UNIVERSIDADE DO ALGARVE

Affine Image Registration Using Genetic Algorithms and Evolutionary Strategies

Mosab Bazargani

Mastrado em Engenharia Informática

2012

UNIVERSIDADE DO ALGARVE

Affine Image Registration Using Genetic Algorithms and Evolutionary Strategies

Mosab Bazargani

Tese orientada por
Fernando Miguel Pais de Graça Lobo

Thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
Universidade do Algarve

2012

Abstract

This thesis investigates the application of evolutionary algorithms to align two or more 2-D images by means of image registration. The proposed search strategy is a transformation parameters-based approach involving the affine transform. A noisy objective function is proposed and tested using two well-known evolutionary algorithms (EAs), the genetic algorithm (GA) as well as the evolutionary strategies (ES) that are suitable for this particular ill-posed problem. In contrast with GA, which was originally designed to work on binary representation, ES was originally developed to work in continuous search spaces. Surprisingly, results of the proposed real coded genetic algorithm are far superior when compared to results obtained from evolutionary strategies' framework for the problem at hand. The real coded GA uses Simulated Binary Crossover (SBX), a parent-centric recombination operator that has shown to deliver a good performance in many optimization problems in the continuous domain. In addition, a new technique for matching points, between a warped and static images by using a randomized ordering when visiting the points during the matching procedure, is proposed. This new technique makes the evaluation of the objective function somewhat noisy, but GAs and other population-based search algorithms have been shown to cope well with noisy fitness evaluations. The results obtained from GA formulation are competitive to those obtained by the state-of-the-art classical methods in image registration, confirming the usefulness of the proposed noisy objective function and the suitability of SBX as a recombination operator for this type of problem.

Keywords: Evolutionary algorithm (EA), image registration (IR), affine transform, point-pattern matching, genetic algorithm (GA), evolutionary strategies (ES), simulated binary crossover (SBX).

I dedicate this work to the most beautiful expression of my life;

"my mother"

Acknowledgements

I would like to thank all those collaborators who helped me in the process that now comes to completion with this thesis.

First and foremost, I want to thank my supervisor Fernando G. Lobo. Without his support and patience, this thesis would not have been possible. I am indebted to him for his help and support, and will always appreciate the time and consideration that he devoted to my work.

Kind gratitude to my beloved family, my mother who passed away in my embrace and on her peaceful face was pictured the whole world's love, my father and two sisters who taught me to be myself. They taught me to never give up in achieving my ultimate goals. They are the spiritual support of my life. A very special thanks goes to Houman Samim, my brother in law, who has always been there for me.

I am very thankful to Ali Mollahosseini, who I first met two years ago, and since then until he left to the USA we were working in the same laboratory. We interacted intensely on a daily basis. I would also like to thank him for all the scientific and social discussions we had.

I am also very grateful to António dos Anjos and his family, especially his father whom I take my hat off to. Some of the ideas presented in this thesis resulted from discussions António and I had. He always tried to encourage me to learn more and more in computer science area.

Kind gratitude to Ana Cristina Pires and Gil Guilherme, my Salsa teachers, who showed me a new path in my life. I would also like to thank all my salsa friends, for their understanding, kindness and companionship, for all the great times we shared.

How can I forget Denise Candeias for all her help at the final stage of my writing.

Last but not the least, I would like to thank all my colleagues and friends, for being there always with me whenever needed.

Contents

List of Figures

List of Algorithms

List of Tables

The aim of science is not to open the door to infinite wisdom, but to set a limit to infinite error.

Bertolt Brecht

Chapter 1

Introduction

1.1 Motivation

> *"If God had wanted to put everything into the universe from the beginning, he would have created a universe without change, without organisms and evolution, and without man and man's experience of change. But he seems to have thought that a live universe with events unexpected even by himself would be more interesting than a dead one."*

In the above mentioned quote, Karl Popper, one of the greatest philosophers of science of the 20^{th} century, brilliantly explains the reason of the existence of evolution of everything surrounding us. In our world today, there are problems with characteristics which are similar to the adaptation problems encountered in Nature which have been solved through evolution. Such problems occur when the task is to find the best (or a reasonably good solution) out of many possible solutions to a given problem. These kind of problems have been reported in a variety of fields such as computer science, management science, industrial engineering, biology, and many others.

Since these kind of problems are solved in Nature, it is quite rational to seek a solution for them which is inspired by Nature. Evolution provides inspiration to compute solutions

to problems that have previously appeared intractable.

All evolutionary systems process through generations, where a small change at one stage can result in large differences at a later stage. This feature is called *butterfly effect*. The butterfly effect is a common trope in fiction when presenting scenarios involving time travel and with hypotheses where one storyline diverges at the moment of a seemingly minor event resulting in two significantly different outcomes.

According to this theory, we would need to germinate another planet and wait several millions of years to study how life could possibly be over there after solving one problem. Since it would be impossible and also irrational we could use computer science to simulate such a world through evolutionary algorithms.

To assess how evolutionary algorithms can cope with real world problems, especially ill-posed problems, in this thesis the image registration problem is optimized by means of evolutionary algorithms.

Image registration (IR) is the process of finding the transformation that aligns one image to the other image. This is a key problem in computer vision encountered in many areas, e.g., medical image analysis, pattern recognition, face tracking, handwriting recognition, astro- and geophysics, and analyzing images from satellites. Image registration can be defined in a simple language with only a few words: given a *static* and a *deformed* image, find a suitable *transformation* such that the transformed deformed image becomes *similar* to the static image. However, it is easy to state the problem but hard to solve it. The main reason comes from the fact that the problem is *ill-posed*. Small changes of the input images can cause completely different registration results. Furthermore, the solution may not be unique. Suppose we have to register the deformed image to the static image that are depicted in Fig. 1.1. For the sake of simplicity, only rigid transformations are allowed, i.e., rotation and transformations. Several solutions can be immediately discovered, e.g., a pure translation, a rotation of 45 degrees, a rotation of 90 degrees followed by a translation, and so on.

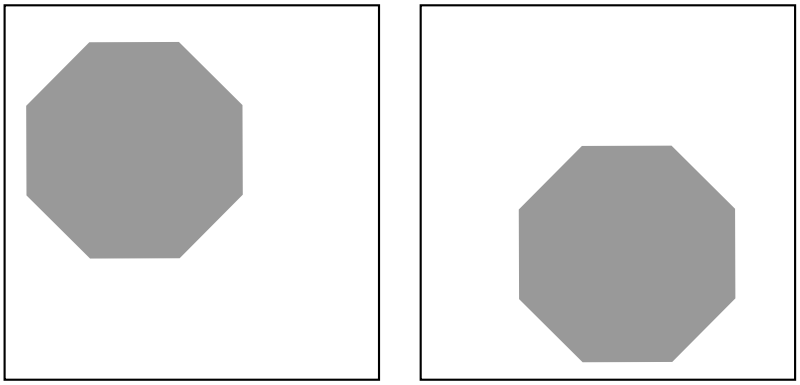


Figure 1.1: Two octagons; left: static image; right: deformed image.

In this thesis, the goal is to find the best mapping function, also called transform, that warps a *Deformed* image (D) in the direction of a *Static* image (S), based on the images' features (e.g. point positions).

1.2 Objectives

The thesis has the following main objectives:

- Application of evolutionary algorithms to image registration.
- Design and program a framework for evolutionary strategies in C++.
- Study a new noisy objective function using a real coded representation for image registration by means of genetic algorithm and evolutionary strategies.

1.3 Contributions

The main contributions of this thesis are:

- Propose a new technique for matching points between a warped and static images by using a randomized ordering when visiting the points during the matching procedure [7].

- Application of a real coded GA to align two or more 2-D images by means of image registration. The proposed search strategy is a transformation parameters-based approach involving affine transformation. The real coded GA uses simulated binary crossover (SBX). This work has been accepted as a poster publication at ACM GECCO 2012, one of the most prestigious conferences in the EC field [7].

- Design and program a generic framework for evolutionary strategies in C++. This framework is fully object oriented and includes the classical *1/5 success rule* and the self-adaptive *uncorrelated mutation strategies*. Moreover, *discrete*, *intermediate*, and *simulated binary crossover (SBX)* as recombination operators and *bit-flip*, *geometrical*, *Gaussian*, and *polynomial* as mutation operators are included. Implemented ingredients of the evolutionary strategies (ES) are elaborated in Appendix A.

- Investigate the effect of simultaneous usage of SBX and real-valued mutation operators on the applications' behavior (Sect. 5.1).

1.4 Organization of the Thesis

The thesis is composed of six chapters. Chapter 1 starts off by explaining the motivation of this work, and continues clarifying the main objectives and contributions of the thesis. It is finalized by the details of organization of the thesis.

Chapter 2 presents a brief overview of the natural evolution and the history of evolutionary computation. Afterwards, it introduces the evolutionary algorithms and its key ingredients as search methods inspired by natural selection and genetics. It continues by addressing genetic algorithms and its major representations. This chapter ends by reviewing the basic procedures of standard evolutionary strategies and its terminology. Moreover, the operators used in this work are explained in this chapter.

Chapter 3 reviews the basics of image registration. Thereupon, it introduces two

classes, rigid and non-rigid, geometric transforms. Affine and polynomial transforms and thin plate splines (TPS) are given as examples of those classes. Additionally, some well known techniques for solving image registration problems, including both classical as well as evolutionary computation and metaheuristic based approaches are discussed.

Chapter 4 proposes a new objective function for matching points between deformed and static images by using a randomized ordering when visiting the points during the matching procedure. Then, the new objective function is studied by means of genetic algorithm and evolutionary strategies. The control parameters for both evolutionary algorithms are discussed in this chapter. It investigates the behavior of SBX in the noisy objective function. Furthermore, it discusses the behavior of evolutionary strategies and genetic algorithm for the image registration problem.

Chapter 5 concludes the thesis. Just as the thesis itself, this chapter ends suggesting some topics for future work in the application of EAs to image registration.

Chapter 2

Genetic Algorithms and Evolution Strategies

2.1 Introduction

Evolutionary Computation (EC) covers all aspects of the simulation of evolutionary process in computer systems. It is quite a recent and tremendously growing field as well as an optimization process. The term itself was invented as recently as 1991, and represents an effort to bring together researchers who have been following different approaches to simulating various aspects of evolution [3]. Those aspects are *genetic algorithms* (GAs) [40, 59, 61], *evolutionary strategies* (ESs) [5, 12], and *evolutionary programing* (EP) [36, 37, 62]. Note, since last decade these aspects are extended. Although simulations of the natural evolution have been used by biologists to study adaptation in changing environments to gain insight into the evolution of the complex organisms found on Earth, it has also been shown that complex optimization problems can be solved with simulated evolution. EC techniques have been successfully applied to various optimization problems in engineering, economics, biology, chemistry, physics, and computer science.

There are many computational problems that require searching through a huge num-

ber of possible solutions. Moreover, many computational problems require complex solutions that are difficult to program by hand. The classical techniques for solving complex optimization problems have been generally unsatisfactory when applied to nonlinear optimization problems especially those with temporal, stochastic, or chaotic elements. Nonetheless, these problems can be classified under the same group of problems that Nature solves itself. In other words, biological evolution is an appealing source of inspiration for computing the solutions to problems that have previously appeared intractable.

This chapter is devoted to two classes of algorithms in EC (ESs and GAs) which have been implemented and used in this work. First of all, natural evolution and history of genesis of evolutionary computation are addressed. A general scheme that forms the common basis of all evolutionary algorithms variants and its key ingredients are elaborated. Afterward, genetic algorithms are briefly described. Furthermore, it details GA operators that are applicable to representations in the continuous domain, as that will be the case for the image registration problem that is addressed in this thesis. Finally, evolutionary strategies involving self-adaptation and its components are elaborated.

2.2 Natural Evolution

In 1859, Darwin came up with *the origin of species* [23] which presented a theory for existence and evolution of life on Earth. According to his theory, the vast majority of the history of life can be fully accounted for by physical (evolution) processes operating on and within populations and species [48]. These processes are: *replication*, *variation*, and *selection*. Replication is an obvious property of extant species. In other words, it increases the population size of species that would have reproductive potential at an exponential rate if all individuals were to reproduce successfully. Variation comes about through the transfer of an individual's genetic program (asexually or/and sexually) to progeny. Variation is introduced due to errors in the replication process, resulting in a

gradual development of new organisms [58]. These changes often occur due to coping errors. Sexual recombination is another form of variation and is itself a product of evolution. Competition is a consequence of expanding populations in a finite resource space, because of the limited resources on Earth, replication can not go on infinitely; individuals of the same species or other have to compete with each other and only the fittest survive. Thus, natural evolution implicitly causes the adaptation of life forms to their environment once only the fittest have a chance to reproduce.

Natural evolution is an open-ended dynamic process in which the fitness of an individual can only be defined in relation to the environment. For example, an Indian elephant has a high fitness in its native environment, since it is well adapted to the weather of mainland Asia. Bringing the Indian elephant to the North Pole would certainly reduce its fitness. Sometimes, species become extinct when they are not able to react to rapid changes in their environment [23].

At this point it is useful to get formally across to the principle of natural evolution by means of some biological terms. In the context of evolutionary algorithms, these biological terms are used in the spirit of analogy with real biology, though the entities they refer to are much simpler than the real biological ones [59].

All living organisms consist of cells, and each cell contains a copy of a set of one or more *chromosome(s)*, which are strings of DNA. The chromosome serves as a "blue print" for the organism. It can be conceptually divided into *genes*, each of them encodes a particular protein and, is also located at a specific *locus* on the chromosome. Very roughly, the genes can be spotted as encoding of a *trait*, such as eye color. The different possible settings of a trait (e.g. black, white, green) are called *alleles*. Most complex organisms have more than a single chromosome in each cell. All chromosomes taken together make up an organism's *genome* which is the complete collection of genetic material. Each organism carries its genetic information referred to as the *genotype*. In other words, genotype refers to the particular set of genes contained in a genome. The organism's traits, which are

developed while the organism grows up, constitute the *phenotype*. Genotype gives rise to the phenotype of the organism under fetal or later development.

The two types of reproduction which are mentioned above can be found in nature (asexual, and sexual). The asexual recombination, also known as *haploid*, where an organism reproduces itself by cell division and the replication of its chromosome(s). Offspring are subject to *mutation* during this process in which one or more alleles of gene are changed, genes are deleted, or they are reinserted at other *loci* on the chromosome. *Diploid* points out the sexual recombination (the second type of reproduction) which has paired chromosomes. In Nature, most sexually reproducing species are diploid, including human beings, having 23 pairs of chromosomes in each cell. In this type of recombination also well known as crossover, genes are exchanged between the chromosomes of the two parents to form a new set of chromosome(s). The *fitness* of a particular organism is mostly defined as the probability the organism has to live and reproduce, called *viability*, or defined by the number of offspring the organism has, called *fertility*.

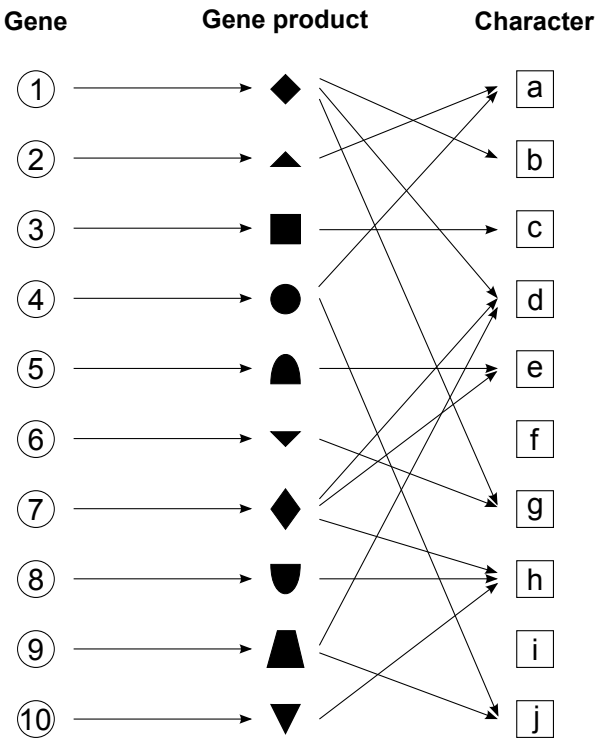


Figure 2.1: Epistatic gene interaction, and the behavior of pleiotropy and polygeny.

Due to the universal effects of gene interaction which is called *epistasis*, the results of genetic variations are hardly predictable. The effect that a single gene may simultaneously influence several phenotypic traits is called *pleiotropy*. And conversely, a single phenotypic characteristics may be determined by the simultaneous interaction of many genes. This effect is called *polygeny*. There are no one-gene, one-trait relationships in natural evolved systems [34]. Fig. 2.1 illustrates the pleiotropy and polygeny. Epistatic interaction in form of pleiotropy and polygeny are always found in living organisms, consequently, the phenotype varies as a complex, nonlinear function of the interaction between the underlying genetic structures and the environmental conditions.

2.3 History of Evolutionary Computation

Writing history is one of the most difficult works. It becomes more complicated when it dates back further down in the past. On the contrary, the evolutionary computation is a recent area for scientific research and most of its initiators are still around. It started in the mid-1950s when several scientists used digital computer models to understand the natural process of evolution better [51, 59].

In the 1960's decade, on both sides of the Atlantic Ocean (Germany and USA) the basis of what we today identify as evolutionary algorithms (EAs) were clearly established [51]. ESs were a joint development of a group of three students, Bienert, Rechenberg, and Schwefel, in Berlin (1965). On the other side of the ocean, the roots of EP were laid by Fogel, Owens, and Walsh in San Diego, California (1966). GAs were developed by Holland, his colleagues, and his students at the University of Michigan in Ann Arbor (1967). Over the following 25 years each of these branches developed quite independently from each other [51], resulting in unique parallel fields which are described in more detail in the following paragraphs. However, after 1990 the boundaries between the three main EC streams have broken down to some extent. Nowadays, there is a widespread inter-

action among researchers studying various EC methods, and there are more than three EC methods. Moreover, in 1996, Bäck introduced a common algorithmic scheme for all brands of current evolutionary algorithms [1].

In 1964, three students at the Technical University of Berlin introduced evolutionary strategies (*Evolutionsstrategie* in German). They developed an approach to optimize the real-valued parameters for devices such as airfoils [13, 63, 71]. Only then did Rechenberg (1965) hit upon the idea to use dice for random decision [51]. After the first computer experiment on implementing ES by Schwefel (1965), the use of normally instead of binomially distributed mutations become standard in most of later computer experiments with real-valued parameters. It was Rechenberg (1973) who formulated a 1/5 success rule for adapting the standard deviation of mutation. Self-adaptation with respect to correlation coefficients and mutation step-size was achieved with the (μ, λ) ES in 1975 by Schwefel and published in his Dr.-Ing. thesis [72]. During 1980s the notion of *self-adaptation by collective learning* first came up and the importance of recombination and soft selection was clearly demonstrated. In 1996, Hansen and Ostermeier invented the new robust method, called *covariance matrix adaptation* (CMA–ES) for governing the individual step-sizes for each coordinate or correlations between coordinates [42].

Genetic algorithms were invented by Holland [49] and were developed by his students and colleagues. Holland's original goal was to study the phenomenon of adaptation as it occurs in Nature and to develop ways in which the mechanisms might be imported into computer systems. Compared to ES and EP, Holland's GA was the first algorithm incorporating a form of recombination (crossover). During the following two decades Holland and his students kept working on the general theory of adaptive systems, but the idea did not spread around the world the before publication of Goldberg's book [40]. That book, in particular, served as a significant catalyst by presenting current GA theory and applications in a clear and precise form easily understood by a broad audience of scientists and engineers, and it's one of the high cited documents in the field of EC.

By the mid-1980s, the first international conference on GAs was held in Pittsburgh, Pennsylvania, USA. In 1993, Juels, Baluja, and Sinclair came up with the idea [52] of replacing the population by a probability vector [57]. It was the base of a new robust approach for GAs and EAs, known as *Estimation of Distribution Algorithms* (EDAs) [46].

2.4 Evolutionary Algorithms

Evolutionary algorithms (EA) have several branches including genetic algorithms, evolutionary strategies, evolutionary programing, and many others. Algorithm 2.1 shows a general template of EA without referring to a particular algorithm. All proposed methods are special cases of this scheme. It starts with initializing a population randomly, i.e., a set of candidate solutions. All candidate solutions are applied to a quality function (to be maximized/minimized) as an abstract fitness measure, the higher/lower the better. Then, in the main loop, a temporary population is selected from the current population (survival of the fittest), which causes a rise in the fitness of the population. Thereupon, the evolutionary operators including *mutation* and *recombination* are applied to all members (individuals) of the temporary population. Recombination is an operator applied to two or more selected individuals (the so-called parents) and results in one or more new candidate(s) (offspring). Mutation is applied to one candidate and results in a new candidate. The main loop is repeated until a termination criterion is fulfilled; for example, if the number of generations evolved exceeds a predefined limit. The newly created individuals are evaluated by calculating their fitness. Before a new generation is processed, the new population is selected from the old and temporary populations.

The evolutionary process makes the population increasingly better at adapting to the environment. Variation operators — recombination and mutation — create the necessary diversity and thereby facilitate novelty while selection acts as a force pushing toward quality. In general, the combined application of variation and selection leads to

Algorithm 2.1 The general scheme of an evolutionary algorithm

Input: A problem at hand with an objective function f to optimize

Output: A solution or a set of solutions

1: $g \leftarrow 0$;
2: `initialize-population(`P^0`)`;
3: `evaluate(`P^0 `using` f`)`;
4: **while** `termination-condition` = **false do**
5: $P' \leftarrow$ `select-for-variation(`P^0`)`;
6: $P' \leftarrow$ `recombine(`P'`)`;
7: $P' \leftarrow$ `mutate(`P'`)`;
8: `evaluate(`P' `using` f`)`;
9: $P^{g+1} \leftarrow$ `select-for-survival(`$P^{(g)}, P'$`)`;
10: $g \leftarrow g + 1$;
11: **end while**

improving fitness values in consecutive populations.

The fitness evaluation is the central part of an evolutionary algorithm. It is usually the objective function of the problem to be solved by the evolutionary algorithm. In other words, the objective function is an expression of environmental requirements.

2.4.1 Components of Evolutionary Algorithms

This section discusses EA in detail. EAs have a number of components and operators that must be specified in order to define a particular EA. The most important components are:

- Representation
- Selection
- Recombination

- Mutation
- Replacement (Survivor Selection)
- Termination Condition

In the following subsections these components are addressed.

2.4.1.1 Representation

The first step in evolutionary algorithms is defining a representation for a given optimization problem. Every search and optimization algorithm deals with solutions, each of which represents an instantiation of the underlying problem. For instance, given an optimization problem defined over n real-valued variables, the set of all possible instantiations of these variables (i.e. the set of all n-dimensional real valued vectors) would form the set of all possible solutions, or *search space*. Representation can be binary, integer, and real, permutations, and even more complex such as lists, trees, and other variable-length structures. In some optimization problems, the solutions may contain different types of variables, called mixed representation.

2.4.1.2 Selection

In each iteration of the EAs, selection is the first step, which consists of selecting a set of promising solutions from the current population based on the quality of each solution (objective function value). The basic idea of this operator is to make more copies of the solutions that perform better according to the objective function value than those that perform worse. There are two main types of selection methods, *fitness proportionate selection* and *ordinal selection*. In the fitness proportionate selection methods (e.g. roulette-wheel selection), each selected solution is drawn from the same probability distribution and the probability of selecting the solution is proportional to its objective function value. Whilst, in the ordinal selection, the probability of selecting a particular

member of the population does not directly depend on its objective function value but it depends on the relative quality of this solution compared to other members of the current population. Ordinal selection methods are generally more popular than fitness proportionate selection methods [41]. There are two main reasons for that; firstly, ordinal selection methods are invariant to linear transformation of fitness and they pose fewer restrictions on the fitness function than fitness proportionate selection methods do. Secondly, ordinal selection methods enable a sustained pressure toward solutions of higher quality and the strength of this pressure is often easier to control. *Tournament selection* is one of the most popular ordinal selection methods. Different selection operators can be found in [31, 41].

2.4.1.3 Recombination

Recombination combines subsets of parent population by exchanging, merging or interacting some of their parts. In biological systems, recombination is a complex process that occurs between pairs of chromosomes which are physically aligned; and breakage occurs at one or more corresponding locations on each chromosome, an homologous chromosome fragments are exchanged before the breaks are repaired [14]. Recombination is guided by the two following requirements (e.g. [10]):

- **Building block hypothesis (BBH):** The BBH [40] explains the different good building blocks from different parents mixed together, thus combining the good properties of the parents in the offspring.

- **Genetic repair (GR):** It is not the different features of the different parents that flow through the application of the recombination operator into the offspring, but their common features [9]. In other words, recombination extracts the similarities from the parents.

For more information on recombination operators, the reader is directed to [11, 14, 26]

2.4.1.4 Mutation

Mutation is responsible for introducing small variation(s) to the chromosomes, which is achieved by performing random modifications locally around a solution. It generally refers to the creation of a new solution from one and only one parent [4], otherwise the creation is referred to as a blend of two or more chromosomes which is recombination. In general, mutation is guided by the following requirements (e.g. [1, 10]):

- **Accessibility:** Every state of the search space should be accessible from any other state by means of a finite number of applications of the mutation operators;
- **Feasibility:** The mutation should produce feasible individuals. This guideline can be crucial in search spaces with a high number of infeasible solutions;
- **Symmetry:** No additional bias should be introduced by the mutation operators;
- **Similarity:** Evolutionary algorithms are based on the assumption that a solution can be gradually improved. This means it must be possible to generate similar solutions by means of mutation.

2.4.1.5 Replacement (Survivor Selection)

The main aim of the replacement is to form the new population for the next generation. There are two main approaches for replacement, *full replacement* and *steady-state*. In full replacement, all of the new candidate solutions replace the original solutions. In contrast, in steady-state replacement, the new population is drawn from the union of the old population and new candidate solutions.

2.4.1.6 Termination Condition

As termination conditions the following standard stopping rules can be used:

1. resource criteria:

- maximum number of generation;
- maximum cpu–time.

2. convergence criteria:

   - in the space of the objective function values;
   - without improving the objective function values after a certain number of generations.

2.5 Genetic Algorithms

Among all different evolutionary algorithms, genetic algorithms (GAs) have a significant similarity to the general scheme of evolutionary algorithm (Algorithm 2.1). As it was addressed in the introduction section, GAs are stochastic optimization methods [32, 40, 49, 59, 61]. This section gives a basic overview of the standard GA and its key ingredients. In addition, more details are introduced about the operators' types that are applied in this thesis. An optimization problem in GA is normally defined by:

- Representation of potential solutions to the problem (chromosome's type).
- An objective function to evaluate the quality of each candidate solution.

GAs work with a population or a set of candidate solutions (chromosomes), in order to find a solution or a set of solutions that perform(s) best with respect to the specified measure (objective function value). Then, the population (candidate solutions) is updated for a specific number of iterations. Each iteration uses the following operators:

1. Selection
2. Variation — crossover (recombination) and mutation
3. Replacement

Similarly to all kinds of EAs, GAs can have different kinds of representation. Here, we are just focused on the real-valued representation and operators as our representation in this work is based on that. In the following subsections those operators are described.

2.5.1 Tournament Selection

The main idea of tournament selection is to sample a population subset of size s, and then select the best solution out of this subset. In what concerns choosing chromosomes (s times) randomly to involve to the subset, it can be done with or without replacement. This process usually repeats N times, where N is the population size.

2.5.2 Variation

Crossover and *mutation* are applied to the set of solutions that are selected in the previous operator (selection). Crossover combines chromosomes by exchanging some of their parts. Mutation is responsible for introducing small variation(s) to the chromosomes, which is achieved by performing random modifications locally around a solution. The next two subsections addresses two popular variation operators for real coded representations, *simulated binary crossover* (SBX) and *Gaussian mutation*. These operators are used in the GA application to image registration.

2.5.2.1 Simulated Binary Crossover (SBX)

The SBX operator was designed to work with real-coded EAs [27]. This recombination scheme involves two parent values (p_1 and p_2) that create two offspring values (c_1 and c_2), relatively speaking; it is variable-wise operator, where each variable, from participating parent solutions, is recombined independently with a certain pre-specified probability to create two new values. The resulting offspring solutions are then formed by concatenating the new values from recombinations of one of the existing parent values, as the case may be. It is also a parent-centric operator, where the offspring solutions are created around

the parents solution. The user-specific control is achieved by means of a parameter n. The effect of n is shown in Fig. 2.2.

SBX uses a probability density function given by

$$\mathcal{P}(\beta_i) = \begin{cases} 0.5(n+1) \times \beta_i^n & \text{if } \beta \leq 1, \\ 0.5(n+1) \times \dfrac{1}{\beta_i^{n+2}} & \text{otherwise }, \end{cases} \tag{2.1}$$

where $\beta_i \in [0, \infty]$ is called the spread factor and is defined as the ratio of the absolute difference in offspring values to that of the parents of the i-th variable. For each variable of the participating parents, the spread factor is calculated using an appropriate mapping from a randomly generated number and then $\mathcal{P}$ decides the location of the offspring. The SBX probability distribution is shown in Fig. 2.2.

This distribution can easily be obtained from a uniform random number $u_i \in U(0, 1)$ by the transformation:

$$\beta(u_i) = \begin{cases} (2 \times u_i)^{\frac{1}{n+1}} & \text{if } u_i \leq 0.5 \\ (2 \times (1 - u_i))^{\frac{-1}{n+1}} & \text{if } u_i > 0.5 \end{cases} . \tag{2.2}$$

For the two participating parent values (p_1 and p_2), two offspring values (c_1 and c_2) can be created as a linear combination of parent values for an event with $\beta(u_i)$ value drawn from (2.2), as follows:

$$c_1 = 0.5(1 + \beta_i) \times p_1 + 0.5(1 - \beta_i) \times p_2 \ , \tag{2.3}$$

$$c_2 = 0.5(1 - \beta_i) \times p_1 + 0.5(1 + \beta_i) \times p_2 \ . \tag{2.4}$$

The resulting weights to p_1 and p_2 are biased in such a way that offspring values close to the parent values are more likely than offspring values away from them (see Fig. 2.2).

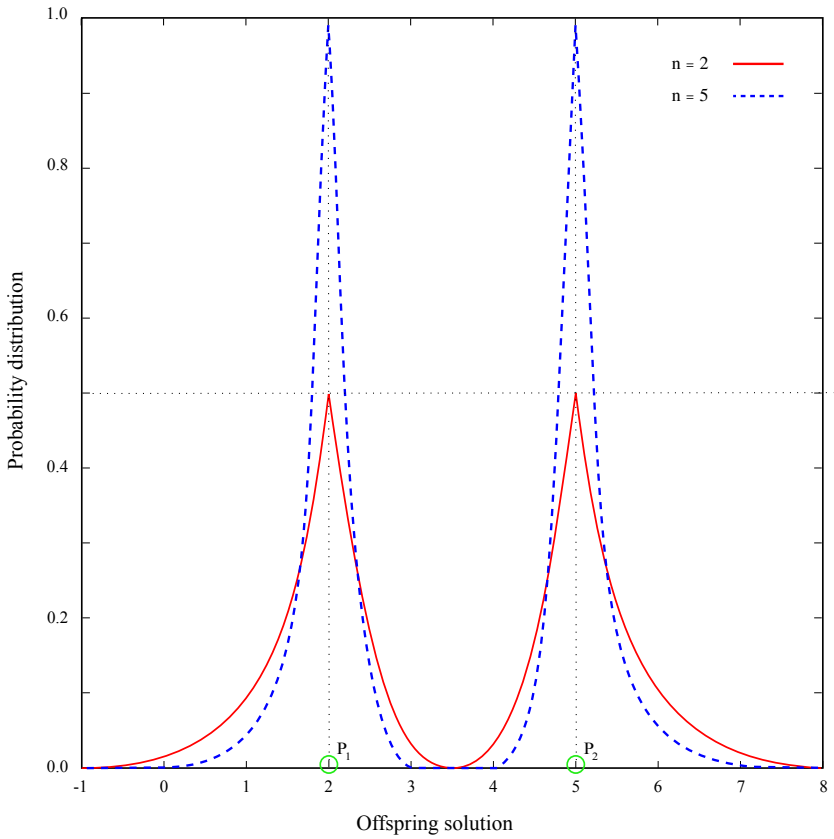


Figure 2.2: Probability distribution, used in the simulated binary crossover (SBX), are shown for different values of the distribution index n. Figure courtesy of Deb and Jain [26].

2.5.2.2 Gaussian Mutation

In this mutation, each variable is modified by adding a random number according to a Gaussian distribution with zero mean. Significant mutation needs high variance of the Gaussian distribution and vice versa.

2.6 Evolutionary Strategies

This section introduces evolutionary strategies (ES), another member of the evolutionary algorithm family. ESs are typically used for continuous parameters optimization [31]. Nevertheless, just as in GAs it can also be used in binary and integer search spaces. Here, we just go through real-valued representation. Nowadays, almost all of ESs algorithms

use *self-adaptation* techniques (on-line adaptation). In general, self adaptivity means that some strategy parameters of the EA are varied during the search process by incorporating them into the genetic representation of the individuals [5]. In ESs the parameters are included in the chromosomes and co-evolve with the solutions. Compared to the GA, the ES stands out by using *endogenous parameters* included in each chromosome that allows the population in an ES to self-adapt in the direction of the optimal solution(s) [1, 76].

ES as all other EC methods is started by random initialization. After initializing population and evaluation of individuals, they are randomly and uniformly selected to be parents for producing children via recombination. This process of selection and recombining the selected parents to produce children continuously generate a certain number of individuals — the number of children is greater than the number of parents. The children are further perturbed via mutation. Finally, the survival selection picks a certain number of the best children to survive. For a comprehensive introduction to ESs see [12].

In the following subsections, the main components of ESs as well as some operators for real-valued representation are described.

2.6.1 Representation

Standard evolutionary strategies are typically used for continues parameter optimizations ($\mathbb{R}^n \to \mathbb{R}$). An individual of the evolutionary strategies typically consists of two components, a candidate solution or *object parameters* and *endogenous strategy parameters*.

The object parameters are presented as $\vec{x} = \langle x_1, \ldots, x_n \rangle \in \mathbb{R}^n$. Endogenous strategy parameters, $\vec{\sigma}$, essentially encoded the n-dimensional normal distribution and are to be used to control certain statistical properties of the mutation operators. Endogenous strategy parameters are very special in ES and can evolve during the whole evolution process, while GAs do not have that. With adding strategy parameters to the vector $\vec{x}$, the ES's individuals shape as follows:

$$\langle \underbrace{x_1, \ldots, x_n}_{\vec{x}}, \underbrace{\sigma_1, \ldots, \sigma_n}_{\vec{\sigma}} \rangle \quad . \tag{2.5}$$

2.6.2 Parent Selection

Parent selection in evolutionary strategies is independent of the parental objective function values. Whenever a recombination operator requires a parent, it is drawn randomly with uniform distribution from population of μ individuals. This contrasts to standard selection techniques in genetic algorithms [40], where the selection relies on the objective function values. Here, it should be considered that in ES terminology, the word "parent" hints the whole population — often called *parent population* — while in GA terminology, it refers to a member of the population that has been selected to undergo variation.

2.6.3 Recombination

Both part of the selected individuals (object parameters and strategy parameters) from parent selection undergo recombination. The basic recombination scheme in evolutionary strategies involve two or more parents that create one child. To obtain λ offspring, recombination is performed λ times. There are two well-known recombination operators in evolutionary strategies, *discrete* and *intermediate* recombinations. In discrete recombination, one of the parent alleles is randomly chosen with equal chance for either parents, while, using intermediate recombination, the values of the parents' alleles are averaged. According to the literature, discrete and intermediate recombinations are recommended for object and strategy parameters, respectively [1, 31].

2.6.4 Mutation

For applying mutation, step-size values (strategy parameters) are needed, which represent standard deviation values to be used in the sampling of values drawn from a Normal

distribution. In practice, the mutation step-sizes, $\vec{\sigma}$, are not set by the user — nevertheless, they are initialized by the user — rather they are coevolving with the solutions according to the self-adaptation. To achieve this, it is essential to modify the strategy parameter(s) first and afterward mutate the object parameters with the new strategy parameter(s). The rationale behind this is that a new individual is evaluated twice. Firstly, it is evaluated directly for its viability during survival selection and secondly, it is evaluated for its ability to create good offspring. Different types of variables can have different kinds of mutation operators. Mutation of strategy parameters are explained in section 2.6.6. Real-valued representation has two well-known mutation operators for object parameters, *Gaussian* and *Polynomial* mutation. Gaussian mutation is addressed in section 2.5.2.2 and in the following subsections, polynomial mutation is elaborated.

2.6.4.1 Polynomial Mutation

One of the popular mutation in real-valued search spaces is *polynomial mutation* which was designed by Deb and Goyal [25]. It has one controllable parameter, a so-called mutation distribution parameter, m. That parameter in ESs is provided as a strategy parameter σ_i, and controls the magnitude of the expected mutation of the candidate solution variable; relatively speaking, small values of σ_i produce large mutations on average while large values of σ_i produce small mutations.

This mutation uses a polynomial probability distribution with mean at the current value and variance as a function of the distribution index m. The probability distribution used in this mutation is defined as follows:

$$\mathcal{P}(\delta) = 0.5 \times (m+1) \times (1 - |\delta|)^m \ , \tag{2.6}$$

where δ is a perturbation factor, and m, as defined before, is the distribution index. This distribution is shown in Fig. 2.3 for different values of m. Furthermore, this probability

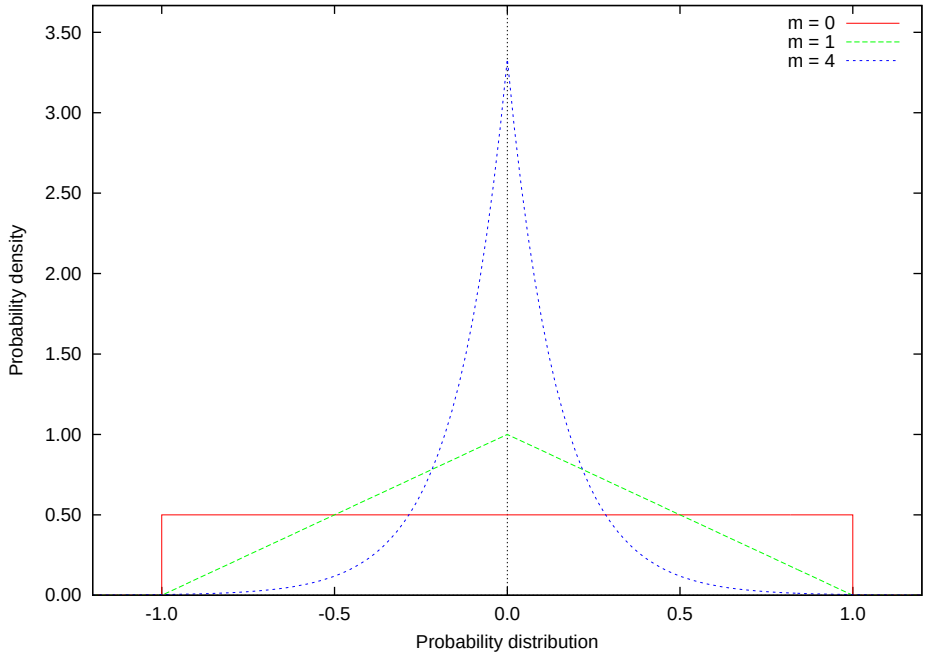


Figure 2.3: Probability distribution to create a mutated value for continuous variables. Figure courtesy of Deb and Goyal [25].

distribution is valid in range $\delta \in (-1, 1)$. It can easily be obtained from a uniform random number u by the transformation:

$$\delta(u) = \begin{cases} (2 \times u)^{\frac{1}{m+1}} - 1 & \text{if } u < 0.5 \\ 1 - (2 \times (1-u))^{\frac{1}{m+1}} & \text{if } u \geq 0.5 \end{cases} . \tag{2.7}$$

Thereafter, the mutated value is calculated as follows:

$$x_i' = x_i + \delta(u_i) \times \Delta_{max} \ . \tag{2.8}$$

where Δ_{max} is a fixed quantity which represents the maximum permissible perturbation in the current value x_i and x_i' is the mutated value.

2.6.5 Survivor Selection

Parent selection and variation operators yield a certain number of offspring (λ), afterward their objective function values are calculated. For the next generation, the best μ of them are chosen *deterministically*, either from a combined pool comprising current population and offspring population, called $(\mu + \lambda)$-selection (elitist selection), or from the offspring only, called (μ, λ)-selection. These notations were introduced by Schwefel in 1977 [73].

At first blush, the $(\mu + \lambda)$-selection seems to be more effective than (μ, λ)-selection. As $(\mu + \lambda)$-selection always guarantees the survival of the best individual, a monotonous course of evolution is achieved this way. However, this selection scheme has some disadvantages when compared to the (μ, λ)-selection — which restricts lifetimes of individuals to one generation. Some reasons of preference of (μ, λ)-selection rather than $(\mu + \lambda)$-selection are now presented:

- The (μ, λ)-selection discards all parent population which enables it to prevent getting stuck in the local optima, it is therefore an advantage in cases of multi-modal topologies.

- If the objective function is noisy (it changes in time), the $(\mu+\lambda)$-selection preserves outdated solutions, so it is not partially able to follow toward optimum. In other words, it is not theoretically admissible to compare two sets with the same objective function affected by different noises.

- The $(\mu + \lambda)$-selection hinders the self-adaptation mechanism with respect to strategy parameters to work effectively, because misadapted strategy parameters may survive for a relatively large number of generations when an individual has good object parameters which cause a good objective function value, while in contrast, it has bad strategy parameters. These kind of individuals mostly yield bad offspring. In general, with elitist selection, the individuals with bad strategy parameters may survive.

The survivor selection has a high pressure in ESs. Practically, λ is much higher than μ, because the extreme case $\mu = \lambda$ for (μ, λ)-selection leads to a random walk behavior of the algorithm, i.e., no selection takes place. A ratio of $\mu/\lambda \approx 1/7$ is recommended [1]. Therefore, μ also has to clearly be chosen, larger than one (e.g. $\mu = 15$) [1, 75].

2.6.6 Self-Adaptation

The strategy parameters determine the distance that the mutated object parameters will lie from the original parameters in the search space. They are also called *step-sizes*. Step-size(s) should get smaller as long as the objective function values get closer to the optimum solution. The aim of self-adaptation is to modify those strategy parameters by means of applying evolutionary operators to them in a similar way as to the solution representations. The competitive process of evolutionary algorithms is then exploited to determine if the changes of the parameters are advantageous concerning their impact on the objective function value of individuals. For a comprehensive introduction to the self-adaptation see the special issue of the *Evolution Computation Journal* (2001) and moreover see [2, 11, 30, 54, 55, 60].

This section is devoted to mutation of the strategy parameters. Firstly, *1/5 success rule* is explained, it is considered as one of the famous on-line adjustment of the strategy parameters. Then, *uncorrelated mutation with one or n step-size(s)* of strategy parameters in real-valued search space are addressed.

2.6.6.1 1/5 Success Rule

Theoretical studies motivated a self-adaptation of step-sizes by the famous 1/5 success rule of Rechenberg since 1973 [64]. This rule states that the ratio of successful mutations should be 1/5. If it is greater than 1/5, the step-size should be increased; if it is less, the step-size should be decreased. The rule is executed at periodic intervals, and step-size,

σ, is reset by the following equation:

$$\sigma = \begin{cases} \sigma & \text{if } p_s > 1/5, \\ \sigma \times c & \text{if } p_s < 1/5, \\ \sigma & \text{if } p_s = 1/5, \end{cases} \tag{2.9}$$

where p_s is the relative frequency of successful mutations measured over a number of trials, and the parameter c, as an adjustment factor, is in the range $0.817 \leq c \leq 1$ [31]. Schwefel suggested the factor $c = 0.82$ in 1981 [74], and later on, he came up with reasons to use the factor $c = 0.85$ in 1995 [76], which should take place every certain number of generations.

2.6.6.2 Uncorrelated Mutation with One or n Step-Size(s)

Choosing an appropriate mutation rate is known to have an important impact on the performance of an EA, and to avoid inappropriate settings of the mutation rate, which can lead to poor performance of an EA, the self-adaptation is applied. There are two types of uncorrelated mutation rules, one step-size and n step-sizes. The former, has only one strategy parameter in each individual while in the later form, each gene within individual has its own strategy parameter.

The mutation mechanism for uncorrelated mutation with one step-size is described as follows:

$$\sigma' = \sigma \times e^{\tau . \mathcal{N}(0,1)} \ , \tag{2.10}$$

$$x_i' = x_i + \sigma' \times \mathcal{N}_i(0,1) \ , \tag{2.11}$$

where σ is mutated each time step by multiplying it by a term $e^{\tau . \mathcal{N}(0,1)}$. $\mathcal{N}(0,1)$ denotes a draw from the standard normal distribution, while $\mathcal{N}_i(0,1)$ denotes a separate draw

from the standard normal distribution for each variable x_i. The parameters τ can be interpreted as *learning rate*.

The mutation mechanism for uncorrelated mutation with one step-size is described as follows:

$$\sigma_i' = \sigma_i \times e^{\tau \cdot \mathcal{N}(0,1) + \tau' \cdot \mathcal{N}_i(0,1)} \ , \tag{2.12}$$

$$x_i' = x_i + \sigma_i' \times \mathcal{N}_i(0,1) \ , \tag{2.13}$$

where τ and τ' are called global and local learning rate, respectively. The common base mutation $e^{\mathcal{N}_i(0,1)}$ provides the flexibility to use different mutation strategies in different directions.

Chapter 3

Image Registration

3.1 Introduction

Image registration (IR) has been applied in a large number of research areas, including medical image analysis, computer vision and pattern recognition [87]. The goal of IR is to find a geometric, or elastic transformation that makes one image similar to the other (Fig. 3.1) [28]. In all IR problems, there are at least two images, a *Static* (S) and *Deformed* (D), that represent the same object, or scene viewed from a different perspective, and/or with different deformation. Defining it more formally, IR aims to find the best mapping function T to warp D towards S, as shown below:

$$W = T(D) \approx S \ . \tag{3.1}$$

W is a warped image that should be as closely shaped to S as possible. IR typically has the four following steps [87]:

1. Feature detection;
2. Feature matching;
3. Mapping function design;

4. Image transformation and re-sampling.

In order to find the correct transformation, image features such as closed-boundary regions, edges, line intersections, corners, and so on, should be extracted (feature detection). These features can be used as control points. Correspondences have to be set between the extracted features of S and D (feature matching). Then, the type of transforming model has to be chosen and its parameters estimated (mapping function). Finally, D is transformed by means of the mapping function (image transformation).

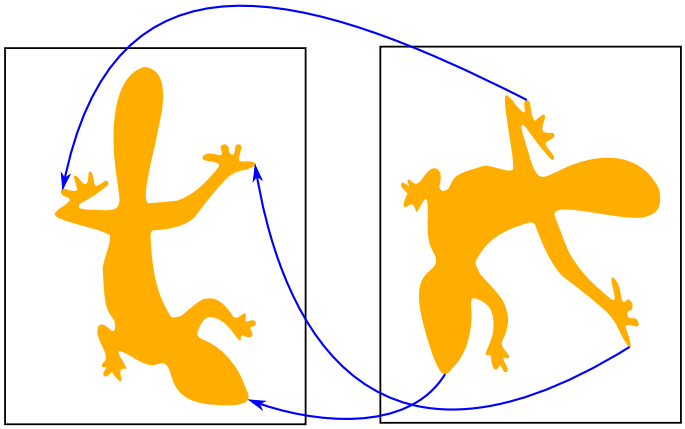


Figure 3.1: Finding a transformation by point correspondence.

IR can be seen as a function approximation method [78], and it is a NP-Complete problem [53]. The most important aspect of parametric IR is the discovery of the unknown parametric transformation that relates the two images. Two different approaches can be found in the literature:

- Matching-based approaches
- Transformation parameters-based approaches

Matching-based approaches conduct a search within the space of possible feature correspondences (typically point matching) between the two images. Thereafter, the parameters for the transformation are calculated based on the correspondence found.

In contrast, *transformation parameters-based* approaches perform a direct search in the space of the parameters of the transformation.

IR methods can also be classified according to the type of models that they allow to transform D into S (sometimes also referred in the literature as the *scene* and *model* images). Two major types of models used are: *linear* and *non-linear transformations*. Linear transformations preserve the operations of vector addition and scalar multiplication. The same does not hold for non-linear (or *elastic*) transformations, which allow local deformations of the image.

The chapter starts by describing several geometrical warping models (or transformation). The estimation of the difference between warped and static images is considered in Sect. 3.3. The chapter ends with a brief literature review of related work that has been proposed to address the IR problem, both with classical, EC and Metaheuristic (MH) methods in Sect. 3.4.

3.2 Warping

Independent from the IR algorithm, *warping* is a very important step in the registration process. Image warping is the application of the calculated transform to the deformed image or, in other words, the process of geometrically transforming a given image. In order to apply geometric transformations to the image, a transform function has to be defined. Transforms may be *rigid* or *non-rigid*. Non-rigid warping is also called elastic [28].

One way of applying the transformation is by transforming each position of the deformed image D and setting the corresponding position in the warped image W, the following way:

$$W(T(i)) \leftarrow D(i) \ , \tag{3.2}$$

where i represents the corresponding positions. This process is referred to as *forward*

warping. Another way of applying the transformation to the image, is by finding a value in the deformed image for each position of the warped image. This approach is called *backward warping*. Therefore, T which is used in Equation (3.2) is not suitable anymore, and the inverse of the transformation should be used the following way:

$$W(i) \leftarrow D(T^{-1}(i)) \ . \tag{3.3}$$

With this approach all the positions of the warped image W are visited once.

3.2.1 Geometric Transform

A common and straightforward approach for doing IR is to deal with the deformation as if it was global. Thus, the transformation is applied globally to the image. Rotation and translation are the most common differences between static and deformed point-sets, and these, very often, affect the image globally.

Transformations that are frequently used to correct global misalignments are described in the following subsections.

3.2.1.1 Rigid

Transformation can be classified as rigid and non-rigid according to the transform used in the process of registration. Rigid transformations preserve the straightness and size of all lines, as well as the angles between them. A rigid transformation can be split in two parts, i.e., *translation* and *rotation*. The affine transform can be seen as a special case of the rigid classification which furthermore is able to do *shearing*.

3.2.1.1.1 Affine Transform

The affine transform is a linear transformation that includes the following elementary transformations: translation, rotation, scaling, stretching, and shearing [50]. These elementary transformations are illustrated in Fig. 3.2.

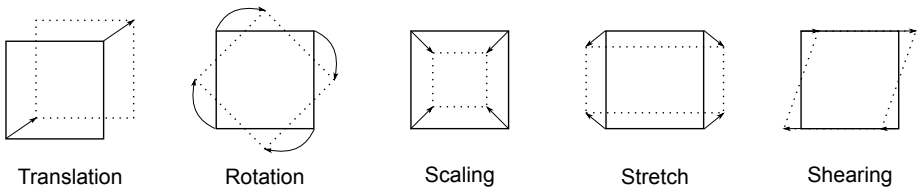


Figure 3.2: Elementary geometric transforms for a planar surface element used in the affine transform: translation, rotation, scaling, stretching, and shearing.

A geometric operation transforms a given image D into a new image W by modifying the coordinates of the image points, as follows:

$$D(x, y) \xrightarrow{T} W(x', y') \ . \tag{3.4}$$

The original values of image D, located at (x, y), warp to the new positions (x', y') in the new image W. To model this process, we first need a mapping function T that is a continuous coordinate transform. An affine transformation function works in the 2-D space, thus, the search space is:

$$T : \mathbb{R}^2 \longrightarrow \mathbb{R}^2 \ . \tag{3.5}$$

The mapping function can be redefined as:

$$\begin{aligned} W &= T(D) \\ T &: \mathbb{R}^2 \longrightarrow \mathbb{R}^2 \ . \end{aligned} \tag{3.6}$$

The warped image $W(x', y')$, in the case of the affine transformation, can be specified as the following two separated functions for the x and y components:

$$x' = T_x(x, y) \tag{3.7}$$

$$y' = T_y(x, y) \ . \tag{3.8}$$

An affine transformation can be expressed by vector addition and matrix multiplication as shown in Equation 3.9,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = S \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3.9}$$

where S is the scaling parameter. By multiplying S with the rotation matrix, Equation 3.9 can be written as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad . \tag{3.10}$$

Finally, by using homogeneous coordinates, the affine transformation can be rewritten as Equation 3.11.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 \\ \theta_3 & \theta_4 & \theta_5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad . \tag{3.11}$$

The affine transform has six parameters: θ_0, θ_1, θ_2, θ_3, θ_4, and θ_5. θ_2 and θ_5 specify the translation and θ_0, θ_1, θ_3, and θ_4 aggregate rotation, scaling, stretching, and shearing.

3.2.1.2 Non-Rigid

Non-rigid transforms, also called deformable and elastic, allow more complex distortions in the image. These include the stretching and curving of the image. There are different kinds of transforms that are classified as non-rigid transformations, e.g., *projective transform* [45], *quadratic and cubic polynomial* [15], and *thin plate splines* (TPS) [29], just to mention a few.

3.3 Estimation of the Difference

After estimating the transformation, and warping the image, the similarity between the static image and the warped image should be evaluated in order to assess the quality of the registration. This is a crucial step if image registration is done automatically, and not by manually selecting and placing landmarks. There are many choices for similarity estimators. The *sum of squared differences* is probably the most commonly used. It is defined $\forall \vec{x} \in S \cap W$ by the following equation:

$$\mathrm{SSD}(S, W) = \sum_{i}^{N} (S(\vec{x}) - W(\vec{x}))^2 \ , \tag{3.12}$$

where N is the number of points under analysis. The SSD is often normalized by dividing it by N, resulting in the *mean squared error* (MSE) as follows:

$$\mathrm{MSE}(S, W) = \mathrm{SSD}(S, W)/N \ . \tag{3.13}$$

The SSD and MSE measure the distances' difference between corresponding points in two images, thereafter, it is equal to 0 when two similar images are perfectly aligned and increases as misalignment increases. When the images to be registered differ only by geometry and/or Gaussian noise, the SSD is a well suited similarity measure [81].

3.4 Related Work

There are a variety of techniques for solving IR problems. This section presents a brief review of some of the most important ones, including both classical, as well as evolutionary computation and metaheuristic (MH) based approaches.

3.4.1 Classical Methods

Two state-of-the-art approaches from this category of methods are Robust Point Matching (TPS-RPM) [17], and Shape Context (SC) [8].

TPS-RPM is a method for matching two point-sets in a Deterministic Annealing (DA) setting. It uses a fuzzy-like matrix instead of a binary permutation matrix to find the matching between two sets of points. In TPS-RPM, both the point correspondences and the transformations are computed interchangeably. Therefore, RPM can be viewed as a general framework for point matching and can accept different transformation models [17] like affine, and even more complicated models like Thin Plate Splines (TPS) [29]. This method is a kind of a hybrid in the sense that it can be considered both a matching-based and a transformation-based approach for IR.

Shape Context (SC) is a matching-based approach that is usually used to estimate the transformation between two images, by finding matches between samples from the edges of the objects in the images. It basically consists of analyzing the spacial relationship between points. It uses four main parameters. The first defines the number of radial bins for the creation of the histograms, the second is the number of theta bins that defines how many slices the histograms should be divided into, and the third and fourth parameters, the minimum and maximum width of the bins, respectively. For more information on these and other classical IR methods, the reader is directed to [15, 18, 87].

3.4.2 EC and MH methods

Evolutionary Algorithms (EAs) and other metaheuristics (MHs) methods have been applied to solve IR problems. EAs and MHs are stochastic optimization methods which aim at finding a solution or a set of solutions that perform(s) best with respect to a certain objective(s). During the last decades these algorithms have been successful in solving a variety of search and optimization problems, and the domain of image registration has

been no exception. As opposed to the classical methods, which are typically based on gradient-based search, EAs and MHs tend to escape more easily from local optima and can be considered, in general, robust methods.

The first known application of evolutionary computation to image registration is due to Fitzpatrick et al. [33] who applied a genetic algorithm (GA) to relate angiographic images. For the subsequent 15 years or so, other EC approaches have been proposed by different authors, but most of them were based on the canonical GA with proportionate selection and a binary representation for solutions. Such a GA has severe limitations when solving optimization problems in the continuous domain, especially due to the problem of Hamming cliffs originated from the discretization of real valued variables into binary coded values, to the fixed precision that depends on the number of bits used for each decision variable, and for imposing lower and upper bounds for a variable's value. Moreover, it is known for several years that fitness proportionate selection methods have several drawbacks when compared to ordinal-based selection methods such as ranking, tournament, or truncation selection [41]. Nonetheless, most of the early EC approaches for IR used such kind of GA setup [79, 80, 38, 85, 86]. Another limitation of the early approaches was that they only dealt with translation and rotation [33, 38, 47, 85], ignoring scaling, stretching, and shearing.

Most modern EC applications to IR use a direct real coded representation of solutions [66, 47, 39, 16, 19, 84, 78]. Besides EC, other MH approaches have been applied to IR, namely Tabu Search [82], Particle Swarm Optimization [83], Iterated Local Search [21], and Scatter Search [20, 70] just to name a few. A detailed review of these works cannot be made in this thesis, but the interested reader can consult recent surveys on the topic [22, 69].

Chapter 4

Application of Evolutionary Algorithms to Image Registration

4.1 Introduction

This section introduces a real coded genetic algorithm for the optimization of the parameters of an affine transformation for the case of 2-D images. The proposed algorithm is a transformation parameters-based approach, since we are performing a direct search for the parameters that define the registration transformation. For the sake of simplicity, we assume we have two 2-D synthetic point-sets representing features from the two images. In other words, it is assumed that the feature detection step of the IR pipeline has been solved beforehand. The problem undergoes two different evolutionary algorithms, ES and GA. In the next sections, the representation, the operators, and the objective function that were used in this study are described.

In this chapter, the new noisy objective function is proposed (Sect. 4.3). Sect. 4.4 presents a real coded GA formulation and operators for the IR problem. The experimental results of the GA formulation are presented and discussed in Sect. 4.4.1. Once the ES is more used for continuous search space, it was also applied to the problem at hand.

Sect. 4.5 presents an ES representation and operators for the problem, and the ES results are presented in Sect. 4.5.1. Sect. 4.6 discusses the results that are obtained from GA and ES formulation for IR.

4.2 Representation

The representation is straightforward. For the 2-D case, the affine transformation is defined by six parameters, $\theta_0 \ldots \theta_5$, as explained in Subsect. 3.2.1.1.1. A candidate solution for the GA is therefore represented by a chromosome vector with six genes, each a real number.

4.3 Objective Function

In order to guide the search for an appropriate set of parameters for the affine transformation, we need to measure the proximity between the static and the warped image (the deformed image after the affine transformation is performed). The closer the two images are, the better the affine transformation is. Since each image is represented by a point-set, we need a way to find the similarity between two point-sets. To do so we first find a correspondence between points in the warped and static images. Once the correspondence is obtained, the objective function value is the weighted similarity of the two point-sets using the Euclidean distance of the matched points.

Algorithm 4.1 gives details of the steps involved in evaluating a candidate solution. In the algorithm, upper case letters denote matrices or vectors, and lower case letters with subscripts denote a specific element of the matrix or vector. The next paragraphs describe the major steps of Algorithm 4.1.

To compute the objective function value of a candidate solution, we start by warping the deformed image D according to the parameters of the affine transformation specified in the candidate solution C, yielding a new point-set W (line 1 of Algorithm 4.1). Then

Algorithm 4.1 Objective function

Input: S, D, C

/* S is the static image points */

/* D is the deformed image points */

/* C is a chromosome */

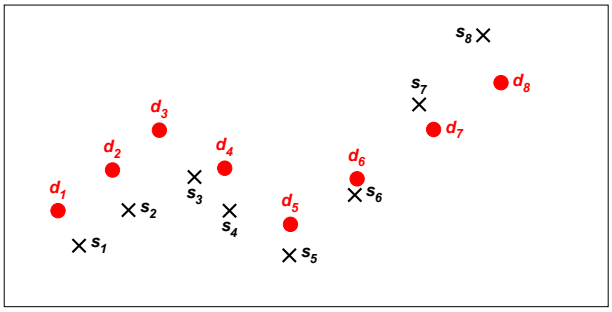
Output: Objective function value

$W \leftarrow T(D, C)$;
/* Euclidean distance between the point-sets */
for all $i \in \{1, \ldots, n\}$ **do**
  for all $j \in \{1, \ldots, k\}$ **do**
    $\delta_{ij} \leftarrow \|w_i, s_j\|$;
  end for
end for
/* Initialize correspondence matrices */
for all $i \in \{1, \ldots, n\}$ **do**
  for all $j \in \{1, \ldots, k\}$ **do**
    $m'_{ij} \leftarrow 0$;
    $m''_{ij} \leftarrow 0$;
  end for
end for
/* Find the closest non-assigned point */
/* O is the matched-order vector */
for all $i \in O$ **do**
  $j \leftarrow \mathrm{W2S}(\Delta, M', i)$; // $(W \rightarrow S)$
  $M'_{ij} \leftarrow 1$;
end for
for all $j \in O$ **do**
  $i \leftarrow \mathrm{S2W}(\Delta, M'', j)$; // $(S \rightarrow W)$
  $M''_{ij} \leftarrow 1$;
end for
$Q = M' + M''$;
/* Calculate weights */
for all $i \in \{1, \ldots, n\}$ **do**
  for all $j \in \{1, \ldots, k\}$ **do**
    if $q_{ij} \neq 0$ **then**
      $q^*_{ij} \leftarrow q^{-1}_{ij}$;
    else
      $q^*_{ij} \leftarrow 0$;
    end if
  end for
end for
/* Weighting matches */
for all $i \in \{1, \ldots, n\}$ **do**
  for all $j \in \{1, \ldots, k\}$ **do**
    $m_{ij} \leftarrow m'_{ij} \times q^*_{ij}$;
  end for
end for
$fitness \leftarrow 0$;
for all $i \in \{1, \ldots, n\}$ **do**
  for all $j \in \{1, \ldots, k\}$ **do**
    $fitness \leftarrow fitness + (m_{ij} \times \delta_{ij})$;
  end for
end for
return $fitness$;

the matching of points in W into points in S is modeled using a correspondence binary matrix $M(n, k)$ based on the closest-point rule, where n and k correspond to the number of points in the warped and static images, respectively. The closest-point is measured using the Euclidean distance between matched points. Each point in any set corresponds, at most, to one point in the other set. To find the correspondence for each point, the closest point in the other set is chosen. If the nearest point has already been assigned to another point, the next non-assigned nearest point is chosen. This procedure is performed once to find the correspondence matrix $M'(n, k)$ from the warped set to the static set ($W \rightarrow S$), and then a second time to find the correspondence matrix $M''(n, k)$ from the static set to the warped set ($S \rightarrow W$). This is achieved by lines 3–24 of Algorithm 4.1. Figure 4.1 shows the possible M' and M'' matrices of two different point-sets, as well as the calculated correspondence.

The order in which the correspondence points are found (lines 17 and 21 of Algorithm 4.1), plays a vital role in the resulting correspondence matrix. A match-order vector is proposed to specify the order in which the points of a given set are visited when finding its closest-point match from the other set ($W \rightarrow S$ and $S \rightarrow W$). For different orderings, different correspondences may be found. Therefore, the match-order vector is randomly created in each generation. This makes the evaluation of a candidate solution a somewhat noisy process. In a given generation, two identical solutions obtain the same objective function value. But the same thing is not necessarily true for two identical solutions from different generations. For instance in Fig. 4.1, if the order s_8, s_5, s_3, s_2, s_1, s_6, s_7, s_4 is used instead of s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, then, the resulting M'' will be different as shown in Fig. 4.2.

Fortunately, GAs are well known for being able to handle well noisy fitness evaluations due to the processing of a population of solutions. Note that we could have used a fixed pre-determined ordering for all evaluations but decided not to do so because the point matching procedure would be somewhat biased with respect to the used ordering.



	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
d_1	1	0	0	0	0	0	0	0
d_2	0	1	0	0	0	0	0	0
d_3	0	0	1	0	0	0	0	0
d_4	0	0	0	1	0	0	0	0
d_5	0	0	0	0	1	0	0	0
d_6	0	0	0	0	0	1	0	0
d_7	0	0	0	0	0	0	1	0
d_8	0	0	0	0	0	0	0	1

$M'(n,k)$
$D \longrightarrow S$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
d_1	1	0	0	0	0	0	0	0
d_2	0	1	0	0	0	0	0	0
d_3	0	0	0	0	0	0	0	1
d_4	0	0	1	0	0	0	0	0
d_5	0	0	0	1	0	0	0	0
d_6	0	0	0	0	1	0	0	0
d_7	0	0	0	0	0	1	0	0
d_8	0	0	0	0	0	0	1	0

$M''(n,k)$
$S \longrightarrow D$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
d_1	2	0	0	0	0	0	0	0
d_2	0	2	0	0	0	0	0	0
d_3	0	0	1	0	0	0	0	1
d_4	0	0	1	1	0	0	0	0
d_5	0	0	0	1	1	0	0	0
d_6	0	0	0	0	1	1	0	0
d_7	0	0	0	0	0	1	1	0
d_8	0	0	0	0	0	0	1	1

$Q(n,k)$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
d_1	.5	0	0	0	0	0	0	0
d_2	0	.5	0	0	0	0	0	0
d_3	0	0	1	0	0	0	0	1
d_4	0	0	1	1	0	0	0	0
d_5	0	0	0	1	1	0	0	0
d_6	0	0	0	0	1	1	0	0
d_7	0	0	0	0	0	1	1	0
d_8	0	0	0	0	0	0	1	1

$Q^*(n,k)$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
d_1	.5	0	0	0	0	0	0	0
d_2	0	.5	0	0	0	0	0	0
d_3	0	0	1	0	0	0	0	0
d_4	0	0	0	1	0	0	0	0
d_5	0	0	0	0	1	0	0	0
d_6	0	0	0	0	0	1	0	0
d_7	0	0	0	0	0	0	1	0
d_8	0	0	0	0	0	0	0	1

$M(n,k)$
$D \longrightarrow S$

Figure 4.1: An example of the correspondence matrices: points s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8 correspond to d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, respectively, for mapping point-sets D to S. However, if mapping S to D, points s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8 correspond to d_1, d_2, d_4, d_5, d_6, d_7, d_8, d_3, respectively.

At this point (line 24 of Algorithm 4.1), matrices M' and M'' specify the point-matchings from $W \rightarrow S$ and $S \rightarrow W$, respectively. We then obtain matrices Q and Q^*. Q is simply the sum of M' and M'', thus each q_{ij} can have a value of 2, 1, or 0, depending on whether point i matches point j in both directions, in a single direction, or has no match at all. Matrix Q^* is obtained from Q by inverting the non-zero elements.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
d_1	1	0	0	0	0	0	0	0
d_2	0	1	0	0	0	0	0	0
d_3	0	0	0	1	0	0	0	0
d_4	0	0	1	0	0	0	0	0
d_5	0	0	0	0	1	0	0	0
d_6	0	0	0	0	0	1	0	0
d_7	0	0	0	0	0	0	1	0
d_8	0	0	0	0	0	0	0	1

$M''(n,k)$
$S \longrightarrow D$

Figure 4.2: Correspondence matrix $M''(S \rightarrow R)$: based on new match-order vector; points s_8, s_5, s_3, s_2, s_1, s_6, s_7, s_4 correspond to points d_8, d_5, d_4, d_2, d_1, d_6, d_7, d_3.

Finally, matrix M is calculated from the element-wise multiplication of matrices M' and Q^*. Fig. 4.1 further illustrates matrices Q, Q^*, and M.

The objective function is based on the weighed similarity of two point-sets using the Euclidean distance of the matched points. The points that are connected exclusively from one direction (either $W \rightarrow S$ or $S \rightarrow W$) are penalized, and those that are connected in both directions are given half weight in terms of Euclidean distance. In other words, if the connection exists in both directions the objective function value decreases.

This objective function is very similar to the one used by Seixas et al. [78]. The main difference is the use of a newly generated matched order vector in each generation, which makes the point-matching procedure less dependent on a fixed ordering of visiting the points.

4.4 GA Operators

With respect to the GA variation operators, we use Simulated Binary Crossover (SBX) proposed by Deb and Agrawal [27] (Sect. A.7.1) and Gaussian mutation (Sect. 2.5.2.2). The utilization of SBX crossover and Gaussian mutation is a natural choice because the problem has a continuous search space. SBX uses a probability distribution to create the offspring, and it does so by biasing the offspring to be created near the parents. SBX

is a *parent-centric* recombination operator because the offspring it produces are located around the parents. This behavior contrasts with *mean-centric* recombination operators whose offspring are located at the center of mass of parents.

It has been shown that *parent-centric* operators have in general a better performance than *mean-centric* operators [24]. This has motivated our choice of SBX as a crossover operator. Surprisingly, none of the real coded GAs proposed in the literature for addressing the IR problem have used SBX or other parent-centric crossover operators. Instead most used recombination operators such as uniform [66, 16], arithmetic [47] and blend crossover [19].

4.4.1 GA Experimental Results

This section describes the experimental results from testing the proposed GA formulation. Five point-sets available at `http://noodle.med.yale.edu/~chui/rpm/TPS-RPM.zip` are used. Each set is composed at most by 105 points. They include the deformed and static points' locations. The deformed points were generated from the static ones by a non-affined (i.e. free-form) transform. This means that it will not be possible to obtain a perfect matching of the images by using an affine transformation model alone. All point coordinate values have a precision of 16 floating point.

The GA setup was the same for all data sets. Most parameter settings were tuned beforehand, and held fixed for all the experiments. We use tournament selection without replacement of size 5, SBX crossover with distribution index 2 [27], and Gaussian mutation with mean zero and standard deviation 1/3 for all the genes. The crossover probability was set to 1.0 and each gene undergoes SBX with probability 0.5. For replacement we use a *replace worst* strategy, with the worst half of the individuals of the current population being replaced by the best half of the newly generated solutions. This replacement strategy makes the GA elitist, never losing the best solution found so far. The GA ran for 500 iterations. The experiments were performed with populations of

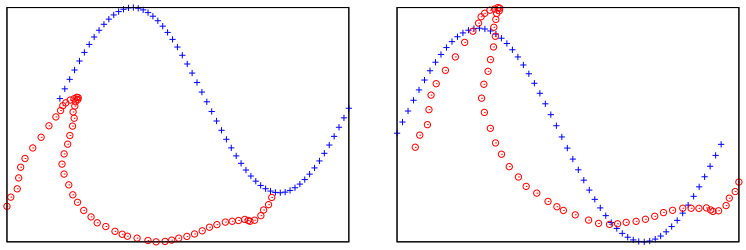
size 30, 60, 120, 240, and 480 individuals, and for each size, 100 independent runs were executed.

According to population sizing theory of GAs [44], larger populations sizes tend to produce a better solution quality, but also at the expense of more processing time. Figure 4.4 shows the objective function value of the best individual in the population at every generation, averaged over the 100 runs, for the various population sizes and for the various point-sets. The performance behavior is more or less identical for all point-sets. We can observe a substantial progress for the first 50 generations, still some progress between generations 50–200, and from there on the improvements are minor. As expected, larger population sizes give better solution quality but the improvements are negligible for population sizes larger than 120.

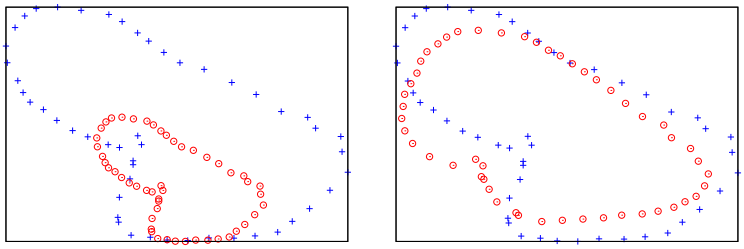
Figure 4.3 illustrates the five point-sets before and after warping. It should be noticed that these are sets where deformations are non-affine, therefore the resulting warped images will never match perfectly. Nevertheless, they present a very good approximation. When compared to the approach from [78], it is possible to observe that our results are more precise.

This can be seen visually by comparing Figures 4.3 and 4.6, and is especially obvious for point-set 4. For point-set 1 it is hard to assess which one is better. Point-set 3 is not comparable because the transformation was made (accidentally) in the reverse order in [78]. Experiments with point-sets 2 and 5 were not reported in [78].

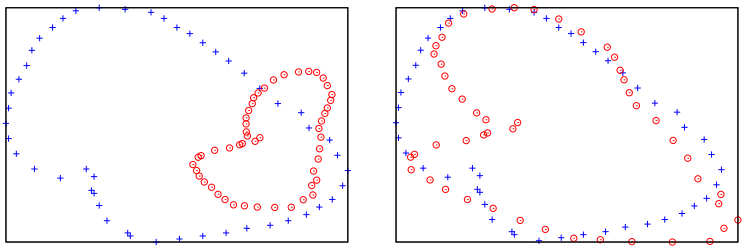
In order to assess the quality of the proposed approach in what concerns affine deformations, affine deformed images were generated from the same five static images, and used in subsequent experiments. The results were compared to those produced by well known classical state-of-the-art approaches such as SC and TPS-RPM, and are illustrated in Figure 4.5. It can be seen that the results produced by the proposed GA are slightly better than those of the SC and are very competitive with those obtained by TPS-RPM. Moreover, the resulting registered images can be observed in Fig. 4.5(a).



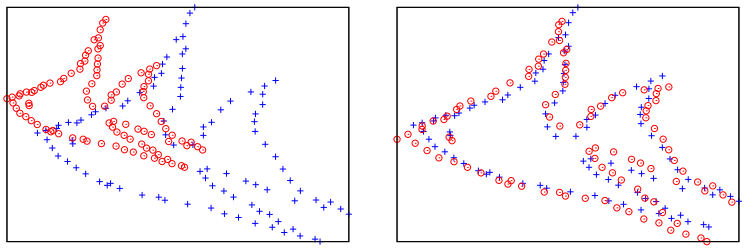
(a) point-set 1



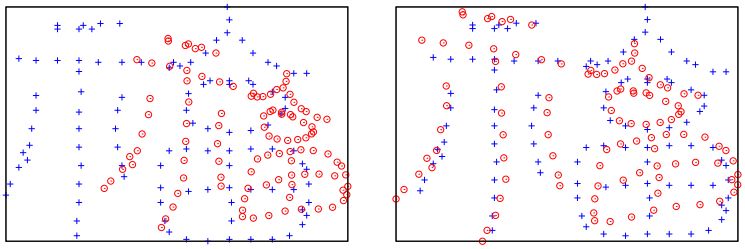
(b) point-set 2



(c) point-set 3

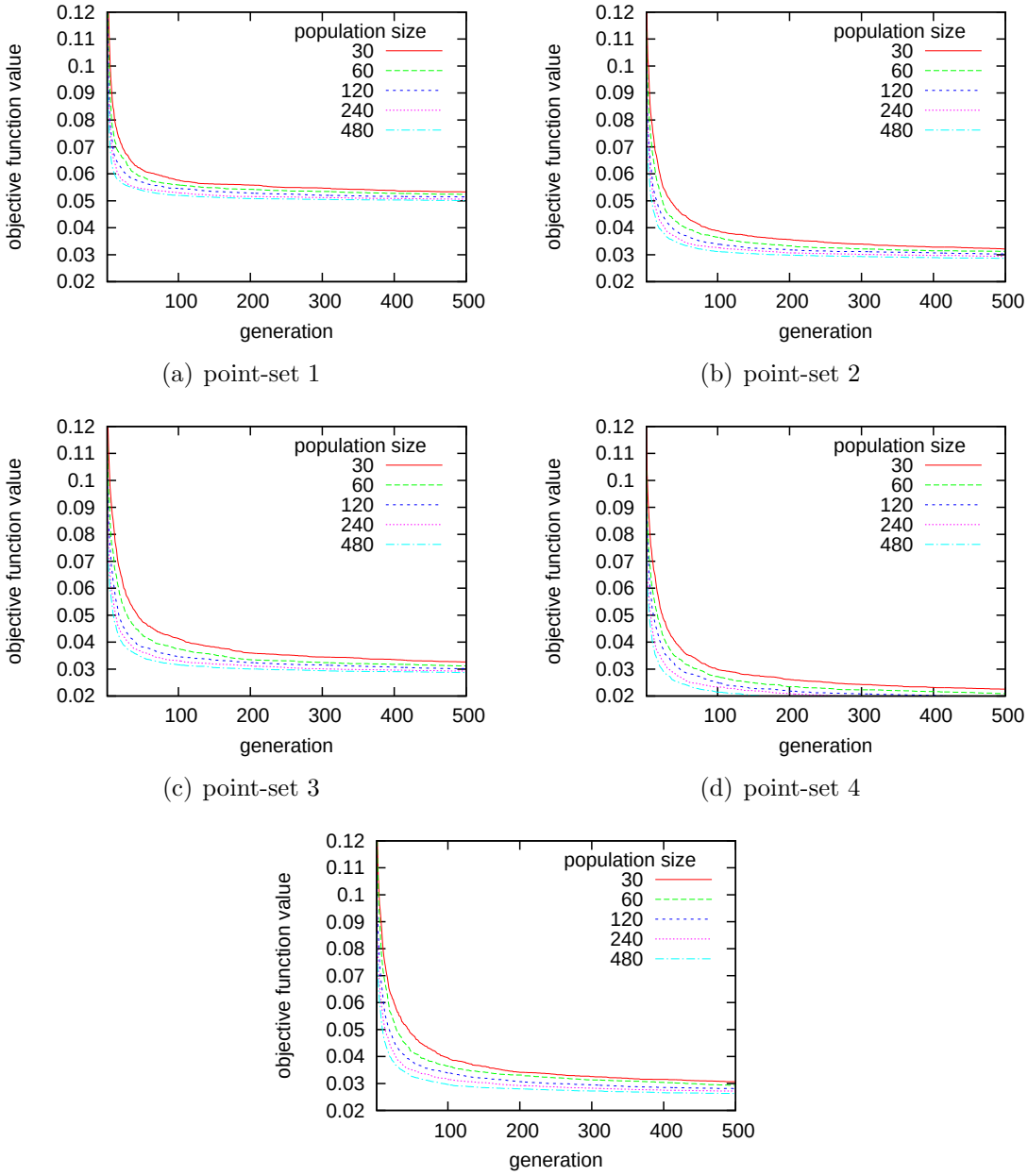


(d) point-set 4



(e) point-set 5

Figure 4.3: Non-affine distorted point-sets (left, blue dots for static image points, red dots for deformed image points) and GA affine image registration (right, red dots are the warped image points) results obtained after 500 generations using population size 120. The warped images are zoomed for better visualization. Note that even better matching could be obtained with larger population sizes, but the improvements are negligible as shown in Figure 4.4.



(a) point-set 1

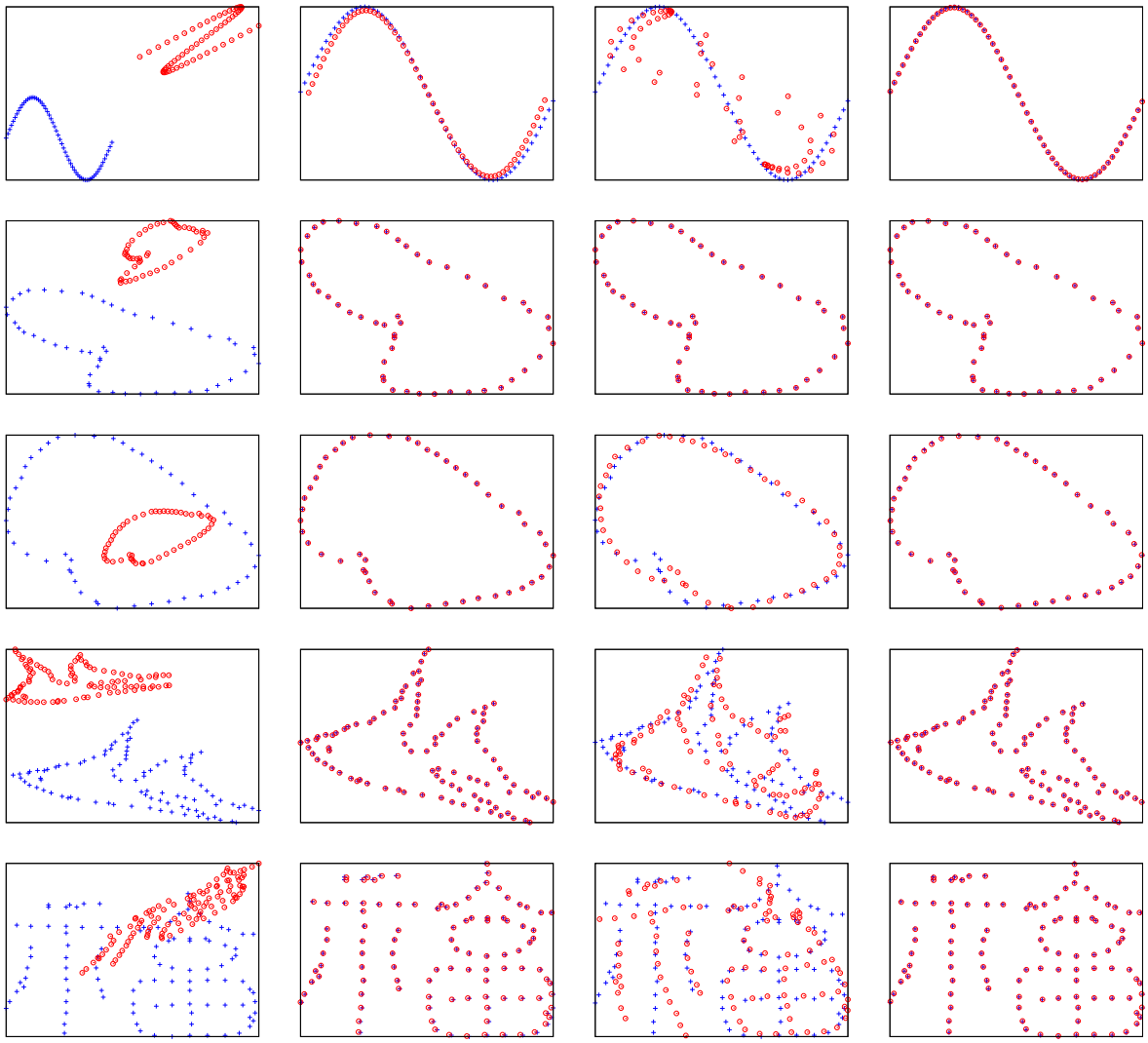
(b) point-set 2

(c) point-set 3

(d) point-set 4

(e) point-set 5

Figure 4.4: Best objective function value through generations for various population sizes obtained for various point-sets. The results are averaged over 100 independent runs.



(a) point-sets (b) GA (c) SC (d) TPS-RPM

Figure 4.5: Affine distorted point-sets and respective registration results. On the left column, (a), the blue dots are the static image points and the red dots are the deformed image points. On the other columns, (b), (c), and (d), the red dots are the warped image points. The warped images are zoomed for better visualization. The GA results were obtained after 500 generations using population size 120. Observe that for the case of GA and TPS-RPM, the deformed and static points are almost on top of each other, meaning that the match is almost perfect. For SC the results are slightly inferior compared with those obtained by the GA and TPS-RPM.

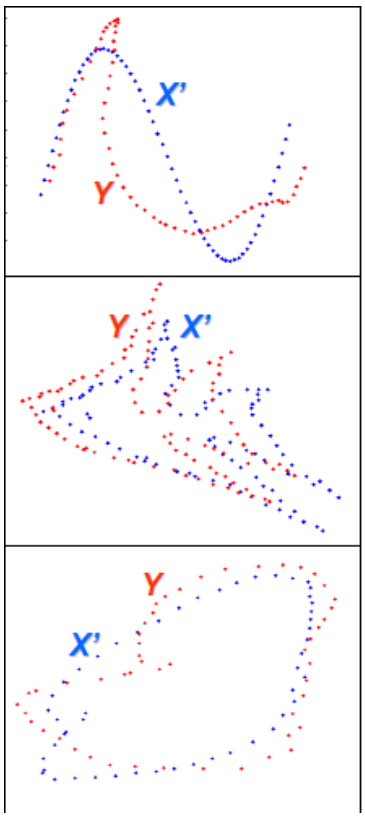


Figure 4.6: These images are obtained directly from [78]. The top, middle, and bottom images, correspond to our point-sets 1, 4, and 3, respectively. The blue dots are the static image points and the red dots are the warped image points.

4.5 ES Operators

Just as in GA representation (Sect. 4.4), a candidate solution for the ES is represented by a chromosome vector with six genes, each a real number. Each gene of the solutions, corresponds to one of the six parameters of the affine transformation (Subsec. 3.2.1.1.1). With respect to the self-adaptation, the number of strategy parameter(s), are chosen, that can either be 1 in the case of 1/5 success rule or uncorrelated mutation with 1 step-size, or 6 in the case of uncorrelated mutation with n step-size (see Sect. 2.6.6).

Several kinds of operators were used to study the ES behavior for this particular problem. Further ahead in this section these operators are listed.

Three different kinds of self-adaptation strategies are used 1/5 success rule, uncorrelated mutation with 1 step-size, and uncorrelated mutation with n step-size.

With respect to the ES variation operators, three different recombination and two different mutation operators that are suitable for the real-valued search spaces are tested. The recombination operators are intermediate recombination, discrete recombination, and SBX (Sect. 2.6.3). With respect to mutation operators, Gaussian and Polynomial mutation are used (Sect. 2.6.4). Finally, the survivor selection, (μ, λ) are $(\mu + \lambda)$ (Sect. 2.6.5), are examined.

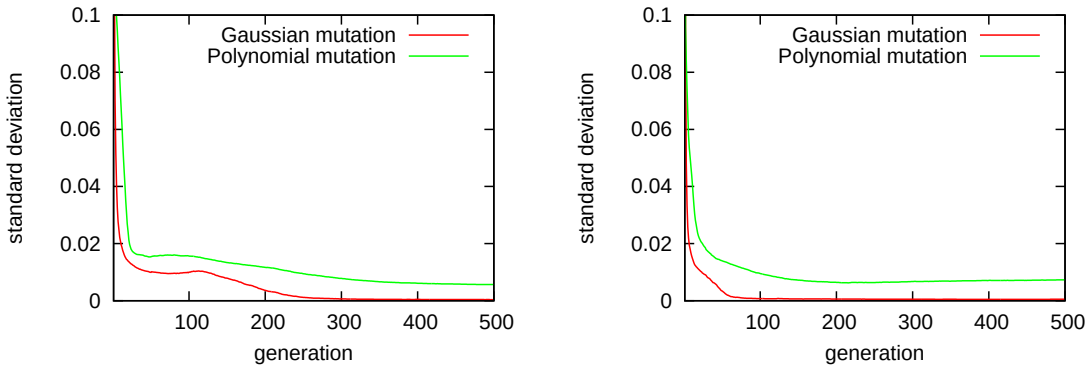
4.5.1 ES Experimental Results

This section presents the experimental results found by the ES for the five point-sets available at `http://noodle.med.yale.edu/~chui/rpm/TPS-RPM.zip`. The point-sets are the same to what was used in the GA experiments.

To find a reliable ES setup, different combination of parameter settings and operators are tested. These different combination make 210 various setup, and for each combination, 10 independent runs were executed. The best setup is almost identical for all point-sets.

The experiments were performed with populations of size 10, 20, 40, 80, and 160 individuals where according to Schwefel's empirical research, the ratio between the population size and the generated number of offspring, μ/λ, should be at least 1/7 [77]. So, for the above population sizes, the number of offspring are 70, 140, 280, 560, and 1120, respectively. As it was expected according to the population size theory [44], large population sizes obtain a better solution, however, with an increased processing time. Three different self-adaptation were used and the uncorrelated mutation with 1 step-size compared to other kinds of self-adaptation has a better performance for all point-sets.

The ES procedure starts with parent selection (Sect. 2.6.2), where the number of the parental individuals are drawn randomly with uniform distribution and using them as inputs of recombination operators. For the parent selection, three different sizes of 2, 4, and 8 for each point-set were used. Parent selection size of 2 has better performance



(a) point-set 4 (b) point-set 5

Figure 4.7: Standard deviation of the objective function value of the population members averaged over 100 runs with a $(160 + 1120)$-strategy using different mutation operators for point-sets 4 and 5. The other point-sets have similar behavior.

for all point-sets except point-set 4, in that, size 4 performs better. Both parts of individuals, object parameters and strategy parameters, undergo recombination operators. Intermediate recombination is used as a recommended operator for the strategy parameters [1, 31]. Among those recombination tested in this work, the discrete recombination has the best performance for all point-sets except point-set 1 where SBX recombination got better results. Two different mutation operators were studied for various point-sets, Gaussian mutation and polynomial mutation. Gaussian mutation has a better performance for all point-sets. Fig. 4.7 shows that the Gaussian mutation has less perturbation around results compared to the polynomial mutation, which makes the EA formulation more adequate for this particular problem.

Fig. 4.8 shows the objective function value of the best individual in the population of size 160 at every generation, averaged over the 100 runs, for various point-sets. Parameter settings for each point-set was tuned as explained before. The performance behavior is more or less identical for point-sets 1 and 4. A substantial progress for the first 200 generations can be observed and still some progress between generations 200-300, from there on the improvements are minor. Point-set 2 has a consequential progress for the first 100 generations. Point-sets 3 and 5 have almost the same behavior which have a

considerable progress for the first 50 generations.

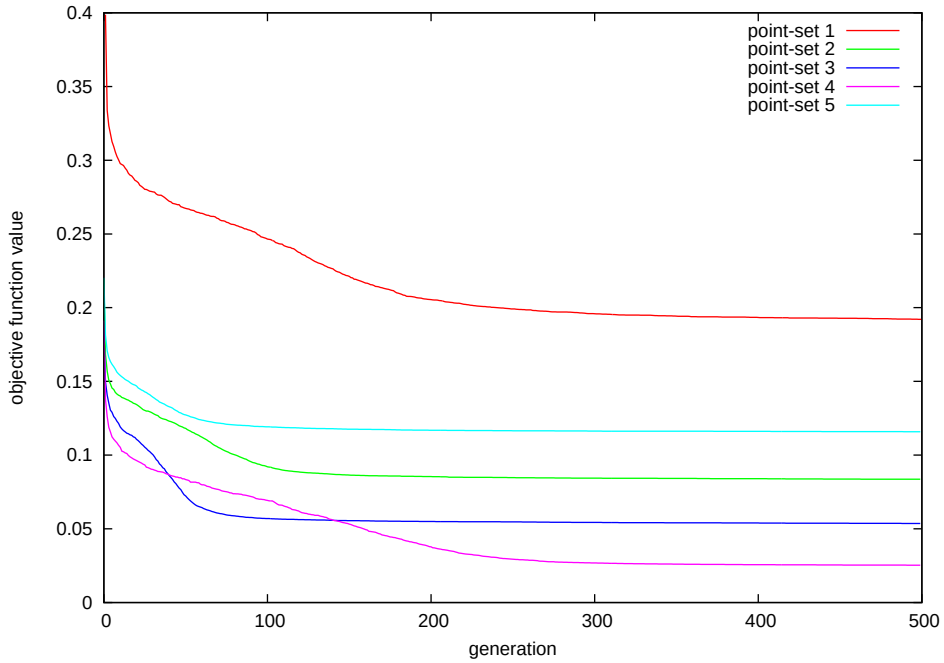
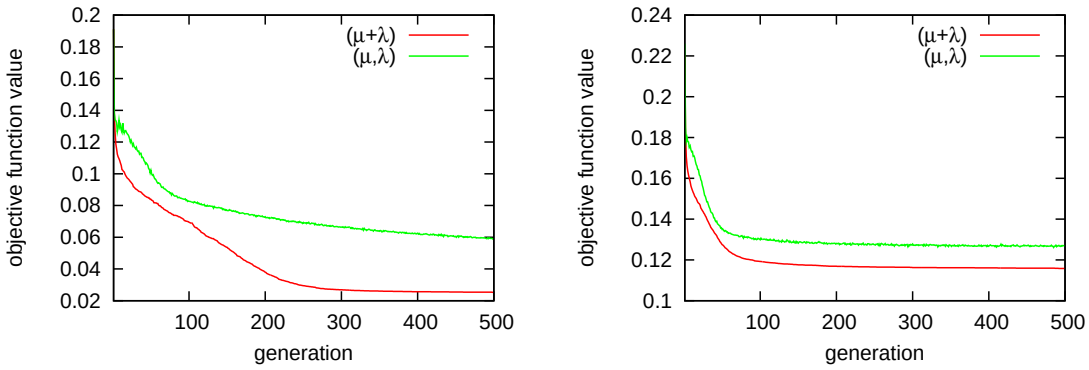


Figure 4.8: Best objective function value through generations for $(160 + 1120)$-strategy for various point-sets. The results are averaged over 100 independent runs.

There are two methods for survivor selection in ES, (μ, λ)-strategy and $(\mu + \lambda)$-strategy. On the contrary to literature where (μ, λ)-strategy is recommended [1, 31, 65, 76], for this particular problem $(\mu + \lambda)$-strategy had a better performance. The $(\mu + \lambda)$-strategy is kind of elitist selection, it chooses the best μ from the union of parents and offspring. Fig. 4.9 shows the behavior of the different survivor selection for point-sets 4 and 5. This behavior is identical for all other point-sets.

4.6 Discussion

The results of ES representation shows that the ES formulation for the IR is not promising when using large perturbation through individuals, since it had a better results when using discrete recombination and small parent selection size (size of 2). Being more efficient using Gaussian mutation rather than polynomial mutation, shows the formulation



(a) point-set 4 (b) point-set 5

Figure 4.9: Best objective function value through generations for $(160 + 1120)$-strategy and $(160, 1120)$-strategy for point-sets 4 and 5. The results are averaged over 100 independent runs.

is more promising by searching a wide space around the current solution than the perturbation through individuals or tied space around the solution. Considering what the results show, the uncorrelated mutation with 1 step-size has a significant higher standard deviation of objective function values of individuals in each generation.

In ES the strategy parameters are used to control the mutation perturbation and restrict that closely to the current solution through executions. Taking the results shown in Fig. 4.7 into account, it seems that decreasing the strategy parameter values are faster than the progressing of the objective function values. It leads the ES formulation for IR to end up having premature results. In contrast to ES approach, GA approach keeps the standard deviation of the mutation operator constant through executions.

Table 4.1: Minimum square errors of affine deformed point-sets

Point set	GA (best solutions found by GA and ES)	ES	SC	TPS-RPM
1	0.0193379395226184	0.1901773795341200	0.2171168813846202	0.0021012035584104
2	0.0028343261311595	0.0846977804666287	0.0000000000000014	0.0007527405941700
3	0.0016668807905935	0.0519722652700338	0.0962479821502197	0.0017388923566028
4	0.0013677616851059	0.0250264560914390	0.2057605902210515	0.0014588099414611
5	0.0046540173682426	0.1158731130896308	0.0450579276900900	0.0017750298978483

Table 4.1 shows the resulting minimum square errors (MSEs) after applying ES as

well as GA, SC, and TPS-RPM to the set of affine deformed images, represented in Fig. 4.5(b), 4.5(c), and 4.5(d), respectively.

4.7 Summary

This chapter proposed a new noisy objective function for the image registration. Two well-known types of the evolutionary algorithms were used to test the proposed objective function, ES and GA. The resulting algorithm was applied to 2-D synthetic point-sets, with deformed images' points obtained from both affine and non-affine transformation. Although the ES is basically designed to work on continuous search spaces, for this particular problem the GA presents better performance (Table 4.1).

The GA formulation uses Simulated Binary Crossover (SBX), and Gaussian mutation produced good results compared to the classical methods as well as ES. Results are superior when compared to those presented in [78]. Furthermore, GA experimental results show that the proposed approach achieves a very precise registration. When compared to other approaches, in general, ours present a smaller MSEs. The GA results will soon be published in [7].

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Image Registration is the process of aligning two or more images of the same scene taken at different times, from different directions, and/or by different sensors, by finding the best mapping function between them [15, 87]. There are many complicated image transforms such as polynomial transforms, however, in many practical problems, usually for the sake of simplicity, only affine transforms are used for image registration. For example, for the registration of X-ray images of bones, one may consider only rigid transformations. Even when non-rigid transforms are applied, in the majority of cases the images are first registered through affine transforms and only then do the non-rigid transforms apply, otherwise the elastic transform may be extremely complex [28]. Therefore, affine transform is chosen in this work to do image registration. Most research in this area is based on classical algorithms and methods, but during the past two decades or so, there has been a growing interest in the application of Evolutionary Computation (EC) and other Metaheuristic (MH) methods to solve the problem.

Evolutionary Algorithms (EAs) are stochastic optimization methods inspired by principles of natural selection and genetics. The goal of EAs is to find a solution or a set of

solutions that perform(s) best with respect to an objective function.

This work proposed a real coded EA that is especially suited for doing image registration of affine distorted images. Furthermore, a scheme called match-order vector to increase the randomness of point selection and prevent from getting trapped into local minima, is proposed. To study the behavior of the designed method, two types of evolutionary algorithms, GA and ES, are used. Although ES is basically designed for working in continuous search space, it didn't work as well as a real coded GA formulation.

As opposed to previous EC approaches for solving IR, our GA method uses Simulated Binary Crossover (SBX), a parent-centric recombination operator that has been giving good results on a variety of continuous real world optimization problems within a GA framework. The use of a randomized ordering when visiting points during the point-matching procedure was also proposed, and although this technique yields a noisy fitness function evaluation, the results obtained show that the GA is capable of dealing with it quite well.

The resulting algorithm was applied to 2-D synthetic point-sets, with deformed images points obtained from both affine and non-affine transformations. For the case of non-affine distorted points, our method produces a more precise registration than previously published results by means of an evolutionary algorithm on the same point-sets [78]. For the case of affine distorted points, the proposed real coded GA produced better results than shape context (SC), and is competitive with TPS-RPM, two well known classical state-of-the-art image registration methods. Although those classical methods are further able to transform non-affine distorted images, we can compare our approach with those methods just in the case of affine deformation where the proposed EA formulation is designed for that. Results of the GA application has been accepted as a publication at GECCO 2012 [7].

5.2 Future Work

As experimental results show, both GA and ES were capable of dealing with this problem very well and are competitive to other advanced methods, however, there is still room for improvement. The work can be extended in both image processing and evolutionary computation.

The presented approach is specialized only for affine transformation. The representation and objective function can be modified in such way that it covers non-affine (free-form) transformation such as the polynomial transform. Two well-known forms of the polynomial transform are *quadratic* and *cubic* transforms. Something else that can be done is to explore the image processing field deeper in order to be able to do complete experiment (four steps of doing image registration) from beginning to the end, whereas, here the main effort was to find the mapping function, the third step of the image registration procedure (Sect. 3.1).

There are also some future studies in the evolutionary computation part that would be done. Recently it has been suggested that it can be beneficial to use a modified version of SBX which adaptively shifts between a parent-centric and a mean-centric recombination operator, by using population statistics gathered during the execution of the optimization run [6]. As an alternative to a GA and ES approaches, a modern evolution strategy such as CMA-ES [43] is also likely to deliver very good results for this type of problem.

Although the provided evolutionary strategies framework is generic and includes several operators for binary, integer, and real-valued representations, it can still be extended with more operators, and spread in the Internet with a manual for that.

Appendix A

Framework of evolutionary Strategies

A.1 Introduction

The ESs was introduced in Chapter 2. Here, we give details about algorithms, operators, and components that are needed in terms of implementing a generic framework of evolutionary strategies. As there is not any available ES framework in the Internet a generic framework of standard ES is designed. The framework covers mixed binary, integer, and real-valued representations and has more than a 5000 lines code. It is written in object-oriented C++ and available in `http://www.deei.fct.ualg.pt/~a38477/es.tar.gz`.

This appendix is organized as follows. ESs representation is the subject of Sect. A.2. The standard $(\mu/\rho \dagger \lambda)$–ES algorithm is outlined in Sect. A.3. Self adaptation is elaborated in Sect. A.4. Sects. A.5 and A.6 are devoted to initialization and parent selection, respectively. Finally, different variation operators and their algorithms are addressed.

A.2 The ESs Representation

ESs are typically used for continuous parameters optimization, but they can also be applied to problems with binary and integer search spaces. The usual goal of an evolution strategy is to optimize the given *objective function*, F, with respect to a set of *object parameters*. Standard representation of the object parameters of the n-dimensional search points, $\vec{x} = \langle x_1, \ldots, x_n \rangle$ is very straightforward, when each x_i is represented by any data type.

$$F(\vec{x}) \longrightarrow \text{optimization}, \quad x_i \in \mathcal{X} \ . \tag{A.1}$$

In principle, let $\mathcal{X}$ be any data type of finite but not necessarily fixed length [12]. The binary n-dimensional search space $\mathbb{B}^n$, the integer search space $\mathbb{Z}^n$, and the real-valued search space $\mathbb{R}^n$ are examples for $\mathcal{X}$. In contemporary ESs, the problem at hand can have n-dimensional object parameters with a mixture of various data structures [77]. In general, the objective function can be defined as follow:

$$F: \ \mathcal{X}^n \longrightarrow \mathbb{R}, \quad \mathcal{X} \in \{\text{any data type }\} \ . \tag{A.2}$$

Nowadays, ESs use self-adaptation almost entirely: they are used to control certain statistical properties of the genetic operators. The vector $\vec{x} = \langle x_1, \ldots, x_n \rangle$ forms only part of a typical ES genotype, but usually, individuals contain a set of endogenous (i.e. evolvable) *strategy parameters*, in particular, parameters of the mutation operator. Strategy parameters, σ value(s), represent the *mutation step-size*, and their number n_σ is usually either 1 or n. For any reasonable self-adaptation mechanism at least one σ must be presented [31].

Individuals with mixture data structures have a separated set of strategy parameter(s) for each single data type. The general form of individuals in ES for a mixture data structure of binary, integer, and real-valued with different dimensional search space is

represented by:

$$\underbrace{\langle \underbrace{b_1, \ldots, b_{n_b}}_{\vec{b}}, \underbrace{z_1, \ldots, z_{n_z}}_{\vec{z}}, \underbrace{r_1, \ldots, r_{n_r}}_{\vec{r}}, \underbrace{\varrho_1, \ldots, \varrho_{n_\varrho}}_{\vec{\varrho}}, \underbrace{\varsigma_1, \ldots, \varsigma_{n_\varsigma}}_{\vec{\varsigma}}, \underbrace{\sigma_1, \ldots, \sigma_{n_\sigma}}_{\vec{\sigma}} \rangle}_{\text{object parameters}} \underbrace{\qquad\qquad}_{\text{strategy parameters}} . \tag{A.3}$$

An ES individual shown in (A.3) can be represented as a 6-tuple $\vec{a}$ of the following form:

$$\vec{a} = (\vec{b}, \vec{z}, \vec{r},\ \vec{\varrho}, \vec{\varsigma}, \vec{\sigma}) \ . \tag{A.4}$$

where:

$\vec{b} = \langle b_1 \ldots, b_{n_b} \rangle$	n_b binary object parameters;
$\vec{z} = \langle z_1 \ldots, z_{n_z} \rangle$	n_z integer object parameters;
$\vec{r} = \langle r_1 \ldots, r_{n_r} \rangle$	n_r real-valued object parameters;
$\vec{\varrho} = \langle \varrho_1 \ldots, \varrho_{n_\varrho} \rangle$	n_ϱ step-size(s) for binary object parameters;
$\vec{\varsigma} = \langle \varsigma_1 \ldots, \varsigma_{n_\varsigma} \rangle$	n_ς step-size(s) for integer object parameters;
$\vec{\sigma} = \langle \sigma_1 \ldots, \sigma_{n_\sigma} \rangle$	n_σ step-size(s) for real-valued object parameters.

$\vec{\varrho}$, $\vec{\varsigma}$, and $\vec{\sigma}$ form the strategy parameters set of an individual and are used in mutation of its various types of object parameters (see Sect. A.4).

A.3 The Standard $(\mu/\rho \overset{+}{,} \lambda)$–ESs Algorithm

The general algorithm frame of ESs is developed by Schwefel in 1981 [35]. It uses multiple parents and offspring in the ESs' procedure. Two main approaches were explored, denoted by $(\mu+\lambda)$–ES and (μ, λ)–ES. Those refer to an ES parametrized according to the relation $1 \leq \mu < \lambda < \infty$ [68]. Within one ES generation step, λ individuals (offspring) are generated from the set of μ parent individuals.

The $(\mu/\rho \overset{+}{,} \lambda)$–ES notation expressed the way the offspring population is generated. The specific strategy parameters μ, λ, and ρ are kept constant during the evolution run. These parameters are so-called *exogenous strategy parameters*. The ρ refers to the number of parents involved in the procreation of *one* or *two* — in simulated binary crossover (SBX) recombination, ρ is 2 and it produces 2 offspring (see Sect. A.7.1) — offspring. There is a special ES case without recombination for $\rho = 1$. All other cases ($\rho > 1$) are strategies with recombination. The "$\overset{+}{,}$" in the notation refers to the used survival selection type, i.e., $(\mu + \lambda)$–ES and (μ, λ)–ES.

The outline of the $(\mu/\rho \overset{+}{,} \lambda)$–ES is given in Algorithm A.1. In what follows of this section a brief description of the algorithm is given.

The algorithm starts with generation zero (line #1), when all individuals of the parental population are initialized through lines #2-5 (see Sect. A.5). Each initialized individual is evaluated by means of an objective function (line #4). After initialization a while-loop is entered (lines #6-18) to generate the next parental population through produced offspring. This loop — which is generating new trials and selecting those with least error — continues until a sufficiently good solution (or solutions) is reached or the available computation is exhausted (line #6). From parental population, $P^{(g)}$, at generation g a new offspring population, $O^{(g)}$, is produced by running variation operators λ times through lines #7-12. Each iteration of this interior loop generates one or two offspring, depends on the selected recombination. Nonetheless, the Algorithm A.1 is designed for generating one offspring at a time. However, for the SBX recombination, the interior loop should be run only $\lambda/2$ times — in this case λ should be even. In the *marriage* step, line #8, a direct ancestors, S of size ρ is randomly chosen from the parent population size μ. Recombination of the strategy parameter(s) as well as recombination of the object parameter(s) take place in line #9. Whereas the recombination of the strategy parameter(s) does not have any effect on the recombination of the object parameter(s), their application order can be exchanged. The mutation which is included in the strategy

Algorithm A.1 Outline of the $(\mu/\rho \dagger \lambda)$–ES

Input: μ, ρ, λ, A problem at hand with an objective function f to optimize

/* μ Population size */

/* ρ Parent selection size */

/* λ Offspring size */

Output: A solution or a set of solutions

1: $g \leftarrow 0$;
2: **for all** $i \in \{1, \ldots, \mu\}$ **do**
3: $P_i^{(g)} \leftarrow$ `initialize`$(n_b, n_z, n_r, n_\varrho, n_\varsigma, n_\sigma)$;
4: `evaluate`$(P_i^{(g)}, n_b, n_z, n_r)$;
5: **end for**
6: **while** `termination-condition` = **false do**
7: **for all** $i \in \{1, \ldots, \lambda\}$ **do**
8: $S \leftarrow$ `marriage`$(P^{(g)}, \rho)$;
9: $O_i^{(g)} \leftarrow$ `recombine`$(S, \rho, n_b, n_z, n_r, n_\varrho, n_\varsigma, n_\sigma)$;
10: $O_i^{(g)} \leftarrow$ `mutate`$(O_i^{(g)}, n_b, n_z, n_r, n_\varrho, n_\varsigma, n_\sigma)$;
11: `evaluate`$(O_i^{(g)})$;
12: **end for**
13: **switch** `selection-type` **of**
14: **case** $(\mu\ ,\ \lambda)$**:** $P^{(g+1)} \leftarrow$ `selection`$(O^{(g)}, \mu)$;
15: **case** $(\mu + \lambda)$**:** $P^{(g+1)} \leftarrow$ `selection`$(P^{(g)}, O^{(g)}, \mu)$;
16: **end switch**
17: $g \leftarrow g + 1$;
18: **end while**

and object parameters' mutations, is done in line #10. In order to ensure the correctness of self-adaptation, the mutation of the strategy parameter(s) should be done before the mutation of the object parameter(s) [12]. After having a complete offspring population $O^{(g)}$, survival selection, in lines #13-16, is performed resulting in a new parent population $P^{(g+1)}$.

A.4 Self-Adaptation

There are two evolutionary operators, recombination and mutation, that are applied to the strategy parameters. The order of applying recombination operators on the object parameters and strategy parameters is not important while the mutation operator should apply firstly on the strategy parameters and then object parameters. The recombination operators that are applicable to the strategy parameters [11] are the same as the recombination operators of the real-valued objective parameters.

A.4.1 1/5 Success Rule

1/5 self-adaptive success rule is introduced in Sect. 2.6.6.1. Algorithm A.2 shows the effect of the 1/5 success rule on the $(\mu/\rho \dagger \lambda)$–ES (Algorithm A.1).

A.4.2 Uncorrelated Mutation with One or n Step-Size(s)

Uncorrelated mutation with one and n step-size(s) are introduced in Sect. 2.6.6.2. Following this subsection the mutation of the strategy parameter(s) in binary, integer, and real-valued search spaces for both uncorrelated mutation with one step-size as well as n step-sizes, are described.

A.4.2.1 Mutation Rate in Binary Search Space

In ES, those chromosomes that involve binary search spaces have a common feature, they have a probabilistic bit-flip scheme as mutation operator, where each bit is mutated with a certain probability ϱ_i. The operator used for mutating the strategy parameters of the variables with the binary search spaces is

$$\varrho_i' = \frac{1}{1 + \frac{1-\varrho_i}{\varrho_i} \times \exp(\gamma \times \mathcal{N}(0,1))} \tag{A.5}$$

Algorithm A.2 Outline of the $(\mu/\rho \stackrel{+}{,} \lambda)$–ES with 1/5 success rule

Input: μ, ρ, λ, c, q, A problem at hand with an objective function f to optimize

/* μ is population size */

/* ρ is parent selection size */

/* λ is offspring size */

/* c is adjustment factor, $c = 0.85$ is suggested by Schwefel [77] */

/* q is restart frequency*/

Output: A solution or a set of solutions

$g \leftarrow 0$; $counter \leftarrow 0$; $reset \leftarrow 0$;
for all $i \in \{1, \ldots, \mu\}$ **do**
  $P_i^{(g)} \leftarrow$ `initialize`$(n_b, n_z, n_r, n_\varrho, n_\varsigma, n_\sigma)$;
  `evaluate`$(P_i^{(g)}, n_b, n_z, n_r)$;
end for
while `termination-condition` = **false do**
  for all $i \in \{1, \ldots, \lambda\}$ **do**
    $S \leftarrow$ `marriage`$(P^{(g)}, \rho)$;
    $O_i^{(g)} \leftarrow$ `recombine`$(S, \rho, n_b, n_z, n_r, n_\varrho, n_\varsigma, n_\sigma)$;
    $O_i^{(g)} \leftarrow$ `mutate`$(O_i^{(g)}, n_b, n_z, n_r, n_\varrho, n_\varsigma, n_\sigma)$;
    `evaluate`$(O_i^{(g)})$;
    if $F(O_i^{(g)}) > F(S)$ **then**
      $counter \leftarrow counter + 1$;
    end if
  end for
  switch `selection-type` **of**
    case $(\mu\ ,\ \lambda)$**:** $P^{(g+1)} \leftarrow$ `selection`$(O^{(g)}, \mu)$;
    case $(\mu + \lambda)$**:** $P^{(g+1)} \leftarrow$ `selection`$(P^{(g)}, O^{(g)}, \mu)$;
  end switch
  if $reset = q$ **then**
    $reset \leftarrow 0$;
    $p_s \leftarrow \dfrac{counter}{\lambda \times q}$;
    if $p_s > 1/5$ **then**
      $m_s \leftarrow 1/c$;
    else if $p_s < 1/5$ **then**
      $m_s \leftarrow c$;
    else
      $m_s \leftarrow 1$;
    end if
    for all $i \in \{1, \ldots, \mu\}$ **do**
      $\vec{\varrho_i} \leftarrow \vec{\varrho_i} \times m_s$;
      $\vec{\varsigma_i} \leftarrow \vec{\varsigma_i} \times m_s$;
      $\vec{\sigma_i} \leftarrow \vec{\sigma_i} \times m_s$;
    end for
  else
    $reset \leftarrow reset + 1$;
  end if
  $g \leftarrow g + 1$;
end while

where ϱ_i' is the bit-flip mutation probability which is applied to each binary variable and $\mathcal{N}(0,1)$ denotes a draw from log–normal distribution in the interval of $]0,1[$. A value $\gamma = 0.22$ is empirically determined [55].

By applying the pure form of (A.5) to the binary strategy parameters, the population is likely to converge to the local optimal solutions, caused by the mutation rates that prematurely collapse to very small values and thereby effectively yielding no mutation. An important fix included in [55] is to restrict the value of ϱ_i to the interval $[\frac{1}{n}, \frac{1}{2}]$, for $n > 3$, and (A.5) is redefined as,

$$\varrho_i' = \min\left\{\frac{1}{2}, \max\left\{\frac{1}{n}, \frac{1}{1 + \frac{1-\varrho_i}{\varrho_i} \times \exp(\gamma \times \mathcal{N}(0,1))}\right\}\right\} \quad . \tag{A.6}$$

A.4.2.2 Mutation Rate in Integer Search Space

The mutation of strategy parameters of the integer genomes is given in Algorithm A.3.

Algorithm A.3 Integer step-size mutation

Input: ς_i

/* ς_i is the step-size of i^{th} gene */

Output: $\varsigma_i{}'$

/* $\varsigma_i{}'$ is updated of the step-size of i^{th} gene */

1: $\mathcal{N}_c \leftarrow \mathcal{N}(0,1)$;
2: $\tau \leftarrow \dfrac{1}{\sqrt{2 \times n_z}}$;
3: $\tau' \leftarrow \dfrac{1}{\sqrt{2\sqrt{n_z}}}$;
4: **if** $n_\varsigma = 1$ **then**
5: $\quad \varsigma_1{}' \leftarrow \max(1, \varsigma_1 \times \exp(\tau \times \mathcal{N}_c))$;
6: **else**
7: $\quad$ **for all** $i \in \{1, \ldots, n_\varsigma\}$ **do**
8: $\quad\quad \varsigma_i{}' \leftarrow \max(1, \varsigma_i \times \exp(\tau \times \mathcal{N}_c + \tau' \times \mathcal{N}(0,1)))$;
9: $\quad$ **end for**
10: **end if**

In lines #1-3 of the algorithm, a constant random number $\mathcal{N}(0,1)$, learning rates τ (global learning rate) and τ' (local learning rate) are being defined. $\mathcal{N}(0,1)$ represents a random number sampled from a normal distribution with mean 0 and variance 1. These variables are used in the mutation of the integer step-sizes in lines #4-10 where the global factor $\exp(\tau \times \mathcal{N}_c)$ allows for an overall change of the mutability, while the local factor $\exp(\tau' \times \mathcal{N}(0,1))$ enables individual changes to step-size ς_i'. In lines #5 and #8, a minimum step-size value of 1 is enforced.

A.4.2.3 Mutation Rate in Real-Valued Search Space

The mutation of strategy parameters of the real-valued genomes is given in Algorithm A.4.

Algorithm A.4 Real-valued step-size mutation

Input: σ_i

/* σ_i is the step-size of i^{th} gene */

Output: $\sigma_i{}'$

/* $\sigma_i{}'$ is updated of the step-size of i^{th} gene */

1: $\mathcal{N}_c \leftarrow \mathcal{N}(0,1)$;
2: $\tau \leftarrow \dfrac{1}{\sqrt{2 \times n_r}}$;
3: $\tau' \leftarrow \dfrac{1}{\sqrt{2\sqrt{n_r}}}$;
4: $\varepsilon \leftarrow 10^{-30}$;
5: **if** $n_\sigma = 1$ **then**
6: $\quad \sigma_1{}' \leftarrow \max(\varepsilon, \sigma_1 \times \exp(\tau \times \mathcal{N}_c))$;
7: **else**
8: $\quad$ **for all** $i \in \{1, \ldots, n_\sigma\}$ **do**
9: $\quad\quad \sigma_i{}' \leftarrow \max(\varepsilon, \sigma_i \times \exp(\tau \times \mathcal{N}_c + \tau' \times \mathcal{N}(0,1)))$;
10: $\quad$ **end for**
11: **end if**

Similar to the integer case, in lines #1-4 of the algorithm, a constant random number $\mathcal{N}(0,1)$, learning rates τ (global learning rate) and τ' (local learning rate), and threshold

rate ε are being defined. $\mathcal{N}(0,1)$ represents a random number sampled from a normal distribution with mean 0 and variance 1. These variables are used in the mutation of the real-valued step-sizes in lines #5-11 where the global factor $\exp(\tau \times \mathcal{N}_c)$ allows for an overall change of the mutability, while the local factor $\exp(\tau' \times \mathcal{N}(0,1))$ enables individual changes to step-size σ_i'. Furthermore, since standard deviations very close to zero are unwanted —to avoid convergence to local optimal solutions, the following boundary rule is used to force step-sizes to be no smaller than a threshold:

$$\sigma' < \varepsilon \;\Rightarrow\; \sigma' = \varepsilon \;\; . \tag{A.7}$$

The minimum value $\varepsilon = 10^{-30}$ is suggested by [65].

A.5 Initialization

Initialization of the individuals of the population in ESs includes object and strategy parameters. All genotypes — included binary, integer, and real-valued — have real-valued strategy parameters. The object parameters and strategy parameters are obtained as follows:

$$\vec{b} = (b_i = \|P(0.5)\|)_{i=1}^{n_b} \;\; , \tag{A.8}$$

$$\vec{z} = (z_i = \|U\left(lBound\_z_i, uBound\_z_i\right)\|)_{i=1}^{n_z} \;\; , \tag{A.9}$$

$$\vec{r} = (r_i = U\left(lBound\_r_i, uBound\_r_i\right))_{i=1}^{n_r} \;\; , \tag{A.10}$$

$$\vec{\varrho} = \left(\varrho_i = U\left(\frac{1}{n_b}, \frac{1}{2}\right)\right)_{i=1}^{n_\varrho} \;\; , \tag{A.11}$$

$$\vec{\varsigma} = \left(\varsigma_i = \left(uBound\_z_i - lBound\_z_i\right)/\sqrt{n_z}\right)_{i=1}^{n_\varsigma} \;\; , \tag{A.12}$$

$$\vec{\sigma} = \left(\sigma_i = \left(uBound\_r_i - lBound\_r_i\right)/\sqrt{n_r}\right)_{i=1}^{n_\sigma} \;\; , \tag{A.13}$$

where:

$P(0.5)$	Bernoulli trial with probability of success 0.5;
$U(a, b)$	random number sampled from a uniform distribution with lower bound a and upper band b;
$lBound\_z_i$	lower bound of the domain of integer object parameter z_i;
$uBound\_z_i$	upper bound of the domain of integer object parameter z_i;
$lBound\_r_i$	lower bound of the domain of real-valued object parameter r_i;
$uBound\_r_i$	upper bound of the domain of real-valued object parameter r_i.

The binary object parameters in $\vec{b}$ are initialized with Bernoulli trial with fixed probability of success 0.5. The integer and real-valued object parameters in $\vec{z}$ and $\vec{r}$, respectively, are initialized uniform randomly to values in their allowed domains.

The lower and upper bound of $1/n_b$ and $1/2$, respectively, for the mutation probabilities in $\vec{\varrho}$ are motivated by the observation that mutation loses its causality [56]. The lower bound of $1/n_b$ assures a minimum of one bit-flip mutation in every application of the mutation operator, and upper bound of $1/2$ mutates about half of the binary object parameters. Integer step-sizes $\vec{\varsigma}$, and real-valued step-sizes $\vec{\sigma}$ are initialized to $\Delta x_i/\sqrt{n}$, where Δx_i denotes the estimated distance between starting point and optimum [73]. This rule explicitly chooses small initial standard deviation (strategy parameter(s)) because, depending of the topology of the objective function, a combination of a large standard deviation and weak selective pressure (a too large value of μ) may cause the algorithm to diverge [1]. Nevertheless, the self-adaptation process quickly scales them into the appropriate range.

While using a single mutation probability for each genotype (i.e., $n_b = 1$ and/or $n_z = 1$ and/or $n_r = 1$), each position in related object parameters is decided independently from mutation decision, but with equal probability for all positions.

A.6 Parent Selection

The first selection of the ES algorithm (line #8 of Algorithm A.1, the operator is called 'marriage') is independent of the parental objective function (F) values. This contrasts to standard selection techniques in genetic algorithms [40], where the selection relies on the objective function values. In this step of the algorithm, ρ number of the parental individuals of size μ are drawn randomly with uniform distribution (with or without replacement) and using them as input(s) of variation operators. In the special case of $\rho = \mu$, all parents become members of the parent family S. Note, if $\rho = 1$, then, recombination is simply a copy of a random selected parent.

A.7 Variation

In the following sections, algorithms of several variation operators (recombination and mutation) for binary, integer, and real-valued search spaces are addressed.

A.7.1 Simulated Binary Crossover (SBX)

SBX is introduced in Sect. A.7.1. The complete SBX algorithm can be found in Algorithm A.5.

A.7.2 Bit-Flip Mutation

The probabilistic bit-flip scheme, is used in canonical EAs, as a mutation operator, to manipulate binary object parameters. In the multi strategy parameters, each binary object parameter is mutated with a certain probability ϱ_i. In contrast, in the single strategy parameter just as in GAs, all binary object parameters are mutated with the same probability. Given a binary individual of the form $\vec{b} = \langle b_1, \ldots, b_{n_b} \rangle \in \mathbb{B}^{n_b}$, the

Algorithm A.5 Simulated binary crossover (SBX).

Input: $\vec{r}_1, \vec{r}_2, \vec{\sigma}\,'$

/* $\vec{r}_1 = \langle r_{11}. \ldots, r_{1n_r} \rangle$ is the first parent */

/* $\vec{r}_2 = \langle r_{21}. \ldots, r_{2n_r} \rangle$ is the second parent */

/* $\vec{\sigma}\,' = \langle \sigma'_1. \ldots, \sigma'_{n_\sigma} \rangle$ are updated step-sizes for real objective parameters */

Output: $\vec{r}_1', \vec{r}_2'$

/* $\vec{r}_1' = \langle r'_{11}. \ldots, r'_{1n_r} \rangle$ is the updated of the first parent */

/* $\vec{r}_2' = \langle r'_{21}. \ldots, r'_{2n_r} \rangle$ is the updated of the second parent */

1: **for all** $i \in \{1, \ldots, n_r\}$ **do**
2: $\varepsilon \leftarrow 10^{-6}$;
3: **if** $r_{1i} > r_{2i}$ **then**
4: $r_{1i} \leftrightarrow r_{2i}$;
5: **end if**
6: $\Delta \leftarrow r_{2i} - r_{1i}$;
7: **if** $r_{1i} - lBound\_r_i < uBound\_r_i - r_{2i}$ **then**
8: $\delta \leftarrow r_{1i} - lBound\_r_i$;
9: **else**
10: $\delta \leftarrow uBound\_r_i - r_{2i}$;
11: **end if**
12: **if** $\delta < 0$ **then**
13: $\delta \leftarrow 0$;
14: **end if**
15: $u \leftarrow U(0, 1)$;
16: **if** (rigid_boundary = **true and** $\Delta > \varepsilon$) **then**
17: $u \leftarrow u \times \left(1 - \frac{1}{2} \times \left(1 + \frac{2\delta}{\Delta}\right)^{\sigma_i + 1}\right)$;
18: **end if**
19: **if** $u < 0.5$ **then**
20: $\beta \leftarrow (2u)^{\frac{1}{\sigma_i + 1}}$;
21: **else**
22: $\beta \leftarrow \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\sigma_i + 1}}$;
23: **end if**
24: $r'_{1i} \leftarrow 0.5\,(r_{1i} + r_{2i}) + 0.5\beta\Delta$;
25: $r'_{2i} \leftarrow 0.5\,(r_{1i} + r_{2i}) - 0.5\beta\Delta$;
26: **end for**

bit-flip mutation operator is

$$b'_i = \begin{cases} 1 - b_i & \text{if } U(0,1) \leq \varrho_i, \\ b_i & \text{if } U(0,1) > \varrho_i, \end{cases} \tag{A.14}$$

where $U(0,1)$ denotes a new uniform random variable sampled for each $i \in \{1, \ldots, n_b\}$.

A.7.3 Geometrical Mutation

As opposed to real-valued object parameters, integer object parameters are less commonly used in evolutionary strategies. In 1994, Rudolph came up with a method of generating a discrete geometrical counterpart of the continuous normal probability distribution by taking the difference of two geometrically distributed random variables G_1 and G_2 [67]. Note that among different kind of distributions defined on integer search spaces, the multidimensional geometric distribution is the one with maximum entropy and finite variance. The strength of the mutation is controlled by a set of strategy parameter(s) $\vec{\varsigma}'$, which represent the mean value of the absolute variation of the integer object parameters. Details of this mutation operator are found in Algorithm A.6.

Line #4 of Algorithm A.6 is taking care of single or multi strategy parameters(s). The geometrical distributed random value with mean strategy parameter ς'_i can be generated by transforming a uniformly distributed random value u, by using:

$$\psi = 1 - (\frac{\varsigma'}{n_z}) \times \left(1 + \sqrt{1 + \left(\frac{\varsigma'}{n_z} \right)^2} \right)^{-1} , \tag{A.15}$$

$$G = \left\lfloor \frac{\ln(1-u)}{\ln(1-\psi)} \right\rfloor . \tag{A.16}$$

The doubly geometrically distributed random number $G_1 - G_2$ is used for updating the object variables z_i (lines #6-12). Then, if the boundary is set to *rigid*, the updated integer object parameters should be within their feasible interval. To accomplish that, the

Algorithm A.6 Integer object parameters mutation

Input: $\vec{z}, \vec{\varsigma}'$

/* $\vec{z} = \langle z_1 \ldots, z_{n_z} \rangle$ are integer object parameters */

/* $\vec{\varsigma}' = \langle \varsigma'_1 \ldots, \varsigma'_{n_\varsigma} \rangle$ are updated step-sizes for integer object parameters */

Output: $\vec{z}\,'$

/* $\vec{z}\,' = \langle z'_1 \ldots, z'_{n_z} \rangle$ are updated integer object parameters */

1: **for all** $i \in \{1, \ldots, n_z\}$ **do**

2: $u_1 \leftarrow U(0,1)$;

3: $u_2 \leftarrow U(0,1)$;

4: $s \leftarrow \varsigma'_{min(n_\varsigma, i)}$;

5: $\psi \leftarrow 1 - (s/n_z) \times \left(1 + \sqrt{1 + (s/n_z)^2}\right)^{-1}$;

6: $G_1 \leftarrow \left\lfloor \dfrac{\ln(1-u_1)}{\ln(1-\psi)} \right\rfloor$;

7: $G_2 \leftarrow \left\lfloor \dfrac{\ln(1-u_2)}{\ln(1-\psi)} \right\rfloor$;

8: **if** rigid_boundary = **true then**

9: $z'_i \leftarrow T^z_{[lBound\_z_i, uBound\_z_i]}(z_i + G_1 - G_2)$;

10: **else**

11: $z'_i \leftarrow z_i + G_1 - G_2$;

12: **end if**

13: **end for**

mutation operator needs to be extended and exploit a *transformation function* $T_{[a,b]}$ (line #9). In other words, the transformation function brings parameters beyond boundaries back into the feasible domain. The transformation function can be viewed as a reflection at the interval boundaries. An example of how the transformation function works can be found in Fig. A.1.

The transformation function can be used for both continuous and integer search spaces. This function starts in the direction of the original unbounded mutation, whenever it meets with an interval boundary the direction is inverted until the total length of the unbounded mutation has been covered. The method can be efficiently implemented

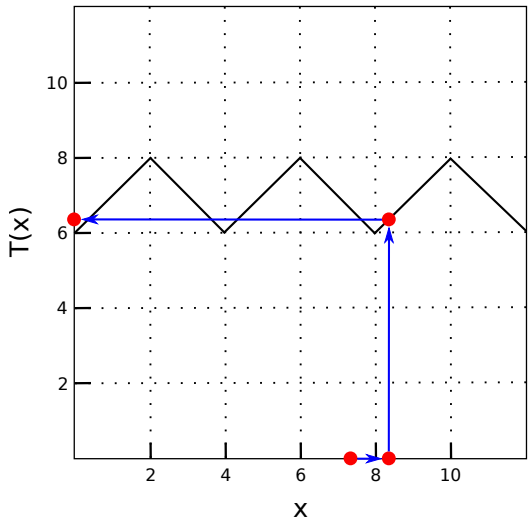


Figure A.1: The mechanism of the transformation function ($a = 6$, $b = 8$). Figure courtesy of Li [56].

as seen in Algorithm A.7.

Algorithm A.7 Transformation function $T^r_{[a,b]}(x)$, for interval boundaries a and b.

Input: x, a, b
/* x is the original value */
/* a is the lower bound of the value */
/* b is the upper bound of the value */
Output: x'
/* x' value checked or transformed to be in $[a, b]$ */
1: $y \leftarrow (x - a)/(b - a)$;
2: **if** $\lfloor y \rfloor \bmod 2 = 0$ **then**
3: $y' \leftarrow |y - \lfloor y \rfloor|$;
4: **else**
5: $y' \leftarrow 1 - |y - \lfloor y \rfloor|$;
6: **end if**
7: $x' \leftarrow a + (b - a) \times y'$;

Algorithm A.7 lists the transformation function $T^r_{[a,b]}(x)$ for real-valued x (where a and b are the lower and upper bounds of the value, respectively); for an integer value the transformation function $T^z_{[a,b]}(x)$ is obtained as follows:

$$T^z_{[lBound\_z_i, uBound\_z_i]}(x) = \lfloor T^r_{[lBound\_z_i, uBound\_z_i]}(x) \rfloor \ , \tag{A.17}$$

A.7.4 Polynomial Mutation

Details of the polynomial mutation can be found in Sect. 2.6.4.1. The mutation procedure for given real-valued object parameters is given in Algorithm A.8. Lines #10-23 represent the rigid type of the polynomial mutation.

Algorithm A.8 Polynomial mutation.

Input: $\vec{r}, \vec{\sigma}\,'$

/* $\vec{r} = \langle r_1 \ldots, r_{n_r} \rangle$ are real objective parameters */

/* $\vec{\sigma}\,' = \langle \sigma'_1 \ldots, \sigma'_{n_\sigma} \rangle$ are updated step-sizes for real-valued objective parameters */

Output: $\vec{r}\,'$

/* $\vec{r}\,' = \langle r'_1 \ldots, r'_{n_r} \rangle$ are updated real-valued objective parameters */

1: **for all** $i \in \{1, \ldots, n_r\}$ **do**
2: $u \leftarrow U(0,1)$;
3: $\Delta_{max} \leftarrow uBound\_r_i - lBound\_r_i$;
4: **if** rigid_boundary = **false then**
5: **if** $u < 0.5$ **then**
6: $\delta \leftarrow (2u)^{\frac{1}{\sigma_i+1}} - 1$;
7: **else**
8: $\delta \leftarrow 1 - (2 \times (1-u))^{\frac{1}{n+1}}$;
9: **end if**
10: **else**
11: $lDelta \leftarrow \max\left(-1, \frac{lBound\_r_i - r_i}{\Delta_{max}}\right)$;
12: $uDelta \leftarrow \min\left(1, \frac{uBound\_r_i - r_i}{\Delta_{max}}\right)$;
13: **if** $-lDelta < uDelta$ **then**
14: $uDelta \leftarrow -lDelta$;
15: **else**
16: $lDelta \leftarrow -uDelta$;
17: **end if**
18: **if** $u < 0.5$ **then**
19: $\delta \leftarrow \left(2u + (1-2u) \times (1 - lDelta)^{\sigma_i+1}\right)^{\frac{1}{\sigma_i+1}} - 1$;
20: **else**
21: $\delta \leftarrow 1 - \left(2 \times (1-u) + (u-0.5) \times (1 - uDelta)^{\sigma_i+1}\right)^{\frac{1}{\sigma_i+1}}$;
22: **end if**
23: **end if**
24: $r'_i \leftarrow r_i + \delta \times \Delta_{max}$;
25: **end for**

Appendix B

Application Details

B.1 Execution of the Application

In order to find the best setup setting, the EAs should be tested with several different operators and parameters settings. Since EAs are stochastic optimization methods, it is necessary to average the results through all runs with different seeds.

In this thesis two well-known evolutionary algorithms, GA and ES, have been implemented and applied to IR problem. Implemented GA's software for IR problem can be found in `http://www.deei.fct.ualg.pt/~a38477/ir_ga.tar.gz`. ES framework can be found in `http://www.deei.fct.ualg.pt/~a38477/es.tar.gz`. Both applications are written fully object oriented in std C++, and are available as open-source packages, under GPL licence.

In this thesis two well-known of evolutionary algorithms, GA and ES are programmed and applied to IR problem. To study the behavior of the algorithms, 2-D synthetic point-sets were used. Deformed images' points were obtained from both affine and non-affine transformations. Affined deformed point-sets can be found in `http://www.deei.fct.ualg.pt/~a38477/affined-distortion.tar.gz`, and non-Affine deformed point-sets can be found in `http://noodle.med.yale.edu/~chui/rpm/TPS-RPM.zip`.

In order to run the applications with different parameters, a *bash* script code under linux operating system is written, and is systematically run with all different setup settings. Then a program which is written in python is used to merge and calculate the average of all runs of the setup setting. In the following, the *bash* script code is shown, and the next section shows an example of the input configuration file.

```
POPSIZE=10
SELSIZE=2
NO\_RUNS=100
POPSTEPS=5
SELSTEPS=3
POINT\_SET=02\\_point\\_set.txt
for K in `seq 1 $POPSTEPS`; do
  SELSIZE=2
  for S in `seq 1 $SELSTEPS`; do
    for MR in `seq 0 2`; do
      for R in `seq 0 2`; do
        for M in `seq 0 1`; do
          if [[ $R -ne 2 || $SELSIZE -eq 2 ]]; then
            NAME=P$POPSIZE\\_S$SELSIZE\\_MuR$MR\\_Xt$R\\_MuT$M
            ./es inputs/$NAME $POINT\_SET $NO\_RUNS
            python mergeintoone.py $NO\_RUNS
            for I in `seq 1 $NO\_RUNS`; do
              rm inputfile\_run$I.txt;
            done
          fi
        done
      done
    done
    SELSIZE=$((2 * $SELSIZE))
  done
  POPSIZE=$((2 * $POPSIZE))
done
```

B.2 An Example of the Input File

In this part an example of the input files is shown. Note, line starting with # sign, are comments.

```
#
# This file contains a sample configuration for running a standard Evolutionary Strategy (ES).
# Here you can specify the test function, the parameters of the algorithm,
# and various report options.
#
```

```
# Lines starting with a '#' are comments and are ignored by the program.
#
# Lines starting with a '$' are array inputs for
# the 'n_distrib_SBX' and lower and higher bound of the genes.
# Do not forget to specify the array type exactly after '$'.
# Do not forget to include the '$' after adding array data.
#
# The format of a configuration option is 'option_name:  value'
# Do not forget to include the ':'  after the option name
#

################################################
## GENERAL SETTINGS
################################################
#
# specification of the test function.
#
# 'test_function:'  can be:
# points_matching
# one_max
#
test_function:                      points_matching

#
# objective function needs additional data
#
objfunc_additional_data:            on

#
# output file
#
out_filename:                       P160_S2_MuR1_Xt1_MuT0_out.txt

#
# seed to initialize random number generator.
# must be a real number on 0..1
#
seed:                               0.43642425593

#
# the optimization type can be:
# minimum_optimization
# maximum_optimization
#
optimization_type:                  minimum_optimization

#
# real values floating point precision
# number of digits are appeared after floating point
#
variable_precision:                 16

#
# population size
#
popsize:                            500

#
# selection size
# selection correspond to the first random selection from parents
#
selectionsize:                      2

#
# offspring size
# number of offspring which should be generated from parental population
# WARNING:
# it is more sufficient, if offspring size be 7 times bigger than population size (popsize)
```

```
#
offspringsize:                      3500

#
# standard deviations (step size) very close to zero are unwanted
# it is used to force step size to be no smaller than a threshold
# epsilon value, which is 10^-30 (Reehuis and Thomas Back 2010)
#
step_size_threshold:                10^-30

#
# beta is a variable which is using for mutating of rotation angles
# its default value is 5 degree which is equivalent to 0.0873 radians
# (References:  Tomas Baeck 2010, Eiben 2003, Tomas Baeck 1996)
#
beta:                               0.0873

#
# parameter c has an effect on the 1/5 rule
# parameters c is in the range 0.817 <= c <= 1
# Schwefel 1981 suggested the value 0.82 for the parameter
# Schwefel 1995 and Fogel 2000 suggested the value 0.85 for the parameter
#
c_success_rule:                     0.85

#
# number of generation before resetting step size in 1/5 rule
# the suggested value is 10 (Thomas Baeck, Evolutionary Computation, mutation operators section)
#
no_iteration_before_one_fifth:      10

#
# selection type can be:
# with_replacement
# without_replacement
#
selection_type:                     without_replacement

#
# survivor type can be:
# mu_plus_lambda
# mu_comma_lambda
#
survivor_selection:                 mu_plus_lambda

#
# recombination strategy parameters can be:
# discrete_strategy_parameters_recombination
# intermediate_strategy_parameters_recombination
#
xover_strategy_parameters:          intermediate_strategy_parameters_recombination

#
# mutation rule can be:
# one_fifth_rule
# uncorrelated_mutation_one_ss
# uncorrelated_mutation_n_ss
# correlated_mutation
#
mutation_rule:                      uncorrelated_mutation_one_ss

#
# stop criterion can be:
# max_generations
# max_fitness
# avg_fitness
# max_fit_not_improve
# avg_fit_not_improve
```

```
#
#
stop_criterion:                    max_generations

#
# 'stop_criterion_arg:'  option is used together with 'stop_criterion:'  if
# applicable.  For example, to run a ES for 5 generations set
# 'stop_criterion' to max_generations and 'stop_criterion_arg' to 5
#
stop_criterion_arg:                500

#
# stops the algorithm when 'max_function_evaluations' have elapsed
# do it regardless of the stopping critetion.
#
max_function_evaluations:          10000000

#
# genes lower and higher bound.
#
$genesboundry
0-5,-1,1
$


##############################################
## BINARY GENES SETTINGS
##############################################

#
# binary genes length
#
chlchrom:                          0

#
# xover_type_bin can be:
# discrete_recombination_bin
# compact_formed_recombination
#
xover_type_bin:                    discrete_recombination_bin

#
# mutation probability for binary genes
# NOTE: use -1, for pmut=1/lchrom
#
p_mutation_binary:                 0.1


##############################################
## INTEGER GENES SETTINGS
##############################################

#
# integer genes length
#
intlchrom:                         0

#
# specified that the boundary checking is rigid
#
rigid_int:                         on

#
# number of trying for finding a right mutated gene
# the mutated gene's value has to be inside of the boundary
# if it didn't find after the retried number the bound value will be replaced
#
out_of_bounds_retries_int:         10
```

```
#
# xover_type_int can be:
# intermediate_recombination_int
# discrete_recombination_int
#
xover_type_int:                     intermediate_recombination_int

#
# mutation_type_int can be:
# polynomial_mutation_int
# mixed_integer_mutation
#
mutation_type_int:                  mixed_integer_mutation


##############################################
## REAL GENES SETTINGS
##############################################

#
# real genes length
#
rllchrom:                           6

#
# specified that the boundary checking is rigid
#
rigid_real:                         off

#
# number of trying for finding a right mutated gene
# the mutated gene's value has to be inside of the boundary
# if it didn't find after the retried number the bound value will be replaced
#
out_of_bounds_retries_real:         10

#
# xover_type_real can be:
# intermediate_recombination_real
# discrete_recombination_real
# sbx_recombination
#
xover_type_real:                    discrete_recombination_real

#
# mutation_type_real can be:
# gaussian_mutation_real
# polynomial_mutation_real
#
mutation_type_real:                 gaussian_mutation_real

#
# n_distribution_c is an array of distribution indexes of SBX crossover.
#
$n_distribution_sbx
0-5,2
$


##############################################
## REPORT SETTINGS
##############################################

#
# on/off reporting flags
#
report_to_screen:                   on
```

```
report_to_textfile:                on
report_pop:                        on
report_string:                     on
report_fitness:                    on
report_strategy_parameters:        on
report_stat:                       on
report_best_in_population:         on
report_best_so_far:                on
```

Bibliography

[1] T. Bäck. *Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[2] T. Bäck. Self-adaptation. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C7.1. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[3] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 1997.

[4] T. Bäck, D. B. Fogel, D. Whitley, and P. J. Angeline. Mutation. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C3.2. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[5] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, San Diego, CA, USA, July 1991. Morgan Kaufmann.

[6] S. Bandaru, R. Tulshyan, and K. Deb. Modified sbx and adaptive mutation for real world single objective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011*, pages 1335–1342. IEEE, 2011.

[7] M. Bazargani, A. dos Anjos, F. G. Lobo, A. Mollahosseini, and H. R. Shahbazkia. Affine image registration transformation estimation using a real coded genetic algorithm with sbx. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, Philadelphia, USA, 2012. ACM.

[8] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002.

[9] H.-G. Beyer. Toward a theory of evolution strategies: On the benefit of sex – the $(\mu/\mu, \lambda)$-theory. *Evolutionary Computation*, 3(1):81–111, 1995.

[10] H.-G. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Berlin, Heidelberg, 2001.

[11] H.-G. Beyer and K. Deb. On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 5(3):250–270, 2001.

[12] H.-G. Beyer and H.-P. Schwefel. Evolution strategies — a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[13] P. Bienert. *Aufbau einer optimierungsautomatik fur drei parameter*. Dipl.-Ing. Thesis, Technical University of Berlin, Institute of Measurement and Control Technology, Germany, 1967.

[14] L. B. Booker, D. B. Fogel, D. Whitley, P. J. Angeline, and Á. E. Eiben. Recombination. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C3.3. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[15] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24:325–376, 1992.

[16] C. K. Chow, H. T. Tsui, and T. Lee. Surface registration using a dynamic genetic algorithm. *Pattern Recognition*, 37(1):105–117, 2004.

[17] H. Chui. *Non-rigid point matching: Algorithms, extensions and applications*. PhD thesis, Yale University, 2001.

[18] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2-3):114–141, 2003.

[19] O. Cordón, S. Damas, and J. Santamaría. A CHC evolutionary algorithm for 3D image registration. In *Proceedings of the 10th International Fuzzy Systems Association (IFSA)*, volume 2715 of *Lecture Notes in Computer Science*, pages 404–411. Springer, 2003.

[20] O. Cordón, S. Damas, and J. Santamaría. A scatter search based optimizer for the registration of 3D surfaces. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005*, pages 2738–2744. IEEE, 2005.

[21] O. Cordón, S. Damas, and J. Santamaría. A fast and accurate approach for 3D image registration using the scatter search evolutionary algorithm. *Pattern Recognition Letters*, 27(11):1191–1200, 2006.

[22] S. Damas, O. Cordón, and J. Santamaría. Medical image registration using evolutionary computation: An experimental survey. *IEEE Computational Intelligence Magazine*, 6(4):26–42, 2011.

[23] C. Darwin. *On the origin of species*. John Murray, 1859.

[24] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, 10(4):371–395, 2002.

[25] K. Deb and M. Goyal. A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996.

[26] K. Deb and H. Jain. Parent to mean-centric self-adaptation in single and multi-objective real-parameter genetic algorithms with SBX operator. Technical Report 2011017, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, PIN 208016, India, Sept. 2011.

[27] K. Deb and A. Kumar. Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems. *Complex Systems*, 9(6):431–454, 1995.

[28] A. dos Anjos. *Automatic processing of bi-dimensional images of "omic" expression blobs*. PhD thesis, Department of Electronic Engineering and Informatics, Faculty of Sciences and Technology, University of Algarve, 2011.

[29] J. Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive Theory of Functions of Several Variables*, volume 571 of *Lecture Notes in Mathematics*, chapter 7, pages 85–100. Springer Berlin/Heidelberg, 1977.

[30] Á. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[31] Á. E. Eiben and J. E. Smith. *Introduction to evolutionary computation*. Natural Computing Series. Springer, 2003.

[32] L. J. Eshelman. Genetic algorithms. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter B1.2. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[33] J. Fitzpatrick, J. J. Grefenstette, and D. Gucht. Image registration by genetic search. In *Proceedings of IEEE Southeast Conference*, pages 460–464, 1984.

[34] D. B. Fogel. Principles of evolutionary processes. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter A2.1. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[35] D. B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*. Wiley-VCH, third edition, 2006.

[36] G. B. Fogel, D. B. Fogel, and L. Fogel. Evolutionary programming. *Scholarpedia*, 6(4):1818, 2011.

[37] L. J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[38] G. Garai and B. B. Chaudhuri. A cascaded genetic algorithm for efficient optimization and pattern matching. *Image and Vision Computing*, 20(4):265–277, 2002.

[39] H. F. G. García, A. G. Vega, A. H. Aguirre, J. L. M. Zaleta, and C. A. C. Coello. Robust multiscale affine 2D-image registration through evolutionary strategies. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, PPSN VII, pages 740–748. Springer-Verlag, 2002.

[40] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Boston, MA, USA, 1989.

[41] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[42] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *International Conference on Evolutionary Computation*, pages 312–317. Morgan Kaufmann, 1996.

[43] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[44] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.

[45] R. I. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2000.

[46] M. Hauschild and M. Pelikan. A survey of estimation of distribution algorithms. Technical Report 2011004, University of Missouri–St. Louis, Department of Mathematics and Computer Science, Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), Mar. 2011.

[47] R. He and P. A. Narayana. Global optimization of mutual information: application to three-dimensional retrospective registration of magnetic resonance images. *Computerized medical imaging and graphics*, 26(4):277–292, 2002.

[48] A. Hoffman. *Arguments on evolution: a paleontologist's perspective*. Oxford University Press, New York, USA, 1989.

[49] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[50] B. Jähne. *Digital image processing*. Springer-Verlag, London, UK, 6th revised and extended edition, 2005.

[51] K. D. Jong, D. B. Fogel, and H.-P. Schwefel. A history of evolutionary computation. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter A2.3. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[52] A. Juels, S. Baluja, and A. Sinclair. The equilibrium genetic algorithm and the role of crossover. 1993.

[53] D. Keysers and W. Unger. Elastic image matching is NP-Complete. *Pattern Recognition Letters*, 24:445–453, 2003.

[54] O. Kramer. *Self-adaptive heuristics for evolutionary computation*, volume 147 of *Studies in Computational Intelligence*. Springer, 2008.

[55] J. W. Kruisselbrink, R. Li, E. Reehuis, J. Eggermont, and T. Bäck. On the log-normal self-adaptation of the mutation rate in binary search spaces. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 893–900, New York, NY, USA, 2011. ACM.

[56] R. Li. *Mixed-integer evolution strategies for parameter optimization and their applications to medical image analysis*. PhD thesis, Leiden Institute of Advanced Computer Science (LIACS) – Leiden University, 2009.

[57] F. G. Lobo. Lost gems of ec: The equilibrium genetic algorithm and the role of crossover. *SIGEVOlution*, 2(2):14–15, July 2007.

[58] P. Merz. *Memetic algorithms for combinatorial optimization problems: fitness landscapes and effective search strategies*. PhD thesis, University of Siegen, Dec. 2000.

[59] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[60] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1994.

[61] M. Pelikan. Genetic algorithms. Technical Report No. 20010007, Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), University of Missouri St. Louis, USA, 2010.

[62] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Lulu Enterprises, UK Ltd, Mar. 2008.

[63] I. Rechenberg. *Cybernetic solution path of an experimental problem*. Royal Aircraft Establishment Library Translation no. 1122, Farnborongh, Hants, U.K., Aug. 1965.

[64] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.

[65] E. Reehuis and T. Bäck. Mixed-integer evolution strategy using multiobjective selection applied to warehouse design optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1187–1194, New York, NY, USA, 2010. ACM.

[66] J. M. Rouet, J. J. Jacq, and C. Roux. Genetic algorithms for a robust 3-D MR-CT registration. *IEEE Transactions on Information Technology in Biomedicine*, 4(2):126–136, 2000.

[67] G. Rudolph. An evolutionary algorithm for integer programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, volume 866 of *Lecture Notes in Computer Science*, pages 139–148, Jerusalem, Israel, 1994. Springer.

[68] G. Rudolph. Evolution strategies. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter B1.3. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 97/1 edition, 1997.

[69] J. Santamaría, O. Cordón, and S. Damas. A comparative study of state-of-the-art evolutionary image registration methods for 3D modeling. *Computer Vision and Image Understanding*, 115(9):1340–1354, 2011.

[70] J. Santamaría, O. Cordón, S. Damas, J. M. García-Torres, and A. Quirin. Performance evaluation of memetic approaches in 3D reconstruction of forensic objects. *Soft Computing*, 13(8-9):883–904, 2009.

[71] H.-P. Schwefel. *Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik*. Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger–Institute for Fluid Dynamics, Germany, March 1965.

[72] H.-P. Schwefel. *Evolutionsstrategie und numerische optimierung*. Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering, 1975.

[73] H.-P. Schwefel. *Numerische optimierung von computer–modellen mittels der evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research*. Birkhäuser, Basle, 1977.

[74] H.-P. Schwefel. *Numerical optimization of computer models*. John Wiley, New York, USA, 1981.

[75] H.-P. Schwefel. Collective phenomena in evolutionary systems. In P. Checkland and I. Kiss, editors, *Problems of Constancy and Change – The Complementarity of Systems Approaches to Complexity, Proc. 31st Annual Meeting*, volume 2, pages 1025–1033, Budapest, 1987. Int'l Soc. for General System Research.

[76] H.-P. Schwefel. *Evolution and optimum seeking*. Sixth-Generation Computer Technology. Wiley Interscience, New York, 1995.

[77] H.-P. Schwefel and G. Rudolph. Contemporary evolution strategies. In F. Morán, A. Moreno, J. J. M. Guervós, and P. Chacón, editors, *Advances in Artificial Life, Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 893–907. Springer, June 1995.

[78] F. L. Seixas, L. S. Ochi, A. Conci, and D. M. Saade. Image registration using genetic algorithms. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO'08, pages 1145–1146, New York, USA, 2008. ACM.

[79] P. W. M. Tsang. A genetic algorithm for affine invariant object shape recognition. In *1st IEE/IEEE International Symposium on Genetic Algorithms in Engineering Systems, GALESIA*, pages 293–298. IEEE, 1995.

[80] P. W. M. Tsang. A genetic algorithm for aligning object shapes. *Image and Vision Computing*, 15(11):819–831, 1997.

[81] P. A. Viola. *Alignment by maximization of mutual information*. PhD thesis, Massachusetts Institute of Technology, 1995.

[82] M. P. Wachowiak and A. S. Elmaghraby. The continuous tabu search as an optimizer for 2D-to-3D biomedical image registration. In *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention, MICCAI 2001*, volume 2208 of *Lecture Notes in Computer Science*, pages 1273–1274. Springer, 2001.

[83] M. P. Wachowiak, R. Smolíkova, Y. Zheng, J. M. Zurada, and A. S. Elmaghraby. An approach to multimodal biomedical image registration utilizing particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):289–301, 2004.

[84] S. Winter, B. Brendel, I. Pechlivanis, K. Schmieder, and C. Igel. Registration of CT and intraoperative 3-D ultrasound images of the spine using evolutionary and gradient-based methods. *IEEE Transactions on Evolutionary Computation*, 12(3):284–296, 2008.

[85] S. M. Yamany, M. N. Ahmed, and A. A. Farag. A new genetic-based technique for matching 3-D curves and surfaces. *Pattern Recognition*, 32(10):1817–1820, 1999.

[86] L. Zhang, W. Xu, and C. Chang. Genetic algorithm for affine point pattern matching. *Pattern Recognition Letters*, 24(1-3):9–19, 2003.

[87] B. Zitová and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21:977–1000, 2003.