

# Introduction to Evolutionary Algorithms

Felix Streichert, University of Tuebingen

## **Abstract**

Evolutionary Algorithms (EA) consist of several heuristics, which are able to solve optimisation tasks by imitating some aspects of natural evolution. They may use different levels of abstraction, but they are always working on whole populations of possible solutions for a given task. EAs are an approved set of heuristics, which are flexible to use and postulate only neglectible requirements on the optimisation task.

As a practical application, technical trading rules found by the use of EA will be presented.

Keywords: Evolutionary Algorithms, financial applications, technical trading

## 1. Introduction

In real world applications you will most likely encounter problems that are hard to solve. Some examples for these kind of problems are the travelling salesman or the knapsack problem, but also financial applications like the constrained portfolio selection, time series prediction, trading rules for asset and bond markets [DM98], bankruptcy prediction [KF95], credit scoring [DL94], data mining and many more.

Algorithms to solve these kind of problems are either so specialized, that they only can be applied to a small range of problems, or they are more general but rather inefficient. Some general search heuristics like simple enumeration require vast amounts of computation time and will effectively fail if the problem dimension increases. On the other hand Hill-Climbing algorithms are easily deceived in multimodal problem spaces<sup>1</sup> and will most likely get stuck in some sub optima.

To approach such hard problems, a couple of concepts were introduced in the past decades, which were inspired by nature. Most of them originated from research areas like Artificial Intelligence (AI) or Artificial Life (AL). Some of the more popular and successful examples are Neural Nets (NN), Fuzzy Methods (FM) and Evolutionary Algorithms (EA or also known as Evolutionary Computation).

In this paper EA methods will be introduced and their possible applications in finance discussed. One of the major advantages of EA methods compared to other methods is, that they only need little problem specific knowledge and that they can be applied on a broad range of problems. EA methods only need the target (fitness) function for a given problem, which is to be optimised. Additional problem specific knowledge can easily be brought into the EA heuristic to improve performance. EA methods do have neglectable demands on the nature of the problem space, they even can be applied on complex problems with discontinuous, non-differentiable and possibly noisy target functions. Additionally they react robust to external EA process parameters and therefore are easy to use on very different problems without the need of special tuning or expert knowledge. Because of the diversity of EA methods it is easy to select an EA method that is especially well suited for a given problem, regarding the data types that are to be processed, the representation of the solution and the search space topology.

Although EAs belong to a relative new research area, a wide range of possible financial applications have been suggested and examined<sup>2</sup>. For several reasons EA methods appear to be particularly well suited for financial applications:

- They are payoff-driven. Payoffs may mean improvements in predictive power or return over a benchmark and such payoffs can be easily translated to a fitness function for an EA.
- EA methods are inherently quantitative, therefore they are well suited for parameter optimisation.
- EA methods allow a wide variety of extensions and constraints that cannot be provided in traditional methods. They are robust, revealing a remarkable balance between efficiency and efficacy.
- EA methods are easily combined with other optimisation techniques by the use of Memetic Algorithms (MA). With MA you can seamlessly scale between a pure EA approach and any other optimisation technique as long as both algorithms use the same representation as possible solution<sup>3</sup>.
- EA methods can also be extended to Multiobjective optimisation [VL00], which is of special interest in most financial applications.

In this paper the EA methods will be introduced and some of their possible financial applications will be discussed. First a survey on the research in the field of EA with financial applications will be given, but only a small number of papers on constrained portfolio selection, time series prediction and trading rules will be presented, due to the amount of papers that were published. This will be followed by a general overview on EA methods. The most important EA methods, Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Strategies (ES), Evolutionary Programming (EP) and Learning Classifier Systems (LCS) will be introduced. Two major extensions of EA will be described, that can improve the performance of EA methods considerably: Memetic Algorithms and the distributed EA. After this a small EA application example will be explained, in which a GA/P hybrid is used to generate profitable trading rules on the Euro/Dollar exchange rate.

---

<sup>1</sup> Multimodal search spaces contain not only one global optimum but many sub optima, which may deceive a simple search algorithm.

<sup>2</sup> [JA95] gives a small survey about financial applications of EA.

<sup>3</sup> With MA the EA will perform at least as good as the other optimisation technique it was combined with.

## 2. Related work

In this paper only few selected research papers on portfolio selection, time series prediction and the generation of trading rules will be presented. In the following sections these representative papers will be introduced and their results outlined. But the possible applications of EA in finance are so broad, that they cannot be completely outlined in this paper.

The EA methods named here will be discussed in detail in chapter 3.

### 2.1. EA and portfolio selection

Portfolio selection is a prominent example for EA methods in financial applications, since it is a parameter optimisation task and the fitness function can be easily calculated from the achieved return, the risk and additional constraints, which are to be met. For this kind of parameter optimisation task, GA or ES methods are the most suited EA methods and they are most commonly used.

Arnone, Loraschi and Tettamanzi did one of the earliest researches on portfolio selection with EA methods [AL93]. The authors described a basic GA approach to portfolio selection. Two years later, they extended the basic GA to a distributed GA [LT95]. They were able to achieve a considerable increase in performance with bigger GA populations, especially on search spaces with about 10.000 assets to select from.

A later paper on portfolio selection by Chang, Meade, Beasley and Sharaiha [CM98] compared the performance of GA, Tabu Search and Simulated Annealing. They found that the GA method yields the best results of all methods. But overall they favoured a combination of all three methods to solve the portfolio selection problem.

Shaqcot used a GA in combination with quadratic programming for index tracking [SJ92]. He used a boxed optimisation approach<sup>4</sup>: the GA tries to optimise the collection of assets, the so-called subset, from the complete asset universe. Quadratic programming was used, for optimal allocation for a given subset (GA Individual) for each generation. This is a simple example how EA methods can be combined with other optimisation techniques.

### 2.2. EA for time series prediction

There are two basic approaches to implement time series prediction with EA methods. One less common method tries to optimise the selection of input data and the strategy parameters for a standard heuristic for time series prediction. For example Minerva and Poli used a GA to optimise the strategy parameters for an ARMA-model with very good results [MP01].

The other approach to time series prediction uses GP for symbolic regression. In both cases the quadratic deviation of the prediction model to the real time series can be used as a simple target function that is to be minimised.

Koza, the father of GP, gave the first example on time series prediction by symbolic regression in his book "Genetic Programming", by reinventing a known econometric equation [KJ00]. Since he was successful, he proved the liability of symbolic regression with GP. In this example Koza used a canonical GP without further enhancements or tuning.

Yoshihara, Aoyama and Yasunaga suggested a combination of GP and MA for time series prediction [YA00]. This way they removed the restriction of GP to static predetermined constant-terminals as elements of the GP tree. They used a canonical GP for symbolic regression and each generation they used an additional local search algorithm to optimise the constant-terminals used in the GP tree. They basically implemented a Lamarckian model of MA, since the descendants inherited the optimised constant-terminals from their parents.

Santini and Tettamanzi were the winners in a contest initiated by the CEC 2000 to predict the Dow Jones [ST01]. First they planned to use ARCH and GRACH models for prediction and optimise the parameters of these models with EA methods. But later they found that a simple GP for multiple point prediction outperformed the optimised ARCH and GARCH models.

To further increase the predictive power of EA methods, a bagging procedure was introduced by Zemke to utilise the complete EA population of optimised predictors (=individuals) [ZS99]. The bagging procedure is based on a simple majority-voting scheme, but Zemke was able to outperform the single best predictors from a given population with the bagged complete population.

### 2.3. Evolutionary Algorithms to generate trading rules

A different strategy to predict time series would be to develop trading rules that make simple short-term predictions, whether a given time series will rise or fall in the near future. A predictive trading rule

---

<sup>4</sup> This is an example for a MA, which will be discussed in chapter 3.6.1.

will most likely be profitable, if it is used to trade the asset represented by the time series. The collected return minus a given trading cost can be used as target function, which is to be maximised. In this kind of application many GP approaches were used to develop profitable trading rules. Another but less common approach is to use a GA representation for trading rules or Learning Classifier Systems (LCS).

One of the earliest papers using GP was published by Allen and Karjalainen [AK99]<sup>5</sup>. In this paper the authors found that GP was not able to outperform the Buy&Hold strategy on the Out-Of-Sample set. This was also the case if trading costs were added. But the authors noted that the GP was able to identify relevant indices, use them for prediction and to ignore irrelevant indices.

Ready published a critique of the work by Allen and Karjalainen [RM97]. He noted that the transaction costs and the occurrence of the actual transaction relative to the signal produced by the trading rule were not correctly implemented. He noted further on that the rules found by Allen and Karjalainen had expired after five years. He suggested that trading rules should be periodically recreated to meet changes in the market. But in the end he sustained the work of Allen and Karjalainen: In spite of the proposed improvements he made, the GP was unable to beat the Buy&Hold strategy on the Out-Of-Sample set.

Jonsson, Madjid and Mordal used a canonical GP to find trading rules on international currency markets [JM97]. They repeated the optimisation process periodically to prevent over fitting of the trading rules. They also used a realistic trading model with transaction costs on intraday data. They claimed that their GP approach was able to find trading rules that produced a higher return than the simple Buy&Hold on the In- and Out-Of-Sample set.

Iba and Sasaki used a GP to find trading rules on the Nikkei225 [IS99]. Unfortunately they used a trained Neural Net as benchmark, so it becomes impossible to compare their results with the former mentioned research papers. They also seem to be less accurate with the implementation of trading costs and the time lag between the trading signal and the actual trade. But one of the most important results they presented is independent of these inaccuracies. They found that the predictive power of the rules found was dependent on the operators that were allowed to occur in the GP program tree. A GP program tree that was restricted to simple operators like {+, -, \*, /} was performing worse than a GP program tree that could use more sophisticated time series operators like the moving average.

Another interesting paper about trading rules and exchange rates was published by Neely, Weller and Dittmar [ND97]. The trading rules they found with a GP approach performed equally well on the In- and Out-Of-Sample set. They also showed that they were able to find trading rules with GP that couldn't be discovered with statistical methods.

Pictet, Dacorogna, Chopard, Oussaidene, Schirru und Tomassini used a simple GA to develop trading rules [OT95]. They stressed the multimodal nature of the search space when searching for trading rules. To increase the performance of the EA methods they used niching and clustering algorithms to track multiple optima in the search space. In a different paper (also with an application on trading rules) they extended this approach to a distributed EA [OT97].

O'Neill, Brabazon, Ryan and Collins used a variation of the canonical GP, the Grammatical Evolution (GE see [BW93] or [OR01]), to find profitable trading rules on the UK FTSE 100 index [OB01]. They found that the rules developed with GE considerably outperformed the simple Buy&Hold strategy. They criticised the work of Allen and Karjalainen, claiming that Allen and Karjalainen didn't note that the trading rules generated by the GP were only in the market for about 60% of the time. Therefore, the trading rules should have been assigned a more positive risk profile than the simple Buy&Hold strategy.

The other method to develop profitable trading rules with EA used in literature is GA or LCS. One example for this kind of approach is a paper by Schulenburg and Ross [SR01]. The trading model they used allowed the trading agent to choose the allocation in a portfolio of three different forms of investment, a long position and a short position of a given asset and an independent low risk asset. Additionally, they claimed to use realistic trading costs. They found trading rules that outperformed the Buy&Hold strategy and they further noted that the performance of the trading rules found by the LCS depends on the input data that is available for the trading agent and the basic structure of the condition rules of the LCS, which have to be set by the operator<sup>6</sup>.

### 3. Fundamentals of Evolutionary Algorithms

EA are stochastic search and optimisation heuristics derived from the classic evolution theory, which are implemented on computers in the majority of cases. The basic idea is that if only those individuals of a population reproduce, which meet a certain selection criteria, and the other individuals of the population die, the population will converge to those individuals that best meet the selection criteria. If

<sup>5</sup> "Using genetic algorithms to find technical trading rules"; this title is a bit irritating since they actually use a GP approach.

<sup>6</sup> See chapter 3.5 for more details on the basic structure of LCS rules.

imperfect reproduction is added the population can begin to explore the search space and will move to individuals that have an increased selection probability and that inherit this property to their descendants. These population dynamics follow the basic rule of the Darwinistic evolution theory, which can be described in short as the “survival of the fittest” [DC59].

To solve optimisation problems with an evolutionary heuristic the individuals of a population have to represent a possible solution of a given problem and the selection probability is set proportional to the quality of the represented solution.

The quality of the represented solution is also called the fitness  $\Phi$  of the individual. We write capital letters  $A, B, C$  for sets of individuals or populations. The current generation of the evolutionary process will be indicated by the letter  $s$ . A single Individual with the index  $i$  from the population  $A(s)$  will have the shortcut  $a_i(s)$ . The quality of the solution represented by an individual is also called the fitness  $\Phi_i$  of the individual  $a_i(s)$ . The selection probability of an individual  $a_i(s)$  will be  $p_i$ . When a description of a possible solution for a given consists of  $n$  elements, the  $i$ -th element forming a possible solution will be called attribute  $x_i$ , regardless of the necessary data type. Therefore, an individual consists of several attributes  $x$  and will represent a possible solution throughout these attributes, that are to be optimised. An EA heuristic follows this basic scheme:

```

initialise random population  $A(s=0)$ 
repeat
  evaluate fitness of all  $a_i$  from  $A(s)$ 
  select the fittest  $a_i$  as parents  $B(s)$  from  $A(s)$ 
  reproduce descendants  $C(s)$  from  $B(s)$ 
   $A(s+1) = C(s)$ 
until break criteria is met
  
```

In the first step, a population of random possible solutions will be set up. Then the EA generational loop is entered. Each generation the fitness  $\Phi_i$  for each individual  $a_i(s)$  within the current population  $A(s)$  is evaluated. Then the best individuals  $a_i$  are selected from the population  $A(s)$  as parents  $B(s)$  for the next generation. The selection probability  $p_i$  is set proportional to the fitness  $\Phi_i$  of the individual. From the selected parents  $B(s)$  descendants are reproduced to form the population  $C(s)$ . In all EA heuristics the descendants are either imperfect clones of the parents with small variations (this equals naturally occurring mutations) or the descendants are a melange of multiple parents and inherit some attributes from the associated parent (this equals sexual reproduction<sup>7</sup> in nature) or both. The descendants  $C(s)$  form the next generation  $A(s+1) = C(s)$ .

One major property of EA heuristics is that the search space is not explored by starting with only one possible solution but with a whole population of possible solutions and that the individuals of the population can exchange solution attributes between them. This way EA heuristic can outperform a single or a Multi-Start Hill Climbing algorithm, since they are more resistant to premature convergence towards a local optima in multimodal search spaces. But still EA methods are not guaranteed to find the global optimum.

The first experiments to mimic evolutionary processes in computer science were performed in the late fifties and in the early sixties. In the mid sixties, John Holland from the university of Michigan successfully introduced the concept of sexual reproduction to EA

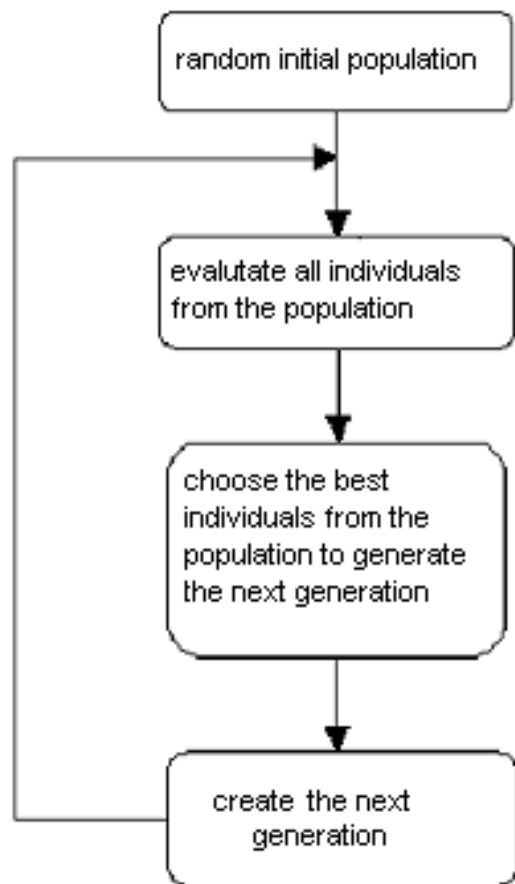


Fig. 1 General EA process.

<sup>7</sup> The sexual reproduction in EA is called crossover, from the natural crossover mechanism that can occur during the DNA reproduction.

[HJ75]. Since then Holland is believed to be the father of the scientific field of EA, which can be separated into several more specialized sub categories.

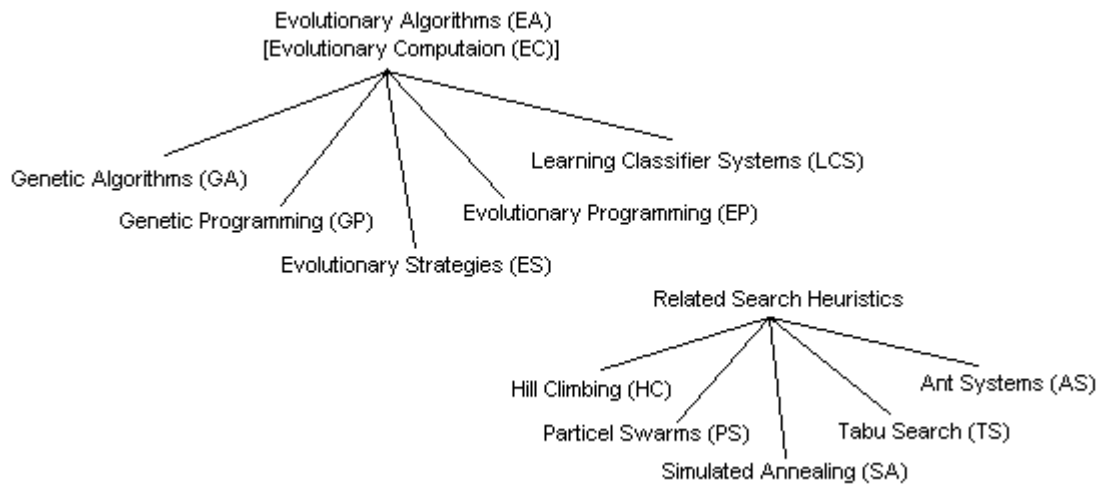


Fig. 2 Classification of EA methods.

Genetic Algorithms (GA) ([HJ92] and [GD89]) exhibit the clearest mapping from the natural process of evolution onto a computer system, because they stress the coding of attributes into a set of genes. One very common coding of attributes is a binary coding into a Bit-String representing the genes.

In contrast to Genetic Algorithms the Evolutionary Strategies (ES) [SH75] stress the actual expression of an attribute and omit any redundant coding. This way they are able to move very efficiently through real valued search spaces by the use of specialised mutation operators.

Genetic Programming (GP) [KJ92a] is related to Genetic Algorithms regarding the general processing scheme. But instead of representing attributes in a general binary coding, Genetic Programming is specialised on representing programs or instruction sets as attributes. And the structures of these programs are altered during the optimisation process.

Evolutionary Programming (EP) [FO66] is similar to the Evolutionary Strategies, but it was developed independently and it has virtually no restrictions regarding the data types of attributes. In EP the evolutionary process is focussed on the level of whole species not on single individuals.

Learning Classifier Systems (LCS) ([HR78], [GS95] and [HJ75]) belong to rule-based machine learning methods, but they are basically an extension of Genetic Algorithms. In the case of LCS an individual represents one or more rules, which have the basic scheme of "if {condition} then {action}". To allow generalisation, the coding of the condition is extended with a "don't care" symbol.

Similar search heuristics, which can be related to EA methods, are: Simulated Annealing, Hill Climbing, Particle Swarms, Ant Systems and Tabu Search.

EA methods are very easy to use even on complex problems with discontinuous, non-differentiable and possibly noisy target functions. They implement a parallel-guided stochastic search strategy and are able to track multimodal or even multiobjective target functions. They can easily be extended to implement constraints or restrictions on the solution attributes that are to be optimised. Further on they can be combined with any existing search strategies for a given search space. EA seamlessly scale between the exploration of the search space through mutation and crossover and the exploitation of known optima through selection of fit individuals.

### 3.1. Genetic Algorithms

As mentioned before, Genetic Algorithms (GA) originated from the work of John Holland ([HJ92] and [GD89]) and are the most obvious mapping of natural evolutionary processes into a computer system. Therefore, some biological terms may be used to illustrate the functionality of Genetic Algorithms.

#### 3.1.1. The GA individual

GA individuals store the solution attributes not directly in clear type but in a coded representation. Most common is the binary coding of an attribute in a chain of Bits, a Bit-String<sup>8</sup>. The Bit-String

<sup>8</sup> This Bit-String would equal the DNA, and a single bit would equal a single DNA base (Adenine, Cytosine, Guanine, Thymine).

consists of  $L$  bits, which are clustered into words  $w_i$ . In a simple version all words are of equal length  $l$ .

The decoded words  $w$  are the solution attributes  $x$ , which are to be optimised. Each attribute  $x_i$  is assigned to the word  $w_i$ .

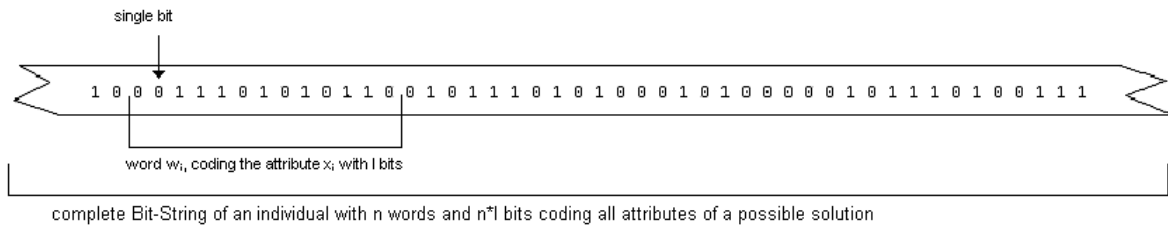


Fig. 3 Bit-String of a GA individual.

In the simplest case a word codes a real number. In this case the real number attributes are limited regarding the possible range of values and the precision, since the length  $l$  of a word is always limited. If the range of an attribute is given and also the length  $l$  of a word, the possible precision of a real number attribute is fix. A simple binary decoding method for a real number attribute  $x_j$  with the upper bound of  $o_j$  and the lower bound of  $u_j$  with the length  $l$  from the word  $w_i$  could be:

$$x_j = \Gamma(w_j) = \Gamma(w_{i,0}, w_{i,1}, \dots, w_{i,l-1}) = u_j + \frac{o_j - u_j}{2^l - 1} \cdot \left( \sum_{z=1}^l w_{i,(l-z+1)} \cdot 2^{z-1} \right)$$

In this notation  $w_{i,n}$  gives the  $n$ -th bit of the word  $w_i$ .

This coding is called standard binary coding. Similar coding styles can be found for nearly every data type. So any data type can be used as an attribute of a GA individual and so be optimised using the GA search heuristic.

After the attributes  $x_i$  of an individual have been determined by decoding the word  $w_i$ , the fitness  $\Phi_i$  can be calculated by using the target function  $F(x_i)$  as fitness function:

$$\Phi(a_i) = F(\Gamma(w_i))$$

After the fitness for every individual  $a_i$  of the population  $A(s)$  has been calculated, the best individuals of the population  $A(s)$  are selected to be the parents of the next generation  $A(s+1)$ .

### 3.1.2. The GA selection

A commonly used selection method for GA is the roulette wheel selection:

In this selection method, an individual  $a_i$  of the population  $A(s)$  is assigned to a small part on a wheel of chance. The size  $p_i$  of this part is proportional to the calculated fitness  $\Phi_i$ . The wheel is then tossed as many times as parents are needed to create the next generation and each winning individual is copied into the parent population  $B(s)$ . With this method a single individual can occur multiple times in the parent population  $B(s)$ .

This stochastic selection mechanism incorporates the Darwinistic principle of the "survival of the fittest" by omitting less fit individuals from the parent population.

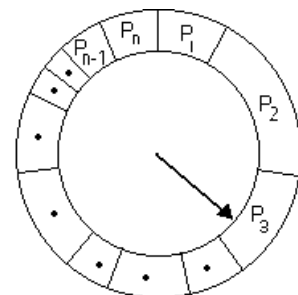


Fig. 4 Roulette-Wheel Selection

Then the descendants are created by performing sexual recombination (crossover) and mutation.

<sup>9</sup> This is called Hollands fixed-length coding.

### 3.1.3. The GA crossover

Crossover exchanges genes, coding the solution attributes, between the parents to generate the descendants. The GA crossover resembles the natural crossover, but instead of DNA, the Bit-String of the parents is cut at a position chosen by chance and the single parts are mixed and chained again to build the Bit-Strings of the descendants.

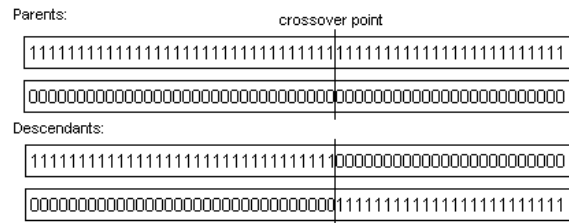


Fig. 5 Crossover with two parents

The cutting takes places without respect of the boundaries of words stored on the Bit-String<sup>10</sup>.

If fixed-length coding is used, both parent's Bit-Strings are cut at the same position to preserve the overall length of the Bit-String.

### 3.1.4. The GA mutation

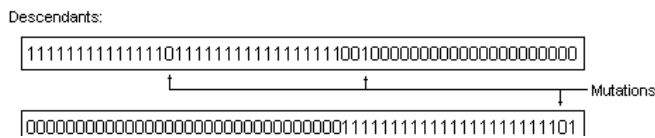


Fig. 6 GA mutation, occurring in the two descendants

After the descendants are created by sexual recombination, the descendants are mutated. This resembles the naturally occurring accidents that can happen when DNA is copied. Mutation on the Bit-String occurs by shifting single bits in the Bit-String selected by chance.

After undergoing these creation and altering processes, the descendants form the next generation  $A(s+1)$ . Then one generation is completed and than the generational process is repeated until a satisfying solution for the given target function is found.

The main advantages of GAs are, that they are very easy to implement and that they can be applied to nearly every kind of optimisation problem. Because of the general binary coding style almost any data type can be stored in an individual and then be optimised by the GA heuristic.

But there are also some drawbacks using the binary coding. For example, if real numbers are used as attributes, they become discretised and because of the non-linear behaviour of the standard binary coding the search space gets disrupted and rugged.

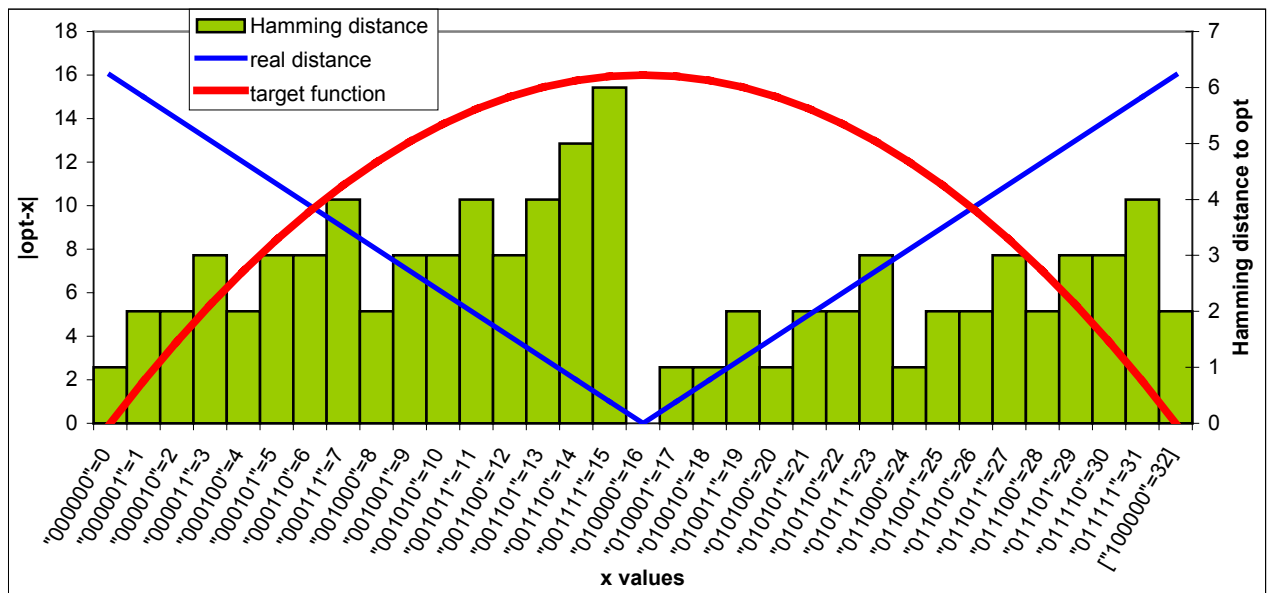


Fig. 7 Distortion of the search space caused by the GA coding

<sup>10</sup> The boundaries of a word within the Bit-String are only important for the coding/decoding methods, not for genotype variation methods.

If we examine the target function

$$f(x) = -\frac{(x-16)^2}{16} + 16 \quad ; \text{ on the range } [0, 31].$$

which is to be maximised and a binary coding with  $l = 5$  bits is used for the single real number solution attribute  $x$ , then a rather harsh discretisation ( $\Delta=1$ ) occurs and the neighbour relationships between possible solutions are disrupted. If the Hamming distance<sup>11</sup> is used and the distance of the optimal attribute value of  $opt = 16$  to all the other possible values is measured, we find that the real distance and the Hamming distance does not correspond. Especially the step from  $x = 15$  to  $opt = 16$ , although small in a real valued search space, is maximal in the binary coded search space. This way a unimodular search space can become multimodular just because of the used coding style. On the other hand the search space gets folded in a way that formerly different attribute values become closer in the binary coded search space<sup>12</sup>.

One way to overcome this problem is to use a different coding style, for example the Gray Coding, see [GF53].

Another possibility is to omit the coding and move from genotype orientated EA operators to phenotype oriented EA operators. This is done in the case of ES.

### 3.2. Evolutionary Strategies

Evolutionary Strategies were developed in the sixties by Rechenberg and Schwefel in Berlin [SH75]. They were conceived as search heuristic for optimisation problems in the field of engineering and they were specialised to optimise real numbers as solution attributes. In contrast to GAs and GPs, ESs were designed from the very beginning as a practice-oriented optimisation method. The main difference to GAs is that the ES method emphasises the phenotype of an individual and that it omits any additional coding. Because of this, the crossover and mutation methods have to change the real value of an attributes and not an abstract coded representation<sup>13</sup>.

This phenotype-oriented approach allows further optimisation of the altering operators. Especially for real number attributes mutation operators have been suggested that can adopt automatically the size and the direction of the mutation relative to the local topology of the search space. The parameters specifying the mutation properties are called strategy parameters and the solution attributes are called decision parameters in ES papers. The ES optimises both of them.

#### 3.2.1. The ES individual

Since the ES is specialised on real numbers as solution attributes, an ES individual consists in most cases of a vector of  $n$  real numbers, the decision parameters that are to be optimised:

$$a_{i,d} = (x_1, x_2, \dots, x_n) \quad , x_i : \text{real numbers}$$

In a simple ES implementation the strategy parameters are stored in an additional vector of  $n$  real numbers:

$$a_{i,s} = (\sigma_1, \sigma_2, \dots, \sigma_n) \quad , \sigma_i : \text{positive real numbers}$$

In this simple case we use one strategy parameter  $\sigma_i$  for each decision parameter  $x_i$ <sup>14</sup>.  $\sigma_i$  assigns a standard deviation to the phenotype mutation for each decision parameter, which is equal to the mean size of a mutation step.

If coding is omitted, the target function can be evaluated without further calculations:

$$\Phi(a_i) = F(a_{i,d})$$

<sup>11</sup> Hamming distance, the number of bits that differ e.g.:  $[001100:011000] = 1$ . In this example the maximum Hamming distance is  $l=6$ .

<sup>12</sup> Note for example that the Hamming distance between  $x = 0$  and  $opt = 16$  is only 1.

<sup>13</sup> A closer investigation on the use of phenotype oriented altering methods can be found in [WA91].

<sup>14</sup> In a more complex version the strategy parameters can be extended to a complete matrix so that the size and the direction of the mutation can be adopted. Very good results were achieved using a derandomised ES, for example with the Covariance Matrix Adaptation (CMA) [HO96].

### 3.2.2. The ES selection

In most ES a deterministic selection procedure is used to select the possible parents for the next generation. Only the  $\lambda$  best individuals are selected from a population  $A(s)$  that is of size  $\mu$  and only these  $\lambda$  best individuals are used for the parent population  $B(S)$ . This approach is called the  $(\mu, \lambda)$  strategy.

If the  $\lambda$  best individuals are treated as elite that enters the next generation unaltered, the strategy is called  $(\mu+\lambda)$  strategy.

### 3.2.3. The ES crossover

ES uses two different crossover methods for strategy and decision parameters to mix two selected parents ( $e1$  and  $e2$ ) from  $B(s)$ :

Discrete crossover is used for the decision parameters: If we use a discrete crossover a descendant  $c$  either inherits the attribute  $x_i$  from one or the other parent. The actual parent that supplies an attribute can be chosen arbitrarily.

E1:	2,5	8,2	7,0	1,3
E2:	4,5	9,4	1,8	2,1
C:	4,5	8,2	7,0	2,1

Intermedium crossover is used for strategy parameters: The intermedium crossover assigns the descendant  $c$  a  $\sigma_i$  value that is the mean of the corresponding  $\sigma_i$  of both parents. This crossover method has a concentration effect, since the strategy parameters of the child are a simple linear interpolation of the strategy parameters of both parents.

E1:	2,5	8,2	7,0	1,3
E2:	4,5	9,4	1,8	2,1
C:	3,5	8,8	4,4	1,7

### 3.2.4. The ES mutation

For the mutation two different methods can be used for strategy and decision parameters: First the strategy parameters are mutated according to:

$$\sigma'_j = \sigma_j \cdot \exp(\tau_1 \cdot N(0,1) + \tau_2 \cdot N_j(0,1))$$

$N(0,1)$  is an evenly distributed random number with the range  $[0,1]$  that is generated for each mutation per individual.  $N_j(0,1)$  is also an evenly distributed random number with the range  $[0,1]$ , but this number is generated for each  $\sigma_i$ .  $\tau_1$  and  $\tau_2$  are exogene strategy parameters<sup>15</sup>.

The new mutation step size  $\sigma'_i$  is used to mutate the decision parameters  $x_i$ :

$$x'_j = x_j \cdot + \sigma'_j \cdot N'_j(0,1)$$

$N'_j(0,1)$  is a Gaussian distributed random number generated for each  $x_i$ .

The descendants form the next generation  $A(s+1)$  after undergoing these altering processes<sup>16</sup>. Again the generational process can be repeated until a satisfying solution is found.

## 3.3. Genetic Programming

In the mid eighties, new concepts were developed to use evolutionary processes to automatically generate programs. The most successful concept was the GP [KJ92a], which has grown since then into independent field of research for program induction.

### 3.3.1. The GP individual

The GP individual is designed to store computer programs in such a way that they can be optimised using an evolutionary approach. As stated before, EA operators like mutation and crossover depend on the representation of the attributes of an individual and also the calculation of the fitness of an individual.

The fitness of a individual, in this case the computer program represented in an attribute, can be evaluated by executing the program under varying conditions and the behaviour or the output of the

<sup>15</sup> Common values for  $\tau_1$  and  $\tau_2$  are:

$$\tau_1 \approx \frac{1}{\sqrt{2 \cdot n}} \quad \tau_2 \approx \frac{1}{\sqrt{2 \cdot \sqrt{n}}} \quad ; n \text{ equals the number of decision parameters.}$$

<sup>16</sup> Eventually the elite  $B(s)$  is added to  $A(s+1)$ .

program can be used to estimate the fitness. Such programs could also be simplified to represent mathematical functions<sup>17</sup>.

In the first experiments, the computer programming language LISP was used to code computer programs into an individual. LISP is an interpreted list oriented language that was commonly used in research areas like AI. A small algebraic term like

$$(x - 1) - x^3$$

would be represented in LISP as:

```
( - ( - x 1 ) ( * x ( * x x ) ) )
```

The first experiments with GP were performed with such a coding and specialized mutation and crossover methods, to ensure that only valid new programs were generated during the evolutionary process.

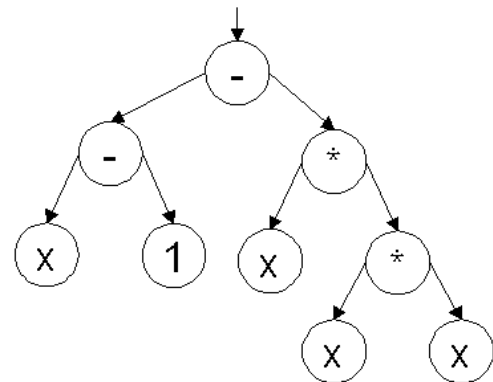


Fig. 8 A GP program tree

Another common representation for programs in GP individuals was derived from LISP: The translation of the LISP code into program trees. These program trees were easier to manipulate and to maintain and they can be implemented in any computer programming language<sup>18</sup>.

A program tree is basically build from nodes with can again point to other nodes. A node without a pointer to a following node is called a leaf node or terminal. It represents a program instruction that doesn't need any further input data. For example a numerical constant terminal (the value 1), an external input value (the current temperature x) or it could be simple command without parameters (for example stop()).

A node with pointers to further nodes is called an operator. For example the operator add(x,y,..) takes at least two inputs values and returns the sum of both. The succeeding nodes can be terminals or more operators.

The number of successor nodes is called the arity of a node. The add(x,y,..) operator needs at least two nodes, so its arity is at least two. The Boolean operator NOT needs only one node and has the arity one. A terminal node has no successor node and has the arity zero.

The collection of all operators and terminals that are allowed to occur in a program tree is called area.

Some examples for operators:

- Arithmetic operators: +, -, \*, /, sin, cos, exp, ....
- Boolean operators: AND, OR, XOR, NOT, NAND
- Problem specific operators: MovingAverage, Max, Min, Varianz, Lag,...
- 

Some restrictions have to be met by the area to solve a given problem with a GP approach:

- Sufficiency: It must be possible to solve a problem using only the elements of the given area<sup>19</sup>.
- Closure: Every operator from the area needs to accept every value that can be generated by any possible following node of the area<sup>20</sup>.

### 3.3.2. The GP selection

The commonly used selection method in GP is the tournament selection. In the tournament selection the parents are selected by doing multiple small tournaments between members of A(s). For each tournament n arbitrarily selected individuals from the population A(s) are put into a tournament group. Then the best individual of the tournament group is selected as possible parent and copied into the parent population B(s). Like in the roulette wheel selection, a single individual can occur multiple times in the parent population B(s).

<sup>17</sup> This is called symbolic regression.

<sup>18</sup> Some other possible structures to evolve programs are:

- Finite State machines in EP [FA95]
- Grammatical Evolution [BW93] or [OR01]
- Linear-Tree GP [KB01]

<sup>19</sup> It would be impossible to get good results with symbolic regression to estimate a periodic data set, when trigonometric functions like (sin, cos,...) are missing in the collection of operators.

<sup>20</sup> For example the DIV(X,Y) = X/Y must accept Y = 0 as possible input and must catch the resulting division by zero error.

There are some GP implementations that prevent the problems that come with the closure requirement by restricting the choice of possible nodes following. For example a Boolean type operator should only take Boolean type operators or Boolean terminals as inputs. This approach is called strongly typed GP [MB94].

### 3.3.3. The GP crossover

The GP crossover is performed by selecting two sub trees by chance from the parents and exchange them to create the descendants.

### 3.3.4. The GP mutation

The GP mutation is performed by selecting a sub tree of the descendant by chance and to exchange the sub tree with an arbitrary generated new sub tree.

Due to the very rugged search space of possible program configurations usually very big populations (> 1.000 individuals) but only a small number of generations (<50) are used for GP.

The GP program trees tend to rapidly increase in size during the evolutionary process. This effect is called bloat and it causes the increase of computation effort, since the programs that are to be evaluated become more complex.

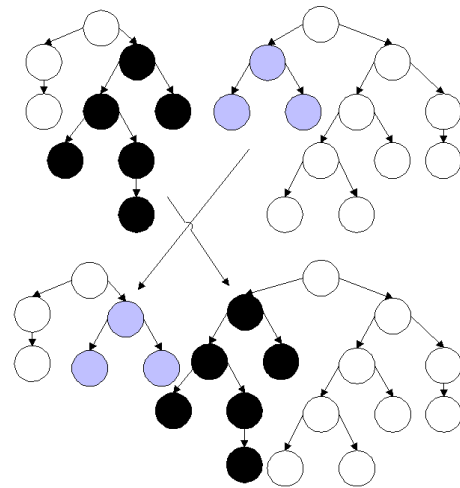


Fig. 9 GP crossover

There are several extensions of the canonical GP that deal with the problem of bloat<sup>21</sup> and try to reduce the destructive effect of altering operators<sup>22</sup>.

## 3.4. Evolutionary Programming

Fogel, Ownes and Walsh developed EP in the mid sixties [FO66]. They stressed the natural example but the evolutionary process is not focussed on the level of single individuals but on the level of whole species. In EP the evolution adopts the general behaviour and not the genotype sub-structures. Because of this EP uses only phenotype oriented altering operators. And since there is no exchange across a species boundary the crossover operator is omitted.

### 3.4.1. The EP individual

The representation of the EP individuals (each representing whole species) is only determined by the given optimisation problem and underlies virtually no restrictions regarding the data types that can be processed.

### 3.4.2. The EP selection

EP uses a mixture of tournament selection and the best of selection as in ES. For each individual  $a_i$  a tournament group of  $n$  individuals selected by chance from  $A(s)$ , then the number of individuals of that tournament group that are inferior to  $a_i$ , regarding the fitness is determined and assigned as score to  $a_i$ . The  $\lambda$  individuals with the highest score are selected as parents  $B(s)$ .

### 3.4.3. The EP mutation

The EP mutation operator is problem or data type specific, because each problem can use different attribute data types for the individuals. If for example an EP individual consists only of a vector of real values as in ES, the same mutation operators as in ES can be applied.

For real number attributes the behaviour of the EP resembles that of ES, if an ES mutation is used.

## 3.5. Learning Classifier Systems

Holland and Reitmann developed learning Classifier Systems, [HR78], [GS95] and [HJ75]. They belong to the inductive learning methods of machine learning and are sometimes called Genetics Based Machine Learning (GBML) methods. LCS are able to find simple rules by interacting with an environment. The method of LCS is based on the GA method, only the Bit-String representation of the individuals was changed that they can represent generalising rules.

<sup>21</sup> One of the easier solutions to restrict the size of the program tree to a certain depth of the tree and prune anything that exceeds that depth. Another approach is to use altering operators that preserve the size of the program trees like the depth-dependant crossover [I198].

<sup>22</sup> Very common is the use to sub routines that are split from the main body of a program, [KJ92b] and [AP93].

### 3.5.1. The LCS individual

LCS are able to learn simple “if {condition} then {action}” style rules<sup>23</sup> by learning from feedback information given by an environment or supervisor. The condition and the action parts of the rules are coded like the GA in Bit-Strings. Only the alphabet of the condition part was extended from the basic binary alphabet {0,1} to {0,1,#}. The ‘#’ symbol represents a ‘don’t care’ to allow generalisation of conditions. These LCS rules can be optimised by using a GA.

Since in most cases a single rule will not be sufficient to cover a given problem, the basic GA was extended to generate a set of heterogeneous rules. One possible way is the Michigan approach: One individual contains a single rule and the whole population represents a set of rules. In this case the basic GA has to be extended with niching algorithms to maintain a heterogeneous population of rules (individuals) [HG94]. The other possibility is the Pitts approach, which puts multiple rules into a single individual and each individual has to contain the complete set of rules that cover the given problem [SS80].

In the basic type of the LCS<sup>24</sup> according to the Michigan approach the individuals are evaluated in two phases: First the performance phases (red), where the individuals interact with the environment, and the reinforcement phase (blue), where the individuals are rewarded for good performance. In the performance phase the current state of the environment is compared to the condition set of each individual. Every individual whose condition is met will be copied into the match set. From the match set an individual  $a_i$  will be selected by chance proportional to the already achieved fitness  $\Phi_i$ . The action part of the selected individual  $a_i$  will then be executed.

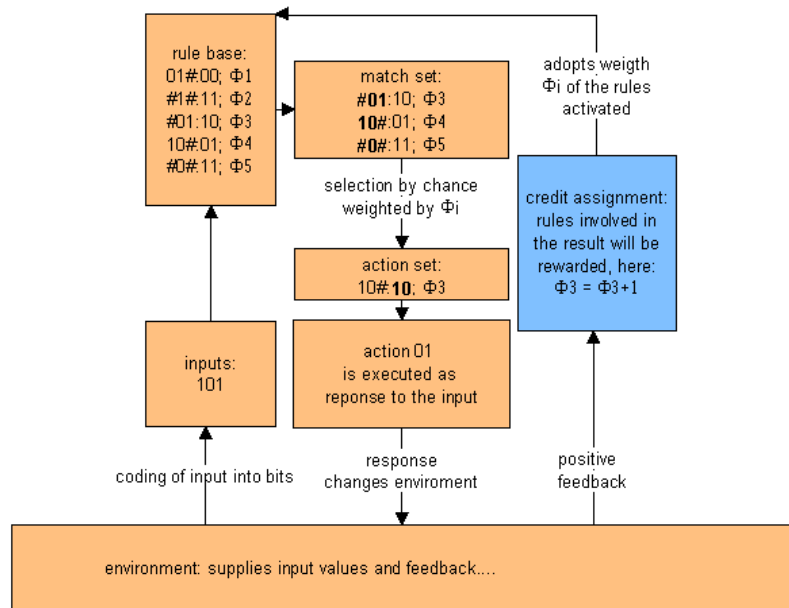


Fig. 10 Performance and reinforcement phase of LCS evaluation

If the action was performed and caused some effect, the feedback (success) can be used in the reinforcement phase to change the fitness of all individuals in the match set that suggested the actually performed action, even if they were not selected for the action set.

The performance and reinforcement phase can be repeated several times to ensure correctly assigned fitness values of the rules evaluated. Then the accumulated fitness can be used to perform a single GA generation step.

### 3.6. Extensions to EA methods

Only two extensions to EA methods which can increase the performance of EA methods considerably will be discussed here. First a method that allows the seamless combination of EA method with any other, possibly even problem specific, optimisation technique: Memetic Algorithms (MA)<sup>25</sup>. Second an easy and cheap method to meet the needs of EA methods for computational resources and increase the quality of the solutions found by EA methods: Distributed EAs.

#### 3.6.1. Memetic algorithms

In the description of EA methods given here, the attributes of an individual are only set once if the individual is created and these attributes are not allowed to change during its lifetime. But in nature it can be observed that individuals can change their properties after birth to adapt to the environment. This phenomenon is called plasticity: the ability to adopt during lifetime.

<sup>23</sup> This is a stimulus-response form.

<sup>24</sup> This type is named „Zeroth-Level LCS“ [WS94]. Other approaches allow the concatenation of rules.

<sup>25</sup> Some times called Hybrid-EA.

In EA such plasticity can easily be given to the individuals and the way how the individuals are allowed to adopt can be chosen arbitrarily. The heuristic of adopting an individual will be called local search and the combination of the local search with the EA method will be called Memetic algorithm (MA)<sup>26</sup>.

The weight between the local search and the EA method can be shifted arbitrarily. This way the EA extended with MA will perform at least as good as a Multi-Start search of the local search heuristic, which was combined with the EA.

If the altered attributes, optimised by the local search heuristic, of an individual are inherited to the descendants, the MA is called Lamarcian style<sup>27</sup>. If the unaltered attributes are passed onto the next generation, the MA has a similar effect as with the Lamarcian evolution, but it is then called the Baldwin effect<sup>28</sup>.

### 3.6.2. Distributed Evolutionary Algorithms

In most cases, EA methods will need a great amount of computational resources, either because the evaluation of the fitness function is very slow or because the population sizes used become too big<sup>29</sup>.

The obvious solution for this problem is to parallelise the EA process and to distribute it over several computers. Many architectures have been suggested how to distribute EAs<sup>30</sup> and only a simple island distribution scheme will be introduced here and the advantages of this scheme over a single process EA will be discussed.

With the island model  $n$  identically configured EA processes are simulated on  $n$  different computers. Each EA process incorporates a sub population of the complete distributed EA. After a certain number of generations each computer is stopped and the currently best individuals of each sub population are exchanged between the sub populations. Then the EA processes are resumed until the next exchange takes place. The final solution of the distributed EA is the best individual of all sub populations after all EA processes have terminated.

Due to the infrequent exchange between the sub populations, the population's dynamics of this distributed EA can be compared to the natural example of remote ocean islands, thus the name "island model". In nature, species tend to specialise, to explore niches on isolated island and develop into species that cannot be found anywhere else. Translated to the EA process, each sub population explores most likely different sub optima in a multimodular search space. By exchanging the currently best individual from time to time, all sub populations will tend to converge on the best solution of all sub optima. Additionally the heterogeneous pool of possible solutions, distributed over the sub populations, combined with the crossover operator of EA methods, will allow the EA process to explore areas of the search space that a simple Multi-Start local search approach could only reach by chance.

Altogether, distributed EAs not only reduce computation time, but also increase the quality of the solution found in multimodular search spaces.

---

<sup>26</sup> The name „Mem“ was taken from Richard Dawkins book „The selfish gene“. In some research papers the name "hybrid EA" is often used for a MA.

<sup>27</sup> Contrary to the classic theory of evolution by Darwin, Lamarck believed that each individual could inherit learned attributes to their children.

<sup>28</sup> A detailed introduction to MA can be found in [TP96] and [WG94]

<sup>29</sup> This may be necessary, to increase the quality of the solution in multimodular search spaces.

<sup>30</sup> An overview on distributed EA and different distribution schemes can be found in [CP97], [CP99], [PC87], [TR89] and [CT98]

#### 4. Technical Trading Rules discovered by EA

An application of GP to technical trading rules on the Euro/Dollar exchange rate will be presented and discussed here as a representative example for EA methods.

Finding profitable trading rules is a very hard optimisation problem, since the search space is very rugged and of multimodular nature. And not all data that may influence the value of the observed asset is available to a computer program. Further on it is difficult to prove that rules found on the training set will produce similar results in the near future. This problem is closely related to the problem of over fitting and is inherent to all machine learning techniques and other heuristics to predict time series. On the other hand even a minor increase in performance over a given benchmark may yield major profits.

In this example a G/P approach was used to find trading rules<sup>31</sup>. The task of the to be optimised G/P program (trading agent) was specified as an simple day trading model:

The trading agent can only decide whether he goes long or short of Dollar. Therefore, the G/P program that controls the behaviour of the agent only needs to indicate the position taken. In this case the program (a mathematical function) indicates a short position if the value is smaller than zero and a long position if the value is equal or greater than zero.

Basic trading agent structure:

```
If (x < 0) then go short
else go long
```

The agent performs a trade, if the sign of the internal math function is changing.

The allowed operators to calculate x were {+; -; \*; /; Max; Min; MovingAverage; TimeLag; If[Condition >0; then action1 : else action2]}. These operators are simplified, but should be able to allow basic decision-making and included some specialised operators for analysing time series like the TimeLag and the MovingAverage operator.

Because of the restriction to be either long or short of the currency traded, the trading model is to be considered a rather simple one. The agent decides once per day whether he wants to change the positioning. He decides after the closing price of the currency is published and is then allowed to buy or sell his position at the closing price with a trading cost of 0,03% of the trading volume<sup>32</sup>. The difference of buy and sell price is added to the return of the agent minus the trading cost. The trading volume of the agent is always one Euro. Achieved profit or loss of the agent is not added to the trading volume, but is stored free of interest.

For this example the Euro/Dollar exchange rate from July 1998 to July 2001 was used as test data. The data was separated with a ratio of 7:3 into In-Sample and Out-Of-Sample set. The Out-Of-Sample set was not used for training, but to test the rules, whether they do generalize.

Two sets of experiments were performed. For each set ten EA runs were simulated. Each EA run was performed over fifty generations, with a population size of one thousand individuals.

The first experiment was done with only open, high, low and closing price data as input.

The second experiment was done with about twenty additional indices all based on or derived from the Random Walk Index<sup>33</sup>.

The benchmark for both experiments was the biased Buy&Hold strategy, separated for the In-Sample and Out-Of-Sample set. Since the Buy&Hold performs very badly on the Out-Of-Sample set, the theoretical maximum return without transaction cost was calculated as an additional benchmark separately for both the In-Sample and the Out-Of-Sample set.

The GP program tree size was limited by the use of depth-dependant crossover to decrease the computation time. There are several implications that come with the use of the depth-dependent crossover and the restriction of program tree size. First of all this removes the bloat from G/P, so computation time is saved. But also with smaller program tree size the programs become more prone to lethal mutations or crossovers. This may reduce the quality of the solutions found by the G/P.

On the other hand the small program trees generated follow the "ockham's razor" principle<sup>34</sup> and are much easier to read.

---

<sup>31</sup> The G/P is a true hybrid from GA and GP to allow the optimisation of the values of non input terminals it further allow s adaptive mapping of linguistic variables to (fuzzy) numbers [HL95].

<sup>32</sup> This simplified trading model is said to be realistic for institutional traders.

<sup>33</sup> For Random Walk index see [PE 92].

<sup>34</sup> Ockham's razor: „The most likely hypothesis is the simplest one”.

**Table 1 Experimental settings**

GP Parameter	
Population Size	1000
Elitism	Yes
Tournament Selection	Group Size: 5
Depth Dependent Crossover	Rate: 2
Insert/Replace Node Mutation	Rate: 5
Area 1 <sup>st</sup> experiment	Open, High, Low, Close + equal number of GA/P optimised constant values
Area 2 <sup>nd</sup> experiment	Open, High, Low, Close, PrimRWI, RWIBB+, RWIBB-, RWIAvg, RWIHigh, DevStop1a, DevStop1b, DevStop1c, DevStop1d, LocHPeakA, LocLPeakA, LocHPeakB, LocLPeakB, LocLPeakC, HIGHA, LOWA, HIGHB, LOWB, LocHPeakC, RWIAvg, POut/dn, POut/dn, GlobHPeak, GlobLPeak + equal number of GA/P optimised constant values
Operators	*, +, -, /, MAX, MIN, DiffXC, LagXC, MovAvIC, IF
Problem	The agent can decide if he is either long or short of Dollar, since he needs to be positioned in one to the two currencies. The GP program that controls the behaviour of the agent indicates the position. It indicates a short position when the return value is smaller than zero and a long position when the value is equal or greater than zero. The agent performs a trade, when the sign of the math function is changing.
Trading cost	0,03%
Trading point	Today's close
Ratio In-Sample/ Out-Of-Sample	7/3

**Table 2 The 1<sup>st</sup> experiment with reduced input data set. Exemplary rules with return on Out-Of-Sample-Set greater than 0,10 \$.**

Absolute		% of max. return		Multiple of Buy&Hold		In-Sample performance Max. possible return: 2,275€
In	Out	In	Out	In	Out	Out-Of-Sample performance Max. possible return: 1,082 \$
0,361 \$	0,119 \$	16,0%	11,0%	1,27	4,25	$X = ((\text{Open} / \text{High}) - (\text{LagXC}(\text{Low}, 8) / 0,9213))$
0,350 \$	0,137 \$	15,5%	12,7%	1,24	4,92	$X = ((\text{Close} / \text{Low}) / \text{MAX}(-6,109, (\text{LagXC}(\text{Low}, 5) - \text{MAX}(\text{Low}, \text{MAX}(\text{High}, \text{High}))))))$
0,329 \$	0,102 \$	14,6%	9,5%	1,16	3,67	$X = (\text{MIN}((-4,3864 * -5,9608), (-5,9608 * \text{Open})) - \text{MIN}((-5,9608 + \text{High}), (-4,3864 / \text{Open})))$
0,310 \$	0,274 \$	13,7%	25,3%	1,10	9,83	$X = ((\text{MovAvIC}(\text{Low}, 5) - \text{Close}) * \text{MAX}(1,2506, \text{Open}))$
0,283 \$	0,028 \$	12,6%	2,6%	1	1	Buy&Hold

**Table 3 The 2<sup>nd</sup> experiment with complete input data set. Exemplary rules with return on Out-Of-Sample-Set greater than 0,10 \$.**

Absolute		% of max. return		Multiple of Buy&Hold		In-Sample performance Max. possible return: 2,275\$
In	Out	In	Out	In	Out	Out-Of-Sample performance Max possible return: 1,082 \$
0,533 \$	0,107 \$	23,6%	9,9%	1,88	3,83	$X = (\text{LocHPeakB} - ((\text{HIGHB}) + \text{MAX}(\text{RWIAvg}, \text{RWIBB-})))$
0,501 \$	0,113 \$	22,2%	10,4%	1,77	4,04	$X = (((\text{LocHPeakB} - \text{Open}) - (\text{RWIAvg} - \text{GlobHPeak})) + \text{HIGHA})$
0,501 \$	0,106 \$	22,2%	9,8%	1,77	3,80	$X = (\text{LocHPeakB} - ((-0,0023 + \text{RWIAvg}) + (\text{HIGHB} - \text{GlobHPeak})))$
0,486 \$	0,192 \$	21,5%	17,7%	1,71	6,88	$X = (\text{MAX}(\text{LocHPeakB}, (\text{RWIBB-} * \text{LocLPeakA})) - \text{MAX}(\text{MovAvIC}(\text{LOWB}, 2), (\text{RWIBB+} * \text{PrimRWI})))$
0,461 \$	0,192 \$	20,4%	17,7%	1,63	6,88	$X = (\text{MAX}(\text{LocHPeakB}, (\text{RWIBB-} * \text{LocLPeakA})) - \text{MAX}(\text{LOWB}, (\text{RWIBB+} * \text{PrimRWI})))$
0,459 \$	0,192 \$	20,3%	17,7%	1,62	6,87	$X = (\text{MAX}(\text{LocHPeakB}, (\text{RWIBB-} * \text{LocLPeakA})) - \text{MAX}(\text{LOWB}, (\text{High} * \text{PrimRWI})))$
0,404 \$	0,157 \$	17,9%	14,5%	1,42	5,62	$X = (\text{MAX}(\text{LocHPeakB}, \text{GlobHPeak}) - \text{MAX}(\text{LOWB}, (\text{High} * \text{PrimRWI})))$
0,394 \$	0,128 \$	17,5%	11,8%	1,39	4,59	$X = (\text{LocHPeakB} - (\text{RWIAvg} + \text{LOWB}))$
0,387 \$	0,139 \$	17,1%	12,9%	1,36	4,99	$X = (\text{LocHPeakB} - \text{MAX}(\text{LOWB}, (\text{High} * \text{PrimRWI})))$
0,283 \$	0,028 \$	12,6%	2,6%	1	1	Buy&Hold

Table 4 The euro/dollar exchange rate.

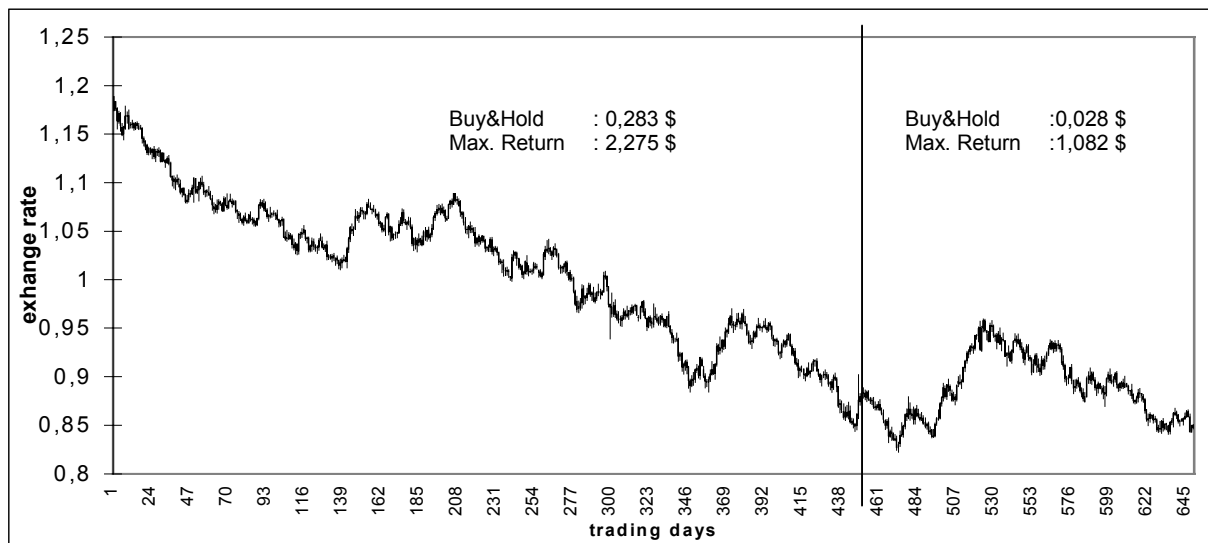


Table 5 The In-Sample performance of both experiments

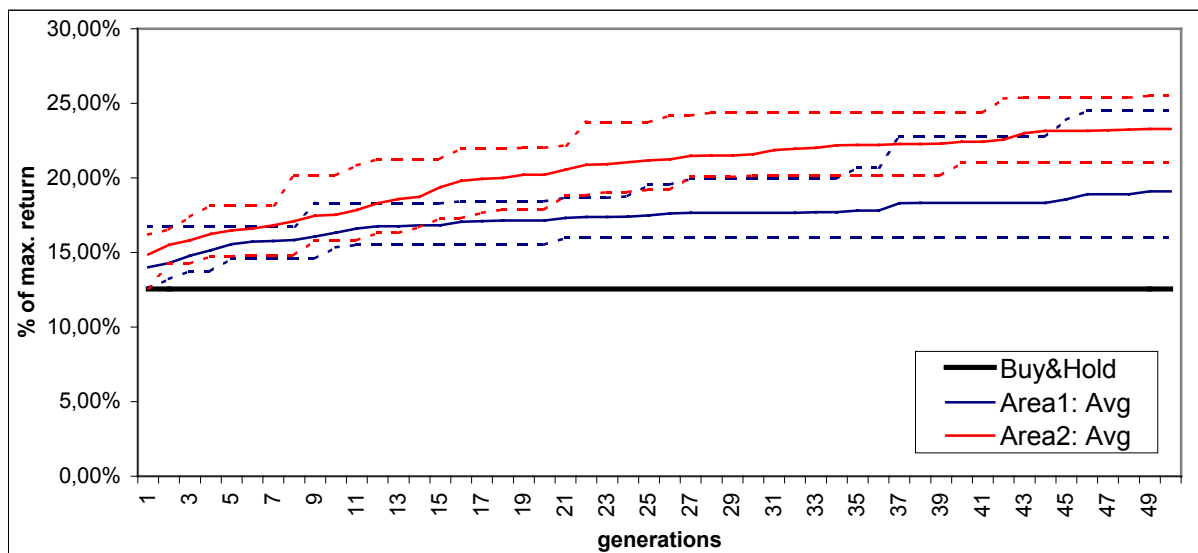
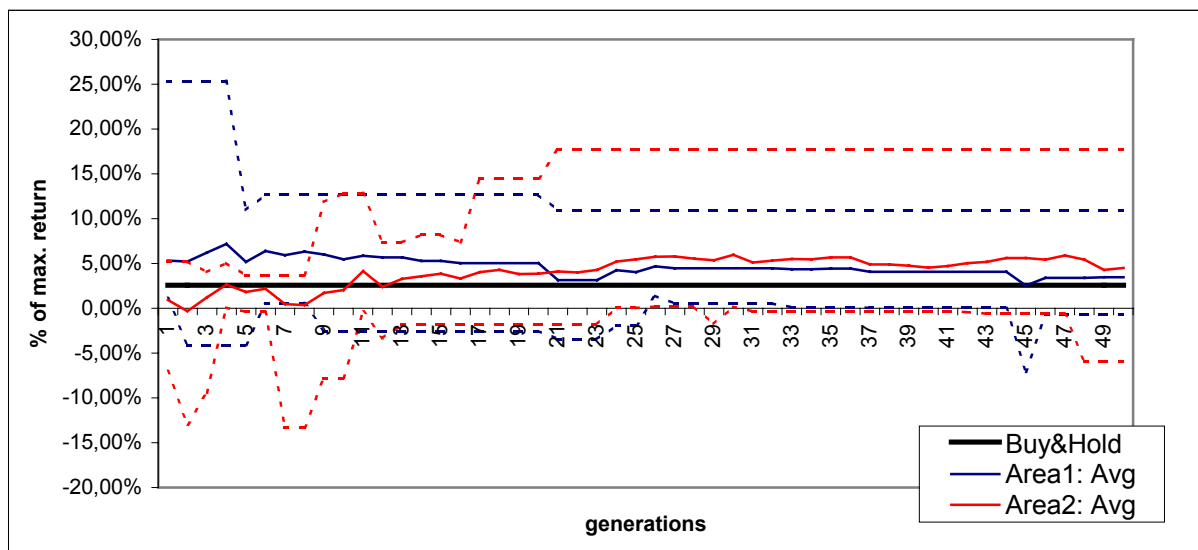


Table 6 The Out-Of-Sample performance of both experiments



These two experiments show, how important the choice of input variables is for the generation of profitable trading rules using GA/P. In the experiment without the RWI the developed rules perform considerably worse than the experiment with RWI data, on the In-Sample and on the Out-Of-Sample set<sup>35</sup>.

This suggests that the RWI does have some predictive power that can be utilised by the GP agents.

But as mentioned before there are some facts that restrict the quality of the solutions:

- Since the quality of the solutions found depend on the input data, it is obvious that the total lack of exogenous input data must limit the predictive power of the rules found.
- The trading agent has no internal memory. It resembles a trader that suffers total amnesia each morning. The agent isn't even conscious what position he currently holds.
- There is no chance in this scenario for the trading agent to move into a risk free position, if the market becomes too volatile for sensible trading.
- As mentioned already, the search space is rugged and highly multimodular. Bigger population sizes, niching algorithms and a distributed GP may increase performance.

But some rules are found that seem to be profitable on the In-Sample and on the Out-Of-Sample set, despite these restrictions and problems<sup>36</sup>.

## 5. Conclusions

Although only an application for generating technical trading rules in a simplified environment was presented here, it needs to be remembered that EA methods can be applied to nearly any kind of financial optimisation problem. And since there are several sub types of the general EA principle, one of them will most likely suit a certain problem best. The most general approaches are, the basic GA and EP. For real number parameter optimisation, ES strategies are most likely the best-suited method. To find simple classification and behaviour rules LCS can be used and a more general approach to optimise functions or computer programs was introduced with GP. Several other problem specific specialisations and extensions of EA methods can be imagined.

Also using MA the EA methods are easy to be combined with already existing, possibly problem specific, search heuristics. They will perform at least as good as the local search with which they were combined with. MA will seamlessly scale between any given standard search heuristic behaviour and the robust parallel population based search of the EA.

Since EA methods only need the fitness function to guide their search, they can also be applied to problems where no search heuristic is know so far. And if EA methods are extended to distributed EA, they perform well even if the search space dimension increases.

EA methods can be imagined as some kind of toolbox to find high quality solution for complex optimisation problems.

## 6. Future work

The presented results in the trading rules example were generated with an EA-Toolbox, which cannot be considered to be the state of the art in some aspects, if compared to modern scientific publications. Especially the ES part of that EA-Toolbox was very weak. But at the University of Tuebingen we are currently working on a modern and improved EA-Toolbox named JavaEvA<sup>37</sup>, which is implemented in JAVA. JavaEvA consists of GA, ES, GP, EP methods and possibly extended problem specific codings for EA. JavaEvA will also include basic schemes for MA to increase performance. The implementation in JAVA and the distributed client/server architecture allows us to process EA methods across a heterogeneous network of computers in parallel and to implement several distributed EA schemes.

JavaEvA can be applied flexible on a wide range of possible problems. Currently it's main field of application is the optimisation to process parameters for "Automated crystallisation"<sup>38</sup>. But we will also continue to work on financial applications to prove the power of JavaEvA in different applications. But in financial application we are dependent on external data and knowledge to develop problem specific operators and local search algorithms.

## 7. Acknowledgements

The GA/P-hybrid used here was developed in a master's thesis sponsored by the Commerzbank AG in Frankfurt. I would like to thank Dr. Koch and Dr. Rathjens for their support and also Mr. Schwerdner and Mr. Prielipp who supplied the raw data and their knowledge.

---

<sup>35</sup> To search effectively for good indices a Meta-EA could be used that optimises the area of indices used in a GP technical trading application, to increase the mean return of the GP application, for Meta-EA see [BT94].

<sup>36</sup> Although the trading model is perhaps too simple and not realistic.

<sup>37</sup> See [http://www-ra.informatik.uni-tuebingen.de/forschung/eva/welcome\\_e.html](http://www-ra.informatik.uni-tuebingen.de/forschung/eva/welcome_e.html) for further details.

<sup>38</sup> See <http://www-ra.informatik.uni-tuebingen.de/forschung/kombikat/welcome.html> for further details.

## 8. References

- [AK 99] Allen F. and Karjalainen R.: "Using genetic algorithms to find technical trading rules", in *Journal of financial economics*, 51, p. 245-271, 1999
- [AL 93] Arnone S., Loraschi A. and Tettamanzi A.: "A Genetic Approach to Portfolio Selection", *Neural Network World* 3(6), p. 597, 1993.
- [AP 93] Angeline, P. J. and Pollack, J. B.: "Evolutionary Module Acquisition" In *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), La Jolla, CA: Evolutionary Programming Society, pp. 154-163, 1993.
- [BN 95] Burke E. K., Newall J.P. and Weare R.F.: "A memetic algorithm for university exam timetabling". In E. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling*, LNCS 1153. Springer Verlag, 1996.
- [BT 94] Bäck T.: "Parallel Optimisation of Evolutionary Algorithms", Y. Davidor, H.-P. Schwefel and R. Männer, editors: *Parallel Problem solving from Nature - PPSN III, International Conference on on Evolutionary Computation*, pp 418-427, Springer, Berlin, 1994.
- [BW 93] Banzhaf W.: "Genetic Programming for Pedestrians" MERL Technical Report, 93-03, Mitsubishi Electric Research Labs, Cambridge, MA, 1993
- [CF 98] Chong F. S.: "A Java based distributed approach to genetic programming on the internet" Master's Thesis, Computer Science, University of Birmingham, 1998.
- [CM 98] Chang T.-J., Meade N., Beasley J.E. and Sharaiha Y.M.: "Heuristics for cardinality constrained portfolio optimisation". working paper available from the third author at The Management School, Imperial College, London SW7 2AZ, 1998.
- [CP 97] Cantu-Paz E.: "A survey of parallel genetic algorithms", *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol 10, number 2, pp. 141-171, 1997.
- [CP 99] Cantu-Paz E.: "Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms". To Appear in: GECCO-99, Genetic and Evolutionary Computation Conference, July 13--17, Orlando FL, 1999.
- [DC 59] Darwin, C.R.: "The Origin of Species" London (John Murray) 1859.
- [DL 94] Davis, L.: "Genetic algorithms and financial applications", in: Deoeck (ed., 1994), pp. 133-147, 1994.
- [DM 98] Drake A.E. and Marks R.E.: "Genetic algorithms in economics and finance: forecasting stock market prices and foreign exchange", Australian Graduate School of Management Working Paper 98-004, February, 1998
- [FA 95] Fogel L., Angeline P., and Fogel D.: "An evolutionary programming approach to self-adaptation on finite state machines" In J. McDonnell, R. Reynolds, and D. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*. MIT Press, 1995
- [FO 66] Fogel L.J., Owens A.J. and Walsh M.J.: "Artificial Intelligence through Simulated Evolution", New York, John Wiley & Sons, 1966
- [GD 89] Goldberg D. E.: "Genetic Algorithms in Search, Optimization and Machine Learning". Addison Wesley, 1989.
- [GF 53] Gray F.: "Pulse Code Communication", U. S. Patent 2 632 058, March 17, 1953.
- [GS 95] Geyer-Schulz A.: "Holland classier systems" *Proceedings of the International Conference on Applied Programming Languages*, June 4.-8., San Antonio, p. 45-55, 1995
- [HG 94] Horn J., Goldberg D.E. and Deb K.: "Implicit niching in a learning classifier system: Nature's way" Technical Report IlliGAL report No. 94001, University of Illinois at Urbana-Champaign, 1994
- [HJ 75] Holland J.: "Adaption in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Systems" The University Press of Michigan Press, Ann Arbor, 1975.
- [HJ 92] Holland J.: "Genetic Algorithms", *Scientific American*, Pages: 44-50, July 1992.
- [HL 95] Howard L.M. and D'Angelo, D. J.: "The GA-P: A Genetic Algorithm and Genetic Programming Hybrid". *IEEE Expert*. Vol 10, N. 3, June 1995
- [HO 96] Hansen, N. and Ostermeier, A.: "Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation" *Proceedings of the 1996 IEEE Intern. Conf. on Evolutionary Computation (ICEC '96)*: 312-317, 1996.
- [II 98] Ito T., Iba H. and Sato S.: "Non-destructive depth-dependant crossover for genetic programming", in *EuroGP'98 Proceedings, Lecture Notes in Computer Science 2038, Genetic Programming*, p. 71-82, 1998.
- [IS 99] Iba H. and Sasaki T.: "Using genetic programming to predict financial data", *CEC 99, Special session on time series prediction*, July 6-9, 1999
- [JA 95] Jarmo T. A.: "Indexed bibliography of genetic algorithms in economics". Report 94-1-ECO, University of Vaasa, Department of Information Technology and Production Economics,

1995

- [JM 97] Jonsson H., Madjidi P. and Nordahl, M.: "Evolution of trading rules for the FX market or how to make money out of GP" TRITA-PDC Report, ISRN KTH/PDC/R--97/5—SE, ISSN 1401-2731, 1997
- [KB 01] Kantschik W. and Banzhaf W.: "Linear-Tree GP and Its Comparison with Other GP Structures", in *EuroGP'01 Proceedings, Lecture Notes in Computer Science 2038, Genetic Programming*, p. 302-312, 2001.
- [KF 95] Kingdon, J. and Feldman, K.: "Genetic algorithms and applications to finance", *Applied Mathematical Finance*, vol. 2, No. 2, June, pp. 89-116, 1995.
- [KJ 00] Koza, J. R.: "A genetic approach to econometric modelling", published at Sixth world congress of the Econometric Society, Barcelona, Spain, August 27, 1990
- [KJ 92a] Koza, J. R.: "Genetic Programming" MIT Press Cambridge, MA 1992 1st Edition HC On the Programming of Computers by Means of Natural Selection, VG/VG Science 03782, 1992.
- [KJ 92b] Koza, J. R.: "Hierarchical Automatic Function Definition in Genetic Programming", in Whitley, Darrell (editor): *Proceedings of Workshop on the Foundation of Genetic Algorithms and Classifier Systems*, Vail, Colorado, San Mateo, CA: Morgan Kaufmann Publishers Inc. p. 297-318, 1992
- [LT 95] Loraschi A., Tettamanzi A., Tomassini M. and Verda P.: "Distributed Genetic Algorithms with an Application to Portfolio Selection Problems", in *Proceedings of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, D.W. Pearson, N.C. Steele and R.F. Albrecht (eds.), Springer-Verlag, 384, 1995.
- [MB 94] Montana D. J., Beranek B. and Newman: "Strongly typed genetic programming" Technical Report 7866, , Inc., March 25, 1994.
- [MP 01] Minerva T. and Poli I.: "Building ARMA models with genetic algorithms", in *EuroGP'01 Applications of Evolutionary Computing, Lecture Notes in Computer Science 2037, Genetic Programming*, p. 335-342, 2001.
- [ND 97] Neely C., Dittmar R. and Weller P.: "Is technical analysis in the foreign exchange market profitable? A GP Approach", in *Journal of Financial and Quantitative Analysis*, p. 405- 426, 1997
- [OB 01] O'Neill M., Brabazon A., Ryan C. and Collins J.J.: "Evolving market index trading rules using grammatical evolution", in *EuroGP'01 Applications of Evolutionary Computing, Lecture Notes in Computer Science 2037, Genetic Programming*, p. 343-351, 2001.
- [OR 01] O'Neill M. and Ryan C.: "Grammatical Evolution", in *IEEE Trans. Evolutionary Computation*, 2001
- [OT 95] Oussaidene M., Tomassini M., Pictet O.V., Dacorogna M.M., Chopard B. and Schirru R.: "Using Genetic Algorithms for Robust Optimization in Financial Applications" , *Neural Network World* 4/95, p. 573-587, 1995
- [OT 97] Oussaidene M., Tomassini M., Chopard B. and Pictet O.V.: "Parallel genetic programming and its application to trading model induction". *Parallel Computing (Netherlands)*, 23(8):1183--1198, 1997.
- [PC 87] Pettey C.S. et al.: "A Parallel Genetic Algorithm," *Proc. of the Second Intl. Conf. for Genetic Algorithms*, Morgan Kaufmann Publishers, p. 155-161, 1987.
- [PE 92] Poulos E.M.: "Are there persistent cycles", *Technical Analysis of Stocks and Commodities*, V10.9, p.385-389, 1992
- [RM 97] Ready M.J.: "Profits from technical trading rules" working paper University of Wisconsin - Madison, Finance, Investment and Banking, 1997
- [SH 75] Schwefel H.-P.: "Evolutionsstrategie und numerische Optimierung", *Dissertation, Technische Universität Berlin*, 1975
- [SJ 92] Shapcott J.: "Genetic algorithms for investment portfolio selection". Technical Report EPCC--SS92--24, Edinburgh Parallel Computing Centre, University of Edinburgh, 1992.
- [SR 01] Schulenburg S. and Ross P.: "An LCS Approach to Increasing Returns: On Market Efficiency and Evolution". To be presented at the Fourth International Workshop on Learning Classifier Systems, IWLCS-2001, held at the Genetic and Evolutionary Computation Conference GECCO 2001, San Francisco, California, July 7-11, 2001.
- [SS 80] Smith S.F.: "A Learning System Based on Genetic Adaptive Algorithms". PhD thesis, University of Pittsburgh, Pittsburgh, 1980.
- [ST 01] Santini M. and Tettamanzi A.: "Genetic Programming for financial time series prediction", in *EuroGP'01 Proceedings, Lecture Notes in Computer Science 2038, Genetic Programming*, p. 361-371, 2001.
- [TP 96] Turney P.: "Myths and legends of the Baldwin Effect". In Fogarty T. and Venturini G. (eds.), *Proceedings of the ICML-96 (13th International Conference on Machine Learning)*, 1996
- [TR 89] Tanese R.: "Distributed genetic algorithms," *Proc. Third ICGA*, pp. 434—439, 1989

- [VL 00] Van Veldhuizen D.A. and Lamont G.B.: "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art", in *Evolutionary Computation* 8 (2), p. 125-147, 2000.
- [WA 91] Wright A. H.: "Genetic algorithms for real parameter optimization". In Rawlins G. J. E. (eds.), *Foundations of Genetic Algorithms*, p. 205--218, 1991
- [WG 94] Whitley D., Gordon V.S. and Mathias.K.: "Lamarckian evolution, the Baldwin effect and function optimization". In Davidor Y., Schwefel H. P. and Berlin Manner R., editors, *Parallel Problem Solving from Nature - PPSN III*, p. 6-15. Springer-Verlag, 1994.
- [WS 94] Wilson S.W.: "Classifier Systems and the animat Problem", in *Machine Learning* 2, p.199-228, 1987
- [YA 00] Yoshihara I., Aoyama T. and Yasunaga M.: "Financial Application of Time Series Prediction based on Genetic Programming" in *Proceedings of the Genetic and Evolutionary Computation Conference* , p. 537, 2000
- [ZS 99] Zemke S.: "Bagging Imperfect Predictors". Presented at ANNIE'99. In C. Dagli eds. *Smart Engineering System Design*, ASME Press, 1999.