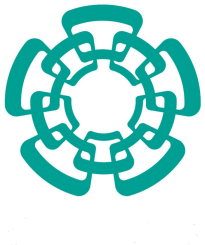


Center for research and advanced studies
of the National Polytechnic Institute

Cinvestav Tamaulipas

Gradient estimation based directions and its use as a local search operator in evolutionary algorithms

Thesis by:

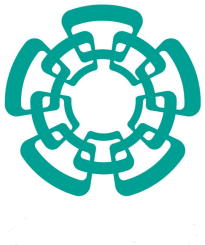
Jose Virgilio Treviño Avalos

as the fulfillment of the
requirement for the degree of:

**Master of Science
in Computer Engineering and
Technologies**

Advisor
Dr. Ricardo Landa Becerra

Cd. Victoria, Tamaulipas, Mexico. September, 2018



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Cinvestav Unidad Tamaulipas

Direcciones basadas en un estimador de gradiente y su uso en algoritmos evolutivos

Tesis que presenta

Jose Virgilio Treviño Avalos

Para obtener el grado de

**Maestro en Ciencias
en Ingeniería y Tecnologías
Computacionales**

Director de tesis:
Dr. Ricardo Landa Becerra

Cd. Victoria, Tamaulipas, México. Septiembre, 2018

The thesis of Jose Virgilio Treviño Avalos is approved by:

Dr. José Juan García Hernández

Dr. Javier Rubio Loyola

Dr. Ricardo Landa Becerra, Committe Chair

Cd. Victoria, Tamaulipas, Mexico., September 14th 2018

To the giants

Acknowledgements

- To my family and beloved pets.
- To my acquaintances of the farm, I will avoid nicknames this time, for the bullying given and received.
- To my thesis advisor for all the patience spent while guiding me through this long year.
- To my thesis reviewers for their constructive feedback that upped the quality of this document.
- To the administrative personnel at Cinvestav Tamaulipas for their help during my stay.
- I acknowledge support from Conacyt through a scholarship to pursue graduate studies at Cinvestav Tamaulipas.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Gradient estimation based directions and its use as a local search operator in evolutionary algorithms

by

Jose Virgilio Treviño Avalos

Cinvestav Tamaulipas

Center for Research and Advanced Studies of the National Polytechnic Institute, 2018

Dr. Ricardo Landa Becerra, Advisor

Numerical optimization problems can be mathematically represented by a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and a value $\mathbf{x}^*$ that satisfies $\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x})$. The objective of numerical optimization is finding this value $\mathbf{x}^*$, which is the global optimum of the function f.

Through the centuries, many techniques have been proposed to find, or at worst to approximate $\mathbf{x}^*$. Calculus inspired the creation of new methods, which use gradient and Hessian information. However the effectiveness of these methods is limited to functions that have continuous second derivatives.

Advances in silicon technology have exponentially incremented the computing power that researchers have available. This power has allowed the development of methods inspired by nature, these methods are called metaheuristics. An important class of metaheuristics are evolutionary algorithms, which are influenced by Darwinian evolution. Bio-inspired techniques require a great number of iterations to find good approximations, but they can operate even in non-derivable functions.

A new class of algorithms that combines classical matemathical methods with metaheuristics has appeared. Memetic algorithms integrate operators based on mathematical techniques into evolutionary algorithms. This class of algorithms has an evolutionary part which finds promising regions of the search, and use local search operators to exploit these regions. Gradient and Hessian approximations have been used in memetic algorithms.

Developed in the fifties, the conjugate gradient method is a numerical optimization technique

that uses first order information and has a convergence rate faster than the steepest descent but slower than methods that use second order information. The convergence speed of conjugate gradient methods is a consequence of the conjugacy of the search directions used, which means that searching in the same direction is avoided. Conjugate gradient methods use gradients as basis vectors, this suggests that it should be possible to construct conjugate directions using gradients computed by an estimator. This thesis proposes the use of conjugate directions as a local search operator in evolutionary algorithms.

Resumen

Direcciones basadas en un estimador de gradiente y su uso en algoritmos evolutivos

por

Jose Virgilio Treviño Avalos
Cinvestav Unidad Tamaulipas
Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional, 2018
Dr. Ricardo Landa Becerra, Director

Los problemas de optimización númerica consisten en una función $f : \mathbb{R}^n \to \mathbb{R}$ para la que se búsca un valor $\mathbf{x}^*$ que satisfaga $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$.

A lo largo de los siglos se han propuesto diversas técnicas para aproximar $\mathbf{x}^*$. Algunas de estas técnicas han encontrado inspiración en el cálculo infinitesimal. En esta clase de técnicas se encuentran incluidos los métodos que hacen uso de informacion tanto del gradiente como de la matriz Hessiana, el uso de estos métodos se limita a funciones que cuentan con derivadas continuas de segundo orden.

En los últimos 50 años el poder de cálculo de las computadoras ha permitido el desarrollo de métodos bio-inspirados, que requieren un gran número de iteraciones para encontrar buenas aproximaciones, y pueden operar incluso en funciones no derivables.

Recientemente, a los algoritmos evolutivos se les han incorporado operadores basados en técnicas matemáticas, lo que dio origen a los algoritmos meméticos, los cuales son un tipo de algoritmo evolutivo que usa un operador de búsqueda local para mejorar la capacidad de explotación del algoritmo. En la literatura se han utilizado estimaciones de gradiente como direcciones de descenso en algoritmos meméticos.

El método del gradiente conjugado es una técnica de optimización matemática con una convergencia más rápida que la del método del descenso de gradiente. Esta mayor velocidad de convergencia viene de las direcciones que por ser conjugadas entre sí evitan repetir búsquedas en una misma dirección. El método del gradiente conjugado utiliza gradientes como vectores base, lo que implica que es posible construir direcciones conjugadas utilizando un estimador de gradiente. Es

dentro de este contexto que se ubica el trabajo de tesis, el cual propone usar direcciones conjugadas como operador de búsqueda local en un algoritmo evolutivo.

Nomenclature

ACO	Ant Colony Optimization
BCO	Bacterial Colony Optimization
DE	Differential Evolution
DECLS	Differential Evolution Cauchy Local Search
EA	Evolutionary Algorithm
EC	Evolutionary Computation
ES	Evolutionary Strategy
EP	Evolutionary Programming
DE	Differential Evolution
GA	Genetic Algorithm
GE	Genetic Evolution
GSA	Gradient Subspace Approximation
HC	Hill Climbing
IFF	If and only if
ILS	Iterated Local Search
MA	Memetic Algorithm
PSO	Particle Swarm Optimization
RS	Random Search
SA	Simulated Annealing
SD	Simulated Diffusion
SS	Scatter Search
TS	Taboo Search
VNS	Variable Neighborhood Search

1

Introduction

1.1 Background

Even in the ancient times, philosophers recognized the importance of optimization in human life. In his magnus opus, 'Elements', Euclid proved that among isoperimetric rectangles a square is the figure with the greatest area [2]. Nearly two thousand years later, Fermat was the first who used the roots of a derivative to find the local optimum of a mathematical function. Derivative based methods are good to finding local optimums but this kind of methods is not able to identify if a solution is the global optimum [3].

A local optimum is a solution that it is the optimal in a neighborhood which corresponds to a subset of the search space, in contrast a global optimum is a solution that not only is better than its neighbors, in fact is the best in the entire search space.

One of the first methods devised after the discovery of calculus was the Newton-Raphson method. This method uses second order derivatives and has a quadratic order of convergence. In 1847, Cauchy

proposed the steepest descent method, which uses only first order derivatives, and is one of the fundamental algorithms in optimization. Nonetheless this method converges slower than Newton-Raphson, which uses information from derivatives of first and second order [4]. Both methods are unable to identify if an optimum is a local optimum [5]. This fact limits the usage of these methods for multi-modal functions, which are functions with multiple local optimums.

One of the greatest issues in numerical optimization is the existence of local optima. These points constitute a strong pole of attraction that difficults the search of the global optimum. Another issue is the existence of the "curse of dimensionality", which in numerical optimization means that the search space grows exponentially with the number of variables. This turns the use of classical methods impractical for high dimensional problems [6].

The unfeasibility of classical methods in conjunction with the increments in computing power have motivated the development of metaheuristics, which are general purpose algorithms that are used when it is impossible to make assumptions about a problem to solve with an ad-hoc method [7]. Many metaheuristics have clear biological inspiration, in this category coexist: genetic algorithms (GAs), evolutionary strategies (ESs), particle swarm optimization (PSO), simulated annealing (SA), and many other techniques.

Recently, classical mathematical optimizations method have been successfully combined with metaheuristics, producing competitive algorithms in comparison with the state of the art. The idea behind this hybridization is to incorporate a specifically designed local search operator in metaheuristics to refine solutions, this operator increments the ability of the method to exploit the search space. For example, in [8] is used the direction of the negative of the gradient as a descent direction to improve the solutions produced by a differential evolution (DE) algorithm.

As discussed below, the conjugate gradient method has a convergence faster than the steepest descent method [9], which suggests than a hybrid method using conjugate directions should be faster than a method that uses the direction of the negative of the gradient as the descent direction.

It is in this context that this thesis work exists. In this thesis conjugate directions, constructed

using a gradient estimator, are used as a local search operator within an evolutionary algorithm (EA).

1.2 Problem Statement

Numerical optimization is the process of finding a value $\mathbf{x}^* \in \mathbb{R}^n$ that for a function $f : \mathbb{R}^n \to \mathbb{R}$ satisfies the condition $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$.

There are mathematical techniques which find the optimum of a function. However these techniques use derivatives and sometimes such information is not available, because the function is non differentiable, the analytical form of the function is unknown, the derivative is hard to obtain, or any other reason. For this kind of situation it is possible to approximate a derivative using, for example, a method based on finite-differences [4].

The gradient vector is a generalization of the concept of derivative for multiple variables and it is defined by the Equation 1.1.

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right) \tag{1.1}$$

where n is the number of variables of the problem.

Recently, there have appeared some alternative methods for approximation of the gradient. Set oriented approximations use a set of neighbor points, which objective function values are already known. A great advantage of these type of approximations over finite-differences methods is that it is possible to avoid additional function evaluations by using the current population of a metaheuristic as neighbors.

However, gradient information can be used one step further: for constructing conjugate gradient directions. Conjugate gradient is a technique that uses gradient vectors to compute conjugate directions and using these directions is able to solve non-convex optimization problems [6].

A set of directions $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$, where every direction $\mathbf{v}_i$ is an n-dimensional vector, it is said to be conjugated if it satisfies the next equation for every $i \neq j$:

$$\mathbf{v}_i^T A \mathbf{v}_j = 0 \tag{1.2}$$

where A is positive definite matrix. If equation 1.2 is satisfied $\mathbf{v}_i$ y $\mathbf{v}_j$ are A-conjugated.

Conjugate gradient method use gradient information to construct conjugate directions and find an optimum in less iterations than using the the steepest descent (the direction of the negative of the gradient) method. In theory, for quadratic problems, this method converges in n steps, where n is the number of variables of the problem [7].

The problem of root-finding is related to numerical optimization [6]. Each optimum in the function f_1 is a root of the function f_2, where the function f_2 is the derivative of the function f_1. Nonetheless there are cases when an objective function is not differentiable. Also computing a derivative may be impractical for a number of reasons. Analytical methods cannot operate without derivatives, to handle these functions were developed methods that use approximations of the derivative.

Classical methods have shown limitations when solving high dimensional problems, however since the decade of 1980, and in part motivated by exponential gains in the computing power available for researchers, general purpose optimization algorithms have become popular. This kind of algorithms are called metaheuristics and are frequently bio-inspired. This type of algorithms are "blind", in the sense that they do not know the form or characteristics of the objective function, and they can only evaluate solutions and finding a global optimum is not guaranteed, this type of algorithms are mostly used when analytical methods cannot be used to find an exact solution.

In this thesis is proposed a a novel approach based on the use of conjugate directions in the context of EAs, as a local search operator in a memetic algorithm (MA).

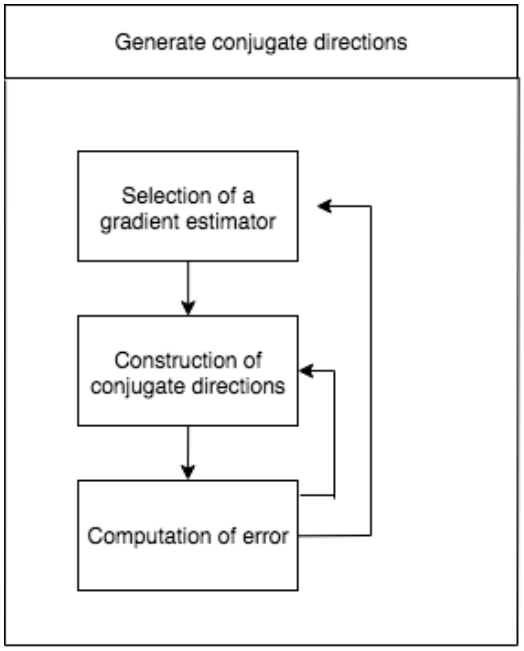


Figure 1.1: Methodology for constructing conjugate directions

1.3 Hypothesis

After describing the problem in the previous sections, it is pertinent to present the hypothesis that will be verified in this research:

Conjugate directions, in conjunction with a gradient estimator based on neighborhood information can be used to design a local search operator in a MA, producing results competitive for high dimensional problems.

1.4 Methodology

The proposed approach consists in constructing the conjugate directions and integrate these directions as an operator in an EA. Figure 1.1 shows the part of the methodology related to the construction of the operator. Figure 1.2 contains the part of the methodology related to the hybridization of the conjugate directions with the EA.

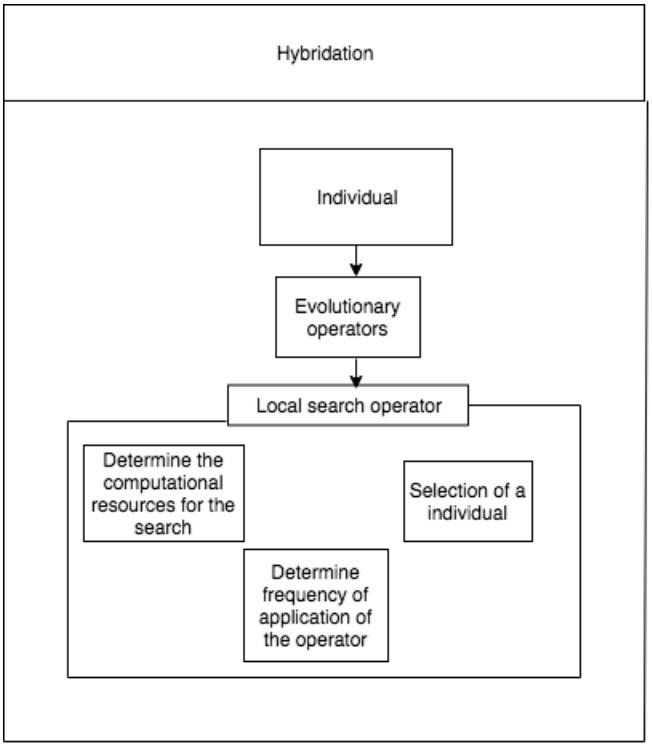


Figure 1.2: Methodology for hybridization

The sequence of tasks mentioned in the two previous figures are as follows:

1. **Selection of a gradient estimator**. There are multiple gradient estimators in the literature, one of these estimators will serve as the basis for the computing of the conjugate directions. The gradient estimator to be selected must be independent of the functions to optimize.

2. **Construction of conjugate directions**. There are multiple algorithms to compute conjugate directions using gradient vectors. One of the difficulties here is that the gradient estimator does not compute exact gradients and this will affect the conjugate directions computed.

3. **Selection of an EA**. An EA will be selected and will serve as the basis of the proposed approach.

4. **Conjugate directions as a local search operator**. The constructed conjugate directions will be incorporated as a local search operator in an EA.

5. **Selection of an individual for local search**. The computational cost of applying the local

search operator in each individual would be very high, this means that it is needed a criterion to determine which individuals will be refined by the operator.

6. **Determine the computational cost used in the search**. The depth of the search performed by the operator should not be excessive, in order to not turn prohibitive the computational cost of the developed algorithm.

7. **Determine the frequency of application of local search**. The frequency of application is also related to the computational cost, to avoid the waste computational resources an schedule for the application of the operator should be determined.

1.5 General Objective

- To develop and use a local search operator based on conjugate gradient, using relatively low-cost gradient estimations. This operator will be used in EAs and must produce competitive results.

1.6 Specific Objectives

- To develop a procedure to generate conjugate directions based on a gradient estimator.
- To design a local search operator for EAs using conjugate gradient estimations.

1.7 Document organization

This document is comprised of 6 chapters. In Chapter 2 fundamental concepts to grasp this study are introduced. The focus of the next chapter, Chapter 3, is on works, described in the literature that incorporate local search strategies in metaheuristic algorithms. In Chapter 4 the approach proposed in this thesis, to solve global optimization problems, is described. In Chapter 5 the results generated

by the experimentation performed are discussed. Finally, Chapter 6 includes conclusions and future work.

2

Theoretical framework

This chapter introduces basic mathematical and algorithmic concepts related to this thesis work. Section 2.1 briefly introduces some basic concepts that are fundamental to grasp the algorithms and techniques described in this thesis. The next sections in this chapter are focused on methods that solve optimization problems. Section 2.2 introduces the concept of EA and its historically three main forms, including a modern paradigm: DE, which is the EA where the local search operator proposed in this thesis is integrated. Section 2.3 presents two methods that use gradients directions to built descent directions.

2.1 Basic concepts

2.1.1 Optimization problems

According to the Merriam-Webster dictionary [10], optimization is defined as:

> An act, process, or methodology of making something (such as a design, system,

or decision) as fully perfect, functional, or effective as possible; specifically: the mathematical procedures (such as finding the maximum of a function) involved in this.

In continuous optimization, which operates in euclidean spaces, there are two types of problems: global optimization and constrained optimization. In global optimization the solution space spans the entire n-dimensional space. In constrained optimization there one or more functions that delimit the region of the n-dimensional space where every possible solution is located.

In global optimization problems the goal is to find the $\mathbf{x}^*$ that satisfies $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$ for a multivariate function, often referred as objective function $f : \mathbb{R}^n \to \mathbb{R}$, sometimes with one or more functions delimiting the region of the search space [6].

2.1.2 Neighborhoods in Euclidean spaces

The neighborhood of $\mathbf{x}$ is defined as a subset of $\mathbb{R}^n$ which contains an open ball centered at $\mathbf{x}$, this open ball is comprised of all the points $\hat{\mathbf{x}}$ that satisfy $||\mathbf{x} - \hat{\mathbf{x}}|| < \epsilon$, where $\epsilon > 0$ [11].

2.1.3 Local search

The idea behind local search is that good solutions in the search space are grouped together and neighborhoods where good solutions have been found could contain better solutions. Search steps should not be too large, to avoid the randomization of the search process [12].

2.1.4 Gradient

The gradient vector is a multi-variate generalization of the derivative. Each entry $\frac{\partial f}{\partial \mathbf{x}_i}$ in the gradient vector is a partial derivative of the variable $\mathbf{x}_i$, more formally this can be stated as:

$$\nabla \mathbf{f} = (\frac{\partial f}{\partial \mathbf{x}_1}, \frac{\partial f}{\partial \mathbf{x}_2}, \ldots, \frac{\partial f}{\partial \mathbf{x}_n}) \tag{2.1}$$

2.1.5 Descent direction

A direction $\mathbf{p}$ is a descent direction at $\mathbf{x}$ if the inequality $\nabla \mathbf{f}(\mathbf{x}) \cdot \mathbf{p} < 0$ is satisfied [6].

2.1.6 Deterministic methods

This type of methods use deterministic variables, and guarantee that the output will be the optimum computed at an arbitrary precision. In this category are included methods that use derivatives and direct search algorithms, like simplex, Nelder-Mead, Hooke and Jeeves, and Powell´s method [6].

2.1.7 Stochastic methods

This type of optimization methods generate and use random variables. The convergence proofs of these methods state that given infinite time the global optimum will be found with a probability of 1. This category includes Markov chain Monte-Carlo algorithms, random search (RS), hill climbing (HC), SA, taboo search (TS), EAs, and many others [13].

2.1.8 Metaheuristics

Metaheuristics are approximated stochastic methods which objective is to explore the search space to find the best solution. In contrast to methods inspired by calculus, metaheuristics are capable of extracting themselves from local optimums [14].

There are cases where supposing particular properties regarding a problem is not possible. To handle this kind of issue, a general purpose algorithm can be implemented to solve the problem or at least to quickly detect solutions. This type of techniques are referred to as metaheuristics, which etymologically comes from Greek "search beyond", this alludes to the increased level of abstraction used to solve the problem [7]. These type of techniques usually have biological (EA, TS), ethological (ACO, PSO) or physical inspiration (SA).

Metaheuristics have proliferated in recent decades powered by the advances in computational hardware. This category includes evolutionary programming (EP), PSO, SA, and many others [15].

There are two type of metaheuristics [1]: single solution based metaheuristics and population based metaheuristics, the former includes noise methods (NMs), SA, taboo search (TS), iterated local search (ILS), variable neighborhood search (VNS), etc. The latter includes EA, ant colony optimization (ACO), scatter search (SS), etc [16].

Metaheuristics most important disadvantages are that this class of methods does not guarantee to find the best solution, adjustment of parameters is not trivial, and large computation times required [14].

2.2 Evolutionary algorithms

After traveling, for years, across the world, British naturalist Charles Darwin was shuddered by the seemingly perfection in nature. Nonetheless evolution does not imply perfection, evolution is an optimization process. Living beings compete with each other and with their environment to survive and as time passes phenotypes come as close to the optimum as possible, given initial conditions and environment constraints. However nature is not static, and the environment is always changing which implies that the optimum is also moving and no living being is perfectly adapted.

The classic techniques of gradient descent, deterministic hill climbing, and purely random search were generally unsatisfactory when applied to nonlinear optimization problems. Biological evolution provided an inspiration to solve these problems [17].

Evolutionary algorithms are a group of metaheuristics which are inspired by neo-darwinian evolution. Like occurs in nature, evolutionary algorithms operate over populations, with individuals being combined to create new ones, with every individual having trials to pass. Each individual is represented by its genotype, which is formed by genes, evolutionary operators work at this level. In numerical optimization, a genotype corresponds to a vector and a phenotype to the value of a

objective function, this mechanism determines how fit is an individual. In this class of algorithms two main operators are involved [7]:

Variation operators: which create the necessary diversity in the population.

Selection operator: which increment the average solution quality in the population. In contrast to variation operators these operators reduce population diversity.

One consequence of the no-free lunch theorem is that a method that solve all optimization problems better than others cannot exist. It has been observed that classical optimization methods are more efficient than evolutionary algorithms while solving linear, quadratic, strongly convex, unimodal, separable, and many other special problems. EAs shine when solving noisy, multimodal, discontinuous, and not differentiable functions [17].

The three main forms of evolutionary algorithms were proposed in the sixties. Fogel laid the roots of EP [18], Holland developed GAs while working at the University of Michigan [19]. In Europe, evolution strategies (ES) were developed by a group of students, Bienert, Rechenber, and Schwefel [20]. DE was specifically designed for numerical optimization, and is is one of the most successful methods to solve real-valued black box problems, it was proposed in the nineties [21].

2.2.1 Evolutionary programming

Lawrence J Fogel proposed EP in 1960 [18]. He observed the limitations in the approaches used in that era and concluded that these limitations were a byproduct of simulating human models and not the process that created these models: evolution. Recombination is generally not used, the mutation operators are usually powerful enough to generate similar perturbations [22]. EP appeared as an attempt to generate machine intelligence but it has been used as an optimizer.

A basic EP starts with a population larger than one, commonly randomly initialized. Each individual is perturbed to produce new individuals, this mutation operator can modify all variables at the same time.

There are three variants of this paradigm [17].:

- Original EP
- Continuous EP, where individuals are inserted at the same as they are generated.
- Self-adaptative EP, individuals incorporate one or more parameters of the optimization process.

2.2.2 Genetic algorithms

GAs were popularized by Holland and his students in the sixties [19]. In this type of EA, the main operator for producing variations is crossover. As proposed by Holland, GAs used a representation for the genotypes based in bitstrings, and the method of selection was proportional selection. Later implementations of GAs have used real-valued genotypes and different methods of selection. GAs manipulate genotype and transform these genotypes into phenotypes which are evaluated using an objective function. Each genotype is formed by alleles which are also called genes [23]. Algorithm 1 describes a basic genetic algorithm with operators of recombination and and mutation used to generate new genotypes each generation.

Algorithm 1 function genetic_algorithm(x_0)

```
Initialize population
Evaluate population
while Is a stopping criterion not satisfied? do
    t ← t + 1
    Select parents from generation t − 1
    Recombination and mutation to generate new individuals
    Evaluate individuals
    Select individuals to form generation t
end while
```

The initial population is usually randomly generated. Until a stopping criterion is satisfied a cycle

is repeated, in this cycle parents are selected. In canonical GA, proportional selection is used and the probability of selecting an individual is proportional to its fitness value. In Holland´s proposal two children were produced each generation, which randomly replace individuals of old population. GAs using proportional selection often suffer convergence problems caused by an individual that it is much better than any other individuals in the population, this produces a strong pressure towards a local optimal. To address the limitations in proportional selection, was proposed ranking selection which uses the rank of the individuals in a linear function to determine the probability of selection during the reproduction phase. GAs can have their bias on the reproduction step or during the replacement selection [17].

2.2.3 Evolution strategies

ESs were proposed by Bienert, Rechenberg, and Schwefel in the sixties while trying to minimize the drag of a robot in a wind tunnel [20]. The first version of evolutionary strategies, which nowadays is called the (1+1) ES, used discrete variables with binomial distributed mutations, this version did not include recombination. Recombination and the population principle were incorporated to more closely mimic evolution. Rechenberg later work, popularized the real-valued version [17].

ESs work with individuals, each individual contains a set of decision variables, and a set of endogenous parameters which is adapted in the evolution process. In the archetypic ES, to generate a new solution an individual is mutated by adding a normal distributed random vector. If the new solution is not worse than the old solution, then it replaces the old solution,this is (1+1)-ES.

Rechenberg extended ES and proposed multimembered ES, which is commonly denoted as $(\mu+1)-ES$, in this variant $\mu > 1$ parents are used to create one children per generation. Extending ES to more than one parent allowed the incorporation of recombination.

Parallel computers motivated the development of $(\lambda + \mu)$-ES and (λ, μ)-ES. These variants are very similar, except that the symbol $+$ indicates that the selection of survivors includes the μ parents and the λ offspring. In (λ, μ)-ES each individual lives only one generation, this avoids long stagnation

caused by misadapted strategy parameters [20].

2.2.4 Differential evolution

DE is an evolutionary algorithm for continuous optimization with applications in electrical engineering [24], [25], aeronautics [26], simulation of proteins [27], among many other applications [28].

In contrast to classical GAs [29], DE represents individuals using real valued vectors. The population in a DE algorithm is described by Equation 2.2.

$$\mathbf{x}_{i,G} = [x_{1,i,G}, x_{2,i,G} \dots x_{n,i,G}] \tag{2.2}$$

where $\mathbf{x}_{i,G}$ is the i-th individual of the population in the G-th generation and $x_{k,i,G}$ represents the k-th variable in $\mathbf{x}_{i,G}$.

DE is named after its mutation operator, which uses scaled differences of vectors to perturb individuals. This mutation operator gives adaptability to DE [30], which explains its success since its introduction in the nineties [31]. As originally proposed DE uses a mutation operator referred as rand/1 and binomial crossover to produce trial vectors, there is also a survival mechanism where a trial vector competes against its parent. Mathematically rand/1 is defined as Equation 2.3, the name of rand/1 comes from the criterion used to select individuals from the population which is random and the number after the bar indicates the number of vectors of differences used.

$$\mathbf{v}_{i,G} = \mathbf{x}_{r1,G} + F(\mathbf{x}_{r2,G} - \mathbf{x}_{r3,G}) \tag{2.3}$$

where $\mathbf{v}_{i,G}$ is a donor vector, $F \in (0, 1)$, $r1, r2, r3$ are different random integers that denote individuals in the population.

Binomial crossover is an operator which goal is to increase diversity [31], it creates a trial vector

Algorithm 2 function DE_algorithm(f)

```
Initialize random population
while Is a stopping criterion not satisfied? do
  Copy previous generation to u_{j,i,G}
  for every individual i in the population do
    Apply mutation operator (Equation 2.3)
    Apply crossover operator with a probability of Cr (Equation 2.4)
  end for
  for every individual i in the population do
    if f(x_{i,G}) <= f(u_{i,G}) then
      x_{i,G+1} ← u_{i,G}
    else
      x_{i,G+1} ← x_{i,G}
    end if
  end for
end while
```

by combination of the target and the donor vector. The operator is defined as:

$$\mathbf{u}_{j,i,G} \begin{cases} \mathbf{v}_{j,i,G} \text{ if } rand_{ij}[0,1] \leq Cr \text{ or } j = j_{rand} \\ \mathbf{x}_{j,i,G} \text{ otherwise} \end{cases} \tag{2.4}$$

where $\mathbf{u}_{i,G}$ is a trial vector, $\mathbf{v}_{i,G}$ a donor vector, and $\mathbf{u}_{i,G}$ the target vector. $rand_{i,j}$ are random numbers and Cr is the crossover rate.

Algorithm 2 describes canonical DE. In contrast to GAs, DE applies mutation before crossover. After the application of the variation operators comes the selection of a survivor. In this selection a individual do not need to be fitter than its parent to survive, only not worse [32].

2.3 Gradient based optimization methods

The gradient vector has the property that it is the direction of steepest ascent, by symmetry is also the direction of steepest descent. Both properties are local which makes methods using only gradient information not very effective in most problems [6].

2.3.1 Steepest descent method

Steepest descent is an iterative method that use the negative of gradient as a direction for minimization. The methods starts at an initial guess $\mathbf{x}_0$ and uses Equation 2.5 to generate new points.

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha_i \nabla f(\mathbf{x}_i) \tag{2.5}$$

where i is the iteration number and $\alpha_\mathbf{i} = \arg\min_{\alpha \in \mathbb{R}} \mathbf{x}_{i+1}$

The algorithm stops when the norm of the gradient is zero or is within a tolerance margin [6].

2.3.2 Conjugate direction methods

Conjugate directions methods form a class of global optimization methods with low-memory requirements and good convergence properties. Conjugate directions methods will optimize a quadratic function in n steps, where n is the size of the problem. One consequence of Taylor theorem is that a general nonlinear function can be approximated by a quadratic near to the optimum, this allows conjugate gradient methods to operate even in nonlinear functions, despite having been conceived for solving linear equation systems [9].

The Powell method extends the basic pattern search method and it was proposed in 1964 [33]. Conjugate directions are constructed using a set of line searches, to minimize a function f. To find the n conjugate directions in quadratic functions, the method performs n^2 line searches. For more complicated functions, the number of computations required is higher. However the proof of quadratic convergence of the algorithm assumes that exact minimums are found, which does not happen in practice, this increments the number of required iterations to solve a problem. One of the advantages of this method is that it does not require a function to be differentiable [6].

In contrast to the method of Powell, which uses canonical base unit vectors, conjugate gradient

methods use gradient as basis vectors to compute the descent directions, which reduces the number of line searches required from n^2 to n but it requires functions to be differentiable.

Conjugate gradient methods generate a sequence of points using Equation 2.6.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{2.6}$$

where α_k is a scalar that minimizes $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$ and $\mathbf{d}_k$ is the k-th search direction.

The set of search directions $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_k\}$ is generated using Equation 2.7.

$$\mathbf{d}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k \tag{2.7}$$

where β_k is a scalar computed using a formula that varies according to the conjugate gradient method used.

Hestenes and Stiefel proposed the first conjugate gradient method for the solution of linear equation systems characterized by symmetric positive definite matrices. This method solves linear equation systems by minimizing a quadratic function. Fletcher and Reeves extended this method for more general functions [6].

The formulas for Hestenes-Stiefel, Fletcher-Reeves and Polak-Riebiere–Polyak to calculate β_k are given [34]:

$$\beta_k^{HS} = \frac{\nabla f(\mathbf{x}_{k+1})^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}{\mathbf{d}_k^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)} \tag{2.8}$$

$$\beta_k^{FR} = \frac{\|\nabla f(\mathbf{x}_{k+1})\|^2}{\|\nabla f(\mathbf{x}_k)\|^2} \tag{2.9}$$

$$\beta_k^{PRP} = \frac{\nabla f(\mathbf{x}_{k+1})^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}{\|\nabla f(\mathbf{x}_k)\|^2} \tag{2.10}$$

where $\|.\|$ represents a vector norm.

If f is a quadratic function each formula is equivalent with an exact line search. For non-quadratic functions each formula leads to different performance. In particular, the Polak–Ribière–Polyak conjugate gradient method has been well studied and is considered one of the most efficient conjugate gradient methods [35]. The current best conjugate gradient methods are hybrid methods, which dynamically adjust the formula for β [36].

Conjugate gradient vectors have been successfully applied to solve civil engineering problems [37], chemistry [38], machine learning [39], and many others [40].

2.4 Chapter summary

This chapter introduced the concept of optimization, with a focus on global optimization. Definitions for neighborhood and local search were also given, these two concepts are necessary to understand what a local search operator does. The condition that a search direction has to satisfy to be a direction of improvement was also stated. This chapter also introduced the historical paradigms in EAs, and a modern paradigm created in the nineties, DE. It is in this EA that the local search operator proposed is incorporated. The descent directions used in the local search operator are computed using a conjugate gradient algorithm, this algorithm and its relation to gradients is also described.

3

State of the Art

In this chapter are presented concepts and works related to the solution of continuous optimization problems using algorithms that integrate local search techniques into EA. First in Section 3.1 is introduced the concept of MAs. Section 3.2 is about MAs that use DE as the population-based metaheuristic. Section 3.3 is focus on MAs that use gradient directions, the underlying population-based metaheuristic of these works is not necessarily DE.

3.1 Memetic algorithms

In 1997, the no-free lunch theorem showed that any two optimization algorithms are equivalent in performance when averaged across every possible problem [41]. A corollary from this theorem is that specialization, design problem specific algorithms should yield best results.

Cooperative metaheuristics are a class of stochastic global search algorithms which combine population-based metaheuristics with problem specific solvers, where the idea is to exploit problem specific knowledge to accelerate the finding of good solutions. Cooperative metaheuristics are

classified according to the interactions of their parts:

Low/High Level

- Low level: a given function of a metaheuristic is replaced by another method.
- High level: the algorithms are self-contained

Relay/Teamwork

- Relay: set of methods that act in a pipeline fashion
- Teamwork: represents cooperative optimization methods.

MAs have been classified as LTH. According to the definition of MA described in [1], in this class of algorithms a transformation operator, commonly the mutation operator, is replaced by an exact search method within a subspace of the search space and it is possible that the GA part and the exact search method are executed at the same time.

A taxonomy of cooperative metaheuristics, where MAs are classified as low level team heuristics is presented in Figure 3.1.

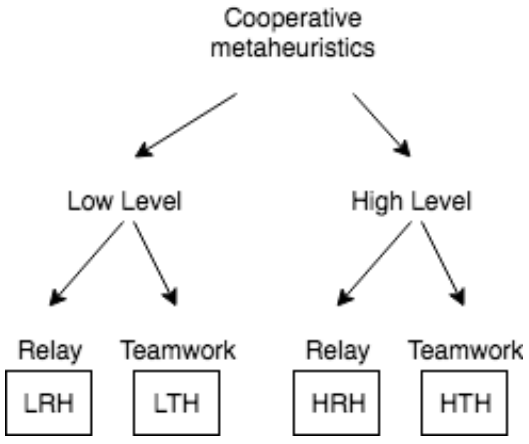


Figure 3.1: Taxonomy of cooperative metaheuristics by [1]

Moscato proposed MAs inspired by Lamarckian ideas of individual lifetime learning [7]. This learning consists of domain-specific information to produce a better performing algorithm [42].

The most common type of MA uses heuristic local searches to improve some solutions. The evolutionary part of this type of algorithms identifies promising regions of the search space, while the local search mechanism explores the neighborhood of a particular solution. MAs have been used in discrete optimization [43] [44], continuous optimization, constrained optimization [45].

There is no universal agreement on the use of the term "memetic algorithm" due to the large existing number of variations, some names that have been used to call MAs are hybrid GAs, Baldwinian EAs, Lamarckian EAs, genetic local search algorithms, and many other names. One of the more pressing concerns in MAs design is diversity maintenance while using local search mechanisms [7].

Lamarckian algorithms have a faster convergence speed than Baldwian algorithms, but Lamarckianism tends to suffer premature convergence, and Baldwinian learning is less likely to produce stagnation. This different behavior is a consequence of the rule used to replace solutions. In Lamarckian algorithms, both the fitness value and the individual change, while in Baldwinian algorithms only the fitness value evolve.

Algorithm 3 describes a local search based MA. In this algorithm the local search operator is applied after the application of recombination or mutation to a fixed number of individuals per generation.

This thesis is focused in continuous optimization, so the studies discussed belong to this type of problems.

3.2 Memetic differential evolution

DE has been one of the most competitive methods for numeric optimization, even since its introduction [31]. MAs were originally proposed for solving combinatorial optimization problems,

Algorithm 3 function basic_MA(par, P)

```
1: Initialize population
2: Evaluate fitness of the population
3: while Is a stopping criterion not satisfied? do
4:    for i ← 1 upto recombination number do
5:       Select parents
6:       Apply recombination
7:       Apply local search operator
8:       Insert individual
9:    end for
10:   for i ← 1 upto mutation number do
11:      Select individual
12:      Apply mutation
13:      Apply local search operator
14:      Insert individual
15:   end for
16: end while
17: return fittest individual
```

eventually this methodology was applicated to solve continuous optimization problems, and since DE is a successful EA it was a natural step to introduce memetic strategies to DE. The works presented in this section are part of this class of algorithms.

In [46] is proposed a directional mutation operator, their motivation is that the best fitness found in DE cannot be improved in every generation. The mutation operator in DE use difference vectors, which are used as the descent directions. The main idea is that difference vectors constructed using trial solutions that are better than the global best have a higher probability of being good descent directions than different vectors that are selected randomly. A pool of difference vectors is constructed these directions, which are used instead of the randomly formed descent directions of canonical differential evolution. One limitation of this approach is that each vector of the pool is only used once and with no fitness improvement happening the pool is empty and the operator is not activated.

Multiple offspring sampling (MOS) is a framework for the development of MAs based in DE that allows the development of both HTH and HRH algorithms, it was proposed in [47]. This

framework refers to each mechanism to generate new candidate solutions as a technique and adjust the partition ratio of these tecniques using two main approaches: a central approach that uses quality and participation functions and a self-adapting approach where the participation ratio is encoded within individuals. Among these two approaches, the self-adapting approach performed worse that the former, as it suffers premature convergence in the selection of techniques, as the best techniques change during the execution of an evolutionary algorithm. DE is combined with the Hookes-Jeeves method to study the Baldwin effect in [48]. A different direction was taken in [49] where the main idea was to optimize DE scale factor F using a hill climber, this decision decoupled the cost of the local search with the number of variables in the problems.

There are methods that rely on randomness, in [50] is proposed a MA that relies on Eager Random Search to enhanced DE. In the paper are presented three strategies based on three probability distributions to explore the search space of a trial solution x, the best performing strategy used was named differential evolution Cauchy local search (DECLS), this variant used a Cauchy distribution to have more chances to escape local optima. Compared to a normal distribution, a Cauchy distribution has a wider distribution, which means that values distant of the mean have a higher chance and this allows bigger moves than a normal distribution.

Chaos is a typical nonlinear phenomenon in nature which is characterized by ergodicity, randomicity and sensitivity to its initial conditions, in [51] is proposed an algorithm that incorporates a local search mechanism that use chaotic local search, based on logistic chaotic function, with a non-linear shrinking strategy. The algorithm starts its execution with a relatively big population which is reduced each generation. A more elitist strategy is pursuit in [52], in this approach the local search strategy is based on information from the elite individuals to exploit promising regions of the search.

3.3 Memetic algorithms using gradients

In this section are mentioned methods that use the gradient as the descent direction, in most of these algorithms, additional operators were incorporated to balance the tendency of falling trapped in local optima that the steepest descent method has.

Gradient evolution (GE) [53] explores the search space using a set of vectors with three main operators: update, jump and refresh. The main operator is the modification of a vector using the direction of the gradient. The direction of the steepest descent tends to attract the algorithm to local optima, the jump operator allows the algorithm to escape these optima. A refresh operator helps the algorithm to avoid getting locked in certain regions of the space as a side effect of the elitist search. Search directions are determined using the Newton-Raphson method. In the experimentation performed the GE obtained better performance in term of number of iterations, compared to PSO, and bacterial colony optimization (BCO), in the majority of the benchmark functions.

In [8] is proposed a hybrid algorithm with three levels, DE hybridized with SA and traditional gradient optimization (DE-SA Newton) and DE with gradient optimization (DE-Newton). The element of SA used in DE-SA Newton is in the variation of the parameter F of DE using an equation reminiscent of the acceptation criterion of SA. DE-Newton has a balance between convergence speed to a local optimum and the ability to find the global optimum. Searching near a optimal is accelerated using gradients as directions of descent, which can be integrated as an additional step in DE. The algorithm DE-Newton proposed is faster than the other variant (SA-DE-Newton) but it might stagnate before finding the optimal solution, both algorithms are superior to canonical DE.

The direction of the steepest descent can drive populations to stagnation, to avoid this drawback, in [54] are proposed search strategies based on the gradient and polar coordinates. This approach combines DE with the steepest descent method and polar coordinates, which expands the search range and avoid local optima while conserving population diversity.

Gradient subspace approximation (GSA) [55] uses neighborhood information of a point x_k to

Table 3.1: Comparison of Memetic Algorithms using gradients

Reference	Hybridation type	Advantages	Disadvantages
Schuetze et al. 2016 [55].	Differential Evolution with gradient based local search.	The descent directions calculated are free in term of functions evaluations	Only feasible for problems with a moderate number of variables
Kuo et al. 2015 [53].	Genetic Algorithm with gradient based local search	Competitive in multimodal functions	Elitist strategy is sensitive to parameter choices.
Rafajowicz 2015 [8].	Differential Evolution with gradient based local search.	Fast convergence speed balanced with global search power	Tendency to stagnation, compared to other hybrid methods
Yang et al. 2015	Differential evolution with local search using gradient and polar coordinates	Good convergence speed	In functions with many uniformly distributed local optima, this algorithm did not performed as well

compute the direction of the maximum descent in a induced subspace. When the induced space coincides with the complete search space, the computed vector is an approximation of the gradient vector. The generated descent direction can be used both a standalone method or in a local search operator in population-based metaheuristics. In the case of population-based metaheuristics the neighborhood information is obtained for free, which is cheaper than estimators based in finite differences.

Table 3.1 summarizes advantages and disadvantages of the methods presented in this section.

3.4 Chapter Summary

This chapter presented a taxonomy of cooperative metaheuristics and introduced MAs as a class of algorithms of this category. MAs are commonly seen as an EA with some kind of local search incorporated. In these algorithms there are two types of learning: Baldiwnian learning and Lamarkian learning, which is the type of learning used in the algorithms presented in this chapter. Even if the type of learning was the same, the local search mechanisms used took different forms, for example in one of the studies mentioned, researchers hypothesized that individuals that outperform the global best can be pooled and used to produce better search directions.

Local search can be applied to parameters, in of the studies mentioned in this chapter, a hill climber was used to optimize the scale factor in DE. A Aggressive local search strategies often lead to local optima, random and chaos phenomena based operators can be used to escape these optima.

In MAs, the ideal local search operator can change during the differences phases of the algorithm, so there are MAs which incorporate more than one local search operator. MOS is a framework specifically designed to develop MAs with multiple local search operators.

Local search operators inspired by calculus also exist. Standalone calculus methods cannot discern whether an optimum is local or global. In this chapter were introduced methods that use the steepest descent direction as the descent direction, this can led to stagnation, most of these methods incorporate mechanisms to escape local optima. The strategies used are varied and included a new coordinate system, a extra metaheuristic (SA) that modifies the parameter F, and operators of reset, triggered when is detected stagnation.

4

Proposed approach

The proposed approach consists of a local search operator based on conjugate directions, which is integrated in an EA. The directions used are constructed with gradients computed by an estimator, in Section 4.1 this estimator is introduced. Section 4.2 presents the conjugate gradient method. Section 4.3 is devoted to the method of interpolation used to compute the step lengths required by the conjugate gradient method. Finally Section 4.4 focus on the MA developed.

4.1 Gradient estimator

The estimator used to compute gradients is GSA, which was proposed in [55]. GSA is a set oriented gradient estimator, which main advantage is that it does not need a set of coordinate aligned points. This feature allows GSA to repurpose fitness values computed by EAs to avoid additional function evaluations.

The idea behind this estimator is to compute the greediest descent direction in a subspace spanned by $\mathbf{x}_0$ and its neighbors. The direction generated is an approximation of the gradient when

the spanned space covers the entire search space. In Euclidean spaces a neighborhood is defined by a radius ϵ, in the proposal the neighborhood radius selected is adjusted to include n solutions, in order to span the entire search space and produce an estimation of the gradient vector. The search of the greediest descent direction is formulated as an optimization problem that it is analytically solved with the Karush–Kuhn–Tucker conditions. These conditions are used are to solve optimization problems with inequality constraints. These conditions are necessary but not sufficient to ensure the presence of local optima in non convex functions [6].

A gradient estimation at $\mathbf{x}_0$ is computed using Equation 4.1.

$$\mathbf{g} = V(V^T V)^{-1}\mathbf{d} \tag{4.1}$$

where $V = \{\mathbf{v}_i, \ldots, \mathbf{v}_r\}$ is a matrix with r linearly independent n-dimensional vectors, $\mathbf{v}_i = \frac{\mathbf{x}_i - \mathbf{x}_0}{||\mathbf{x}_i - \mathbf{x}_0||}$, $d_i = \frac{f(\mathbf{x}_i) - f(\mathbf{x}_0)}{||\mathbf{x}_i - \mathbf{x}_0||}$ and $\mathbf{g}$ is a gradient estimation at point $\mathbf{x}_0$.

This descent direction can be used in an optimization method to generate a succession of monotonically better solutions. Even used as a standalone method, GSA was proven superior to the Nelder-Mead method in optimization problems. This standalone GSA is roughly equivalent to the steepest descent method and generates a sequence of points in the form given by Equation. 4.2.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) \tag{4.2}$$

where $\mathbf{x}_0$ is an initial solution, $\nabla f(\mathbf{x}_k)$ is the gradient at $\mathbf{x}_k$,and α_k is an scalar that minimizes $f(\mathbf{x}_{k+1})$.

Algorithm 4 describes standalone GSA. Each iteration a new $\mathbf{x}_{k+1}$ is generated by linear combination of $\mathbf{x}_k$, and a search direction $\mathbf{p}_k$ with a parameter α_k, which is the solution of a smaller optimization problem $\alpha_k = \arg\min_{\alpha_k \in \mathbb{R}} f(\mathbf{x}_k - \alpha_k \mathbf{p}_k)$. The execution of this procedure continues until $|\nabla f(\mathbf{x}_k)|$ is within a tolerance margin or the number of iterations set is exhausted

Algorithm 4 function standalone_GSA(par, P)

while Is a stopping criterion not satisfied? **do**
  $fxk \leftarrow f(\mathbf{x}_k)$
  $\mathbf{p}_k \leftarrow$ gradient estimation at $\mathbf{x}_k$
  Find an α_k that minimizes $f(\mathbf{x}_k - \alpha_k \mathbf{p}_k)$
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha_k \mathbf{p}_k$
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k+1}$
end while
return fittest individual, ac

[6]. In the implementation used in this thesis $\nabla f(\mathbf{x}_k)$ is computed using a gradient estimator.

4.2 Conjugate gradient

There are many conjugate gradient methods described in the literature [36], but these methods operate using exact gradients, and the gradients computed by GSA are only approximations. This implies that in practice the performance of these methods can differ from the results published in the literature, which means that it is pertinent to perform experiments to select a conjugate gradient formula. These experiments included three conjugate gradient methods: Hestenes-Stiefel (Equation 4.3), Fletcher-Reeves (Equation 4.4), and Polak-Ribiere-Polyak (Equation 4.5). The first experiment executed involved an environment where the conjugate gradient methods were used as a standalone optimization method in a unimodal function, this experiment used a neighborhood with radius ϵ and confirmed Polak-Ribiere-Polyak as the best performing formula among the ones tested. A similar second experiment was performed, in this experiment a multimodal function was used, and the neighborhood consisted of the individuals of a DE algorithm. The results of this experimentation differed from the previous one and Fletcher-Reeves was the best method. This can be attributed to the inaccuracy in the gradients used to construct the conjugate directions, which favored the simplest of the formulas tested, the Fletcher-Reeves formula. Algorithm 5 displays the conjugate gradient method used as a local search operator in this thesis. In this algorithm, each conjugate direction

Algorithm 5 function conjugate_gradient(x_0)

Initialize population
$\mathbf{x}_k \leftarrow \mathbf{x}_0$
$\mathbf{d}_k \leftarrow$ Estimate gradient at $\mathbf{x}_k$
while Is a stopping criterion not satisfied? **do**
  Compute α_k using quadratic interpolations
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha_k \mathbf{d}_k$
  Estimate gradient at $\mathbf{x}_{k+1}$
  Compute β_k using Equation 4.4
  $\mathbf{d}_{k+1} \leftarrow -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k$
  $k \leftarrow k + 1$
end while
return x_k

is a linear combination of the previous conjugate directions, and the first iteration is equivalent to the steepest descent method. These directions are used as descent directions each iteration and the algorithm stops when a fixed number of iterations is reached.

$$\beta_k^{HS} = \frac{\nabla f(\mathbf{x}_{k+1})^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}{\mathbf{d}_k^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)} \tag{4.3}$$

$$\beta_k^{FR} = \frac{\|\nabla f(\mathbf{x}_{k+1})\|^2}{\|\nabla f(\mathbf{x}_k)\|^2} \tag{4.4}$$

$$\beta_k^{PRP} = \frac{\nabla f(\mathbf{x}_{k+1})^T (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}{\|\nabla f(\mathbf{x}_k)\|^2} \tag{4.5}$$

4.3 Step length optimization

The method of the quadratic interpolations can be used to find the step lengths required by the local search operator. Let f be a multivariate objective function, the problem of minimization along a direction given by an unitary vector v, is finding the λ^* which minimizes Equation 4.6 [6].

$$f(\lambda) = f(\mathbf{x}_0 + \lambda \mathbf{v}) \tag{4.6}$$

where $\lambda \in \mathbb{R}$, $\mathbf{v}, \mathbf{x}_0 \in \mathbb{R}^n$. Point $\mathbf{x}_0$ is the initial point in the search, λ^* is a factor multiplying the search direction, unitary vector $\mathbf{v}$.

The three points used to interpolate the quadratic curve are obtained by evaluating the objective function at three different points, using these points, linear system formed is described by the next three equations:

$$f(\mathbf{x}_0 + \lambda_1 \mathbf{v}) = A\lambda_1^2 + B\lambda_1 + C \tag{4.7}$$

$$f(\mathbf{x}_0 + \lambda_2 \mathbf{v}) = A\lambda_2^2 + B\lambda_2 + C \tag{4.8}$$

$$f(\mathbf{x}_0 + \lambda_3 \mathbf{v}) = A\lambda_3^2 + B\lambda_3 + C \tag{4.9}$$

The initial points are arbitrary values for λ, in order to save one function evaluation, λ_1 was set to 0. The remaining values of λ were set to 0.5 and 1 in order to not move too far of $\mathbf{x}_0$. The system formed is solved and the best point in this quadratic is found in the geometric place given by Equation 4.10 if and only if $A > 0$.

$$\lambda^* = \frac{-B}{2A} \tag{4.10}$$

This λ^* and the two best points of the previous iteration are used to fit another quadratic until the maximum number of iterations is reached or until is not possible to generate a better point using the same procedure.

The pseudocode of the algorithm implemented is listed in Algorithm 6. Each iteration, a new point is generated and is tested if it is better or not, it ends when the number of iterations is spent

Algorithm 6 function qinterpolations_algorithm()

$(\lambda_1, \lambda_2, \lambda_3) \leftarrow (0, 0.5, 1)$
$(f1, f2, f3) \leftarrow (f(\lambda_1), f(\lambda_2), f(\lambda_3))$
while Is the number of iterations already spent? **do**
  Solve linear system to obtain A, B, C
  if $A > 0$ **then**
    $\lambda^* \leftarrow \frac{-B}{2A}$
    $\mathbf{x}^* \leftarrow \mathbf{x}_0 + \lambda^* \mathbf{v}$
    $fn \leftarrow f(\lambda^*)$
    if fn is the worst in a set that also includes $f1, f2, f3$ **then**
      break
    end if
    Add $\mathbf{x}^*$ to a pool of solutions
    $a \leftarrow$ Sort $[(f1, \lambda_1), (f2, \lambda_2), (f3, \lambda_3), (fn, \lambda^*)]$ in ascending order
    $(f1, \lambda_1) \leftarrow a_1$
    $(f2, \lambda_2) \leftarrow a_2$
    $(f3, \lambda_3) \leftarrow a_3$
  else
    break
  end if
end while
return (f1, λ_1)

or the best point of the interpolated parabola is worst than the previously computed points. At the end of each iteration the current solution is added to a pool of solutions that for the purposes of the gradient estimator is temporally added to the points used in the estimator.

4.4 Development of memetic algorithm

In a MA, an EA is endowed with some kind of local search [56]. This mechanism enhances the exploitation in the promising regions found by an EA. This exploitation should not be so aggressive that the loss of diversity causes premature convergence. Among many others, aspects that need to be taken into account when designing MAs are [7]:

- The frequency of the application of the local search

- The depth of the local search
- The individuals that the operator will refine

These aspects are related to the relatively high cost of using a local search operator compared with the evolutionary operators. As Figure 4.1 shows, in the developed approach the local search is executed each generation after the evolutionary operators.

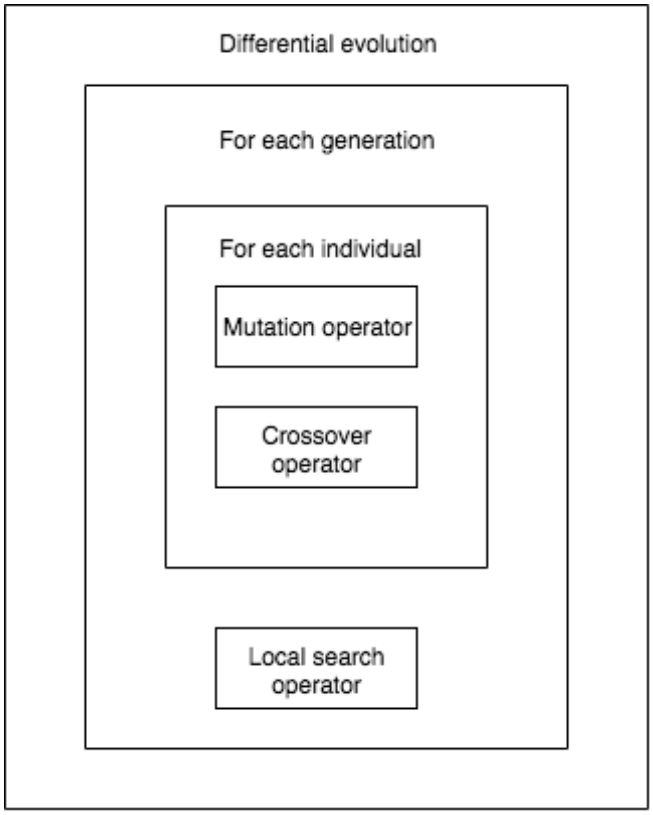


Figure 4.1: Local search operator in the proposed approach

The proposed approach is sketched in Algorithm 7, in the proposal the individual selected for the application of the operator can be either the individual with the best fitness value in the population or an individual of the first tercile with the best solutions in the population. This decision discriminates against not promising regions of the search space. The algorithm maintains a list with a boolean value indicating if the local search operator has been applied to an individual and the individual

Algorithm 7 function hybrid_algorithm(par, P)

```
Initialize random population
ac ← 0
while Is a stopping criterion not satisfied? do
  Execute one generation of canonical DE
  ac ← ac + 1
  while ac > 0 do
    if if best individual is not in taboo list then
      The individual selected is the best individual
    else
      Select a random individual in percentile 33
      while if the individual selected is in the taboo list do
        Select a random individual in percentile 33
      end while
    end if
    if a individual was selected then
      Apply local search operator to selected individual
      ac ← ac − 1
    else
      break
    end if
  end while
end while
return fittest individual, ac
```

selected is avoided. The discrimination of these individuals prevents the repetition of a failed local search. In the proposed approach the individual with the best fitness is the default choice, otherwise an individual of the first tercile is selected. If there is not a single individual that is eligible for the application of the operator, the schedule application of the generation accumulates for the next generation, in average the operator is applied one time per generation.

Two variants of Algorithm 7 are proposed, in Chapter 5 these variants are compared. One of these variants uses only a gradient estimation as the descent direction, which can be seen as a steepest descent operator using only one iteration (DE/GSA). The second algorithms uses conjugate directions (DE/GSA-CG) with a stopping criterion given by a percentage of improvement set to 2.5%. This percentage is computed using Equation 4.11.

$$imp = 100\frac{f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})}{\mathbf{x}_k} \tag{4.11}$$

GSA produces greedy directions that are approximations of the gradient when the number of points in the neighborhood is n. This value is used to construct the gradients used in the Fletcher-Reaves method. Table 4.1 lists the parameters used for the the algorithm of quadratic interpolations.

Parameter	Value
Maximum iterations	10
Initial λ_1	0
Initial λ_2	0.5
Initial λ_3	1

Table 4.1: Parameters used in the proposed approach

4.5 Chapter summary

This chapter detailed the local search operator proposed in this thesis work. This local search operator is based on directions generated by GSA, a gradient estimator. GSA can repurpose the fitness values stored in EAs, which means that it is free in terms of additional function evaluations.

In this thesis, the version of GSA used a neighborhood with n elements, which is a subset of the union of the current population and the individuals generated during the current application of the operator. These directions are used to construct conjugate directions with the Fletcher-Reeves formula. The conjugate directions are incorporated as a local operator in a MA based on DE. In the proposal, the local search operator is applied in average one time per generation and it is applied to the best individual if certain condition is satisfied. Otherwise the same condition is randomly tested in individuals ranked in the first tercile according to their fitness values. If no individual satisfies the selection criteria the pending application is executed in the next generations.

5

Experimentation and Results

In this chapter are presented the results generated by the experimentation performed. In Section 5.1 are listed the parameters used in the experimentation. Three different algorithms and the proposed approach were compared in these experiments. In Section 5.2 DE and the proposed approach are compared and the superiority of the developed approach over DE is shown. The proposed algorithm is compared in Section 5.3 with DE/GSA and DECLS.

5.1 Experimental settings

Benchmark problems retrieved from the competition of the Congress on Evolutionary Computation 2017 [57] are used to compare the four algorithms studied, descriptions of these problems have been annexed to Appendix A.1. Table 5.1 indicates whether a function has a local optimum, which is also the global optimum (unimodal) or the function has many local optima (multimodal).

Function	Modality	Function	Modality
$f1$	unimodal	$f2$	unimodal
$f3$	unimodal	$f4$	unimodal
$f5$	multimodal	$f6$	multimodal
$f5$	multimodal	$f6$	multimodal
$f7$	multimodal	$f8$	multimodal
$f9$	multimodal	$f10$	multimodal
$f11$	multimodal	$f12$	multimodal
$f13$	multimodal	$f14$	multimodal
$f15$	multimodal	$f16$	multimodal
$f17$	multimodal	$f18$	multimodal

Table 5.1: Modality of bechmark functions

The parameters used for the experiments performed are included in Table 5.2. These parameters include population size [58], F and CR which correspond to the parameters of a DE algorithm [59]. The number of variables in the experimentation performed was also varied in order to see how well the performance of the algorithms scale. To produce a fair comparison between the approaches employed, the computational cost is defined in terms of the number of function evaluations, which is the number of times the objective function is called. The performance of stochastic algorithms is usually analyzed using non-parametric test. Nonetheless normality is still tested using a Shapiro-Wilks test. Two tests are used to test whether the samples generated from the results have the same distribution. Either the Friedman test or Wilcoxon sign-rank test are executed to detect whether an algorithm A is really better than an algorithm B or the statistics generated are not meaningful [60]. The confidence level used for each test is set to 0.05, which is a commonly used value.

The number of independent runs for the experimentation performed is 25, and the settings used are summarized in Table 5.2.

Parameters	Value
Domain	$x_i \in [-100, 100]$
Number of variables	$n = [20, 50, 100]$
Function evaluations	$5000n$
Population size	$10n$
F	$\frac{3}{5}$
CR	$\frac{4}{5}$

Table 5.2: Parameters used for the evolutionary part of the algorithm

Parameters	Value
F	0.9
CR	0.85
M	5
j	0.1
t	0.2

Table 5.3: Parameters used for DECLS

For the experimentation with DECLS most parameters remained the same, but the ones that changed were adjusted to match what the authors proposed, these settings are included in Table 5.3.

.

5.2 Comparison between DE and the proposed approach

5.2.1 Results

This comparison involved canonical DE and DE/GSA-CG. For reasons of space, the results of each experiment are included in separated tables, and this arrangement is indexed in Table 5.4. These indexes include the number of variables, the number of function evaluations and the problems included.

Function		DE/GSA-CG	DE	Wilcoxon
f1	Mean	**4.288E+05**	4.742E+05	
	Median	4.856E+05	**4.853E+05**	
	SW	**3.323E-02**	6.514E-01	4.926E-01
f2	Mean	3.819E+10	**1.552E+10**	
	Median	2.629E+10	**1.092E+10**	
	SW	**4.410E-02**	**9.697E-06**	**9.417E-03**
f3	Mean	1.120E+04	**1.114E+04**	
	Median	1.108E+04	**1.103E+04**	
	SW	**2.010E-03**	6.997E-01	7.164E-01

Table 5.5: Fitness values after 100000 function evaluations for functions f1-f3 with $n = 20$

n	Function evaluations	Functions	Tables
20	100000	f1-f3	5.5
20	100000	f4-f18	5.6
50	250000	f1-f8	5.7
50	250000	f9-f16	5.8
50	250000	f16-18	5.9
100	500000	f1-f6	5.10
100	500000	f7-f12	5.11
100	500000	f13-18	5.12

Table 5.4: Index of tables with the generated results

The tables mentioned in Table 5.4 are composed of two columns which represent the compared algorithms, and rows for three statistical properties: mean, median, and the p-value produced by a Shapiro-Wilks test. An additional column contains the p-value of a Wilcoxon signed-rank test, the level of confidence was set to 0.05 and the p-values that satisfy this condition and the test of Shapiro-Wilks are marked in bold. Also in bold are the fitness values of the best algorithm for each function.

Function		DE/GSA-CG	DE	Wilcoxon
f4	Mean	4.732E+04	**2.824E+04**	
	Median	3.912E+04	**2.380E+04**	
	SW	**1.044E-03**	**1.308E-02**	**1.382E-02**
f5	Mean	**1.253E+02**	1.323E+02	
	Median	**1.250E+02**	1.334E+02	
	SW	1.136E-01	4.181E-01	**3.955E-02**
f6	Mean	**8.278E+00**	8.302E+00	
	Median	8.327E+00	**8.324E+00**	
	SW	5.405E-02	4.496E-01	6.571E-01
f7	Mean	4.926E+01	**4.672E+01**	
	Median	5.211E+01	**4.644E+01**	
	SW	1.824E-01	**2.786E-02**	3.002E-01
f8	Mean	**1.165E+03**	1.285E+03	
	Median	**1.226E+03**	1.341E+03	
	SW	**1.705E-05**	6.272E-01	9.797E-02
f9	Mean	**3.284E+03**	3.727E+03	
	Median	**3.251E+03**	3.373E+03	
	SW	1.062E-01	**1.194E-02**	1.425E-01
f10	Mean	**3.333E+00**	4.124E+00	
	Median	**3.354E+00**	4.333E+00	
	SW	6.964E-01	7.794E-01	**6.848E-03**
f11	Mean	**2.060E+01**	2.064E+01	
	Median	**2.060E+01**	2.065E+01	
	SW	**1.207E-03**	3.963E-01	**3.507E-03**
f12	Mean	**2.079E+01**	2.192E+01	
	Median	**2.100E+01**	2.241E+01	
	SW	5.376E-01	6.935E-02	**1.100E-02**
f13	Mean	6.013E-01	**5.959E-01**	
	Median	**5.947E-01**	5.974E-01	
	SW	1.882E-01	6.445E-01	5.272E-01
f14	Mean	**1.637E+00**	1.751E+00	
	Median	**1.658E+00**	1.765E+00	
	SW	**1.010E-02**	6.362E-01	2.209E-01
f15	Mean	**5.326E-01**	6.205E-01	
	Median	**5.359E-01**	6.219E-01	
	SW	6.469E-01	6.852E-01	**1.186E-03**
f16	Mean	**3.661E-01**	3.844E-01	
	Median	**3.659E-01**	3.889E-01	
	SW	8.984E-01	8.420E-01	2.758E-01
f17	Mean	1.571E+07	**1.359E+07**	
	Median	1.349E+07	**1.029E+07**	
	SW	**2.030E-03**	**1.034E-03**	7.570E-01
f18	Mean	**6.053E-01**	8.295E-01	
	Median	**5.994E-01**	8.032E-01	
	SW	9.698E-01	8.582E-01	**2.958E-04**

Table 5.6: Fitness values after 100000 function evaluations for functions f4-f18 with $n = 20$

Function		DE/GSA-CG	DE	Wilcoxon
f1	Mean	**3.803E+10**	3.890E+10	
	Median	**3.827E+10**	3.839E+10	
	SW	1.287E-01	6.946E-01	5.272E-01
f2	Mean	**5.908E+67**	5.986E+67	
	Median	**7.789E+66**	1.801E+67	
	SW	1.000E+00	1.000E+00	8.824E-01
f3	Mean	**1.143E+05**	1.291E+05	
	Median	**1.172E+05**	1.305E+05	
	SW	**7.171E-08**	9.676E-01	**1.721E-03**
f4	Mean	**1.627E+10**	2.178E+10	
	Median	**1.965E+10**	2.177E+10	
	SW	**3.692E-04**	4.281E-01	1.829E-01
f5	Mean	4.492E+04	**4.273E+04**	
	Median	4.500E+04	**4.329E+04**	
	SW	2.996E-01	5.025E-01	7.800E-02
f6	Mean	**2.287E+01**	2.306E+01	
	Median	**2.291E+01**	2.309E+01	
	SW	**1.949E-02**	2.836E-01	**1.489E-02**
f7	Mean	**1.934E+04**	2.200E+04	
	Median	**1.987E+04**	2.263E+04	
	SW	**1.471E-06**	6.067E-02	**2.466E-02**
f8	Mean	**7.130E+03**	7.415E+03	
	Median	**7.235E+03**	7.535E+03	
	SW	**2.944E-02**	**1.547E-02**	**4.797E-02**

Table 5.7: Fitness values after 250000 function evaluations for functions f1-f8 with $n = 50$

Function		DE/GSA-CG	DE	Wilcoxon
f9	Mean	**4.092E+08**	4.252E+08	
	Median	**4.161E+08**	4.279E+08	
	SW	6.014E-01	6.272E-01	4.118E-01
f10	Mean	**1.129E+05**	1.310E+05	
	Median	**1.101E+05**	1.291E+05	
	SW	6.923E-01	1.413E-01	**9.804E-04**
f11	Mean	**2.104E+01**	2.106E+01	
	Median	**2.105E+01**	2.107E+01	
	SW	2.492E-01	5.805E-01	**2.643E-02**
f12	Mean	**6.901E+01**	7.006E+01	
	Median	**6.947E+01**	6.998E+01	
	SW	**2.090E-02**	7.823E-01	6.934E-02
f13	Mean	**1.176E+01**	1.177E+01	
	Median	1.190E+01	**1.171E+01**	
	SW	1.122E-01	7.796E-01	5.998E-01
f14	Mean	**3.074E+00**	3.216E+00	
	Median	**3.109E+00**	3.262E+00	
	SW	3.346E-01	2.009E-01	9.797E-02
f15	Mean	**4.353E+02**	4.557E+02	
	Median	**4.369E+02**	4.533E+02	
	SW	7.440E-01	7.729E-01	8.265E-02
f16	Mean	4.228E+04	**4.202E+04**	
	Median	**4.221E+04**	4.354E+04	
	SW	5.604E-01	5.009E-02	7.570E-01

Table 5.8: Fitness values after 250000 function evaluations for functions f9-f16 with $n = 50$

Function		DE/GSA-CG	DE	Wilcoxon
f17	Mean	**3.186E+16**	3.192E+16	
	Median	**3.194E+16**	3.297E+16	
	SW	4.541E-01	8.397E-01	9.464E-01
f18	Mean	**9.329E-01**	1.026E+00	
	Median	**9.551E-01**	1.044E+00	
	SW	1.014E-01	**4.821E-02**	**2.259E-03**

Table 5.9: Fitness values after 250000 function evaluations for functions f17-f18 with $n = 50$

Function		DE/GSA-CG	DE	Wilcoxon
f1	Mean	2.233E+11	**2.106E+11**	
	Median	2.214E+11	**2.164E+11**	
	SW	**3.546E-02**	5.866E-02	**1.382E-02**
f2	Mean	**2.379E+161**	5.230E+161	
	Median	**1.514E+159**	4.643E+160	
	SW	1.000E+00	1.000E+00	**5.816E-03**
f3	Mean	**2.522E+05**	2.611E+05	
	Median	**2.624E+05**	2.759E+05	
	SW	**7.793E-09**	**8.484E-09**	**3.467E-02**
f4	Mean	1.182E+11	**1.137E+11**	
	Median	**1.195E+11**	1.198E+11	
	SW	9.535E-02	**1.316E-08**	4.593E-01
f5	Mean	**2.240E+05**	2.278E+05	
	Median	**2.259E+05**	2.298E+05	
	SW	**3.177E-02**	2.553E-01	2.418E-01
f6	Mean	**4.733E+01**	4.843E+01	
	Median	**4.736E+01**	4.757E+01	
	SW	2.917E-01	**2.253E-10**	**1.303E-03**

Table 5.10: Fitness values after 500000 function evaluations for functions f1-f6 with $n = 100$

Function		DE/GSA-CG	DE	Wilcoxon
f7	Mean	7.290E+04	**7.012E+04**	
	Median	7.312E+04	**7.290E+04**	
	SW	1.800E-01	**1.781E-07**	4.432E-01
f8	Mean	**1.848E+04**	4.829E+09	
	Median	**1.860E+04**	1.882E+04	
	SW	7.496E-01	**1.214E-10**	1.742E-01
f9	Mean	4.573E+09	**4.420E+09**	
	Median	**4.577E+09**	4.640E+09	
	SW	6.007E-01	**5.516E-08**	9.893E-01
f10	Mean	**2.988E+05**	3.148E+05	
	Median	**3.023E+05**	3.221E+05	
	SW	7.228E-01	**3.840E-06**	**2.831E-02**
f11	Mean	**2.039E+01**	2.042E+01	
	Median	**2.125E+01**	2.128E+01	
	SW	**1.335E-10**	**1.327E-10**	**1.382E-02**
f12	Mean	1.546E+02	**1.507E+02**	
	Median	**1.548E+02**	1.572E+02	
	SW	7.285E-02	**3.156E-10**	**2.831E-02**

Table 5.11: Fitness values after 500000 function evaluations for functions f17-f12 with $n = 100$

Function		DE/GSA-CG	DE	Wilcoxon
f13	Mean	5.805E+01	**5.611E+01**	
	Median	**5.791E+01**	5.850E+01	
	SW	9.605E-01	**1.976E-08**	9.250E-01
f14	Mean	**3.717E+00**	3.905E+00	
	Median	**3.729E+00**	4.000E+00	
	SW	**8.146E-03**	**1.564E-03**	**1.018E-02**
f15	Mean	1.144E+03	**1.108E+03**	
	Median	1.168E+03	**1.142E+03**	
	SW	5.798E-02	**4.203E-08**	3.002E-01
f16	Mean	2.229E+05	**2.179E+05**	
	Median	2.279E+05	**2.277E+05**	
	SW	8.096E-02	**5.181E-08**	5.449E-01
f17	Mean	**1.184E+17**	1.231E+17	
	Median	**1.240E+17**	1.285E+17	
	SW	**2.565E-02**	**1.798E-07**	9.263E-02
f18	Mean	**6.460E-01**	7.105E-01	
	Median	**6.820E-01**	6.979E-01	
	SW	**4.352E-09**	**3.207E-07**	**1.603E-02**

Table 5.12: Fitness values after 500000 function evaluations for functions f13-f18 with $n = 100$

5.2.2 Discussion

Tables 5.5 and 5.6 show that DE is superior in functions $f2$ and $f4$, which are unimodal functions with exponential growth and narrow valleys respectively. The different between the two algorithms in the remaining unimodal functions is not statistically significant. DE/GSA-CG has a slight edge

that is statistically significant in functions $f5$, $f10$, $f11$, $f12$, $f15$ and $f18$.

In the experiments with $n = 50$, which data is included in tables 5.7, 5.8, and 5.9, DE/GSA-CG was better in functions $f3$, $f6$, $f7$, $f8$, $f10$, $f11$, and $f18$, these were the functions which exhibited statistical significance. In the remaining functions neither algorithm achieved a significant difference. In the experiments with $n = 100$, which data is included in tables 5.10, 5.11, and 5.12, DE/GSA-CG was better in functions $f2$, $f3$, $f6$, $f10$, $f11$, $f12$, $f14$, $f18$, while DE won in function $f1$.

In unimodal functions, DE/GSA-CG performance improved in comparison with DE when more dimensions were added to the problems. In multimodal functions was not repeated the same trend, but the number of functions in which DE/GSA-CG was better remained constant. From these studies it can be concluded that DE/GSA-CG is clearly superior to DE in multimodal problems and also scales better for the solution of unimodal problems. This superiority of the proposal is expected, as DE/GSA-CG enhanced the ability of its evolutionary part to exploit promising regions of the search space, even if the generated directions are not as accurate as conjugate directions constructed with exact gradients.

5.3 Comparison between the developed approach, DE/GSA and DECLS

5.3.1 Results

This comparison involves the approach developed (DE/GSA-CG) and two more algorithms. For reasons of space, the results of each experiment are included in separated tables. Table 5.13 indexes the content of the tables included. In these tables, the rows contained statistical properties of the data: mean and median, and the p-value obtained with a Shapiro-Wilks test. The columns in these table are devoted to the three algorithms compared and the p-value of the Friedman test performed. In bold font are the p-values that satisfy the level of confidence to be statistically significant. Also

in bold are the fitness values generated by the best algorithm in the comparisons performed for each function.

n	Function evaluations	Functions	Table
20	100000	f1-f2	5.14
20	100000	f3-f10	5.15
20	100000	f11-f18	5.16
50	250000	f1-f5	5.17
50	250000	f6-f13	5.18
50	250000	f14-f18	5.19
100	500000	f1-f8	5.20
100	500000	f9-16	5.21
100	500000	f17-18	5.22

Table 5.13: Index of tables with the generated results

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f1	Mean	**4.288E+05**	4.776E+05	6.549E+05	
	Median	4.856E+05	**4.756E+05**	6.432E+05	
	SW	**3.323E-02**	6.391E-01	3.323E-02	**3.492E-04**
f2	Mean	3.819E+10	**2.816E+10**	2.925E+10	
	Median	2.629E+10	**1.696E+10**	2.586E+10	
	SW	**4.410E-02**	**5.275E-07**	4.410E-02	4.317E-01

Table 5.14: Fitness values after 100000 function evaluations for functions f1-f2 with $n = 20$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f3	Mean	1.120E+04	**1.050E+04**	1.098E+04	
	Median	1.108E+04	**1.052E+04**	1.113E+04	
	SW	**2.010E-03**	9.190E-01	**7.543E-03**	4.317E-01
f4	Mean	4.732E+04	4.666E+04	**4.158E+04**	
	Median	3.912E+04	3.968E+04	**3.482E+04**	
	SW	**1.044E-03**	**3.676E-02**	**4.393E-02**	7.558E-01
f5	Mean	1.253E+02	**1.246E+02**	1.297E+02	
	Median	**1.250E+02**	1.294E+02	1.296E+02	
	SW	1.136E-01	**7.654E-03**	1.136E-01	2.369E-01
f6	Mean	**8.278E+00**	8.282E+00	8.303E+00	
	Median	8.327E+00	**8.289E+00**	8.310E+00	
	SW	5.405E-02	1.475E-01	5.405E-02	5.945E-01
f7	Mean	4.926E+01	**4.841E+01**	5.141E+01	
	Median	5.211E+01	**4.457E+01**	5.085E+01	
	SW	1.824E-01	8.386E-02	1.824E-01	8.869E-01
f8	Mean	**1.165E+03**	1.229E+03	1.259E+03	
	Median	**1.226E+03**	1.253E+03	1.364E+03	
	SW	**1.705E-05**	**3.648E-02**	**6.150E-04**	1.249E-01
f9	Mean	**3.284E+03**	3.300E+03	5.202E+03	
	Median	3.251E+03	**3.139E+03**	5.002E+03	
	SW	1.062E-01	3.913E-01	1.062E-01	**1.960E-05**
f10	Mean	**3.333E+00**	3.597E+00	5.712E+00	
	Median	**3.354E+00**	3.486E+00	5.349E+00	
	SW	6.964E-01	7.318E-01	6.964E-01	**5.903E-06**

Table 5.15: Fitness values after 100000 function evaluations for functions f3-f10 with $n = 20$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f11	Mean	**2.060E+01**	2.060E+01	2.063E+01	
	Median	**2.060E+01**	2.061E+01	2.066E+01	
	SW	**1.207E-03**	**1.555E-03**	**2.244E-02**	1.249E-01
f12	Mean	**2.079E+01**	2.123E+01	2.185E+01	
	Median	**2.100E+01**	2.129E+01	2.196E+01	
	SW	5.376E-01	1.005E-01	5.376E-01	**6.220E-03**
f13	Mean	6.013E-01	**5.839E-01**	5.987E-01	
	Median	5.947E-01	**5.920E-01**	6.090E-01	
	SW	1.882E-01	2.657E-01	**1.662E-03**	4.317E-01
f14	Mean	**1.637E+00**	1.651E+00	1.726E+00	
	Median	1.658E+00	**1.629E+00**	1.710E+00	
	SW	**1.010E-02**	6.805E-01	1.010E-02	6.977E-01
f15	Mean	**5.326E-01**	5.789E-01	6.151E-01	
	Median	**5.359E-01**	5.918E-01	6.304E-01	
	SW	6.469E-01	2.242E-01	6.469E-01	**5.742E-03**
f16	Mean	**3.661E-01**	3.823E-01	4.247E-01	
	Median	**3.659E-01**	3.768E-01	4.098E-01	
	SW	8.984E-01	6.887E-01	8.984E-01	8.716E-02
f17	Mean	1.571E+07	**9.691E+06**	2.227E+07	
	Median	1.349E+07	**7.265E+06**	1.867E+07	
	SW	**2.030E-03**	6.139E-02	**3.629E-02**	**1.624E-02**
f18	Mean	**6.053E-01**	6.647E-01	6.879E-01	
	Median	**5.994E-01**	7.428E-01	6.945E-01	
	SW	9.698E-01	**4.723E-02**	9.698E-01	5.945E-01

Table 5.16: Fitness values after 100000 function evaluations for functions f11-f18 with $n = 20$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f1	Mean	3.803E+10	**3.768E+10**	3.809E+10	
	Median	**3.827E+10**	3.908E+10	3.831E+10	
	SW	1.287E-01	**1.514E-07**	1.287E-01	9.608E-01
f2	Mean	5.908E+67	5.160E+67	**5.051E+67**	
	Median	**7.789E+66**	2.844E+67	1.950E+67	
	SW	1.000E+00	1.000E+00	1.000E+00	2.101E-01
f3	Mean	**1.143E+05**	1.243E+05	1.261E+05	
	Median	**1.172E+05**	1.250E+05	1.295E+05	
	SW	**7.171E-08**	9.917E-01	**4.740E-02**	**4.979E-02**
f4	Mean	**1.627E+10**	2.149E+10	2.044E+10	
	Median	**1.965E+10**	2.115E+10	2.035E+10	
	SW	**3.692E-04**	1.882E-01	3.692E-04	5.273E-01
f5	Mean	4.492E+04	**4.110E+04**	4.347E+04	
	Median	4.500E+04	4.381E+04	**4.317E+04**	
	SW	2.996E-01	**2.592E-06**	2.996E-01	3.679E-01

Table 5.17: Fitness values after 250000 function evaluations for functions f1-f5 with $n = 50$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f6	Mean	**2.287E+01**	2.291E+01	2.300E+01	
	Median	2.291E+01	**2.291E+01**	2.300E+01	
	SW	**1.949E-02**	4.216E-01	**3.055E-03**	1.466E-01
f7	Mean	**1.934E+04**	2.140E+04	2.080E+04	
	Median	**1.987E+04**	2.143E+04	2.166E+04	
	SW	**1.471E-06**	6.839E-01	**7.635E-04**	8.046E-02
f8	Mean	**7.130E+03**	7.170E+03	7.257E+03	
	Median	7.235E+03	**7.180E+03**	7.312E+03	
	SW	**2.944E-02**	7.252E-01	2.944E-02	2.894E-01
f9	Mean	4.092E+08	**3.626E+08**	4.316E+08	
	Median	4.161E+08	**3.848E+08**	4.442E+08	
	SW	6.014E-01	**2.471E-06**	**3.032E-02**	7.730E-02
f10	Mean	**1.129E+05**	1.155E+05	1.359E+05	
	Median	**1.101E+05**	1.144E+05	1.396E+05	
	SW	6.923E-01	3.673E-01	6.923E-01	**1.159E-03**
f11	Mean	**2.104E+01**	2.105E+01	2.106E+01	
	Median	**2.105E+01**	2.106E+01	2.108E+01	
	SW	2.492E-01	3.084E-01	**3.010E-02**	1.023E-01
f12	Mean	**6.901E+01**	6.938E+01	7.048E+01	
	Median	**6.947E+01**	6.951E+01	7.053E+01	
	SW	**2.090E-02**	2.111E-01	2.090E-02	8.716E-02
f13	Mean	1.176E+01	**1.148E+01**	1.152E+01	
	Median	1.190E+01	**1.137E+01**	1.198E+01	
	SW	1.122E-01	**2.683E-02**	**6.979E-08**	1.409E-01

Table 5.18: Fitness values after 250000 function evaluations for functions f6-f13 with $n = 50$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f14	Mean	**3.074E+00**	3.132E+00	3.291E+00	
	Median	**3.109E+00**	3.179E+00	3.264E+00	
	SW	3.346E-01	7.161E-01	3.346E-01	8.716E-02
f15	Mean	4.353E+02	**4.203E+02**	4.298E+02	
	Median	4.369E+02	4.357E+02	**4.308E+02**	
	SW	7.440E-01	**7.173E-07**	7.440E-01	8.869E-01
f16	Mean	4.228E+04	**4.076E+04**	4.278E+04	
	Median	4.221E+04	**4.141E+04**	4.323E+04	
	SW	5.604E-01	2.458E-01	5.604E-01	2.894E-01
f17	Mean	3.186E+16	3.319E+16	**2.780E+16**	
	Median	3.194E+16	3.336E+16	**2.772E+16**	
	SW	4.541E-01	5.783E-01	4.541E-01	1.409E-01
f18	Mean	**9.329E-01**	9.792E-01	9.687E-01	
	Median	**9.551E-01**	9.792E-01	1.007E+00	
	SW	1.014E-01	6.829E-01	**3.156E-02**	4.317E-01

Table 5.19: Fitness values after 250000 function evaluations for functions f14-f18 with $n = 50$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f1	Mean	2.233E+11	2.260E+11	**2.184E+11**	
	Median	**2.214E+11**	2.271E+11	2.218E+11	
	SW	**3.546E-02**	4.915E-01	3.546E-02	2.101E-01
f2	Mean	2.379E+161	**9.480E+160**	8.340E+161	
	Median	**1.514E+159**	3.980E+159	1.140E+161	
	SW	1.000E+00	1.000E+00	1.000E+00	**1.873E-03**
f3	Mean	**2.522E+05**	2.642E+05	2.611E+05	
	Median	**2.624E+05**	2.647E+05	2.742E+05	
	SW	**7.793E-09**	9.518E-01	**6.512E-08**	3.263E-01
f4	Mean	1.182E+11	1.189E+11	**1.173E+11**	
	Median	1.195E+11	1.200E+11	**1.174E+11**	
	SW	9.535E-02	**3.526E-02**	9.535E-02	2.894E-01
f5	Mean	**2.240E+05**	2.302E+05	2.305E+05	
	Median	**2.259E+05**	2.314E+05	2.332E+05	
	SW	**3.177E-02**	4.301E-01	3.177E-02	2.276E-01
f6	Mean	**4.733E+01**	4.735E+01	4.737E+01	
	Median	**4.736E+01**	4.742E+01	4.741E+01	
	SW	2.917E-01	**3.480E-02**	2.917E-01	4.677E-01
f7	Mean	7.290E+04	7.168E+04	**6.999E+04**	
	Median	7.312E+04	7.379E+04	**7.054E+04**	
	SW	1.800E-01	**9.254E-04**	1.800E-01	5.273E-01
f8	Mean	**1.848E+04**	1.859E+04	1.859E+04	
	Median	**1.860E+04**	1.876E+04	1.862E+04	
	SW	7.496E-01	**1.974E-02**	7.496E-01	8.521E-01

Table 5.20: Fitness values after 500000 function evaluations for functions f1-f8 with $n = 100$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f9	Mean	4.573E+09	**4.497E+09**	4.570E+09	
	Median	4.577E+09	**4.511E+09**	4.695E+09	
	SW	6.007E-01	5.170E-01	6.007E-01	7.558E-01
f10	Mean	2.988E+05	**2.934E+05**	3.140E+05	
	Median	3.023E+05	**2.890E+05**	3.162E+05	
	SW	7.228E-01	3.447E-01	7.228E-01	**3.474E-02**
f11	Mean	**2.039E+01**	2.126E+01	2.127E+01	
	Median	**2.125E+01**	2.126E+01	2.128E+01	
	SW	**1.335E-10**	2.121E-01	1.335E-10	**1.832E-02**
f12	Mean	**1.546E+02**	1.553E+02	1.572E+02	
	Median	**1.548E+02**	1.557E+02	1.572E+02	
	SW	7.285E-02	3.409E-01	7.285E-02	**9.279E-03**
f13	Mean	5.805E+01	5.753E+01	**5.549E+01**	
	Median	**5.791E+01**	5.843E+01	5.835E+01	
	SW	9.605E-01	6.036E-02	**7.055E-09**	8.521E-01
f14	Mean	3.717E+00	**3.707E+00**	3.962E+00	
	Median	**3.729E+00**	3.788E+00	3.946E+00	
	SW	**8.146E-03**	2.105E-01	8.146E-03	**1.950E-03**
f15	Mean	1.144E+03	1.164E+03	**1.126E+03**	
	Median	1.168E+03	1.167E+03	**1.149E+03**	
	SW	5.798E-02	**2.936E-04**	**1.443E-03**	4.317E-01
f16	Mean	**2.229E+05**	2.236E+05	2.233E+05	
	Median	2.279E+05	2.293E+05	**2.245E+05**	
	SW	8.096E-02	**3.101E-03**	**5.586E-04**	8.869E-01

Table 5.21: Fitness values after 500000 function evaluations for functions f9-f16 with $n = 100$

Function		DE/GSA-CG	DE/GSA	DECLS	Friedman
f17	Mean	**1.184E+17**	1.201E+17	1.194E+17	
	Median	1.240E+17	**1.199E+17**	1.221E+17	
	SW	**2.565E-02**	5.718E-01	2.565E-02	8.869E-01
f18	Mean	**6.460E-01**	6.645E-01	6.561E-01	
	Median	6.820E-01	6.755E-01	**6.659E-01**	
	SW	**4.352E-09**	9.791E-02	**6.992E-03**	4.317E-01

Table 5.22: Fitness values after 500000 function evaluations for functions f17-f18 with $n = 100$

5.3.2 Discussion

For $n = 20$, the statistics data show that both the proposed approach and DE/GSA were superior to DECLS in functions $f1$, $f9$, $f10$, $f12$, $f15$, and $f17$. The same happened in the case for $n = 50$ in functions $f3$, and $f10$. in the experiments that used 100 variables, the statistically significant difference in favor of DE/GSA and DE/GSA-CG expanded to include $f2$, $f10$, $f11$, $f12$, $f14$. It can be seen from these results that the best performing function for DE/GSA and DE/GSA-CG was $f10$, which was included in the three lists and corresponds to the Ackley function. This function is a problem known to cause issues to hill climbers, Ackley´s function has many local optima. The Griewank´s function also appeared to show statistical difference in the cases where $n = 20$ and $n = 100$. The results presented suggest that both DE/GSA and DE/GSA-CG were superior to DECLS, which could be attributed to the power of exploitation possessed by GSA and the balance achieved by the search strategies applied in these two algorithms.

For the comparison that involves DE/GSA a Wilcoxon test was performed, in this test the proposed approach did not achieve a statistically significant superiority over DE/GSA. These two algorithms were very near in every problem, which can be attributed to the exit criterion used in the local search operator, which combined with inaccuracies in the estimation of the gradients led to not

very spotted approximations of conjugate directions. This means that there were not used enough conjugate directions to have a statistical difference because overall these directions were not better than the gradients estimated.

6

Conclusions and Future Work

A local search operator based on conjugate gradient is proposed in this thesis work, this operator was integrated in a MA, which is based on DE.

One of the most frequently described hardships in the design of MAs is the balance between exploitation and exploration. Excessive exploitation results in premature convergence while the lack of exploratory power leads to not finding good enough solutions.

The strategy used to achieve balance was to prefer the best individual for the application of the local search operator, but reject this individual if after applying the operator the solutions produced do not improve. In this case the individual to be refined is selected randomly from the best tercile of the population. By selecting the best individual, the algorithm exploits promising regions of the search space. The application of the operator to individuals different from the best one, but in best tercile kept the focus in the promising regions. The operator worked as expected but its application was limited to one application per generation to keep computational time low.

The developed local search operator was tested using DE but it is possible to use it with other

Evolutionary Algorithm. The results generated in the experimentation showed that the developed approach produces better solutions than DE and DECLS and it is only competitive with the algorithm that only uses gradient directions, this is explainable by considering that the gradient conjugate method is sensible to errors in the estimated gradients which accumulate in the process of constructing successive conjugate directions. The strategy used was to set a threeshold of improvement to limit the number of conjugate directions spent in each application of the local search operator.

One of the insights gained while working at this thesis, is that there are parameters that are not as important as others, for example, an amount of time was spent while trying to find a good initial length for the quadratic interpolations. Experimentation to determine how many conjugate directions was optimum was also performed, in this case using n proved to be inefficient, given the approximated nature of the gradients used the error in the conjugate directions constructed was clearly going to accumulate, and this inevitable imprecision limits the exploitation performed by the local search operator.

Further improvements that could be made to this work, is incorporating Cooperative Evolution strategies to handle a higher number of variables. Also QR decomposition could be used to generate better points for the gradient estimator which might result in steeper descent directions, this comes with a trade off, additional points require spending more functions evaluations and the computing time required for the QR decomposition per se. It is possible to incorporate the proposed operator in other Evolutionary Algorithms, this could be one more path that future work could abordate.

A

Appendix

A.1 Benchmark functions

These problems should be treated as black-box problems, the list below includes unimodal and multimodal functions.

Bent Cigar Function

$$f_1(\mathbf{x}) = \mathbf{x}_i + 10^6 \sum_{n=1}^{n} \mathbf{x}_i^2 \tag{A.1}$$

Sum of Different Power Function

$$f_2(\mathbf{x}) = \sum_{n=1}^{n} |\mathbf{x}_i|^{i+1} \tag{A.2}$$

Zakharov function

$$f_3(\mathbf{x}) = \sum_{n=1}^{n} \mathbf{x}_i^2 + (\sum_{n=1}^{n} \frac{\mathbf{x}_i}{2})^2 + (\sum_{n=1}^{n} \frac{\mathbf{x}_i}{2})^4 \tag{A.3}$$

Rosenbrock´s function

$$f_4(\mathbf{x}) = \sum_{n=1}^{n} (100(\mathbf{x}_i^2 - \mathbf{x}_{i+1})^2 + (\mathbf{x}_i - 1)^2) \tag{A.4}$$

Rastrigin´s function

$$f_5(\mathbf{x}) = \sum_{n=1}^{n} (\mathbf{x}_i^2 - 10cos(2\pi\mathbf{x}_i) + 10) \tag{A.5}$$

Expanded Schaffer´s F6 function

$$f_6(\mathbf{x}) = g(\mathbf{x}_1, \mathbf{x}_2) + g(\mathbf{x}_2, \mathbf{x}_3) + \ldots + g(\mathbf{x}_{n-1}, \mathbf{x}_n) + g(\mathbf{x}_n, \mathbf{x}_1) \tag{A.6}$$

$$g(x, y) = 0.5 + \frac{(sin^2(\sqrt{x^2 + y^2}) - 0.5)}{(1 + 0.001(x^2 + y^2)} \tag{A.7}$$

Levy function

$$f_7(\mathbf{x}) = sin^2(\pi w_1) + \sum_{n=1}^{n-1} (w_i - 1)^2[1 + 10sin^2(\pi w_i + 1)] + (w_n - 1)^2[1 + sin^2(2\pi w_n)] \tag{A.8}$$

Modified Schwefel's Function

$$f_8(\mathbf{x}) = 418.9829n - \sum_{n=1}^{n} g(z_i) \tag{A.9}$$

$z_i = \mathbf{x}_i + 4.209687462275036e + 002$

$$g(z_i) = \begin{cases} z_i sin(|z_i|^{\frac{1}{2}}) & \text{if } |z_i| \leq 500 \\ (500 - mod(z_i, 500))sin(\sqrt{|500 - mod(z_i, 500|)}) - \frac{(z_i - 500)^2}{10000n} & \text{if } |z_i| > 500 \\ (mod(z_i, 500) - 500)sin(\sqrt{|mod(z_i, 500) - 500|}) - \frac{(z_i - 500)^2}{10000n} & \text{if } |z_i| > 500 \end{cases}$$

High Conditioned Elliptic Function

$$f_9(\mathbf{x}) = \sum_{n=1}^{n} (10^6)^{\frac{i-1}{n-1}} \mathbf{x}_i^2 \tag{A.10}$$

Discus Function

$$f_{10}(\mathbf{x}) = 10^5 \mathbf{x}_i^2 + \sum_{n=1}^{n} \mathbf{x}_i^2 \tag{A.11}$$

Ackley´s Function

$$f_{11}(\mathbf{x}) = -20exp(-0.2\sqrt{\frac{1}{n}\sum_{n=1}^{n} \mathbf{x}_i^2}) - exp(\frac{1}{n}\sum_{n=1}^{n} cos(2\pi \mathbf{x}_i)) + 20 + e \tag{A.12}$$

Weierstrass Function

$$f_{12}(\mathbf{x}) = \sum_{n=1}^{n} (\sum_{n=1}^{kmax} [a^k cos(2\pi b^k (\mathbf{x}_i + 0.5))]) - n \sum_{n=1}^{kmax} [a^k cos(\pi b^k)] \tag{A.13}$$

$a = 0.5, b = 3, kmax = 20$

Griewank´s Function

$$f_{13}(\mathbf{x}) = \sum_{n=1}^{n} \frac{\mathbf{x}_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{\mathbf{x}_i}{\sqrt{i}}) + 1 \tag{A.14}$$

Katsuura Function

$$f_{14}(\mathbf{x}) = \frac{10}{n^2} prod_{i=1}^{n} (1 + i \sum_{n=1}^{32} \frac{2^j x_i - round(2^j \mathbf{x}_i)}{2^j})^{\frac{10}{D^{1.2}}} - \frac{10}{n^2} \tag{A.15}$$

HappyCat Function

$$f_{15}(\mathbf{x}) = |\sum_{n=1}^{n} \mathbf{x}_i^2 - n|^{\frac{1}{4}} + (0.5\sum_{n=1}^{n} \mathbf{x}_i^2 + \sum_{n=1}^{n} \mathbf{x}_i)/n + 0.5 \tag{A.16}$$

HGBat Function

$$f_{16}(\mathbf{x}) = |(\sum_{n=1}^{n} \mathbf{x}_i^2)^2 - \sum_{n=1}^{n} \mathbf{x}_i)^2|^{\frac{1}{2}} + (0.5\sum_{n=1}^{n} \mathbf{x}_i^2 + \sum_{n=1}^{n} \mathbf{x}_i)/n + 0.5 \tag{A.17}$$

Expanded Griewank´s plus Rosenbrock´s Function

$$f_{17}(\mathbf{x}) = f_{13}(f_4(\mathbf{x}_1, \mathbf{x}_2)) + f_{13}(f_4(\mathbf{x}_2, \mathbf{x}_3)) + \ldots + f_{13}(f_4(\mathbf{x}_{n-1}, \mathbf{x}_n)) + f_{13}(f_4(\mathbf{x}_n, \mathbf{x}_1)) \tag{A.18}$$

Schaffer´s F7 Function

$$f_{18}(\mathbf{x}) = [\frac{1}{n-1}\sum_{n=1}^{n-1}(\sqrt{s_i}(sin(50s_i^{0.2}) + 1]^2 \tag{A.19}$$

$s_i = \sqrt{\mathbf{x}_i^2 + \mathbf{x}_{i+1}^2}$

Bibliography

[1] L. Jourdan, M. Basseur, and E.-G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620 – 629, 2009.

[2] Serkan Kiranyaz. *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. 08 2013.

[3] Felix Feather. The calculus story: A mathematical adventure, by david acheson. *BSHM Bulletin: Journal of the British Society for the History of Mathematics*, 0(0):1–2, 2018.

[4] S. Butenko and P.M. Pardalos. *Numerical Methods and Optimization: An Introduction*. Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series. CRC Press, 2014.

[5] JF Bonnans, Jean Charles Gilbert, C Lemarachal, and Claudia Sagastizabal. *Numerical Optimization – Theoretical and Practical Aspects*. 01 2006.

[6] Singiresu S. Rao. *Engineering Optimization Theory and Practice*. John Wiley Sons, Inc, 2009.

[7] Carlos Cotta et al. Ferrante Neri. *Handbook of Memetic Algorithms*. Springer-Verlag Berlin Heidelber, 2012.

[8] Wojciech Rafajłowicz. A hybrid differential evolution-gradient optimization method. In Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing*, pages 379–388, Cham, 2015. Springer International Publishing.

[9] J. C. Allwright. Conjugate gradient versus steepest descent. *Journal of Optimization Theory and Applications*, 20(1):129–134, 1976.

[10] Merriam-Webster. Optimization, 2018.

[11] Marius Durea, Radu Strugariu, A Nowacka-Leverton, Vicentiu Radulescu, and N Rogers. *An introduction to nonlinear optimization theory*. 01 2014.

[12] Franz Rothlauf. *Design of Modern Heuristics: Principles and Application*, volume 25. 01 2011.

[13] Rupika Delgoda and James Douglas Pulfer. A guided monte carlo search algorithm for global optimization of multidimensional functions. *Journal of Chemical Information and Computer Sciences*, 38(6):1087–1095, 1998.

[14] P. Siarry. *Metaheuristics*. Springer International Publishing, 2016.

[15] Iztok Fister jr, Xin-She Yang, Iztok Fister, Janez Brest, and DuÅ¡an Fister. A brief review of nature-inspired algorithms for optimization. 80, 07 2013.

[16] Ilhem BoussaÃ¯d, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82 – 117, 2013. Prediction, Control and Diagnosis using Advanced Neural Computations.

[17] Thomas Back, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, UK, 1st edition, 1997.

[18] David B Fogel; IEEE Neural Networks Council. *Evolutionary computation : toward a new philosophy of machine intelligence*. IEEE series on computational intelligence. Wiley, 3rd ed edition, 2006.

[19] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[20] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1991.

[21] J. Vesterstrom and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 2, pages 1980–1987 Vol.2, June 2004.

[22] William M. Spears, Kenneth A. De Jong, Thomas Bäck, David B. Fogel, and Hugo de Garis. An overview of evolutionary computation. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, pages 442–459, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[23] M. Srinivas and L. M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, June 1994.

[24] H. R. Cai, C. Y. Chung, and K. P. Wong. Application of differential evolution algorithm for transient stability constrained optimal power flow. *IEEE Transactions on Power Systems*, 23(2):719–728, May 2008.

[25] L. Lakshminarasimman and S. Subramanian. *Applications of Differential Evolution in Power System Optimization*, pages 257–273. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[26] Kai Yit Kok and Parvathy Rajendran. Differential-evolution control parameter optimization for unmanned aerial vehicle path planning. *PLOS ONE*, 11(3):1–12, 03 2016.

[27] Sandra M. Venske, Richard A. GonÃ§alves, Elaine M. Benelli, and Myriam R. Delgado. Ademo/d: An adaptive differential evolution for protein structure prediction problem. *Expert Systems with Applications*, 56:209 – 226, 2016.

[28] Anyong Qing. *A Literature Survey on Differential Evolution*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[29] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[30] K V. Price, R M. Storn, and J Lampinen. *Differential Evolution-A Practical Approach to Global Optimization*, volume 141. 01 2005.

[31] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, Feb 2011.

[32] Rainer Storn and Kenneth Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. 23, 01 1995.

[33] M. J. D. Powell. Restart procedures for the conjugate gradient method. *Mathematical Programming*, 12(1):241–254, Dec 1977.

[34] R. Pytlak. *Conjugate Gradient Algorithms in Nonconvex Optimization*. Nonconvex Optimization and Its Applications. Springer Berlin Heidelberg, 2008.

[35] Gonglin Yuan, Zengxin Wei, and Guoyin Li. A modified polak-ribiere-polyak conjugate gradient algorithm for non smooth convex programs. *Journal of Computational and Applied Mathematics*, 255:86 – 96, 2014.

[36] William W. Hager and Hongchao Zhang. A survey of nonlinear conjugate gradient. 2005.

[37] Der-Horng Lee, Yu Nie, and Anthony Chen. A conjugate gradient projection algorithm for the traffic assignment problem. *Mathematical and Computer Modelling*, 37(7):863 – 878, 2003.

[38] C K Edelman. Curvature in conjugate gradient eigenvalue computationwith applications to materials and chemistry. 2008.

[39] Gábor Takács, István Pilászy, and Domonkos Tikk. Applications of the conjugate gradient method for implicit feedback collaborative filtering. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 297–300, New York, NY, USA, 2011. ACM.

[40] Tapan K. Sarkar, Xiaopu Yang, and Ercument Arvas. A limited survey of various conjugate gradient methods for solving complex matrix equations arising in electromagnetic wave

interactions. *Wave Motion*, 10(6):527 – 546, 1988. Special Issue on Numerical Methods for Electromagnetic Wave Interactions.

[41] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997.

[42] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms, 1989.

[43] Muzaffar Eusuff, Kevin Lansey, and Fayzul Pasha. Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Engineering Optimization*, 38(2):129–154, 2006.

[44] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, Nov 2000.

[45] Vincent Kelner, Florin Capitanescu, Olivier LÃ©onard, and Louis Wehenkel. A hybrid optimization technique coupling an evolutionary and a local search algorithm. *Journal of Computational and Applied Mathematics*, 215(2):448 – 456, 2008. Proceedings of the Third International Conference on Advanced Computational Methods in Engineering (ACOMEN 2005).

[46] Xin Zhang and Shiu Yin Yuen. A directional mutation operator for differential evolution algorithms. *Applied Soft Computing*, 30:529 – 548, 2015.

[47] Antonio LaTorre, Santiago Muelas, and José-María Peña. A mos-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. *Soft Computing*, 15(11):2187–2199, Nov 2011.

[48] Saul Dominguez-Isidro and Efren Mezura-Montes. The baldwin effect on a memetic differential evolution for constrained numerical optimization problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 203–204, New York, NY, USA, 2017. ACM.

[49] Andrea Caponio, Anna V. Kononova, and Ferrante Neri. *Differential Evolution with Scale Factor Local Search for Large Scale Problems*, pages 297–323. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[50] Miguel Leon and Ning Xiong. Differential evolution enhanced with eager random search for solving real-parameter optimization problems. *International Journal of Advanced Research in Artificial Intelligence*, 4(12), 2015.

[51] Dongli Jia, Guoxin Zheng, and Khurram Khan. An effective memetic differential evolution algorithm based on chaotic local search. 181:3175–3187, 08 2011.

[52] Zhaolu Guo, Haixia Huang, Changshou Deng, Xuezhi Yue, and Zhijian Wu. An enhanced differential evolution with elite chaotic local search. In *Comp. Int. and Neurosc.*, 2015.

[53] Ferani E. Zulvia R.J. Kuo. The gradient evolution algorithm: A new metaheuristic. *Information Sciences*, 316:246–265, 2015.

[54] J. Yang, J. Wei, and J. Liu. Adaptive differential evolution algorithm based on gradient and polar coordinates search strategies. In *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pages 274–277, Dec 2015.

[55] Alvarado S. Segura C. et al. Schütze, O. Gradient subspace approximation: a direct search method for memetic computing. *Soft Computing*, 21:6331–6350, 2016.

[56] R. Martí, P. Panos, and M. Resende. *Handbook of Heuristics*. Handbook of Heuristics. Springer International Publishing, 2017.

[57] Guohua Wu, Rammohan Mallipeddi, and Ponnuthurai Suganthan. Problem definitions and evaluation criteria for the cec 2017 competition and special session on constrained single objective real-parameter optimization, 10 2016.

[58] J. Ronkkonen, S. Kukkonen, and K. V. Price. Real-parameter optimization with differential

evolution. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 506–513 Vol.1, Sept 2005.

[59] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523, June 1996.

[60] G.W. Corder and D.I. Foreman. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley, 2009.