $A' >$ fitness $A''$ then
        replace $A''$ with $A'$
      end if
      repeat the same for $B'$
    end while
  else
    while not new population complete do
      $A$ := select individual (using any recombination operator)
      $B$ := select individual (using any recombination operator)
      $A'$ := exchange crossover $A$, $B$
      $B'$ := exchange crossover $B$, $A$
      mutate $A'$, $B'$
      place $A'$, $B'$ into the new population
    end while
    evaluate all individuals in the new population
  end if
end while
```

5 Experimental Results

Two experiments that had proven difficult for the original version of *GeLog* have been conducted in order to verify that the performance of *GeLog* has been improved.

5.1 Tic-Tac-Toe

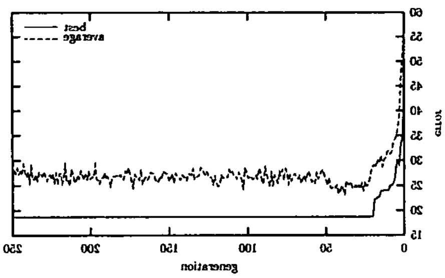
This experiment was introduced by Aha [14] and is based on the Tic-Tac-Toe game. It encodes all possible endgame situations where the player using the "X" symbol has started. The target concept, which takes the nine board squares as variables with values `blank`, `X`, or `O`, is to classify the win situation for player "X". For 626 of the 956 possible constellations player "X" wins.

The performance of the concept learning algorithms on this data set varied remarkably, depending on the used variant of learners: while experiments conducted with decision tree based learners exhibited errors of 20% or more, other algorithms, such as rule based learners performed well on it (errors $\leq 2\%$). With the original version of *GeLog* we could not achieve error results below 24%, which were significantly improved after the modifications.

individual distance	chromosome selection	chromosome storing	selection operator	lowest error (%)	average error (%)
none	ordered	ordered	ts	6.25	9.71
none	**shuffled**	**similarity**	**ts**	**5.26**	**8.66**
sum/min	shuffled	similarity	rts	6.25	10.13
sum/sum	shuffled	similarity	rts	6.25	10.86
original *GeLog* version				24.95	26.91

Table 1: GeLog test results on 'TicTacToe'

Table 1 shows four tests, each consisting of five test runs. All tests have been conducted using ten-cross-validation, that is, the example set is divided into 10 disjunctive subsets. One is declared as test set. This procedure is used in order to avoid over-fitting. The first column indicates the individuals' distance metric. For tournament selection this column contains the word "none", since this selection operator does not utilize the individuals' distances. The second and third columns state the chromosome selection and storing scheme as described in Section 4.4. The selection operator appears in the fourth row, "rts" means restricted tournament selection, whereas "ts" is tournament selection. The depicted settings have been chosen in order to clarify whether restricted tournament selection is in any case necessary to maintain a high level of diversity. Therefore, tournament selection was tested using the ordered selection and storing scheme, thus totally ignoring the distance relation. The second experiment was conducted using random selection but with storage into the slot that exhibits the highest similarity. For restricted tournament experiments the same selection/storing scheme was used with the two different distance metrics, as this might be a critical factor for restricted tournament selection. The mutation probability for all experiments was defined as 0.01, crossover was performed with a probability of 0.6, a population consists of 150 individuals, and the maximum number of generations was 250, however, for tournament selection, the lowest error rate was chiefly obtained after $50-100$ generations.


(a)

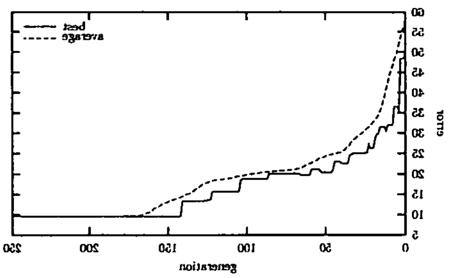

(b)

Figure 4: Minimal versus average error for "TicTacToe": (a) tournament selection, (b) restricted tournament selection

The absence of a significant difference in the results of tournament and restricted tournament selection may indicate that the problem is not difficult enough, i.e., that plain tournament restriction can maintain large enough a diversity.

For the chromosome selection/storing we see a slight difference between "ordered/ordered" and "shuffled/similarity", which can be attributed to higher in-individual diversity, that is, clauses have greater differences.

Figure 4 shows two runs of the same experiment, however only 100 individuals per population are generated. It becomes clear that restricted tournament selection is aiming at improving the fitness situation of the entire population.

individual distance	selection operator	run time (25 generations)
none	ts	≈ 3m50s
sum/sum	rts	≈ 5m
sum/min	rts	≈ 5m8s
original *GeLog* version		≈ 2m10s

Table 2: Durations for different selection operators and distance metrics for 25 generations.

In Table 2 the durations for different operators and distance metrics are demonstrated. The original version of *GeLog* is faster, due to faster decoding times and less fitness evaluations. It is also obvious that restricted tournament selection has a significantly longer run time, owing to the higher number of fitness evaluations. The different distance metrics seem to have little influence on the duration.

All experiments have been executed on an Intel Pentium II computer with 450 MHz. The run time experiments only involved 25 generations. The number of right hand sides (disjunctions) was fixed to three. The length of a chromosome is 27.

5.2 Chess Endgame King-Rook-King

The second experiment we have chosen to validate the performance improvement of *GeLog* is a chess endgame variant, the "White King and Rook vs. Black King" [15]. There are three pieces left on the board: a white king, a white rook, and a black king. The next player to move is white. The objective is to classify legal and illegal constellations, where a situation is illegal when either white has already won or black can capture the rook without being check (draw).

individual distance	chromosome selection	chromosome storing	selection operator	lowest error (%)	average error (%)
none	**ordered**	**ordered**	**ts**	**0.53**	**0.71**
sum/sum	**ordered**	**ordered**	**rts**	**0.53**	**0.71**
sum/sum	shuffled	shuffled	rts	0.53	0.72
sum/min	shuffled	shuffled	rts	0.53	0.72
sum/sum	shuffled	similarity	rts	0.53	0.72
sum/min	shuffled	similarity	rts	0.53	0.74
original *GeLog* version				4.90	8.02

Table 3: GeLog test results on 'King-Rook-King'

There is a total of 28056 entries, each of which consists of the coordinates of the pieces and an attribute for the optimal number of moves for white to win. The attributes are the number of moves (0..17) and "draw".

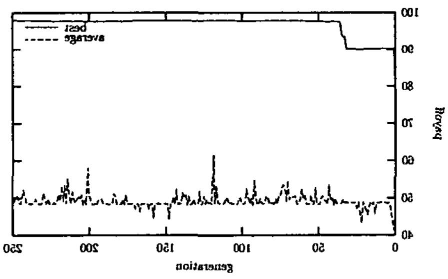
mutation probability	0.01
crossover probability	0.6
population size	150
number of generations	250
termination criterion	–

Table 4: Parameter Settings for the 'King-Rook-King' experiment

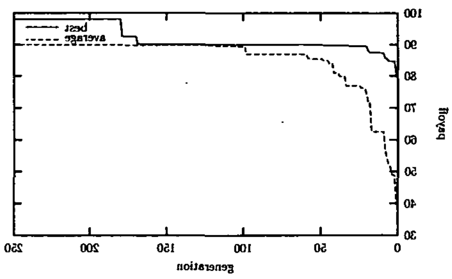
With the original *GeLog* program the minimum error was about 5%. Table 3 shows the experimental results for the modified version of *GeLog*, each entry representing five test runs, all of which are conducted using ten-cross-validation. The experiment settings are summarized in Table 4.

Again, neither the different distance metrics nor the diversity sustaining restricted tournament selection exhibit a remarkable difference with respect to the objective function values.

Also Figure 5 shows that the average payoff of the population is remarkably higher for restricted tournament selection.



(a)



(b)

Figure 5: Best versus average payoff for "King Rook King": (a) tournament selection, (b) restricted tournament selection

In Table 5 the run times for the different operators and metrics are listed. The experiments have been conducted with the same settings as in the previous experiment. However, the number of disjunctions is set to five. The length of chromosome in the new version is 54.

individual distance	selection operator	run time (25 generations)
none	ts	≈ 3m50s
sum/sum	rts	≈ 4m50s
sum/min	rts	≈ 5m10s
original *GeLog* version		≈ 2m6s

Table 5: Run times for different selection operators and distance metrics for 25 generations.

6 Conclusion and Future Work

In this article we presented some modifications to the *GeLog* framework—a genetic logic programming system—in order to improve its underlying genetic algorithm. The original *GeLog* program, linkage learning, and some related approaches were briefly introduced.

It was demonstrated how linkage learning was incorporated into the *GeLog* framework. All necessary changes and the resulting problems were presented. A distance metric, which was developed for the implemented selection operator, was also presented.

Finally, the test results showed that the *GeLog* framework has been significantly improved. Problems that used to be hard for the original program were solved.

Although these first test results are very promising and already demonstrate that the modifications do indeed enhance *GeLog*'s performance, further experiments have to be conducted to quantify the influence of the improvements achieved by the modified *GeLog* framework. In particular, it has to be investigated if under certain conditions linkage learning by the combination of the exchange crossover operator and standard tournament selection can withstand the force of selection.

References

[1] Gabriella Kókai. GeLog—A System Combining Genetic Algorithm with Inductive Logic Programming. In *Proc of the International Conference on Computational Intelligence, 7th Fuzzy Days LNCS*, pages 326–345, Springer Verlag, Dortmund, 2001.

[2] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, New York, 1994.

[3] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[4] Christian Jacob. *Illustrating Evolutionary Computation with Mathematica.* Morgan Kaufmann, San Francisco, CA, 2001.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[6] John J. Grefenstette. Deception considered harmful. In D. Whitley, editor, *Proceedings of the Foundations of Genetic Algorithms Workshop*, Morgan Kauffmann, Vail, CO, 1992.

[7] Stephanie Forrest and Melanie Mitchell. What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation. *Machine Learning*, 13:285–319, 1993.

[8] David E. Goldberg and Clayton L. Bridges. An analysis of a reordering operator on a GA-hard problem. *Biological Cybernetics*, 62(5):397–405, 1990.

[9] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1990.

[10] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid accurate optimization of difficult problems using fast messy genetic algorithms. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 56–64, Morgan Kaufmann, San Mateo, CA, 1993.

[11] Hillol Kargupta. SEARCH, polynomial complexity, and the fast messy genetic algorithm. Technical report, University of Illinois, Illinois Genetic Algorithms Laboratory, Urbana, Il, 1995.

[12] Hillol Kargupta. The gene expression messy genetic algorithm. In *International Conference on Evolutionary Computation*, pages 814–819, Piscataway, NJ, 1996.

[13] George Harik. *Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* PhD thesis, The University of Michigan, Ann Arbor, Michigan, 1997.

[14] David W. Aha. Incremental constructive induction: An instance-based approach. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 117–121, Morgan Kaufmann, Evanston, IL, 1991.

[15] Micheal Bain. Learning optimal KRK strategies. In S. Muggleton, editor, *ILP92: Proc. Intl. Workshop on Inductive Logic Programming*, Report ICOT TM-1182, Tokyo, 1992.