Diplomarbeit

Conception, Implementation and Validation of an Algorithm for the automated Chaining and simultaneously Parameter Optimization of Image Processing Operations

June 1998

angefertigt am
Institut für Werkzeugmaschinen und Fabrikbetrieb
Fachgebiet Steuerungstechnik für Produktionsanlagen
Prof. Dr.-Ing. W. Adam

von
cand.-inform. Stephan Rudolf
Matrikelnummer 100385
Studiengang Informatik
Fachbereich 13
Abstract

One method to detect faults in surfaces is the application of image processing operations, in order to emphasize areas with faults. Usually, a suitable chaining of image processing operations has to be found. During the search, suitable image processing operations are manually sequenced. This has to be repeated till the following goals are reached: Areas with faults should be emphasized and areas without faults should be suppressed. Even in the case of comparable few image processing operations, with each having one or more parameters, this is a tedious task. With increasing number of operations there is a high increase in the effort, which is known as ‘combinatorial explosion’ of choices.

This paper describes a framework for evolutionary algorithms and a concrete realization, which performs this task automatically. It is designed to use a priori knowledge of the user to narrow the possible search space. Also there are given some results by working with this algorithm.

Keywords: evolutionary algorithms, evolutionary computation, image processing, fault detection, genetic algorithms, genetic programming, Evolutionsstrategien, multiobjective optimization.
Contents

1 Zusammenfassung 1 3

2 Introduction 4
  2.1 Global overview ........................................... 4

3 Used Image Processing Operations 6
  3.1 Overview .................................................. 6
  3.2 Definitions ............................................... 6
    3.2.1 Basic ................................................ 6
    3.2.2 Mathematical Morphology .............................. 7
    3.2.3 Boolean ............................................. 8

4 Expressions and expression graphs 9
  4.1 Overview .................................................. 9
  4.2 Expressions .............................................. 9
  4.3 Expression graphs ....................................... 10
  4.4 Image processing expressions (IPEs) ..................... 10
    4.4.1 Source image IPE .................................... 10
    4.4.2 Binarization IPE .................................... 10
    4.4.3 Morphological IPEs .................................. 11
    4.4.4 Mask IPE ........................................... 11
    4.4.5 Boolean IPE ......................................... 12
  4.5 Image processing expression graphs ...................... 12
  4.6 Strategy expressions .................................... 12
  4.7 Strategy expression graphs ................................ 13

5 The Generic Evolutionary Algorithm GEA 14
  5.1 Overview .................................................. 14
  5.2 Motivation ................................................ 14
    5.2.1 Evolutionary Algorithms and Evolutionary Computing 14
    5.2.2 Genetic Algorithms .................................. 15
    5.2.3 Evolutionstrategien ................................ 16
    5.2.4 Genetic Programming ................................ 16
    5.2.5 Idea ................................................ 17
  5.3 Concept .................................................. 17
    5.3.1 Elements ............................................ 17
    5.3.2 Phases ............................................. 18

1Introduction (chapter 2) in German.
5.4 Implementation ................................................. 19
  5.4.1 Important data structures .......................... 19
  5.4.2 Attributes ........................................ 19
  5.4.3 Operators ......................................... 19
  5.4.4 Sequences ....................................... 27
5.5 Relationships with other algorithms .................. 29
  5.5.1 Rechenberg's *Evolutionstrategien* ............ 29
  5.5.2 Genetic Algorithms ............................... 29
  5.5.3 Genetic Programming .............................. 30
5.6 Why ranking? ........................................ 30
5.7 User interface ......................................... 30

6 Working with GEA 32
  6.1 Overview ............................................ 32
  6.2 Optimization goal .................................... 32
    6.2.1 Evaluation function .......................... 33
  6.3 Unique parameters ................................... 33
  6.4 Visualization ....................................... 34
  6.5 Simple problem ..................................... 35
    6.5.1 Goal ........................................... 35
    6.5.2 Parameters .................................... 35
    6.5.3 Result ......................................... 35
    6.5.4 Interpretation .................................. 36
  6.6 Not so simple problem ............................... 38
    6.6.1 Goal ........................................... 38
    6.6.2 Parameters .................................... 38
    6.6.3 Result ......................................... 38
    6.6.4 Interpretation .................................. 40
  6.7 Fault problem ....................................... 41
    6.7.1 Goal ........................................... 41
    6.7.2 Parameters .................................... 41
    6.7.3 Result ......................................... 42
    6.7.4 Interpretation .................................. 43

7 Summary 45
  7.1 Further work ........................................ 45
    7.1.1 To do GEA .................................... 45
    7.1.2 To do IPG ...................................... 45
    7.1.3 To do Application ............................... 45

Figures 46

Tables 48

Bibliography 49
Chapter 1

Zusammenfassung


Der evolutionäre Algorithmus GEA wird vorgestellt: Er ist sehr flexibel und leicht zu erweitern und dadurch an bekannte evolutionäre Algorithmen in dem Sinne anzupassen, daß er sie 'emultiert'. Die Grundidee hinter GEA ist eine objektorientierte und 'generische' Sichtweise auf eine Klasse von Algorithmen, und nicht nur ein spezieller Algorithmus.


Der verwendete Algorithmus wird in dieser Arbeit so genau beschrieben, daß er vom Leser dieser Arbeit nachprogrammiert werden kann, um die Ergebnisse nachzuvollziehen.

---

1Introduction (chapter 2) in German.
2Dieser Begriff ist mit Absicht so abstrakt und allgemein gewählt.
Chapter 2

Introduction

This paper is created from a practical point of view. How to optimize black-box functions without much knowledge about the topology of the search space? With a directed – not random! – probabilistic search. Many concepts and methods to perform such a search come from the field of ‘evolutionary computing’\(^1\).

The generic evolutionary algorithm called ‘GEA’ is described: It is very flexible and easy to extent to adapt it – in an ‘emulating sense’ – to existing algorithms from this area. The core idea behind the algorithm GEA is an object-oriented and ‘generic’ view onto a class of algorithms, not only one special algorithm.

Three concrete optimization tasks from the field of image processing are described in this paper with the results. The user supplies a gray valued source and a binary target image. Starting from the source image, image processing operations will be applied to approximate the target image as exactly as possible. By the concept of image processing expression graphs (IPEGs) together with their corresponding strategy expression graphs (SEGs), a human being is able to strongly support the algorithm in searching for good solutions. So a priori knowledge is used. This can be the difference between success or fail!

The used algorithm will be described in so much detail, that the reader of this paper is able to implement it and to prove the results.

2.1 Global overview

A global overview is given here from the end to the beginning, but it is suggested to read this paper sequentially from the beginning to the end.

In last chapter 7 there is given a short summary about what could be done in future. The work in this paper comprises multiple parts, which all together are used to perform the optimization tasks in chapter 6.

To perform these optimizations an optimization algorithm is needed. Its concepts are described in chapter 5. People which are only interested in evolutionary computing from the evolutionary algorithm view – and want to take the to-be-optimized as black box – could start there.

\(^1\)It is intended to use this abstract and general term.
Also there has to be something to be optimized: these are image processing expression graphs (IPEGs), which are introduced in chapter 4.

IPEGs deal with image processing operations, which are introduced first in chapter 3.
Chapter 3

Used Image Processing Operations

3.1 Overview

In this chapter some important image processing operations are defined, which are the basis for the later explained image processing expressions.

3.2 Definitions

Some basic definitions are given in the following, for later defining image processing operations.

3.2.1 Basic

A gray valued image $p$ is a function which maps a location $l$ in this image to a gray value $g$:

$$p(l) = g, \text{ for } g \in [0, \ldots, g_{\text{max}}].$$

(3.1)

Definitions for gray valued images apply also to binary images with only two gray values $\{0, 1\}$. Variable $l$ (for location) in the following definitions denotes a pixel in an image and can be seen as coordinates of this pixel, e.g. $(col, row)$ or $(x, y)$. ¹ Used coordinates in this paper are denoted $(x, y)$: $x$ stands for the position in the column, $y$ for the row of the image; $(0, 0)$ denotes the location of the pixel in the upper left corner.

The complement $C$ of an image $p$ with a maximum possible gray value $g_{\text{max}}$ is defined as

$$p^C(l) = g_{\text{max}} - p(l).$$

(3.2)

For a binary image this definition leads to a boolean $\text{not}$ for every pixel ($g_{\text{min}} = 0$ and $g_{\text{max}} = 1$).

¹A good naming of items is important: $c$ for coordinates (instead of $l$ for location) could be associated with a constant semantic, $x$ as placeholder for $(x,y)$ is confusing and $(x,y)$ is quiet long. Often there is used $f$ for an image function, but what means $f^2$? Here $i$ and $l$ (for image) are avoided, because they could be confused with indices or the identity function.
The point wise maximum $\vee$ and the point wise minimum $\wedge$ of two gray valued images $p_1$, $p_2$ with identical domain spaces are denoted $p_1 \vee p_2$ respectively $p_1 \wedge p_2$ and are defined for every point $x$ as:

\[
(p_1 \vee p_2)(l) = \max[p_1(l), p_2(l)],
\]

\[
(p_1 \wedge p_2)(l) = \min[p_1(l), p_2(l)].
\]

For binary images $\vee$ corresponds to or’ing and $\wedge$ to and’ing both.

The binarization operation $T$ with thresholds $t_i$ and $t_j$ for a gray valued image $p$ is defined as

\[
[T_{[t_i,t_j]}(p)](l) = \begin{cases} 
1 & \text{for } t_i \leq p(l) \leq t_j \\
0 & \text{otherwise}
\end{cases}
\]

### 3.2.2 Mathematical Morphology

The knowledge about mathematical morphology has mainly come from the books [GW92, So98].

The point reflection $\hat{S}$ for a morphological structuring element $S$ is defined as

\[
\hat{S} = \{-s|s \in S\}.
\]

The erosion $\circ$ of an image $p$ by a structuring element $S$ is defined as

\[
p \circ S = \bigwedge_{s \in S} p_{-s},
\]

the dilation $\oplus$ as

\[
p \oplus S = \bigvee_{s \in \hat{S}} p_{-s}.
\]

A special case is the computation near the borders of the image: If the center of the structuring element is moved over the whole image, parts of the structuring element sometimes cover pixels outside the image. \(^2\) For erosion the values of pixels outside the image are treated as $g_{\text{max}}$, for dilation as $g_{\text{min}}$. This results in simply ignoring areas in the structuring elements, which are outside the image.

Note: In definition 3.8 the structuring element $S$ is reflected. The reason for this choice is not made by me, for a variant see footnote 3 - could be the following: For an asymmetrical structuring element erosion cuts away more from and dilation adds more to the foreground on its longer side - measured with respect to the central point - than on its shorter one. But both 'prefer' the same side. This is easy to memorize and leads to elegant definitions for opening and closing with no reflection of the structuring element. \(^3\)

\(^2\)Except for the structuring element consisting only of the center point.

\(^3\) But there also exists the formulation without reflection in literature (e.g. [So98]). This formulation leads to the need of reflecting the structuring element in the definitions for opening and closing operations, but also to more elegant formulation of other equations.

I don’t prefer one of these formulations, but I have had to choose one.

For symmetrical structuring elements this makes no difference, but there is a difference in case of asymmetrical structuring elements. One has to be aware of this point while reading literature about mathematical morphology, because many equations based on differing basic definitions differ in reflecting the structuring element or not.
Now opening and closing of an image can be defined in terms of erosion and dilation. The opening $\circ$ of an image $p$ by a structuring element $S$ is defined as

$$ p \circ S = (p \ominus S) \oplus S, \quad (3.9) $$

i.e. $p$ will be eroded and then the result dilated.

The closing $\bullet$ of an image $p$ by a structuring element $S$ is defined as

$$ p \bullet S = (p \oplus S) \ominus S, \quad (3.10) $$

i.e. $p$ will be dilated and then the result eroded: this is the reversed order as for the opening case.

### 3.2.3 Boolean

An exclusive or operation $\text{xor}$ with two binary images $p_1$ and $p_2$ as arguments is defined as

$$ p_1 \text{xor} p_2 = (p_1 \lor p_2) \land (p_1 \land p_2)^C. \quad (3.11) $$

A boolean operation $B$ with two binary images $p_1$ and $p_2$ as arguments and a boolean rule $r$, $r \in \{0, \ldots, 15\}$ is defined as

$$ [B_r(p_1,p_2)] = \begin{cases} 
0 & \text{for } r = 0 \\
p_4 \land p_a & \text{for } r = 1 \\
p_a^C \land p_a & \text{for } r = 2 \\
p_a & \text{for } r = 3 \\
p_4 \land p_a^C & \text{for } r = 4 \\
p_4 & \text{for } r = 5 \\
p_4 \text{xor} p_a & \text{for } r = 6 \\
p_4 \lor p_a & \text{for } r = 7 \\
p_4 \lor p_a^C & \text{for } r = 8 \\
p_4 \text{xor} p_a^C & \text{for } r = 9 \\
p_4^C & \text{for } r = 10 \\
p_4^C \lor p_a & \text{for } r = 11 \\
p_4^C \lor p_a^C & \text{for } r = 12 \\
p_4 \lor p_a & \text{for } r = 13 \\
p_4 \lor p_a^C & \text{for } r = 14 \\
1 & \text{for } r = 15 
\end{cases}. \quad (3.12) $$

These rules are covering all possibilities to combine two binarized images.
Chapter 4

Expressions and expression graphs

4.1 Overview

In section 4.2 expressions will be introduced, section 4.3 introduces expression graphs, which consist of expressions. Section 4.4 explains the concept of image processing expressions, which is fundamental for all optimizations in this paper. Section 4.5 describes how to combine these image processing expressions. Strategy expressions, which control the generating and changing of image processing expressions are explained in section 4.6 and their connection to strategy expression graphs in section 4.7.

4.2 Expressions

A single expression (see figure 6.9 for an example of a graph of such expressions) consists of

- zero, one or more subexpressions, which are needed to evaluate such an expression,
- parameters, which control its functionality and
- an evaluation method for local computations at the graph nodes.

To get the evaluation of an expression, two logical \(^1\) steps have to be performed:

1. evaluation of the subexpressions of this expression (if any exists), and

2. performing the evaluate method, which uses the evaluated subexpressions of the previous step.

An expression could be seen as a 'function' \(^2\) with arguments; evaluating it means, evaluating the arguments first and after that computing the 'function'

---

\(^1\) The concrete implementation may perform these two steps in one evaluation method.

\(^2\) Quoted, because it's not a function with mathematical semantics. E.g. an expression with subexpressions could return random numbers which are not affected by the subexpressions. Then it is not a function which has to return the same output if the input is the same.
value. An expression without subexpressions could be seen as a 'function' without arguments.

4.3 Expression graphs

Expression graphs are directed graphs, whose nodes are one or more expressions (for examples see figures 6.3, 6.9 and 6.17). They have one root expression, which returns the result of the whole expression graph. If there are more expressions than only one, then the root expression has one or more subexpressions, these subexpressions could have further subexpressions, too, and so on. The leaves in an expression graph are expressions without any subexpressions.

4.4 Image processing expressions (IPEs)

In this section an overview is given over used image processing expressions or IPEs.

4.4.1 Source image IPE

An IPE $p' = Source$ is constant during one optimization run. It returns the source image, which underlies further processing by other IPEs.

Always used as leaf in IPEGs.

Mutation

No mutation.

4.4.2 Binarization IPE

An IPE $p' = Binarize_{\text{from}, \text{to}}(p)$ is defined as

$$
[Binarize_{\text{from}, \text{to}}(p)](l) = \begin{cases} 
T_{\text{from}, \text{to}}(p)(l) & \text{for } \text{from} \leq \text{to} \\
T_{\text{from}, \text{to}}(p)^C(l) & \text{for } \text{from} > \text{to}
\end{cases}
$$

for from, to, $p(l)$ out of discrete gray level range \{0, ... , 255\}.

It maps every gray value from the actual gray value range to 0 or 1. The second case with condition from > to in equation 4.1 gives the possibility to set the bordering regions of the gray value range to 1 and its interior to 0.

Mutation

Mutable parameters are from and to. They are binary coded 8-bit unsigned integers \(^3\). The used mutation probability is the probability for flipping each bit. This means that every bit will be flipped with this probability.

\(^3\)Gray code would be an alternative.
4.4.3 Morphological IPEs

There are four morphological expressions, which are defined together. An IPE $p' \equiv \text{Morph}_{\text{op}}(p, M)$ is defined as

$$\text{Morph}_{\text{op}}(p, M) = p \text{ op } M, \text{ for } \text{op} \in \{\oplus, \ominus, \circ, \bullet\}. \tag{4.2}$$

**Mutation**

These expressions are not mutable.

4.4.4 Mask IPE

An IPE $M = \text{Mask}_{\text{size}, x_{\text{limit}}, y_{\text{limit}}}$ represents a discrete structuring element for morphological operations. This is realized by a collection of relative point $(x, y)$ offsets: They are relative with respect to a center point $(0, 0)$, which might be included in this mask as well. Examples are given in figure 4.1. The $x_{\text{limit}}$ and $y_{\text{limit}}$ are the absolute values of offsets for the corresponding dimension, e.g. if $x_{\text{limit}} = 1$ and $y_{\text{limit}} = 1$, the point offsets $(-1, 1), (1, -1), (0, 0)$ are allowed, but $(-2, 0), (2, 1), (0, 2)$ are forbidden. Parameter size denotes the size of the mask, if size = 3 three point offsets with different or equal (see below) values are stored.

The result of the expression is a mask with no equal point offsets in it: E.g. if a stored mask of size = 6 includes $((-1, -1), (-1, -1), (0, 0), (1, 0), (1, -1), (1, 0))$ the result is a mask with only $((-1, -1), (0, 0), (1, -1), (1, 0))$.

Because there can be equal point offsets in the stored mask, the resulting mask can be shorter than the stored mask. This gives the possibility to reduce the size – and therefore complexity – of the applied mask while running the optimization. The result $M$ of the Mask IPE is used by morphological operations (see 4.4.3).

The Mask IPE always is used as leaf in IPEGs.

**Mutation**

The stored point offsets are the entities, which are mutable. The used mutation probability is the probability for changing each point offset. If it is – according
to mutation probability – decided to change one point offset \((x,y)\), a new one 
\((x,y)'\) is randomly generated by

\[
(x,y)' = (-R_{\text{int}}(-\text{size}, \text{size}), R_{\text{int}}(-\text{size}, \text{size})),
\]

\(R_{\text{int}}(\text{from}, \text{to})\) is a random generator, which returns uniform distributed random numbers out of the set \(\{\text{from}, \ldots, \text{to}\} \subset Z\).

4.4.5 Boolean IPE

There are several boolean operations, which will be defined at once. An IPE \( p = \text{Bool}_{\text{rule}} \) is defined as

\[
p = \text{Bool}_{\text{rule}}(p_1, p_2) = B_{\text{rule}}(p_1, p_2), \text{rule} \in \{1, \ldots, 14\},
\]

see formula 3.12 for the definition of \(B_{\text{rule}}\). It combines two images \(p_1, p_2\) by rule \text{rule}. Note: Rules 0 and 15 are not used, because they return an image where all bits are equally set (1) or unset (0). If it is wanted \(^4\) – while recursively combining these operations – to filter out one of the parameter images, this could be achieved by rules 3 and 5, which return one of its argument images \(p_1\) or \(p_2\).

Mutation

These expressions are not mutable.

4.5 Image processing expression graphs

An Image processing expression graph (IPEG) consists of expressions, which describe image processing operations. A leaf returns the source image or another object \(^5\), which is processed by other expressions within the IPEG. All expressions, which are not leaves, return an image as – intermediate – result, the root expression returns the global result image of the whole IPEG. \(^6\)

4.6 Strategy expressions

Strategy expressions SEs are used to control the mutation, recombination and initialization of IPE’s. There are two types of SEs:

1. The single operation strategy expression SOSE, which puts exactly one IPE at its corresponding node, and

2. the multi operation strategy expression MOSE, which chooses an IPE by a given probability, which comprises a mapping between an object \(o\) out of a set \(O\) to its probability \(P(o), P(o) \in [0..1], \sum_{o \in O} P(o) = 1\). This is realized in ST (short cut for ‘Smalltalk,’ a parent of object oriented programming languages) by a so-called ProbabilityDictionary class. An instance of a

\(^4\)By a human being or an optimization algorithm.

\(^5\)Currently a mask for morphological operations is the only alternative.

\(^6\)This is the current status and not a constraint: it is not forbidden to build other expressions, which e.g. perform computations on masks.
ProbabilityDictionary stores associations between ST objects and their probabilities and allows – by a given random generator – to retrieve one object according its assigned probability.

SEs consist of

- one IPE (SOSE) or a set of IPEs with probabilities of choice (MOSE),
- a recombination probability weight \( r_{p\text{-weight}} \),
- an operation mutation probability weight \( m_{op\text{-weight}} \) (0 for SOSE),
- a parameter mutation probability weight \( m_{pp\text{-weight}} \).

4.7 Strategy expression graphs

A strategy expression graph (SEG) reflects its corresponding IPEGs by its structure. For each optimization problem considered in this paper, a SEG has to be build, which controls generation, recombination and mutation of each corresponding node in the generated IPEGs.

While traversing the strategy expression graph local weights \( r_{p\text{-weight}} \), \( m_{op\text{-weight}} \) and \( m_{pp\text{-weight}} \) have to be multiplied with the corresponding global \(^7\)

- recombination probability \( r_{p\text{-global}} \),
- operation mutation probability \( m_{op\text{-global}} \) and
- parameter mutation probability \( m_{pp\text{-global}} \),

to get the used probabilities for recombination, operation mutation and parameter mutation for the currently visited node in the IPEG.

Examples for SEG’s can be found in section 6.

\(^7\)They are defined in a ChildGenerator class in ST.
Chapter 5

The Generic Evolutionary Algorithm GEA

5.1 Overview

Section 5.2 shows the idea behind GEA and explains, why to use another evolutionary computing algorithm instead of the existing ones, section 5.3 describes the design goals of GEA. The implementation will be explained in full detail – and so it is quite long - in section 5.4. Relationships to other algorithms out of the field of evolutionary computing are presented in section 5.5. In section 5.6 there are some remarks to the features of ranking. Finally there is given a view onto the user interface in section 5.7.

5.2 Motivation

5.2.1 Evolutionary Algorithms and Evolutionary Computing

The term 'evolutionary algorithms' (EAs) has been chosen to comprise all classes of algorithms on computers, which are inspired by concepts of biology, especially the concepts evolution and genetics. In addition, terms should be avoided which have a special meaning and denote one class of evolutionary algorithms. Such EA classes are e.g. Evolutionsstrategien (ES's) [Re94, Mut82, Sch77], genetic algorithms (GAs) [Go89] and genetic programming (GP) [Koz92] ¹. For either kind there are variants and enhancements of the original algorithms. Also there are many possibilities to combine them.

Evolutionary computing (EC) denotes the application of evolutionary algorithms.

¹Ordering is alphabetically and does not express the importance of one of these schools.
Typical evolutionary algorithm

In figure 5.1 there is shown the typical structure of an evolutionary algorithm (EA). First there is an initialization phase, which could be seen as generating individuals in a Monte Carlo-like manner. Then a loop starts, in which the following happens:

- test for termination criterion (time, fitness (evaluation), etc.) and exit loop if it is fulfilled, otherwise...
- increase the time counter,
- select a sub-population (parent individuals) for offspring production,
- recombine the 'genes' of selected parents (generate childs),
- perturb the mated population stochastically (mutate by a mutation probability),
- evaluate its new fitness (evaluate childs),
- select the survivors from actual fitness (limit size of population).

Algorithm EA is

```
// start with an initial time
t := 0;

// initialize a usually random population of individuals
initpopulation P (t);

// evaluate fitness of all initial individuals in population
evaluate P (t);

// test for termination criterion (time, fitness, etc.)
while not done do

  // increase the time counter
  t := t + 1;

  // select sub-population for offspring production
  P' := selectparents P (t);

  // recombine the "genes" of selected parents
  recombine P' (t);

  // perturb the mated population stochastically
  mutate P' (t);

  // evaluate its new fitness
  evaluate P' (t);

  // select the survivors from actual fitness
  P := survive P,P' (t);

end
```

Figure 5.1: Pseudo code for an evolutionary algorithm, taken from [HB98].

5.2.2 Genetic Algorithms

Genetic algorithms (GAs)

"...owe their name to an early emphasis on representing and manipulating individuals in terms of their genetic makeup rather than using a phenotypic representation." [JS93]

The internal representation are binary strings, which are manipulated by recombination/crossover and mutation operators. For GAs mutation is less important than recombination/crossover, it is used to ensure diversity in the population during the optimization run. To be domain independent there is a focus on the genotype and not on the phenotype, which represents the domain dependent interpretation of the bit strings. This interpretation directly leads to
the so called 'coding problem,' because it has to be chosen a suitable coding of the concrete problem.

Parents are selected according to a probability proportional to their 'fitness,' which is a - to be maximized - scalar evaluation.

The theoretically basis is the so called 'schema theorem' [Ho92], which analyzes the distribution of so called 'schemata' reached after multiple applications of the recombination/crossover operator. Schemata are local bit patterns, which state for adjacent bits in a binary string, if they are 0, 1 or arbitrary. The schema theorem states that schemata in good (high fitness) evaluated bit strings are diffusing through the population exponentially with a rate proportional to their usefulness. They displace the exponentially diminishing schemata in bad (low fitness) bit strings.

5.2.3 Evolutionsstrategien

Evolutionsstrategien

“... (ESs) were developed with a strong focus on building systems capable of solving difficult real valued parameter-optimization problems ... " [JS93]

“They walk (better diffuse) uphill on ways of steepest increase.” 2 [Rec94]

The internal representation are real valued vectors representing the parameters to be optimized. They are manipulated primarily by the mutation operator, which changes the entries according to a Gaussian distribution. Also there are strategy parameters like mutation stepwidths, which will be adapted during an optimization run and are very important for the functionality of an ES, too.

Parents are selected with uniform distributed probability according their ranked 'quality,' which is a - to be maximized - scalar evaluation. The term 'fitness' in GA terminology is just replaced by the term 'quality'.

For some models of quality functions there exist formulas for the success probability and the success speed [Rec94]. For a multimodal \( \mathbb{R}^n \rightarrow \mathbb{R} \) function optimization they are directly usable without any coding problem and therefore the natural choice.

5.2.4 Genetic Programming

Genetic programming (GP) is an extension of the genetic algorithm, in which the evolving population consists of computer programs rather than bitstrings or other fixed data structures. The original GP algorithm and variations of it are discussed at great length in the book of John Koza [Koz92].

The internal representation of the population is a set of randomly generated programs, which are represented as expression trees, which are composed of the available simple programmatic ingredients (node and terminal functions). The algorithm is as follows: GP iterates with iteratively applying the standard genetic operations selection and crossover. Crossover chooses two individuals at random, roughly proportional to fitness or to ranking (mating selection), and

\[2\text{Original: "Sie laufen (besser diffundieren) auf Wegen steilsten Anstiegs bergan."} \]
swaps randomly chosen sub-trees of them. Selection copies a certain percentage recombinated expression trees to the new generation in proportion to their fitness or ranking. Mutation (e.g. by replacing randomly chosen sub-trees with newly generated ones) is seldom used in GPs. As a result of the iteration, the population consists of evolved expression trees which are also executable – hopefully better problem-solving or goal-fulfilling – programs.

This approach offers interesting possibilities to the field of evolutionary computation. The image processing expression graphs in this paper (see section 4.5) have a relationship to these expression trees.

5.2.5 Idea

Through implementing different kinds of evolutionary algorithms\footnote{E.g. \textit{Evolutionstrategien}\ (ES’s) \cite{sch77}, hill climbing with learning \cite{hcWL} \cite{KPP95}, stochastic hill climbing with learning by vectors of normal distributions \cite{SHCLYND} \cite{RK96, Rud97}.} I have made the experience that the concepts are similar, but the concrete algorithms differ, so that I have had to implement for each algorithm its own ST class. This has led to implementing many similar, but not identical methods. Different ST classes are definitely not appropriate to mix them. So the following question has come up:

\textbf{How to be flexible without working twice?}

Therefrom has arisen the idea to work on a more abstract level and split concepts of these differing, but in many aspects similar EAs, into small but flexible modules, which are easy to combine. This should ease the implementation of new and variants of existing EAs once a base of modules is available.

5.3 Concept

In figure 5.2 is shown the principal design of GEA.

The design goals which underly the conception of GEA are

- to be mostly flexible and to avoid the fixation on a special kind of EA,
- to add new parts should be possible and easy and
- there should be a high reusability of the existing ones.

GEA is not a monolithic algorithm, where only the parameters are to be chosen. Instead it has to be seen as a kind of toolbox of evolutionary and other operators. The concrete algorithm – which does the work – arises from the connection of different parts which are working together.

5.3.1 Elements

A GEA maintains operators, attributes and sequences. A schematic overview is given in figure 5.2. Examples for operators are selection, surviving, evaluating, displaying values in a plot window, etc. Attributes are something like population (central), step (of computation), rankings (a measure for the relative
5.3.2 Phases

There are two phases which are needed to get a GEA up and running: construction phase and working phase.

In the construction phase operators have to be registered and connected. Mostly, connections are not established directly by referencing one operator by another. They are established indirectly by working with the same attributes referenced by operator attributes and through the sequences, which give orders of operators in time. After that every operator has to initialize GEA attributes and its own private data structures which it needs to get to the working phase.

In the working phase sequences of operators are started. These sequences are able to contain (call) other sequences. On an abstract level operators could be seen as commands of an evolutionary computation language, attributes as variables and sequences as programs.

4Evaluation is a more neutral term than 'quality' or 'fitness'. Often it is intended to minimize the evaluation, and minimizing quality or fitness sounds like reducing it, but that is not meant.

5There is a self-reference: in the ST implementation an operator is called by a GEA and gets the whole GEA as argument.
5.4 Implementation

5.4.1 Important data structures

Individual
An individual has three components,
1. a genotype, which is generated by generators and changed – by recombination and mutation – by child generators;
2. an evaluation, which is computed by an evaluator; and
3. a birth (-day), which is set to the actual 'time' measured in optimization steps (here attribute #step).

Population
A very central data structure, which is used by many operators as an attribute, is the population. A population is a sequenceable collection of sorted – by a sorter – or unsorted individuals.

5.4.2 Attributes
Attributes are the data of a GEA. They are referenced by a #name. This #name is a unique symbol, so there is exactly one attribute which could be referenced by it. The names of the attributes are stored – once or multiple – in operators as operator attributes (see below).

5.4.3 Operators
GEA operators have attributes, from which they are influenced and/or which are affected by them, parameters, which control their functionality, and a name, by which they are called from sequences. Each operator attribute contains the #name of a GEA attribute or operator by which it is referenced. An operator is not the storage of GEA attributes! The content (storage) of attributes is located in the GEA only. For simplicity it is assumed, that it is clear from the context, if a direct or indirect (over operator attributes) reference to GEA attributes is meant. 11

In the following, a short description of each operator is given for optimization problems considered in this paper.

Table 5.1 gives a summary of all operators together with their parameters and – operator – attributes.

---

11In biology and in a more general sense, there is made a difference between the genotype and the phenotype of an individual. The genotype underlies mutation and recombination. The phenotype is build according to the information in the genotype and then evaluated by nature or an evaluation function. But in this paper, there was no need to make a difference between both.

12Referencing the same attributes by different #names should be avoided to minimize confusion.

13Operator attributes can refer to GEA attributes and GEA operators, e.g. attribute inBestAtAllRanker of BestAtAllStorer refers to an GEA operator. GEA attributes instead are exactly referred from those operator attributes which are not referring to GEA operators. It is wanted to have the possibility of accessing all GEA attributes and GEA operators from GEA operators, but only GEA operators can be organized in GEA sequences.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Parameter</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Survivor</td>
<td>x, y</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>Evaluate/Crossfit</td>
<td>x, y</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>ParentSelector/Ranked</td>
<td>x</td>
<td>inBestAtAll, inPopulation, outPopulation, inPlotModel, inTargetSelector</td>
</tr>
<tr>
<td>ChildSelector/Random</td>
<td>y</td>
<td>inBestAtAll, inPopulation, outPopulation, inPlotModel, inTargetSelector</td>
</tr>
<tr>
<td>Generational/Operations</td>
<td>x, y</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>Generation/Operations</td>
<td>x, y, label</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>Selection/Criterion</td>
<td>x</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>Initial/Random</td>
<td>y</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>Step/Increment</td>
<td>x</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
<tr>
<td>Terminate</td>
<td>y</td>
<td>Population, outPlotModel, inPlotModel</td>
</tr>
</tbody>
</table>

Table 2.1: GEA operator overview. Please seqment and please collecton2D
**Survivaler**

- **Parameter**
  - maxAge: if an individual is older than maxAge, it will be killed.
  - maxIndividuals: after selecting all individuals younger or equal maxAge, only the first maxIndividuals survive.

- **Attributes**
  - inPopulation: individuals, which can survive.
  - outPopulation: individuals, which have survived.

- **Functionality:** Plays god. The age of an individual is the difference between its birth entry and the actual value of attribute #step.

**EvaluatorSinglePerformer**

- **Parameter**
  - performerClassSelector: how to get the responding performer class (ST specific).
  - accessingSelector: how to get the responding performer object from performer class (ST specific).

- **Attributes**
  - inOutPopulation: individuals to evaluate.
  - inOutSingleEvaluations: sum of all individual evaluations so far.

- **Functionality:** EvaluatorSinglePerformer evaluates the phenotypes of all individuals from inOutPopulation one by one.\(^{12}\) It doesn’t evaluate the phenotype itself. Instead of this an object is used as evaluator for the phenotype, which is accessed by performerClassSelector and accessingSelector. At last it stores the result in the evaluation ‘memory’ of each individual.\(^{13}\) Also it sets inOutSingleEvaluations to the sum of all single evaluations so far.\(^{14}\)

**Ranker**

- **Parameter**
  - minimize: a scalar or vector of boolean values; true for mini-, false for maximization.

- **Attributes**

---

\(^{12}\) There are also Evaluators possible, which evaluate a whole population at once, e.g. a tournament with playing each individual against other individuals. In such a scenario there is no lonely evaluation possible.

\(^{13}\) It is possible to perform the evaluation directly in an Evaluator. But for the examples in this paper the performer (accessed by the parameters) is an application, which computes complex evaluation functions – image processing expression graphs, see below – and also visualizes these evaluation, if desired (see also section 5.7). This has led to this indirect way.

\(^{14}\) E.g. used for plots.
- inPopulation: individuals, which are to be ranked.
- outRankings: rankings of individuals.

- Functionality: A ranker ranks individuals by their scalar or vectorial evaluations. It has to know, whether to minimize or to maximize, this means: if minimal or maximal values are good values leading to a higher rank \(^{15}\). If evaluation is a scalar, ranking ranks the scalars of the individuals. This results in a scalar ranking. If evaluation is a vector, vectorial ranking ranks every dimension on its own as in the scalar case and combines the results as a vector. This results in a vectorial ranking. A vector of booleans for the minimize parameter is necessary, if there is the need to minimize some and maximize some other dimensions of an evaluation vector.

ParetoRanker
- Parameter
  - minimize: a scalar or vector of boolean values; true for mini-, false for maximization.
- Attributes
  - inPopulation: individuals, which are to be ranked.
  - outRankings: rankings of individuals.
  - outParetoLevels: pareto levels of individuals.

- Functionality: A pareto ranker first computes rankings like the standard ranker presented in section 5.4.3 (see above). After that it computes the pareto levels of the individuals from the rankings. This only makes sense in case of a multiobjective evaluation resulting in a vectorial ranking. In case of scalar evaluation, pareto levels are the same as the scalar rankings.

Sorter
- Parameter
  - sortCriterionSequence: vector with one or more elements of \{ #ranking, #sumOfRanks, #paretoLevel \}.
- Attributes
  - inPopulation: individuals, which are to be sorted.
  - outPopulation: individuals, which are sorted.
  - inRankings: rankings needed for sorting.
  - inParetoLevels: pareto levels needed for sorting or nil \(^{16}\).

\(^{15}\)high rank has a low number.

\(^{16}\)Smalltalk expression for an undefined object.
• Functionality: A sorter sorts individuals. The sortCriterionSequence determines which sort criterion comes first. Only if two individuals are equal in one criterion, next criterion in the sortCriterionSequence will be applied. Criterion #ranking means, a scalar ranking has to be minimized; criterion #sumOfRanks means, sum of ranks has to be minimized and criterion #paretoLevel means, pareto level has to be minimized.

BestAtAllStorer
• Parameter
  – storeLimit: size limit for the collection of best-at-all individuals.

• Attributes
  – inPopulation: individuals, which could be better than current best-at-all.
  – inBestAtAllRanker: ranker for best-at-all individuals.
  – inBestAtAllSorter: sorter for best-at-all individuals.
  – inOutBestAtAll: best-at-all individuals.

• Functionality: It is interesting to know, which individuals performed best during the whole optimization run. So there is the need to compare the best of every generation with best-at-all obtained so far and to
  – add best of them to best-at-all, if the store limit is not reached; or to
  – replace worst of best-at-all, if the new ones are better and the store limit is reached.

The best-at-all have their own ranker and sorter: they have to be ranked and sorted together with individuals from inPopulation, because ranking is a relative measure and rankings of different populations are not comparable at all.

ParentSelectorRanked
• Parameter
  – $\mu$: Number of possible parents taken from inPopulation.
  – $\eta_{max}$: needed only for #linearRanking (see below), constraint is $1 \leq \eta_{max} \leq 2$.
  – selectionMethod: #uniformRanking or #linearRanking.

• Attributes
  – inPopulation: first $\mu$ of these individuals have to be preselected.

\footnote{If store limit is less than size of inPopulation (the usual case), only ranking and sorting with storeSize best of inPopulation is necessary.}
• Functionality: ParentSelectorRanked first preselects $\mu$ parents. This means it just takes first $\mu$ from inPopulation. To ensure selection pressure which guides the optimization, inPopulation should be evaluated, ranked and sorted in advance. After preselecting it returns parents with probability $p_i, i \in [1, \ldots, \mu]$. If someone (e.g. a ChildGenerator) asks for one $^{18}$ by one of two possible selection methods:

- #uniformRanking simply selects each preselected individual with the same probability $p_i = \frac{1}{\mu}$.

- #linearRanking uses a more complicated variant. It uses

$$p_i = \frac{1}{\mu} \left( \eta_{\text{max}} - 2(\eta_{\text{max}} - 1) \frac{i - 1}{\mu - 1} \right) \quad (5.1)$$

as probability for selecting the $i$'th of the preselected individuals.

ParentSelectorRanked uses random generator attribute named #random for generating random numbers.

GeneratorImageOperations

A GeneratorImageOperations $^{19}$ generates individuals from scratch. This is necessary before using ChildGeneratorImageOperations, because that needs parents for its functionality. If only a GeneratorImageOperations is used during an optimization, the search space will be searched in a MonteCarlo manner.

• Parameter

- $\lambda$: how many individuals are to be generated.

• Attributes

- outPopulation: to which the generated child individuals will be appended to.

- inStrategyExpressionGraph: controls generating process.

• Functionality: GeneratorImageOperations generates individuals from scratch. The structure of and information within inStrategyExpressionGraph is employed as a template. Attribute inStrategyExpressionGraph SEG is traversed. For every strategy expression node SE an image processing expression IPE node will be constructed as follows:

The new IPE will be chosen out of the set of possible IPEs in current SE according to their assigned probabilities. Then its mutable parameters are randomly initialized. GeneratorImageOperations uses attribute named #random for generating random numbers and #step for setting the birth (-day) of the generated individuals.

$^{18}$This is realized over the message #next, which results in returning the next individual, if it has been sent to a ParentSelectorRanked.

$^{19}$This class is a subclass of a more general Generator, which is specialized for the optimization problems in this paper.
ChildGeneratorImageOperations

ChildGeneratorImageOperations (see also footnote 19) generates new individuals (child) dependent on already existing ones (parents).

- **Parameter**
  - $\lambda$: how many child individuals are to be generated.
  - recombinationProbability $r_{p\_global}$: global recombination probability.
  - operationMutationProbability $m_{op\_global}$: global operation mutation probability.
  - parameterMutationProbability $m_{pp\_global}$: global parameter mutation probability.

- **Attributes**
  - outPopulation: to which the generated child individuals will be appended to.
  - inStrategyExpressionGraph: controls generating process.
  - inParentSelector: where to get parents for generating children.

- **Functionality:** ChildGeneratorImageOperations generates new child individuals with information taken from existing parent individuals. For each child it gets two parent individuals from inParentSelector and performs recombination, operation and parameter mutation according to inStrategyExpressionGraph, operationMutationProbability, parameterMutationProbability and recombinationProbability in the following order:

  1. recombination: Attribute inStrategyExpressionGraph SEG and copies of both image processing expression graphs IPEGs taken from parents (their genotypes) are traversed in parallel. For each image processing expression IPE node the local recombination probability $r_{p\_local}$ will be computed by

     $$r_{p\_local} = r_{p\_global} \times r_{p\_weight}.$$ 

     If the recombination is due to a random number from random generator, the IPEs are exchanged, otherwise nothing happens. First of both recombined IPEGs is the resulting child IPEG for the following mutations.

  2. mutation of operation: Child IPEG from the foregoing step will be traversed with SEG in parallel. For each strategy expression SE node in SEG which has different possible IPEGs \(^{20}g\), the local operation mutation probability $m_{op\_local}$ will be computed by

     $$m_{op\_local} = m_{op\_global} \times m_{op\_weight}.$$ 

     If there has to be an operation mutation, the new IPE will be chosen out of the set of possible IPEGs according to their probabilities. After that the mutable parameters are randomly initialized, because a new IPE may have a different number of parameters with a semantic differing from the old one.

---

\(^{20}\)IPEs with only one possible IPEG are not able to mutate – this means change – the kind of IPE expression (this does not affect the possible mutation of parameters!}

25
3. mutation of parameters: Child IPEG from the foregoing step will be traversed with SEG in parallel. The local parameter mutation probability $m_{pp_{local}}$ will be computed by

$$m_{pp_{local}} = m_{pp_{global}} \times m_{pp_{weight}}.$$ 

In case of $m_{pp_{local}} > 0$ all mutable parameters of the current IPE are mutated with this probability. How this is done depends on the kind of parameter and is explained in section 4.4 for every used IPE.

All generated child IPEs will be appended to outPopulation. 21 Child-GeneratorImageOperations uses attribute named #random for generating random numbers and #step for setting the birth (-day) of the generated individuals.

**StepIncrementer**

A StepIncrementer increments an attribute value by 1. Here it is used for attribute #step, which is used by a Survivaler (see 5.4.3).

- Parameter
  - no parameters

- Attributes
  - inOutValue: to be incremented.

- Functionality: Increments inOutValue by 1. Initializes it to 0.

**Terminator**

Terminator is used to terminate optimizations in this paper, if the best-at-all individual (from best-at-all individuals) has evaluation scalar 0 or vectorial (0, \ldots, 0); the optimum is reached then!

- Parameter
  - terminationCriteriumBCString: the to be compiled termination criterion.

- Attributes
  - no attribute

- Functionality: Terminator compiles terminationCriteriumBCString to get a ST terminationCriteriumBC BlockClosure. This BlockClosure evaluated with GEA has to return ST true if termination is intended, false otherwise. Terminator returns #terminate, if termination criterium returns ST true. If Terminator is included in a sequence, GEA stops this sequence, if Terminator returns #terminate.

21To avoid an unlimited growing of outPopulation, a Survivaler is needed to 'kill' some individuals.
InitializerRandom

- Parameter
  - fixed: if true, initialize with random seed, if false, reuse last start seed.
  - generator: random generator.
  - startSeed: seed of random generator.

- Attributes
  - no attribute

- Functionality: Initializes attribute named #random with random generator.

ObjectFromFile

- Parameter
  - fileNameString: file name of file with ST string.

- Attributes
  - outObject: where to store compiled ST string.

- Functionality: ObjectFromFile reads an ST string from a file, compiles it to a ST object and stores this object as attribute outObject. This is used to read strategy expression graphs (SEGs) from files, which are stored by an editing tool \(^22\).

PlotterSequential and PlotterCollection2D

PlotterSequential and PlotterCollection2D are operators for generating different kinds of plots. Their parameters and attributes are not described in detail here, because this would not serve a better understanding of the topics in this paper.

5.4.4 Sequences

Sequences are collections of operators and/or sequences. They are the heart of a running GEA. They allow an automatic batch-like applying of operators and also of other sequences(!) to a GEA. So it is easy to implement nested loops on an operator level.

In the following a brief description of the used sequences in this paper is given together with their relationships to existing evolutionary algorithms.

In this paper attribute #population is used by all operators which work with populations of individuals. So all operators work with the same population.

\(^22\)E.g. StrategyExpressionGraphBrowser.
**Used Sequences**

**_RSS:_** #(_Ranker Sorter BestAtAllStorer_).

This sequence ranks and sorts a population. At last the best-at-all individuals will be stored. It is used as sub-sequence by MonteCarlo and MueLambda sequences.

Operators:

- **Ranker** ranks #population.
- **Sorter** sorts #population.
- **BestAtAllStorer** stores best-at-all individuals from #population.

**MonteCarlo:** #(_StepIncrementer Survivaler GeneratorImageOperations EvaluatorSinglePerformer _RSS _display Terminator_).

This sequence makes a MonteCarlo like optimization step. It can be applied to an freshly initialized GEA, when no individuals exist. Therefore it is suited to *initialize* a #population, which can be used by a MueLambda sequence then.

Operators and sub-sequences:

- **StepIncrementer** increases the optimization step.
- **Survivaler** It 'kills' those individuals from #population, which are too old or not good enough.
- **Generator** generates individuals from scratch.
- **EvaluatorSinglePerformer** evaluates #population.
- **_RSS_** sub-sequence will be performed.
- **_display_** sequence displays something as plots etc.; it is not further described here.
- **Terminator** terminates optimization, if a certain criterium is fulfilled.

**MueLambda:** #(_StepIncrementer Survivaler ParentSelectorRanked ChildGeneratorImageOperations EvaluatorSinglePerformer _RSS _display Terminator_).

This sequence supplies similar functionality as MonteCarlo Sequence, but because it needs existing individuals, it can only be applied after once applying MonteCarlo. Operator Generator from MonteCarlo sequence is replaced by the following operators:

- **ParentSelector** preselects first $\mu$ individuals and supplies parents to the ChildGenerator on request. To ensure a selection pressure, the #population has to be sorted due to a quality measure! This is realized by the _RSS sequence_ in the MonteCarlo (before first call of this sequence) or in this sequence.

- **ChildGenerator** takes parents from ParentSelector and generates child individuals.

**opt:** #(_MonteCarlo MueLambda_).

This sequence makes a full optimization and is used for all problems considered in this paper.

Operators and sub-sequences:
MonteCarlo initializes the #population in a MonteCarlo-like manner.

499 MueLambda performs 499 times the sequence MueLambda. This is a practical feature, which also is available for operators in sequences. 499 seems to be odd, but together with the MonteCarlo-like initialization step there are totally 300 optimization steps and a corresponding number of single evaluations (assumed same λ for all – individual – generators).

5.5 Relationships with other algorithms

This paper is written with an undogmatic view to different ‘schools’ of evolutionary computation. Why to be fixed on one point of view? Concepts used in this paper are wildly taken from the main branches Evolutionsstrategien (ES) [Rec94, Sch77], genetic algorithms (GA) [Go89] and genetic programming (GP) [Koz92].

5.5.1 Rechenberg’s Evolutionsstrategien

Rechenberg’s Evolutionsstrategien (ES) in the classical (μ,λ) and (μ + λ) formulations could be realized as follows: Use opt sequence as described before.

Survivaler gets μ maxIndividuals and a maxAge of 1 for (μ,λ) or maxAge of positive infinity (or a very large number) for (μ + λ). So only μ individuals remain from the foregoing step (maxAge of 1) or all steps so far (maxAge very huge, so-called 'elitist strategy'). Other values for maxAge give a soft mixture of (μ,λ) and (μ + λ) strategies.

ParentSelectorRanked preselects the μ individuals (its attribute has the same name) and uses #uniformRanking as selection method.

That's all!

Operator ChildGenerator appends new childs to #population. The parents will be removed by the survivoral, if there is the (μ,λ) strategy, because after increasing the optimization #step by StepIncrementer they are too old then. If there is the (μ + λ) strategy they are only removed, if they perform too bad, not considering their age. A Rechenberg-like ChildGenerator has to take one (or more for newer ES variants) parent(s) and the genotype is a vector of real numbers which are to be mutated by Gaussian distributed random numbers. This is a task of coding and mutating, which depends on the kind of the optimization problem.

5.5.2 Genetic Algorithms

To realize classical Genetic Algorithms (GAs), it would be necessary to enhance the ParentSelector operator. There a 'fitness' dependent selection is necessary, which differs from ranking based selection. This efforts to look directly onto the evaluations of the individuals, without depending solely on the abstraction of ranking, which eliminates absolute distances between different individual evaluations.

---

20 Not described in this paper.
The coding is problem dependent, too; but e.g. bit wise mutation is used for the Binarization IPE (see section 4.4.2) in this paper.

5.5.3 Genetic Programming

Working with expression graphs is influenced by concepts of genetic programming. But the work presented here underlies a static graph concept; the graph does not change its form \(^{24}\) and does not grow or shrink. Otherwise here are given possibilities to fine tune mutation and recombination probabilities by weights in the nodes, and not only global or operation dependent probabilities.

5.6 Why ranking?

In this paper there is used ranking based selection, which are the reasons?

- Computing a scalar or vectorial ranking of evaluations, and sorting and selecting individuals according to these rankings instead of directly sorting them according to their evaluations, has some advantages: \(^{25}\)
  - If to mini- or maximize has to be stated only for the Ranker (to compute the ranking) and not for the sorter, too;
  - hence sorters always have to minimize the ranking and need not to know which is/are the optimization direction(s) in the scalar/vectorial case.
  - Ranking gives an abstract view onto the evaluation space. No rescaling or weighting to deal with different scaled dimensions of the evaluation space is necessary, as it would be often for fitness dependent selection.

5.7 User interface

An impression of the user interface gives the screenshot in figure 5.3.

\(^{24}\)But its contexts: Nodes can be changed to another kind of expression [IPE] with the same arity.
\(^{25}\)To visualize the optimization success, plotting evaluations instead of their rankings is necessary, of course.
Figure 5.3: Screenshot of Smalltalk user interface with GEA and other tools. The top left VisualWorks window shows steps in a sequence. By the GEA window below a GEA will be controlled, it contains operators, attributes and sequences, which can be manipulated. Two time and one space plots are in the middle. At the top right there is a tool for visualizing an JPEG and viewing source, target and result image; it is possible to visualize temporary image processing results while computing an JPEG, too. With an IPSOperator seen below the space plot, image processing operations can be applied on images in IPSImaginators, two of them are right bottom. The window labeled GenericEA is a so called ST 'Inspector' for inspecting variables of a GEA.
Chapter 6

Working with GEA

6.1 Overview

In this section some optimization problems out of the field of image processing are described, going from less to higher complexity. GEA together with problem specific expression graphs are used to solve these tasks.

The optimization goal with the used evaluation function is explained in section 6.2 first. In section 6.3 parameters are given which are the same for all optimization runs. The plots and other results of the optimization runs are described in section 6.4.

In the following three sections these problems are shown: For a simple problem shown in section 6.5 parameters of a binarization operation have to be found. For a not so simple Problem in section 6.6 a morphological operation and a mask has to be found. The real problem in section 6.7 is about finding faults in an image; two morphological operations and two masks have to be optimized there.

6.2 Optimization goal

In each of the following problems there are a source and a target image. The source image serves as input for the image processing expression graph (IPEG) to be optimized. The target image is a binary image, which is constructed by marking some areas in the source image. The marked pixels are treated as foreground and the others are treated as background of the target image: this means, marked pixels get the value 1, unmarked pixels value 0 in the target image.

The optimization goal is to get an image processing expression graph (IPEG), which transforms the source image into the given target image most exactly. The marked area given by the target image has to be filtered out by the desired IPEG. Because of starting from a gray valued source image yielding to a binary target image, there is the constraint to have a binarization step in the image processing expression graph.
6.2.1 Evaluation function

The last expression in the image processing expression chain returns always a binarized result image, which is comparable to the target image. The foreground is composed of the pixel, which are marked in the target image, the background are those pixels which are not marked (value 0). The evaluation function computes the difference between result and target image. \(^1\) As result it returns a 3-dimensional vector \((f_{\text{wrong}}, b_{\text{wrong}}, |f_{\text{wrong}} - b_{\text{wrong}}|)\):

- \(f_{\text{wrong}}\) says how many foreground pixel are wrong, this means pixel which should be set (value 1), but are not;

- \(b_{\text{wrong}}\) says how many background pixel are wrong, this means pixel which should not be set, but are set;

- \(|f_{\text{wrong}} - b_{\text{wrong}}|\) is the absolute difference between both.

All of these measures have to be minimized, with a – not necessarily reachable – optimum at \((0,0,0)\).

By measuring \(|f_{\text{wrong}} - b_{\text{wrong}}|\), it is achieved is reached preferring \(f_{\text{wrong}}\) and \(b_{\text{wrong}}\) at similar sizes (in opposite to strongly different sizes). Otherwise a total black or white result image, which always reaches the optimum of \(f_{\text{wrong}}\) or \(b_{\text{wrong}}\) (one of both is zero then), evaluates comparable well to an average good image, if we are using the sum of ranks as sorting criterium! But images which are totally black or white, are obviously no good solutions.

Optimizing all three measures of the evaluation function means, that there is a multiobjective optimization.

6.3 Unique parameters

It follows a description of unique operator parameters, which are used with the same values for all problems.

**Survivaler** maxIndividuals: 10.

**Ranker** minimize: true.

**Sorter** sortCriterionSequence: #sumOfRanks.

**BestAtAllStorer** storeLimit: 10.

**ParentSelectorRanked** \(\mu\): 10, selectionMethod: #uniformRanking.

**GeneratorImageOperations, ChildGeneratorImageOperations** \(\lambda\): 50. \(^2\)

**ChildGeneratorImageOperations** parameterMutationProbability: 1/5, recombinationProbability: 0.

Parameters with changed values are described with the respective problem.

---

1 This is realized by combining both images by different boolean rules and counting pixels in the resulting images. This has the effect of comparing every pixel value.

2 Together with used sequence opt this leads to a maximal number of 25000 evaluations.
6.4 Visualization

For every problem there are shown figures of the

- source image together with target image,
- best result image,
- best image processing expression graph IPEG,
- following plots 8:

**time plot** \( f_{\text{wrong}} \) of best-at-all (not full black line) individual together with best-of-population individual (full black line) for each optimization step,

**time plot** \( b_{\text{wrong}} \) of best-at-all (not full black line) individual together with best-of-population (full black line) individual for each optimization step,

**space plot** of first two dimensions (measures) \( f_{\text{wrong}} \) and \( b_{\text{wrong}} \) of evaluations of all generated individuals, the horizontal axis corresponding to \( f_{\text{wrong}} \) (foreground wrong), the vertical axis to \( b_{\text{wrong}} \) (background wrong). 4 The bottom left corner corresponds to the optimum value for these two dimensions. If it is reachable or not depends on the kind of problem.

Both time plots together only show the first two dimensions of the evaluation function. This has to be taken into account, if there are plotted evaluation values for a best-at-all individual, which seem to be worse than values of a best-at-all individual before or values of a best-of-pop individual.

The strategy expression graph (SEG) parameter is explained in the text referring to the numbering of best found – visualized – IPEG, because it has the same structure.

---

8All time and space plots are plotted without coordinate axes, they would rather disturb the figure than illuminate the results. The numbers in the plots have the following meaning: the left number denotes the lowest and the right number the highest horizontal value, the bottom number denotes the lowest and the top number the highest vertical value.

4Some individuals may be covered by other ones.
Figure 6.1: Simple problem source image. Gray bar consisting of gray levels \( \{0, \ldots, 255\} \). A white (value 255) stripe at the bottom (not visible) corresponds to the black (value 0) stripe at the top of the image.

Figure 6.2: Simple problem target image. Also best result image. Goal is to filter gray levels laying in interval \([100 \ldots 120]\). It seems to be larger than the source image, because the white stripe at the bottom of the source image is black (not marked) here.

Figure 6.3: Simple problem IPEG.

### 6.5 Simple problem

#### 6.5.1 Goal

The source image in figure 6.1 shows a gray bar over all gray values for the used gray scale range. In the target image, all pixels which corresponds to gray values from 100 to 120 in the source image are marked. The goal here is to find the correct parameters from and to of the \( \text{Binarize}_{\text{from}, \text{to}}(p) \) IPE.

#### 6.5.2 Parameters

See also section 6.3 for unique parameters.

- **Survivaler maxAge**: 3.
- **ChildGenerator operationMutationProbability**: senseless.
- **StrategyExpressionGraph**

  \[ \text{SE 1: IPE Binarize}_{\text{from}, \text{to}}(p). \quad m_{pp\_weight} = 1, r_{pp\_weight} \text{ is irrelevant.} \]

  \[ \text{SE 1.1: IPE source image.} \]

#### 6.5.3 Result

The resulting IPEs were

- **IPE 1**: \( \text{Binarize}_{100,120} \).
- **IPE 1.1**: Source image.
GEA has found the searched optimum $(\text{from} = 100, \text{to} = 120)$. The number of evaluations needed strongly depend on the random generator.

### 6.5.4 Interpretation

**Time plots**

There is a fast convergence near the optimum ($f_{\text{wrong}} = 0$ and $b_{\text{wrong}} = 0$). But then it takes a long time to complete the search.

**Space plot**

The white area inside the space plot comes from the structure of the problem: If $f_{\text{wrong}}$ is lying between its minimal (0) or maximal (903) value, at least one of the parameters from and to of $Binarize$ lays in the interval $[100, \ldots, 120]$. If $\text{from} < \text{to}$ at least one case of

1. background pixels with values below 100 are correct, or
2. background pixels with values greater than 120 are correct,

happens. If $\text{from} > \text{to}$ at least one case of

1. background pixels with values below 100 are wrong, or
2. background pixels with values greater than 120 are wrong,

happens. So there is a range of values which cannot occur.

The lines of individuals arise from the discrete structure of the problem: If one of $Binarize$ parameters $\text{from}$, $\text{to}$ changes by 1, $f_{\text{wrong}}$ or $b_{\text{wrong}}$ change at
least by the number of pixels in one vertical line in the middle area of the gray bar (which corresponds to one gray level)\textsuperscript{5}.

It can be seen, that there are more generated individuals in direction to the optimal evaluation than by a Monte-Carlo-like search.

\textsuperscript{5} Top (black) and bottom (white) areas of gray bar include much more pixels.
6.6 Not so simple problem

6.6.1 Goal

Goal in this task (see figure 6.7 and 6.8) is to find a morphological operation together with a mask, which results in removing one diagonal line without affecting the other one. The image is enlarged for a better visibility.

6.6.2 Parameters

See also section 6.3 for unique parameters.

Survivaler maxAge: 1.

ChildGenerator operationMutationProbability 0.1.

StrategyExpressionGraph

SE 1: IPEs $\text{Morph}_E$ and $\text{Morph}_O$ (eroding and opening) – because the foreground of the image obviously has to be smaller in the result image –, each with probability 1/2. $m_{op\_weight} = 1, m_{pp\_weight} = 0, r_{p\_weight} = 0$.

SE 1.1: IPE source image.

SE 1.2: IPE $\text{Mask}$: size: 6, xLimit: 2, yLimit: 2. $m_{pp\_weight} = 1$, $r_{p\_weight} = 1$.

6.6.3 Result

The resulting IPEs were

IPE 1: $\text{Morph}_O$ (opening).
Figure 6.9: Not so simple problem IPEG.

Figure 6.10: Not so simple problem time plot; vertical $f_{g \text{ wrong}}$, horizontal evaluations.

Figure 6.11: Not so simple problem time plot; vertical $b_{g \text{ wrong}}$, horizontal evaluations.
IPE 1.1: Source image.

IPE 1.2: Mask $(-2, -2), (-1, -1), (1, 1), (2, 2)$.

GEA has found the optimal mask $(-2, -2), (-1, -1), (1, 1), (2, 2)$ \(^6\) (see figure 6.9) and the correct opening operation very fast – in four different runs ranging from 700 to 1300 evaluations. But there is the need to choose the right parameters! E.g. with Survivor maxAge of 3 GEA performs much worse and it easily runs into a local optimum like e.g. $((-1, -2), (0, -1), (1, 0), (2, 1))$.

6.6.4 Interpretation

Time plots

Optimal evaluation has been reached very fast, best-of-population does not differ from best-at-all individual after half of optimization run (evaluations).

Space plot

The gap between the optimal and second best found solution is big, but this has not disturbed the algorithm.

\(^6\)Mask $((-1, -1), (1, 1))$ is not optimal.

![Figure 6.12: Not so simple problem space plot; horizontal $g\_{wrong}$, vertical $g\_{wrong'}$.](image-url)

40
6.7 Fault problem

6.7.1 Goal

The problem is to detect areas with scratches in a polished aluminium surface. To minimize reflections, the illumination has to be very low and diffuse, so there comes a very dark image (see figure 6.13) from the camera, which is also shown inverted (see figure 6.15). The goal here (see figure 6.14) is to find a sequence of two morphological operations together with their mask, which results in marking the areas as marked in the target image by the user.

6.7.2 Parameters

See also section 6.3 for unique parameters.

Survivaler maxAge 1.

ChildGenerator operationMutationProbability 0.1.
Figure 6.17: Fault problem IPEG.

Figure 6.18: Fault problem time plot; vertical $f_{g\text{wrong}}$, horizontal evaluations.

Figure 6.19: Fault problem time plot; vertical $b_{g\text{wrong}}$, horizontal evaluations.

**StrategyExpressionGraph**

SE 1: IPE $\text{Binarize}_{\text{from, to}}(p)$. $m_{pp\text{-weight}} = 1, r_{p\text{-weight}} = 0$.

SE 1.1: All morphology IPEs $\text{Morph}_0$, $\text{Morph}_2$, $\text{Morph}_\oplus$ and $\text{Morph}_\ominus$ (dilating, eroding, closing and opening), each with probability $1/4$. $m_{op\text{-weight}} = 1, m_{pp\text{-weight}} = 0, r_{p\text{-weight}} = 0$.

SE 1.1.1: Same as SE 1.1.

SE 1.1.2: IPE $\text{Mask}$. size: 6, xLimit: 2, yLimit: 2. $m_{pp\text{-weight}} = 1, r_{p\text{-weight}} = 1$.

SE 1.1.1.1: Source image.

SE 1.1.1.2: Same as SE 1.1.2.

6.7.3 Result

The resulting IPEs were

IPE 1: $\text{Binarize}_{77,247}$.

IPE 1.1: $\text{Morph}_2$ (dilation).
Figure 6.20: Fault problem space plot; horizontal $f_{g_{\text{wrong}}}$, vertical $b_{g_{\text{wrong}}}$.

Figure 6.21: Fault problem space plot, corner detail with equal dimensions; horizontal $f_{g_{\text{wrong}}}$, vertical $b_{g_{\text{wrong}}}$.

IPE 1.1.1: same as IPE 1.1.

IPE 1.1.2: Mask $\{(-2, -2), (-2, 2), (1, 0), (2, -1), (2, 1), (1, -2)\}$.

IPE 1.1.1.1: Source image.

IPE 1.1.1.2: Mask $\{(1, 1), (-1, -1), (1, -2), (-2, 1), (-2, 0), (-2, -2)\}$.

Both masks are visualized in figure 6.17. Note, that there is no redundancy in the masks, all entries are different.

The best solution found was the evaluation vector $(493, 516, 23)$, this means 493 foreground and 516 background pixel remained wrong after 25000 evaluations (see figures 6.20, 6.19 and 6.19). The third entry is the absolute difference between both. The best result image is shown in figure 6.15.

6.7.4 Interpretation

Time plots

The time plots show a very chaotic behavior for the evaluations in dimensions $f_{g_{\text{wrong}}}$ and $b_{g_{\text{wrong}}}$ of the best-of-pop individual. The best-at-all individuals correspond to the mostly straight lines parallel to the horizontal. So it seems to be difficult to improve the evaluation after the first optimization phase.
Space plots

The comparison of the space plot in figure 6.20 with the space plot in figure 6.22 shows very clearly how the selection pressure has influenced the movement of the algorithm within the evaluation space. The space plot in figure 6.22 is generated by a Monte-Carlo search and shows the distribution of individual evaluations for a blind search without any selection pressure. The evaluation space has a certain structure, some areas cannot be hit at all and others are hit with a higher or lower probability. It is very instructive to see such a plot to get an imagination of the structure of the evaluation space. The used algorithm produces selection pressure, which can be seen as a cumulation of evaluations in direction to the theoretic optimum - bottomleft corner - (see figure 6.20) in comparison to the Monte-Carlo case (see figure 6.22). It is instructive to get an imagination of the evaluation space for a ‘stupid’ algorithm and to compare it with a more sophisticated one. With the used algorithm most individuals are generated near the pareto border in direction to the optimal evaluation. The pareto border shows a limitation for concurrently reaching optimal (0) values of $f_g_{wrong}$ and $g_{wrong}$ and the not plotted value $|f_g_{wrong} - g_{wrong}|$!

The comparison of the corner detail space plot in figure 6.21 with the corner detail space plot in figure 6.23 shows how the kind of the evaluation function has influenced the movement of the algorithm within the evaluation space. The corner is magnified in the vertical dimension $g_{wrong}$, which is scaled equal to $f_g_{wrong}$. Both plots rely on the used algorithm and differ only in the evaluation function used. If the third measure $|f_g_{wrong} - g_{wrong}|$ is abandoned, there can be seen an asymmetrical generation of individuals with respect to bottom (background wrong) and left (foreground wrong) side. This has led to bad result images. But with measure $|f_g_{wrong} - g_{wrong}|$ there is a magnetism to the diagonal between left and bottom side and the algorithm tries to minimize the number of wrong pixels of fore- and background with equal strength.

Figure 6.22: For comparison: fault problem space plot generated by a MonteCarlo search (no selection pressure); horizontal $f_g_{wrong}$, vertical $g_{wrong}$.

Figure 6.23: For comparison: fault problem space plot, corner detail with equal dimensions; generated by an evaluation function without measure $|f_g_{wrong} - g_{wrong}|$; horizontal $f_g_{wrong}$, vertical $g_{wrong}$.
Chapter 7

Summary

A program runs, this paper has shown some results. This could be used as ground for further research.

7.1 Further work

7.1.1 To do GEA

- Implement as many wide spreaded EAs as possible by GEA sequences, operators and attributes.
- Compare different EAs systematically by well known test functions.
- Realize multi population EAs
  - inside a GEA with more than one population attribute, or
  - by populations of GEAs.

7.1.2 To do IPEGs

- More IPEs are necessary.
- Implement IPEGs with the feature to (re-)use a temporary result by more than one following IPE.

7.1.3 To do Application

Use this work to solve image processing tasks taken from the real world! How to choose

- SEG (which selects IPEs), and

- $r_p\_{\text{global}}, r_p\_{\text{weight}}, m_{op\_{\text{global}}}, m_{op\_{\text{weight}}}, m_{pp\_{\text{global}}} \text{ and } m_{pp\_{\text{weight}}}$

for different problems?
List of Figures

4.1 Mask examples. The center point has the coordinates $(0,0)$ and is marked as gray circle, mask points are marked with crosses. The left mask $((-1,-1),(0,0),(1,1))$ is point symmetrical and contains its center point, the right mask $((-1,-1),(1,1),(2,2))$ is asymmetrical and does not contain its center point. 

5.1 Pseudo code for an evolutionary algorithm, taken from [HB98].  
5.2 Principal design of GEA. Operators $O_i$ change attributes $A_i$ and use their values. Sequences $S_i$ are orders of operators and other sequences. Right sequence $S_2$ consists of operators $O_3$ and $O_2$ and left sequence $S_1$ as a sub-sequence.  
5.3 Screenshot of Smalltalk user interface with GEA and other tools. The top left VisualWorks window shows steps in a sequence. By the GEA window below a GEA will be controlled, it contains operators, attributes and sequences, which can be manipulated. Two time and one space plots are in the middle. At the top right there is a tool for visualizing an IPEG and viewing source, target and result image; it is possible to visualize temporary image processing results while computing an IPEG, too. With an IPSOperator seen below the space plot, image processing operations can be applied on images in IPSImaginators, two of them are right bottom. The window labeled GenericEA is a so called ST 'Inspector' for inspecting variables of a GEA.  

6.1 Simple problem source image. Gray bar consisting of gray levels \{0,...,255\}. A white (value 255) stripe at the bottom (not visible) corresponds to the black (value 0) stripe at the top of the image.  
6.2 Simple problem target image. Also best result image. Goal is to filter gray levels laying in interval [100...120]. It seems to be larger than the source image, because the white stripe at the bottom of the source image is black (not marked) here.  
6.3 Simple problem IPEG.  
6.4 Simple problem time plot; vertical $f_g$ wrong, horizontal evaluations.  
6.5 Simple problem time plot; vertical $b_g$ wrong, horizontal evaluations.  
6.6 Simple problem space plot; horizontal $f_g$ wrong, vertical $b_g$ wrong.
6.7 Not so simple problem source image. It is a binary image with a cross, which is shown magnified: one of the small squares corresponds to one pixel. .......................... 38
6.8 Not so simple problem target image. Also best result image. One diagonal of the cross (from top-left to bottom-right) has to be left after image processing; shown magnified. .................. 38
6.9 Not so simple problem JPEG. .......................... 39
6.10 Not so simple problem time plot; vertical $f_{g\text{ wrong}}$, horizontal evaluations. .......................... 39
6.11 Not so simple problem time plot; vertical $b_{g\text{ wrong}}$, horizontal evaluations. .......................... 39
6.12 Not so simple problem space plot; horizontal $f_{g\text{ wrong}}$, vertical $b_{g\text{ wrong}}$. .......................... 40
6.13 Fault problem source image. .......................... 41
6.14 Fault problem target image. .......................... 41
6.15 Fault problem inverted source image. Shown to elucidate the erroneous areas. .......................... 41
6.16 Fault problem result image. Best result image reached after 25000 evaluations. .......................... 41
6.17 Fault problem JPEG. .......................... 42
6.18 Fault problem time plot; vertical $f_{g\text{ wrong}}$, horizontal evaluations. .......................... 42
6.19 Fault problem time plot; vertical $b_{g\text{ wrong}}$, horizontal evaluations. .......................... 42
6.20 Fault problem space plot; horizontal $f_{g\text{ wrong}}$, vertical $b_{g\text{ wrong}}$. .......................... 43
6.21 Fault problem space plot, corner detail with equal dimensions; horizontal $f_{g\text{ wrong}}$, vertical $b_{g\text{ wrong}}$. .......................... 43
6.22 For comparison: fault problem space plot generated by a Monte-Carlo search (no selection pressure); horizontal $f_{g\text{ wrong}}$, vertical $b_{g\text{ wrong}}$. .......................... 44
6.23 For comparison: fault problem space plot, corner detail with equal dimensions; generated by an evaluation function without measure $[f_{g\text{ wrong}} - b_{g\text{ wrong}}]$; horizontal $f_{g\text{ wrong}}$, vertical $b_{g\text{ wrong}}$. .......................... 44
List of Tables

5.1 GEA operator overview. PlotterSequential and PlotterCollection2D are displaying operators, which generate graphics. . . . . . 20
Bibliography


the authors as technical report from Fraunhofer Institute for Production Systems and Design Technology (IPK), Pascalstr. 8-9, D-10587 Berlin.

