FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Faculdade de Engenharia da Universidade do Porto

Masters in Mechanical Engineering

Development and implementation of memetic models for the optimal design of mechanical systems

Author:

Hugo Fernando de Sá Gonçalves (up201806233@fe.up.pt)

Advisor:

Prof. Carlos Conceição António

Co-Advisor:

Prof. Gonçalo das Neves Carneiro

October 4, 2023

Resumo

Algoritmos Meméticos são Algoritmos de Optimização que têm por base a ideia de meme de Richard Dawking. De facto, os algoritmos genéticos utilizam conceitos baseados na biologia e na genética, enquanto os algoritmos meméticos incluem a aprendizagem como uma componente importante da evolução, associando-a à evolução cultural.

Nos últimos anos a taxa de aumento da velocidade dos computadores tem vindo a diminuir, o que por sua vez está a levar a um maior foco em usar algoritmos mais rápidos aos invés de haver um foco unicamente na melhoria do hardware.

Frequentemente, a Optimização requer milhares de avaliações do modelo a analizar, cada uma dessas avaliações pode demorar alguns segundos mas também pode demorar dias. Computação Memética é uma forma de melhorar Algoritmos de Optimização através da diminuição do numéro de avaliações necessárias para se chegar a um óptimo global.

Nesta tese começa-se por fazer uma revisão de conceitos básicos de optimização e do estado-da-arte dos Algoritmos Meméticos. Depois uma revisão de algoritmos de optimização clássicos e de algoritmos inspirados na Natureza, com ênfase em Algorimos Genéticos. De seguida são explorados com maior profundidade os Algoritmos Meméticos. Finalmente, é desenvolvido um Algoritmo Memético que é aplicado a três problemas de optimização, dos quais se obtêm resultados que serão analisados.

Com esta tese pretende-se entender melhor os Algoritmos Meméticos e incentivar o seu uso.

Abstract

Memetic Algorithms are Optimization Algorithms based on Richard Dawkins idea of a meme. In fact, genetic algorithms use concepts based on biology and genetics, while memetic algorithms include learning as an important component of evolution, associating it with cultural evolution.

In the last years the rate of evolution of computers has been slowing down, no longer following Moore's Law. This is leading to a shift in the paradigm, no longer focused just on improving the hardware but also the performance of the algorithms.

Optimization often requires thousands of model evaluations, each one of them taking seconds to days to solve. Memetic Computing are one way to improve Optimization Algorithms by decreasing the number of model evaluations necessary to reach the global optimum.

The thesis starts with a review of basic concepts of optimization and the state-of-the-art of Memetic Algorithms. Then a review of some classical optimization algorithms general review of Bio-inspired Algorithms with an emphasis on Genetic Algorithms. This is followed by a deeper study of Memetic Algorithms. Finally is described the algorithm developed, it's applied to three optimization problems and the results are analyzed.

With this thesis it is expected to obtain a further understanding of Memetic Algorithms and spread their use. It is also expected to learn how to better solve optimization algorithms at hand.

Agradecimentos

Quero agradecer ao meu Orientador Prof. Carlos Conceição António que se disponibilizou desde cedo a criar uma proposta de tese na área da optimização e que sempre arranjou tempo para ler e corrigir tanto a tese como o programa em FORTRAN, a sua experiência é inestimável. Agradeço ao meu Co-Orientador Prof. Gonçalo das Neves Carneiro por ter ajudado na recolha inicial de artigos para revisão do estado-da-arte, espero que tenha sorte nas suas aventuras. Agradeço também ao Joâo Festas e à Carolina Fernandes por me terem dado dicas.

Quero também agradecer à minha família por me terem dado espaço e tempo para estudar e fazer a tese. Sei que muitas vezes abdiquei de tempo com eles.

Por fim quero agradecer aos meus amigos, tenho a certeza que sem eles não teria motivação para acabar o curso.

"What is better – To be born good, or to overcome your evil nature through great effort?"

- Paarthurnax, The Elder Scrolls V: Skyrim

Contents

List of Figures

List of Tables

Acronyms

ACO Ant Colony Optimization 21, 23

AMA Adaptive Memetic Algorithm 9, 40, 41, 50

BiA Bio-inspired Algorithms 3, 27

CM Controlled Mutation 58

DE Differential Evolution 21, 24

EA Evolutionary Algorithm 11, 21

FEM Finite Element Method 4, 5, 85

FSD Fully Stressed Design 4

GA Genetic Algorithm xv, xix, 1, 4, 5, 9, 21, 25, 27, 28, 32, 40, 42, 45, 53, 55, 69, 76, 88, 93, 95

HAMA Hyperheuristic Adaptive Memetic Algorithm 40

HGA Hybrid Genetic Algorithm 9

KKT Karush-Kuhn-Tucker Conditions 4

LS Local Search 10, 11, 42, 46, 49, 58, 71, 74, 80, 82, 90, 93, 96

MA Memetic Algorithm xix, 3, 9, 39–42, 45, 47, 48, 50, 51, 53, 96, 97

MMA Multimeme Memetic Algorithm 39

NFL No Free Lunch Theorem 16, 37, 47

PSO Particle Swarm Optimization 21

QP Quadratic Programming 18

SQP Sequential Quadratic Programming 18

TSP Traveling Salesman Problem 9, 23

List of Symbols

α	Step size.
$\boldsymbol{\lambda}$	Penalty coefficients, Lagrange multipliers(inequality constraints).
$\boldsymbol{\mu}$	Lagrange multipliers(equality constraints).
Φ	Penalty function.
Ω	Search space.
$\boldsymbol{d}$	Search direction.
d_H	Hamming distance.
$\boldsymbol{F}$	Vector of external forces
f	Objective function, Cost function.
$\boldsymbol{g}$	Inequality constraints.
$\boldsymbol{H}$	Hessian Matrix.
$\boldsymbol{h}$	Equality constraints.
$[\boldsymbol{K}]$	Stiffness Matrix
$\mathcal{L}$	Lagrangian
p_{mut}	Mutation group size relative to the population size.
p_{off}	Offspring size relative to the population size.
S_i	Genome of individual i.
$\boldsymbol{x}_i$	Phenotype of individual i.

Chapter 1

Introduction

In this work the author intents to study, develop and implement memetic models applied to the optimization of mechanical systems.

Initially the need for this kind of model will be discussed, followed by a revision of the start of the art of evolutionary algorithms and finally the development and application of such models to test functions and optimization problems related to mechanical engineering.

The author will try to build upon what was previously made to address some of the problems found in evolutionary algorithms that have been found over the years.

The programming language used in this work is FORTRAN.

Bio-inspired Algorithms, of which GAs and Particle Swarm stand-out the most, have been successfully applied to all kind of problems: constrained/unconstrained, continuous/discrete/mixed, combinatorial, engineering related, etc. Yet we yearn for more. We want a "faster, better, stronger" algorithm.

In 1989, Pablo Moscato proposed a new type of algorithm: the Memetic Algorithm. His algorithm was based on the idea of a meme proposed by Richard Dawkins 15 years earlier, in 1974, in his book The "Selfish Gene".

At this time GAs were very popular, with merit, as they were able to solve problems much faster than the algorithms previously available, so naturally the original MA took heavy inspiration from the GA.

Since then memetic algorithms concepts have been extended to other areas of problem solving, outside of optimization, creating a new subject of computing:"Memetic Computing".

1.1 Motivation

Anyone can build a bridge, but it takes an engineer to build one that barely stands -

Common saying

This saying summarizes one of engineering's objectives: to do the most with the least resources possible.

To build any mechanical component one could overdimension and make it out of the most expensive materials, but we don't, mostly due to cost but also because of environment issues, weight and time constraints. Instead is used a combination of materials that have a good weigth to load capacity ratio and are also cheap, mostly steel but also composite materials, and components are designed in such a way that they have just enough material to carry the expected loads, with an associated safety coefficient.

This is why Optimization Algorithms are needed, to *build bridges* with the minimum cost possible.

In 1997 Wolpert and Macready proved that any two optimization algorithms had the same average performance across the set of all optimization problems[1]. They called it the **No free lunch theorem**.

This might come as a shock, indeed, one might even ask himself: *Then why should I bother carefully choose an optimization algorithm?*.

What this theorem means is that an algorithm that performs well in a particular **class of problems** does so at the expense of performance in another class of problems.

The class of a problem refers to it's features[2]:

1. the shape of the fitness landscape and it's properties;
2. the presence of multiple local optima, i.e, unimodality vs multimodality;
3. the possibility of separating the problem into subproblems;
4. presence of noise, randomness in the objective function;
5. the dependence of time, i.e, the function is dynamic;
6. shape of the search domain, the presence of discontinuities and connectivity between different search regions;

1.2 Objectives and Organization

The scope of this work includes a further understanding of MAs, research recent developments in such algorithms an then implement some of those techniques to common Mechanical Engineering optimization problems.

To do so this thesis will start by reviewing basic optimization concepts. Then a state of the art review will be done to understand what has been done recently in MAs and how to implement it.

Then Non-linear Programming methods, Bio-inspired Algorithmss (BiAs) and MAs will be presented justifying the emergence of one group after another.

Finally MA will be implemented in test problems and common engineering optimization problems, followed by a discussion of the results.

1.3 Basic principles of optimal design

1.3.1 Three columns

Usually, an optimization problem can be addressed by the *Three-Column-Concept*[3, 4, 5, 6, 7].

1^{st} column - Model of the system

The first column of optimization concerns the making of an analysis model that will describe the physical phenomena or technological process to be optimized. To do so the equations that govern the system to be studied must be established.

In mechanical systems this equations establish relations between state variables such as stresses, strains and displacements and the external loads.

The analysis model evaluates each project solution, to do this we must establish a direct or indirect relation between the input and output variables.

Usually the objective of an optimization model is formulated in terms of a scalar function, but can also be a vector in the case of multiobjective optimization. The evaluation can be done in terms of the state variables or other kinds of variables than influence the project, like weight. Very often the evaluation of the objective function doesn't require the evaluation of the entire model, what requires the evaluation of the entire model is the evaluation of the constraint violations. The constraints are formulated in a vector of functions.

The Finite Element Method (FEM) is often used in Mechanical Engineering to model structures, components and other phenomena. In order to automatize the optimization process the object to be analyzed must be well defined, either mathematically or numerically, by the analysis model. Furthermore the project parameters should be defined independently of the mesh size and the transformation between project parameters and analysis variables should be automatic. This way the mesh can be generated automatically based on project parameters and some mesh quality control parameters

2^{nd} column - Optimization Algorithm

Real-world optimization problems are typically associated with non-linear objective functions with constraints. Optimization algorithms solve those problems. They start with an initial solution (or set of solutions) and apply a algorithm until the optimum (or a good enough solution) is found.

Optimization algorithms can be divided in three groups: Mathematical Programming methods, algorithms based on Optimality Criteria and Bio-inspired Algorithms.

Mathematical Programming are extensively used is non-linear optimization problems. They often use gradients to compute search directions and step sizes, that are then applied to the solutions to iteratively search for the optimum.

Algorithms based on Optimality Criteria use optimality conditions that ensure a minimum has been achieved, commonly after an iterative procedure, such as verifying Karush-Kuhn-Tucker Conditions (KKT) or other problem specific such as Fully Stressed Design (FSD).

For example, when sizing a shaft subject to torsion we know instinctively that the optimum solution is the one that verifies the shear maximum stress constraint exactly.

Bio-inspired algorithms are a group of methods that had an increase in popularity in latter years. They are based in evolution and group behaviour in Nature and don't require gradients. From this methods the GA is the one that stands out the most.

The performance of the method chosen depends on the problem.

3^{rd} column - Optimization model

The optimization model establishes the bridge between the mathematical model and the optimization algorithm. It describes the relation between the state variables and the design variables.

The state variables are the relevant quantities that change during the optimization process chosen from the parameters of the model being analysed.

The behaviour of the system is explicitly or implicitly defined with respect to the design variables.

Often the outputs need to processed so the objective function(s) and the constraints can be evaluated, this is called post processing.

The design variables may be mapped into transformation variables to adapt the optimization model to specific requirements of the optimization algorithm. For example, the genetic algorithm uses binary encoding very often so we need transform real variables to bit-strings and vice-versa.

The parameterization concerns the design and analysis models and the transformation between design variables and analysis variables.

The FEM is regularly used in structural optimization. Solving the system of linear equations yields the displacements in most formulations. From these the forces can be obtained, stresses and strains that are then used in to evaluate the constraints. If a GA were to be used to optimize a truss structure being modeled by FEM:

- the analysis variables would be the lengths, cross section areas, the inclinations of the members and the node displacements;
- the design variables would be lengths and cross sections of the members
- the state variables would be the stresses and displacements
- the transformation variables would be the bit-strings that encode the length and the cross section of each member

1.3.2 Formulation

The general formulation of an optimization problem is given by:

$$\underset{x\in\Omega^n}{\text{minimize}}(f(x)), \tag{1.3.1}$$

subject to:

$$\begin{cases} h_i(x) = 0, \text{if } i \in E \\ g_j(x) \geq 0, \text{if } j \in I \end{cases} \tag{1.3.2}$$

where $f(x)$ is the objective function, $h_i(x) = 0$ are the equality constraints, $g_i(x) \geq 0$ are the inequality constraints and E,I are the respective set of constraint indexes.

1.3.3 Necessary and Sufficient Conditions for Optimality

In the formulation of the optimization problem we search for a global optimum in the domain $\boldsymbol{X}$, but in practice this is often not the case because we can't ensure that a minimum is global. Instead the designer should be satisfied by a good enough local optimum.

1^{st} order necessary conditions

$\boldsymbol{x}^*$ is a global minimum if:

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \text{ for } \forall\, \boldsymbol{x} \in \boldsymbol{X}. \tag{1.3.3}$$

$\boldsymbol{x}^*$ is a local minimum if:

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \text{ for } \forall\, \boldsymbol{x} \in \boldsymbol{X} \cap \boldsymbol{U}_{\varepsilon}(\boldsymbol{x}^*). \tag{1.3.4}$$

where $\boldsymbol{U}_{\varepsilon}(\boldsymbol{x}^*)$ is a neighbourhood of radius ε centered around $\boldsymbol{x}^*$.

In practice, it's only possible to ensure that a certain point is a global minimum for some functions with convexity properties that ensure that if a local minimum was found than it is also a global minimum.

A search direction $\boldsymbol{d}$ is said to be admissible if, for a given $\boldsymbol{x}$, $(\boldsymbol{x} + \alpha \cdot \boldsymbol{d}) \in \boldsymbol{X}$, $\forall\, \alpha \in [0, \bar{\alpha}]$, in other words a search direction is said to be admissible if, when starting in inside the search domain, all points in that direction up to a finite distance fall into the search domain.

Theorem 1.1 (1^{st} order necessary conditions)**.** *Let $\boldsymbol{X}$ be a subset of $\mathbb{R}^n$ and $f \in C^1$ [1] be a function defined in $\boldsymbol{X}$. If $\boldsymbol{x}^*$ is local minimum of f in $\boldsymbol{X}$, then $\nabla f(\boldsymbol{x}^*) \cdot \boldsymbol{d} \geq 0$ for any $\boldsymbol{d} \in \mathbb{R}^n$ that is an admissible direction in $\boldsymbol{x}^*$.*

A particular case of this theorem is when $\boldsymbol{x}$ falls inside $\boldsymbol{X}$, in that case the following corollary is obtained:

Corolary. *Let $\boldsymbol{X}$ be a subset of $\mathbb{R}^n$ and $f \in \mathbb{R}$ a function defined in $\boldsymbol{X}$. If $\boldsymbol{x}^*$ is a local minimum of f in $\boldsymbol{X}$ and $\boldsymbol{x}^*$ is an interior point of $\boldsymbol{X}$, then $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$.*[2]

2^{nd} order necessary conditions

The previous condition is based on a first order approximation of the neighborhood of point $\boldsymbol{x}^*$. If instead a second order approximation is used a stronger condition is obtained.

Theorem 1.2 (2^{nd} order necessary conditions)**.** *Let $\boldsymbol{X}$ be a subset of $\mathbb{R}^n$ and $f \in C^2$ [3] a function defined in $\boldsymbol{X}$. If $\boldsymbol{x}^*$ is local minimum of f in $\boldsymbol{X}$, then, for any $\boldsymbol{d} \in \mathbb{R}^n$ that is an admissible direction in $\boldsymbol{x}^*$, we get:*

- $\nabla f(\boldsymbol{x}^*) \cdot \boldsymbol{d} \geq 0$
- *If $\nabla f(\boldsymbol{x}^*) \cdot \boldsymbol{d} = 0$, then $\boldsymbol{d}^T \boldsymbol{H}_f(\boldsymbol{x}^*) \boldsymbol{d} \geq 0$*

[1] f has first order continuous derivatives.

[2] Note that this is a necessary condition for a local minimum but isn't a sufficient condition, that is, a point can have null gradient(stationary point) and not be a local minimum. Such is the case of maximum and saddle points.

[3] f has second order continuous derivatives.

Theorem 1.3 (2^{nd} order necessary conditions without constraints)**.** *Let $\boldsymbol{x}^*$ be an interior point of the set $\boldsymbol{X}$ and assume $\boldsymbol{x}^*$ is a local minimum in $\boldsymbol{X}$ of $f \in C^2$, then:*

- $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$
- $\boldsymbol{d}^T \boldsymbol{H}_f(\boldsymbol{x}^*)\boldsymbol{d} \geq 0, \forall \boldsymbol{d}$

where $\boldsymbol{H}$ is the Hessian matrix of f.

1^{st} order necessary conditions for problems with constraints

Let $\boldsymbol{x}^*$ be a point that satisfies:

$$\boldsymbol{h}(\boldsymbol{x}^*) = \boldsymbol{0}, \boldsymbol{g}(\boldsymbol{x}^*) \leq \boldsymbol{0}, \tag{1.3.5}$$

and let $\boldsymbol{J}$ be a set of indexes j such that $g_j(\boldsymbol{x}^*) = 0$. Then $\boldsymbol{x}^*$ is designated a regular point with respect to the constraints 1.3.5 if the gradient vectors $\nabla h_i(\boldsymbol{x}^*)$ and $\nabla g_j(\boldsymbol{x}^*)(i = 1, ..., m$ and $j \in \boldsymbol{J})$ are linearly independent.

Karush-Kuhn-Tucker conditions (KKT) Let $\boldsymbol{x}^*$ be a local minimum of:

$$\underset{x \in \Omega^n}{\text{minimize}}(f(x)), \tag{1.3.6}$$

subject to:

$$\begin{cases} h_i(x) = 0, \text{if } i \in E \\ g_j(x) \geq 0, \text{if } j \in I \end{cases} \tag{1.3.7}$$

and assume that $\boldsymbol{x}^*$ is a regular point relative to the constraints. Then there is a vector $\boldsymbol{\lambda} \in \mathbb{R}^m$ and a vector $\boldsymbol{\mu} \in \mathbb{R}^p$, where $\boldsymbol{\mu} \geq 0$, such that:

$$\nabla f(\boldsymbol{x}^*) + \boldsymbol{\lambda}^T \nabla \boldsymbol{h}(\boldsymbol{x}^*) + \boldsymbol{\mu}^T \nabla \boldsymbol{g}(\boldsymbol{x}^*) = \boldsymbol{0}, \tag{1.3.8}$$

and

$$\boldsymbol{\mu} \cdot \boldsymbol{g}(\boldsymbol{x}^*) = 0. \tag{1.3.9}$$

2^{nd} order necessary conditions for problems with constraints

Assume the functions $f, \boldsymbol{g}, \boldsymbol{h} \in \boldsymbol{C}^2$ and $\boldsymbol{x}^*$ is a regular point relative to the constraints 1.3.5. If $\boldsymbol{x}^*$ is a local minimum of 1.3.6 subject to 1.3.7, then there is a $\boldsymbol{\lambda} \in \mathbb{R}^m$ and a $\boldsymbol{\mu} \in \mathbb{R}^p$, where $\boldsymbol{\mu} \geq 0$, such that the KKT conditions are verified and:

$$\boldsymbol{H}_L(\boldsymbol{x}^*) = \boldsymbol{H}_f(\boldsymbol{x}^*) + \lambda^T \boldsymbol{H}_h(\boldsymbol{x}^*) + \mu^T \boldsymbol{H}_g(\boldsymbol{x}^*), \tag{1.3.10}$$

is semi-defined positive in the subspace tangent to the active constraints in $\boldsymbol{x}^*$, where $\boldsymbol{H}_f, \boldsymbol{H}_g$ and $\boldsymbol{H}_h$ are the Hessian matrices of $f, \boldsymbol{g}, \boldsymbol{h}$

2^{nd} order sufficient conditions for problems with constraints

Let $f, \boldsymbol{g}, \boldsymbol{h} \in \boldsymbol{C}^2$. The sufficient conditions, so that a $\boldsymbol{x}^*$ satisfying 1.3.5 is a local minimum of 1.3.6 subject to 1.3.7, are that there is a $\boldsymbol{\lambda} \in \mathbb{R}^m$ and a $\boldsymbol{\mu} \in \mathbb{R}^p$ such that:

$$\boldsymbol{\mu} \cdot \boldsymbol{g} = 0, \tag{1.3.11}$$

$$\nabla f(\boldsymbol{x}^*) + \boldsymbol{\lambda} \cdot \boldsymbol{h}(\boldsymbol{x}^*) + \boldsymbol{\mu} \cdot \boldsymbol{g}(\boldsymbol{x}^*) = 0, \tag{1.3.12}$$

and:

$$\boldsymbol{H}_L(\boldsymbol{x}^*) = \boldsymbol{H}_f(\boldsymbol{x}^*) + \boldsymbol{\lambda}^T \boldsymbol{H}_h(\boldsymbol{x}^*) + \boldsymbol{\mu}^T \boldsymbol{H}_g(\boldsymbol{x}^*), \tag{1.3.13}$$

be defined positive in the subspace:

$$\boldsymbol{M}' = \{\boldsymbol{y} : \nabla \boldsymbol{h}(\boldsymbol{x}^*) = \boldsymbol{0}, \nabla g_j(\boldsymbol{x}^*) = 0 \text{ for all } j \in \boldsymbol{J}\}, \tag{1.3.14}$$

where

$$\boldsymbol{J} = \{j : g_j(\boldsymbol{x}^*) = 0, \mu_j < 0\}. \tag{1.3.15}$$

Chapter 2

State of the art

Multiple frameworks have been proposed since the canonical MA was created by Moscato in 1989 Some of those are: Meta-Lamarckian MA (Ong and Keane 2004) [8],Self-generation MA (Krasnogor and Gustafson 2004)[9] , and Diffusion Memetic Algorithm (Nguyen, Ong *et al.* 2008)[10].

GAs,Hybrid Genetic Algorithms (HGAs) and MAs have been successfully applied to many applications ranging from structural optimization to image classification. Their success stems from their flexibility, ease of use and ability to escape local minima.

2.1 Recent Developments

In 1999, ten years later after the initial proposal, Moscato [11] proposed a MA to address the Traveling Salesman Problem (TSP). In this algorithm *agents* competed and cooperated interleaved between local search stages.

In 2005, Fawaz, Xu and Behdinan [12] publish the use of an hybrid evolutionary algorithm with a local search heuristic applied to a truss structure, showing that a significant reduction of number evaluations can be achieved in such optimization problems.

Yun (2006) [13] proposed an hybrid genetic algorithm with adaptive local search scheme, i.e, an Adaptive Memetic Algorithm (AMA). In this study, two types of adaptive local search techniques are presented. The conditional local search method is the first mode, which can be used to measure the average fitness values obtained from the continuous two generations of the a-hGA, while the similarity coefficient method is the second mode, which can be used to gauge how similar the members of the population of the a-hGA are to one another.

Nghia Le *et al.*(2009)[14] attempted to study the intrinsic properties of benchmark problems. They introduce the concepts of "local optimum structure" and "connectivity structure".

The idea of 'constructive'/'obstructive' local optimum structure is described as a characteristic of the modified fitness landscape as a result of the individual learning process in MAs. 'Constructive'/'obstructive' connectivity is proposed as a trait for exposing the inner workings of an MA solver in search.

It is shown that the connectivity and local optimum structure properties have a significant impact on the search process and efficiency of MAs.

The results on common benchmark problems showed that the altered landscape structure proposed is superior to the original.

The article titled "Memetic algorithms based on local search chains for large scale continuous optimization problems: MA-SSW-Chains" by Molina *et al.*(2011)[15] introduces a variant of memetic algorithms called MA-SSW-Chains. The objective of MA-SSW-Chains is to address large-scale continuous optimization problems. In a previous work [16] a MA with Local Search (LS) chains that assigns each individual a LS sensitivity depending on its features was proposed by the same authors. They define LS chain as act of initializing a LS operator using the final configuration/state/result of a previous call of the LS operator.

In the model developed by these authors, a solution created by an LS invocation may subsequently serve as the starting point for another LS invocation, adopting the final values for the former's strategy parameters as its initial values. By adapting its strategy parameters to the unique characteristics of the search zones in this way, the continuous LS technique can increase the LS effort over the most promising solutions and regions.

To address the issue of large scale optimization a modified local searcher was proposed.

The results were similar to other state-of-the-art algorithms.

Zhang *et al.* (2013)[17] propose a Memetic Differential Evolution Algorithm with both Lamarckian and Baldwinian learning procedures. The local search is done by the Hooke-Jeeves algorithm. The intent of this algorithm is to obtain a better balance between exploration and exploitation. They divide the population into subpopulations based on von Neumann's topology and periodically there is an exchange of information via migration.

António (2014)[18] suggests a memetic-based selfish gene algorithm (MA + SG), a mixed model that employs a number of learning techniques to investigate the interplay between various cultural transmission laws and the evolutionary process. This model applies the selfish gene algorithm instead of a traditional local search operator.

Feng *et al.* (2015)[19] , in their paper, propose a definition to local and global effectiveness of a local search operator. Local effectiveness translates into a numerical value the improvements in convergence made by local and global search operators based on composite objective. Global effectiveness is the number of local search solutions divided by number of global search solutions in the Pareto front.

Yan *et al*(2015) [20] present the GADEDHC, an adaptive memetic algorithm, which combines a genetic algorithm and differential evolution algorithm. It uses a directional hill climbing algorithm as local search method. Additionally, a novel approach is put forth to balance the intensity of local and global search methods as well as the ratio between genetic and differential evolution. Experiments on a variety of benchmark issues of varying complexity have demonstrated the new approach's ability to deliver results that are similar to other state-of-the-art algorithms

Zhang *et al.*, (2016)[21] propose a multi-objective memetic algorithm based on decomposition for big optimization problems (MOMA/DBigOpt) with a gradient based local searcher. They compared it with other MOEA's with satisfying results.

Wang *et al.* (2017)[22] propose an hybrid filter SQP-GA to tackle non-linear constrained optimization problems. They apply it to the minimization of the maximum vertical acceleration of a auxetic [1] jounce bumper in the suspension of a vehicle. This filter transforms the single optimization problem into a dual optimization problem to handle the constrains.

Shi *et al.* (2017)[23] propose a method to escape local minima. In this method the local search is conducted with a penalized objective function.

The penalty function is applied to the candidate solution for every feature that is present in the recorded elite. Every time the local search arrives at a local minimum the penalty is increased.

The Elite Based Guided Local Search(EB-GLS) proposed in this article reduces the penalties near the recorded best to increase the search near this point hoping it is near the global optimum.

In order to characterize biological intelligence using the notions of useful and utility, Luca and Craus (2018)[24] suggest three LS algorithms for hybridization with an existing Evolutionary Algorithm (EA) with Pareto ranking. The hybridization meant to reach solutions that wouldn't be available otherwise.

According to their definition, biological intelligence refers to an individual's capacity to directly or indirectly access information using the ideas of utility and usefulness. Anything that helps complex behaviors become better organized is useful, and anything that is profitable is utility.

[1] Auxetic materials expand transversely when under tension and have the opposite behaviour when under compression, i.e, they have negative Poisson's ratio.

Liu, Yuen and Sung (2022)[25] developed a method to build a portfolio of algorithms that perform well for a class of problems. According to their experimental results, this method of choosing well-coordinated algorithms is effective, and the constructed portfolio has the highest rank when compared to individual algorithms, elite portfolios, and comparative algorithms.

Chen *et al.* (2022)[26] propose a hybrid operator selection technique based on feedback and support vector machine classification. They apply it to the multiobjective optimization algorithm's decomposition-based framework. The technique chooses the operator with the highest promising performance at various phases of evolution, using this operator to produce children based on the past performance of various operators.

Chapter 3

Non-linear programming

3.1 Descent Methods

Classical methods were the first(as the name suggests) try to address minimization problems. They have solid mathematical foundations in algebra and calculus. These methods use derivatives and pseudo-derivatives to move from one point to another with a lower cost/objective function value. This can be interpreted as someone moving down a hill using the slope to guide itself.

In general way in this kind of method the search starts with an initial point, a search direction is computed based on values of the function, derivatives and/ or second order derivatives, then a step size is found in the search direction that ensures the function is being minimized. This is repeated until stopping criteria is achieved.

Algorithm 1 Generic Descent Method

Choose an $\boldsymbol{x}^0$. $k \leftarrow 0$.
while Stopping criteria not met **do**
  Compute $\boldsymbol{d}^k$
  Find an α^k that minimizes f along $\boldsymbol{d}^k$ via linear search
  $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \cdot \boldsymbol{d}^k$
  $k \leftarrow k + 1$
end while

3.1.1 Steepest Descent Method - Pure Gradient Method

This is the simplest first-order method, simply taking finite steps in the opposite direction of the gradient.

Algorithm 2 Steepest Descent Method

Choose an $\boldsymbol{x}^0$. $k \leftarrow 0$.

while Stopping criteria not met **do**

$\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$

Find an α^k that minimizes f along $\boldsymbol{d}^k$ via linear search

$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \cdot \boldsymbol{d}^k$

$k \leftarrow k + 1$

end while

3.1.2 Conjugate Gradient Method

The conjugate gradient method is an iterative numerical algorithm for solving systems of linear equations. It is commonly used for solving large systems of linear equations that arise in various fields such as engineering, physics, and computer science.

The basic idea of the conjugate gradient method is to iteratively improve an initial guess of the solution to a linear system by searching for the solution in a Krylov subspace [1] , which is a vector space generated by the initial guess and a sequence of orthogonal vectors that are obtained by multiplying the initial guess by the system matrix.

At each iteration, the method computes a new search direction by combining the residual of the current solution estimate with the previous search direction in a way that ensures orthogonality. The method then moves along this direction to obtain a new estimate of the solution. This process is repeated until the desired accuracy is achieved or a maximum number of iterations is reached.

There are multiple ways to compute the residual. Two of those are the Fletcher-Reeves (Equation 3.1.1) and the Polack-Ribière (Equation 3.1.2) formulas[4].

$$\beta_{FR}^k = \frac{\left\|\nabla f(\boldsymbol{x}^{k+1})\right\|^2}{\left\|\nabla f(\boldsymbol{x}^k)\right\|^2} \tag{3.1.1}$$

$$\beta_{PR}^k = \frac{\nabla f(\boldsymbol{x}^{k+1}) \cdot (\nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k))}{\left\|\nabla f(\boldsymbol{x}^k)\right\|^2} \tag{3.1.2}$$

[1] In linear algebra, the order-r Krylov subspace generated by an n-by-n matrix A and a vector b of dimension n is the linear subspace obtained by sequentially multiplying r times the vector b by the matrix A (starting from A^0) [27].

The conjugate gradient method can be extended to handle non-linear equations and can also be used for optimization problems where the goal is to minimize a quadratic objective function.

Algorithm 3 Conjugate Gradient Method

Choose an $\boldsymbol{x}^0$. $k \leftarrow 0$

while Stopping criteria not met **do**

  if $k = 0$ **then**

    $\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$

  else

    $\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k) + \beta^{k-1}\boldsymbol{d}^{k-1}$

  end if

  Find an α^k that minimizes f along $\boldsymbol{d}^k$ via linear search

  $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \alpha^k \cdot \boldsymbol{d}^k$

  Compute β^k either according to 3.1.1 or 3.1.2

  $k \leftarrow k + 1$

end while

3.1.3 Newton's Method

Newton's Method is an iterative method based in the quadratic approximation of the function using a Taylor's Polynomial. It yields the exact result for quadratic polynomials in a single iteration but can be extended to other functions with the addition of linear search.

The second order Taylor series approximation of a function f with a single variable around point a is[4]:

$$f(x) \approx f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) \tag{3.1.3}$$

The second order Taylor series approximation of a multivariable function f around point $\boldsymbol{x}_0$ is:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + \nabla f_0^T(\boldsymbol{x} - \boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^T \boldsymbol{H}_0(\boldsymbol{x} - \boldsymbol{x}_0) \tag{3.1.4}$$

where ∇f_0 and $\boldsymbol{H}_0$ are the gradient and the hessian matrix evaluated at $\boldsymbol{x}_0$.

At a **stationary point** of f the first order partial derivatives are null:

$$\frac{\delta f(\boldsymbol{x}^*)}{\delta x_j} = 0,\ j = 1, 2, ..., n \tag{3.1.5}$$

differentiating 3.1.4 and setting ∇f to $\mathbf{0}$ we get:

$$\nabla f \approx \nabla f_0 + \boldsymbol{H}_i(\boldsymbol{x}^* - \boldsymbol{x}_0) = \mathbf{0} \tag{3.1.6}$$

rearranging the equation and assuming that H is nonsingular we get:

$$\boldsymbol{x}^* \approx \boldsymbol{x}_0 - \boldsymbol{H}^{-1}\nabla f_i \tag{3.1.7}$$

which leads to the iterative step:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \boldsymbol{H}_k^{-1}\nabla f^k \tag{3.1.8}$$

If f is a quadratic polynomial it can be written as:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c \tag{3.1.9}$$

The minimum of this function is given by:

$$\nabla f = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{0} \tag{3.1.10}$$

or

$$\boldsymbol{x}^* = -\boldsymbol{A}^{-1}\boldsymbol{b} \tag{3.1.11}$$

By applying 3.1.8 to a quadratic function we get:

$$\boldsymbol{x}^{k+1} = \boldsymbol{A}^{-1}(\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{b}) = -\boldsymbol{A}^{-1}\boldsymbol{b} \tag{3.1.12}$$

, i.e, the exact solution.

On average, Newton's Method has a better performance than 1_{st} order methods, apparently contradicting the No Free Lunch Theorem (NFL). This can be easily justified by the fact that it uses extra information in the form of second order derivatives, *the lunch isn't free*.

One must be aware that this method not always converges to a minimum, it may converge to a maximum or a saddle point, i.e, a stationary point.

One way to ensure that the method leads to a minimum is to impose the monotonous decrease of the function. Algorithm 4 is one way to implement this method.

Algorithm 4 Newton's Method based on linear search

Input: $c_1 \geq 0, c_2 \geq 0, p \geq 2, \geq 3$

Choose $\boldsymbol{x}^k$. $k \leftarrow 0$

while Stopping criteria not met **do**

if $\boldsymbol{d}_N^k$ is a solution to the system of linear equations $\boldsymbol{H}(\boldsymbol{x}^k)\boldsymbol{d}_N = -\nabla f(\boldsymbol{x}^k)$ and satisfies the conditions $\nabla f(\boldsymbol{x}^k) \cdot \boldsymbol{d}_N^k \leq -c_1\left\|\nabla f(\boldsymbol{x}^k)\right\|^q$, $\left\|\boldsymbol{d}_N^k\right\|^p \leq c_2\left\|\nabla f(\boldsymbol{x}^k)\right\|$ **then**

$\boldsymbol{d}^k = \boldsymbol{d}_N^k$

else

$\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$

end if

Find an α^k that minimizes f along $\boldsymbol{d}^k$ via linear search

$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \cdot \boldsymbol{d}^k$

$k \leftarrow k + 1$

end while

3.1.4 Quasi-Newton Methods

As said previously, a disadvantage of Newton's Method is that it requires second order derivatives, in an effort to fight this there were created the Quasi-Newton Methods that try to approximate the (inverse of the) Hessian matrix iteratively with the history of the search direction and information of the gradient in the previous points of the method.

The following equations define the Quasi-Newton methods:[2]

$$\boldsymbol{\delta}^k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k, y^k = \nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k) \tag{3.1.13}$$

$$\boldsymbol{H}^{k+1} = \boldsymbol{H}^k + \Delta \boldsymbol{H}^k \tag{3.1.14}$$

Many methods to update the approximate Hessian matrix ($\boldsymbol{B}^k$) were created, but they can be divided in two types: class 1 and class 2 based on the updating formula. In this work will be used class 2 Broyden's methods (the most used). The updating formula is given by[3, 4, 5]:

$$\Delta \boldsymbol{H}^k = \frac{\boldsymbol{\delta}^k (\boldsymbol{\delta}^k)^T}{\boldsymbol{\delta}^k \cdot \boldsymbol{y}^k} - \frac{\boldsymbol{H}^k \boldsymbol{y}^k (\boldsymbol{H}^k \boldsymbol{y}^k)^T}{(\boldsymbol{y}^k)^T \boldsymbol{H}^k \boldsymbol{y}^k} + c(\boldsymbol{y}^k)^T \boldsymbol{H}^k \boldsymbol{y}^k \boldsymbol{v}^k (\boldsymbol{v}^k)^T \tag{3.1.15}$$

where:

$$\boldsymbol{v}^k = \frac{\boldsymbol{\delta}^k}{\boldsymbol{\delta}^k \cdot \boldsymbol{y}^k} - \frac{\boldsymbol{H}^k \boldsymbol{y}^k (\boldsymbol{H}^k \boldsymbol{y}^k)^T}{(\boldsymbol{y}^k)^T \boldsymbol{H}^k \boldsymbol{y}^k}, \tag{3.1.16}$$

and c is a non-negative scalar. For $c = 0$ the equation degenerates to the Davidon-Fletcher-Powell (DFP) formula and $c = 1$ we get the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula.

Similarly to Newton's Method, precautionary steps must be taken to ensure the method converges to a minimum. To do this one can take advantage of one of this methods properties: if $\boldsymbol{H}^0$ is positive defined [3] (it usually is set as the identity matrix, ensuring this condition) and, at each iteration, $\boldsymbol{\delta}^k \cdot \boldsymbol{y}^k \geq 0$ then $\boldsymbol{H}^k$ is positive defined for all iterations. [4] A generic Quasi-Newton method can be given by:

Quadratic Programming

Quadratic programming deals with a particular set of non-linear problems, the ones that have a quadratic objective function[28].

[2] In this section the inverse of the approximate Hessian matrix is denoted as $\boldsymbol{H}$, an unfortunate naming convention in the opinion of the author.

[3] A positive definite matrix is a square matrix where all its eigenvalues are positive.

[4] If $\boldsymbol{H}^k$ isn't defined then it doesn't make sense to talk about an approximation of the inverse of the Hessian matrix, on the other and if $\boldsymbol{H}^k$ isn't positive we get that $\boldsymbol{d}^k \cdot \nabla f(\boldsymbol{x}^k) > 0$, indicating that we might be moving to a maximum instead of a minimum.

Algorithm 5 Generic Quasi-Newton Method

Choose an $\boldsymbol{x}^k.\boldsymbol{H}^0 = \boldsymbol{I}$, $k \leftarrow 0$

while Stopping criteria not met **do**

$\boldsymbol{d}^k = -\boldsymbol{H}^k \nabla f(\boldsymbol{x}^k)$

Find an α^k that minimizes f along $\boldsymbol{d}^k$ via linear search

$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \cdot \boldsymbol{d}^k$

Compute $\Delta \boldsymbol{H}^k$

$\boldsymbol{H}^{k+1} = \boldsymbol{H}^k + \Delta \boldsymbol{H}^k$

$k \leftarrow k + 1$

end while

A generic Quadratic Programming (QP) can be stated as:

$$\underset{x}{\text{minimize}}(f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{G}\boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{c}) \tag{3.1.17}$$

subject to:

$$\begin{cases} \boldsymbol{a}_i^T \boldsymbol{x} = b_i, i \in E \\ \boldsymbol{a}_j^T \boldsymbol{x} \geq b_j, j \in I \end{cases} \tag{3.1.18}$$

where G is a symmetric matrix, $\boldsymbol{c}$ is a vector of constants, E and I are the indices of the constraints and $\boldsymbol{a}_i$,$\boldsymbol{a}_j$ are constraint related constants.

Methods have been developed to solve this kind of problems, both constrained and unconstrained.

Sequential Quadratic Programming

The point of sequential quadratic programming is to approximate the objective function in the vicinity of the current point by a quadratic problem and solve that subproblem using Quadratic Programming. It differs from the Newton's and Quasi-Newton's methods referred previously by the inclusion of restriction in it's formulation in the form of Lagrange Multipliers. That being said, for an unconstrained problem, Quadratic Programming methods degenerate into Newton's (and Quasi-Newton's) methods[28].

The Sequential Quadratic Programming (SQP) framework to solve optimization problem in the presence of equality and inequality constraints is:

$$\underset{x}{\text{minimize}}(f(\boldsymbol{x})) \tag{3.1.19}$$

subject to:

$$\begin{cases} h_i(\boldsymbol{x}) = 0, i \in E \\ g_i(\boldsymbol{x}) \geq 0, i \in I \end{cases} \tag{3.1.20}$$

By linearizing the constraints we can transform our problem into:

$$\underset{p}{\text{minimize}}(f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2 \mathcal{L}_k p) \tag{3.1.21}$$

where $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda h}(\boldsymbol{x}) + \max(0, \boldsymbol{\mu g}(\boldsymbol{x}))$ is the Lagrangian function of the problem, subject to:

$$\begin{cases} \nabla h_i(\boldsymbol{x}^k)^T p + h_i(\boldsymbol{x}_k) = 0, i \in E \\ \nabla g_j(\boldsymbol{x}^k)^T p + g_j(\boldsymbol{x}_k) \geq 0, j \in I \end{cases} \tag{3.1.22}$$

where E and I are the sets of equality and inequality constraint indexes.

3.2 Constraint Handling

3.2.1 Side Constraints

The most common way to implement side constrains in non-linear programming is during line search. Before enacting any kind of function evaluation it's possible to backtrack the search until it lands inside the search space.

3.2.2 Linear Equality Constraints

Linear Equality Constraints can be easily included by solving them with respect to one variable, therefore that variable can be obtained knowing the value of the other, effectively reducing the complexity of the problem.

3.2.3 Non-linear Constraints

Penalty Methods

Penalty methods are the simplest way of including constraints, they do so by penalizing non-admissible points[3, 4].

Quadratic Penalty

This is the most used penalty method, the penalty function $\Psi(\boldsymbol{x})$ is defined has:

$$\Psi(\boldsymbol{x}) = \sum_{i=1}^{n} \lambda_i \cdot g_i(\boldsymbol{x}^2) \tag{3.2.1}$$

where λ_i are constants defined by the designer (often all equal when the constraints are normalized).

Augmented Lagrangian Method

Because λ_i's defined in the previous method don't have a physical sense they are hard to define when the user lacks knowledge about the problem.

The Augmented Lagrangian Method circumvents this by updating the penalties based on the constraint violations until they reach an acceptable value defined by the user. This value has a physical sense so the designer can easily define it. The difference between the Lagrangian Method and the Augmented Lagrangian Method is the addition of a quadratic penalty component.

When using this method the objective function, f, is replaced by the Augmented Lagrangian, L_a, of the problem. When we have equality and inequality constraints the Augmented Lagrangian can be given by[3]:

$$L_a(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\lambda}; \varepsilon) = f(\boldsymbol{x}) + \boldsymbol{\mu} \cdot \boldsymbol{h}(\boldsymbol{x}) + \frac{1}{\varepsilon}||\boldsymbol{h}(\boldsymbol{x})||^2 + \boldsymbol{\lambda}^T max\{\boldsymbol{g}, -\frac{\varepsilon}{2}\boldsymbol{\lambda}\} + \frac{1}{\varepsilon}||max\{\boldsymbol{g}(\boldsymbol{x}), -\frac{\varepsilon}{2}\boldsymbol{\lambda}\}||^2 \quad (3.2.2)$$

where

To update ε we can use a rule as simple as:

$$\varepsilon^{k+1} = \rho\varepsilon^k \quad (3.2.3)$$

where

$$\begin{cases} \rho = 0 & \text{, if } ||\boldsymbol{h}(\boldsymbol{x}^{k-1})||^2 \text{ is small enough.} \\ \rho < 1 & \text{, if } ||\boldsymbol{h}(\boldsymbol{x}^{k-1})||^2 \text{ otherwise.} \end{cases} \quad (3.2.4)$$

To update the multipliers we can use a 1^{st} order formula:

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \frac{2}{\varepsilon^k}\boldsymbol{h}(\boldsymbol{x}^k) \quad (3.2.5)$$

and:

$$\boldsymbol{\lambda}^{k+1} = max\{0, \boldsymbol{\lambda}^k + \frac{2}{\varepsilon^k}\boldsymbol{g}(\boldsymbol{x}^k)\} \quad (3.2.6)$$

Chapter 4

Bio-inspired Algorithms

When dealing with Classical methods the first question arises almost immediately: *What if I can't calculate the derivatives?* either because they are too cumbersome or impossible to obtain (as is the case of a *blackbox* objective function). That's were Bio-inspired Algorithms come into play.

These algorithms can be divided into two big categories:evolutionary and swarm intelligence-based. The former is inspired Darwinism while the latter is inspired is Behavioural Biology. Two examples of evolutionary algorithms are GA' and Differential Evolution (DE); two examples of swarm intelligence-based algorithms are Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO).

In general, there are a couple of common characteristics to EAs:

- they are population based, i.e, they process a set of solutions in parallel
- they use recombination, taking two ore more solutions to produce a new one
- they are stochastic

The most important components of EAs are:

- Representation
- Fitness function
- Selection operator for mating
- Recombination
- Mutation
- Selection operator for survival
- Initialization and stopping criteria

The term swarm is used for any aggregation of agents or individuals that interact as group. They don't have an hierarchy, each one of them acting using information obtained by other individuals. Separated, each individual has low or no intelligence. The strength of this algorithms comes from the group interactions.

Generally the group behaviour of swarm intelligence methods follows a set of properties[29]:

- Proximity - Agents should be able to interact between themselves directly or indirectly while they process information about space-time.
- Quality - Agents should be able to evaluate their their own behaviour when answering to quality factors in the ambient context.
- Response diversity - Allows the system to react to unexpected situations by not allocating all the search resources to to a restricted search space.
- Stability - The system shouldn't change it's behaviour with any minor ambient fluctuation.
- Adaptability - The system should be able to adapt to ambient variations.

The basic characteristics of self organization are[30]:

- Positive feedback - The use of rules that encourage agents to pursue premising regions in the search space.
- Negative feedback - To counterbalance positive feedback, stabilizing the system. Mechanisms such as limiting the number of foragers, satiety, food exhaustion, competition for food,etc.
- Fluctuations - Randomness is essential to ensure diversity, creativity and innovation. They allow the search of regions that wouldn't be available by a greedy approach.
- Multiple interactions - Self organization requires a minimum population density of individuals that tolerate each other just enough to take advantage of the results of their own activities and their neighbour's.

Bio-inspired algorithms are characterized by their ability to scan over a big search space making it more unlikely that they will get stuck in local minima.

4.1 Particle Swarm Optimization (PSO)

From the family of of swarm intelligence algorithms, the particle swarm is the most popular.

An initial random population with small initial speeds is generated.

Then at each step, for each agents, the speed is updated based on the speed in the last step, the current position of the agent, the position of the best agent in the swarm and the best position ever found by swarm:

$$v_i(t+1) = \omega v_i(t) + c_1\, rand(.)(p_i^{best} - p_i^t) + c_2\, rand(.)(p_{swarm\,best} - p_i^t) \tag{4.1.1}$$

where $rand(.)$ is a function that generates a random number $r \in [0,1]$, and ω, c_1 and c_2 are constants specified by the user.

The positions are then updated using:

$$p_i(t+1) = p_i(t) + v_i(t) \tag{4.1.2}$$

This two steps are repeated until a good enough solution is found. The simplicity of the algorithm justifies it's popularity.

4.2 Ant Colony Optimization (ACO)

The point of ACO is to mimic the cooperation between ants in Nature. The interest in their behaviour comes from the fact that they can carry out complex tasks as a group while performing simple tasks as individuals. It was originally designed as an heuristic to solve TSP kind of problems[31, 32].

The algorithm is composed by two steps:

1. **Solution construction/Edge selection -**This is achieved by a probabilistic state change. Each ant/individual performs a greedy construction of a solution using stochastic mechanisms, adding parts of a solution until they have a complete solution.

2. **Pheromones update -**This step is done using the generated solutions. It is divided into two smaller steps:

   (a) **Evaporation phase -**The pheromones of a trail decrease automatically with time. This can be done with a fixed proportion:

   $$\tau_{ij} = (1-\rho)\tau_{ij},\ \forall i,j \in [1,n] \tag{4.2.1}$$

   (b) **Reinforcement phase -**The pheromones are updated according to the generated solutions. This can be done using a variety of strategies.

4.3 Differential Evolution (DE)

Differential evolution is part of the evolutionary category, it was first proposed by Storn and Price[33].

It uses real encoding and it's main selling point is that, in it most basic form, only needs three parameters:population size-NP, scale factor-F and crossover rate-CR.

The basic search mechanisms of the original DE are as follows:

1. **Mutation** Two vectors,$\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, from the population are randomly chosen. From these is drawn a vector that is then multiplied by a constant between 0 and 2 and added to a third vector,$\boldsymbol{x}_3$, from the population. If the newly generated solution,$\boldsymbol{x}'_3$, is better than a fourth randomly chosen vector from the population, $\boldsymbol{x}_4$, than it replaces it. This process is called mutation, and can be written as: $\boldsymbol{x}' = \boldsymbol{x}_3 + F \cdot (\boldsymbol{x}_2 - \boldsymbol{x}_1), F \in [0, 2]$.

2. **Crossover** A vector obtained from mutation and another from the population are selected. Parameters are taken from one vector or the other with a probability CR, Crossover Rate, given by the user, while making sure that at least one parameter from the mutant vector is chosen.

3. **Selection** To decide if the vector resultant from crossover should be included in the next generation it is compared with the vector from the population that gave origin to it. If it has a better fitness it is kept, otherwise the original vector is kept.

4.4 Genetic Algorithms (GA)

For over 3.5 billion years, life has evolved and thrived, constantly adapting to various environments and challenges. From the deep ocean trenches to the scorching deserts, living organisms have developed ingenious solutions to survive and flourish. By studying and imitating nature's designs, engineers seek to harness the efficiency, resilience, and elegance found in biological systems. From mimicking the structure of bird wings to enhance aircraft aerodynamics, to replicating the self-cleaning mechanism of lotus leaves for surface coatings, biomimetic engineering seeks to unlock the secrets of nature and apply them in creating innovative and sustainable solutions for the challenges we face today.

But why stop there? Why not imitate evolution itself?

GAs draw inspiration from Darwin's Theory of Evolution, they are intended to copy the mechanisms that lead to evolution of species and apply them to the *evolution* of solutions. Selection, Mutation and Crossover are the operators common to all of them. The following flowchart show's a generic implementation:

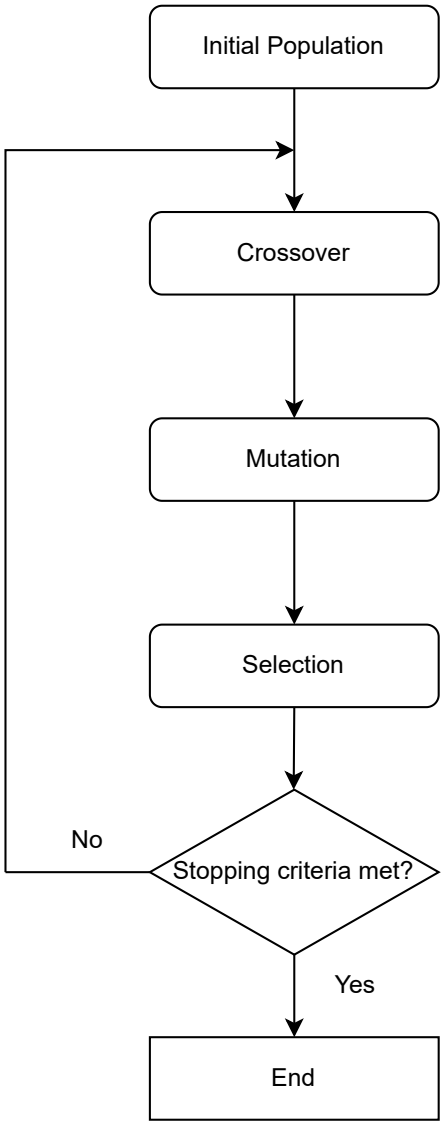

Figure 4.1: Generic Genetic Algorithm

The stopping criteria may contain various metrics, such as:

- Maximum number of generations
- Maximum number of function evaluations
- Time since the search started
- Number of generations since an improvement has been found

- Fitness threshold
- Diversity

4.4.1 Representation of the search space

After the project variables and the search space that represents the phenotype of the population are obtained, one needs to represent the individuals/solutions with an appropriate encoding, the genotype. In Nature, the genotype is encoded in the DNA .

There are many ways to represent the search space:

- Binary encoding
- Binary encoding real
- Integer encoding
- Real encoding
- Vectors, lists and matrices (of integers, characters, etc.)

Experimental studies have shown there isn't a single best representation as it is problem dependent. Furthermore, one representation might lead to the existence of local minima while other representation doesn't.

Following the example given by Moscato[34], let's say we have the following problem:

$$\underset{x\in\Omega}{\text{minimize}}(f(x) = (x-8)^2) \tag{4.4.1}$$

where:

$$\Omega = \{0, 1, 2, ..., 15\} \tag{4.4.2}$$

Now assume we use a representation where we connect 0 and 15 allow moves in one of two directions:$i+1$ and $i-1$. We take the move that produces the best value of f. In this case, no matter were we start, we will always reach $x = 8(f(8) = 0)$, the global minimum.

If instead we choose a traditional binary representation where each integer is represented by a 4-bit string and allow moves by flipping one bit, at each point we get four possible moves. Our first instinct would be to assume that having more moves would make it easier to not get stuck, but that's not the case

The configuration 0111(7) ($f(7) = 1$) has the following neighbours:

- 1111(15) - $f(15) = 49$
- 0011(3) - $f(3) = 25$
- 0101(5) - $f(5) = 9$
- 0110(6) - $f(6) = 4$

As we can see, even tho the new representation introduces more possible moves, it also introduces a local minimum, so we can't easily choose a representation a priori.

The type of representation is usually chosen by it's convenience.

Traditional binary encoding

Most GAs use a traditional binary encoding. This further exacerbates the problem of BiAs having difficulties at arriving at the global optimum as it is very unlikely that whatever grid size is chosen happens to have a point exactly at the global optimum.

Each solution is composed by a set of n project variables, x_i,:

$$\boldsymbol{x} = (x_1, x_2, ..., x_n), x_i \in \Omega_i \subseteq \mathbb{R}^n \tag{4.4.3}$$

which can be easily transformed into the genotype and vice-versa:

$$x_{i,bin} = \sum_{k=0}^{l_i - 1} 2^k b_{i,k} \tag{4.4.4}$$

$$x_i = x_i^{inf} + \frac{x_i^{sup} - x_i^{inf}}{2^{l_i} - 1} x_{i_b in} \tag{4.4.5}$$

where $b_{i,k}$ are the values at the k^{th} position of the i^{th} sub-string, l^i is the length of the sub-string and x_i^{sup}, x_i^{inf} are the upper and lower limits(size constraints) of each project variable.

Hamming Space

The Hamming Space uses a bit-string representation. In this space the distance, d_h, is defined as the number of moves(bit-flips) needed to go from one point to another:

$$d_H(\boldsymbol{v}, \boldsymbol{z}) = \sum_{i=1}^{n} |v_i - z_i| \tag{4.4.6}$$

For example, lets say $\boldsymbol{v} = (1, 0, 1, 0)$ and $\boldsymbol{z} = (1, 1, 0, 1)$, then the distance between $\boldsymbol{v}$ and $\boldsymbol{z}$ is 3 because we would need to flip the 2^{nd}, 3^{rd} and 4^{th} bits to go from $\boldsymbol{v}$ to $\boldsymbol{z}$, and vice versa.

4.4.2 Exploration vs Exploitation

The balance between exploitation and exploration is a fundamental concept in the field of GAs. Exploitation refers to the process of leveraging the current knowledge to exploit promising areas of the search space. It involves intensifying the search in regions where good solutions have been found, refining and improving them further. Exploitation is crucial for converging towards optimal solutions efficiently and exploiting the knowledge gained during the search.

Relying solely on exploitation can lead to premature convergence and getting stuck in local optima. This is where exploration comes into play. Exploration involves venturing into unexplored regions of the search space to discover potentially better solutions. By promoting diversity and allowing for the discovery of new and unexplored areas, exploration prevents the algorithm from being trapped in suboptimal solutions.

Achieving the right balance between exploitation and exploration is a challenging task in genetic algorithms. Initially, exploration should be prioritized to ensure a diverse exploration of the search space. As the algorithm progresses and gains knowledge, exploitation should become more intense to refine the solutions. Balancing these two strategies effectively allows genetic algorithms to escape local optima while efficiently converging towards globally optimal solutions.

Various techniques can be employed to strike the exploitation-exploration balance, such as using different selection mechanisms, employing mutation operators with varying exploration levels, and incorporating adaptive strategies that dynamically adjust the balance based on the algorithm's performance.

Overall, finding the optimal trade-off between exploitation and exploration is essential in genetic algorithms to ensure their robustness, adaptability, and ability to discover high-quality solutions across a wide range of problem domains.

4.4.3 Operators

One operator, one landscape [35, 36]

There is an important aspect when selecting the operators: each operator produces it's own landscape. Because any two different operators produce a different mapping of input solutions to output solutions with an associated probability, when we change one of the operators we also change search procedure and therefore the landscape. What this implies is that the existence of local minima under one operator does not imply the existence of local minima under another operator, for example, if our mutation operator generates a completely random solution then any individual can become any other solution, meaning that under this operator there are no local minima.

There is another fact that might *fly under the radar*, the concept of search landscape isn't restricted to mutation. Crossover, Selection and all the other type of operators also have an associated search landscape, as they produce one or multiple solutions when given an input solution(s).

Selection Operators

Together with the fitness function they determine how likely it is that good or bad solutions should keep existing and reproducing. If we only allow the best solutions to reproduce and survive then the diversity of the population will decrease rapidly leading to premature convergence. On the other hand if the probability of being selected of both good and bad solutions is too similar the algorithm will struggle to converge.

Two approaches can be taken when selecting individuals for crossover and survival:

- Proportional selection- the individuals are selected with a probability calculated based on their fitness.
- Sorting based selection- the individuals are sorted and then their selection probability is calculated based on their ranking.

There are infinitely many ways to define the probability of selection an individual for crossover and survival.

Mutation Operators

The operator is responsible to introduce new genetic information into the population ensuring that, eventually, the algorithm will reach optimum solution.

Let's say all the individuals in the population at some stage of the search had the same value in the n^{th} bit, 1 for example. If we didn't have a way to randomly generate information then there would be no way to get a 0 in that bit, so there would be no way to ensure we could reach the global minimum. In fact half the search space couldn't be searched.

Flip-Bit mutation

The simplest was to implement mutation is, after selecting a target individual, to go over each bit and flip it with a given probability. That can be achieved by looping over the bits generating a random numbers between 0 and 1, if the randomly generated number is smaller than the probability then the bit is flipped.

Implicit mutation

This is the extreme case of Flip-Bit mutation where every bit has a 50/50 chance to flip, essentially creating a completely random solution.

Real Randomly Uniform Mutation

This kind of mutation generates and individual inside an hyper-cube centered around a target individual. The size of the edges of the hyper-cube are given by the user.

Real Normally Distributed Mutation

This mutation applies a displacement to the target individual where the components of the displacement vector are given by a Gaussian distribution with 0 mean and standard deviation given by the user. This mutation ensures the new individual is more likely to be generated close by the target individual than further away.

Real Polynomial Mutation

Similarly to Real Normal Distributed Polynomial Mutation also generates new individuals close by with a higher probability, differing in the probability distribution function.

Crossover Operators

The crossover operator mainly affects how fast the algorithm converges to a solution. They can have real or binary based encoding. This operators can take inputs like the number of cutoff points, the bias towards one of the parents and the number of parents or children.

Single point crossover (1PX)

This algorithm uses binary encoding.

In this algorithm a random point in the genetic material is chosen, then the offspring is generated taking genetic material from one parent to the first part and genetic material from the other parent to the other part.

Algorithm 6 Single Point Crossover

Step 1: Generate a random number k between 1 and l_c

Step 2: **For** $i = 1$ to l_c

If $i \leq$k

$s_{1,i} = p_{1,i}, s_{2,i} = p_{2,i}$

Else

$s_{1,i} = p_{2,i}, s_{2,i} = p_{1,i}$

End If

End For

where $\boldsymbol{s}$ and $\boldsymbol{p}_1, \boldsymbol{p}_2$ are the genome of the offspring and the parents, respectively. Algorithm 6 generates two children, one takes the genetic material from the first parent before the cutoff point and the remaining genetic material comes from the second parent. The other child takes the opposite approach, it takes the genetic material from the second point until the cut-off point and the it takes from the first parent.

Multi-point crossover (MPX)

This algorithm takes a similar approach to Single Point Crossover, but instead generates multiple cutoff points.

Uniform crossover (UX)

In the extreme case of a multi-point crossover where every point has a 50% probability of becoming a cutoff point we have a uniform crossover, where each gene has a probability of being chosen form one individual or the other.

Segmented Crossover (SX)

Segmented Crossover is similar to Multipoint Crossover but the number of cutoff points is randomly generated, we do so by deciding if each gene should become a cutoff point given a segment switching rate.

Crossover based on real encoding

Usually one of two approaches is taken:

1. Mean-centric recombination The offspring is generated near the centroid of the two parents plus a variation.
2. Parent-centric recombination The offspring is generated around the parents based on some probability distribution that ensure a higher probability of the offprinsg being generated near the parents.

Similarity control

Elitist strategies (or any other that has an heavy exploitation component) have a serious risk of converging too early to a local minimum. Similarity control is a family of operators that remove or penalize solutions that are too alike.

The similarity can be of variables or objective function(in the case of multi-objective optimization).

The measure of difference may be done using any convenient distance that is appropriate to the optimization problem(euclidean distance, Hamming distance or any other).

An extreme case of similarity control is when only clones (all the variables/bits are the same) are eliminated.

4.5 Constraint handling

4.5.1 Side constraints

Side constrains are included naturally in the encoding when the domain intervals for each design variable are defined.

4.5.2 Penalty

The same methods that are used in classical methods to handle constraints can also be used in GA's(See section 3.2). Beyond these some other can also be used.

Feasibility based penalty

In this kind of penalty any feasible solution has a better fitness than one that isn't this can be easily achieved by adding a penalty to the unfeasible solutions with the same numerical value as the worst feasible solution in the populations.

4.5.3 Repair

Individuals outside the feasible space are very likely to have a terrible fitness making it improbable they will be selected to crossover and/or survive until the next generation. So, across the years, many attempts have been done to recover part of the information of such solutions.

The repair can be partial, if the constraint violations are reduced, or total, if the constraints are completely eliminated.

We can take a Lamarckian approach, if we correct the genome, or a Baldwiniam approach if we only change the fitness.

4.5.4 Lagrangian Method

The Lagrangian Method can be used together with GAs. Adeli and Cheng [37] propose a schedule were the mean constraint violation after a fixed amount of generations is used to update the multipliers (in a similar fashion of what was discussed in 3.2.3). Their algorithm can be interpreted as a Lagrangian Method that uses a GA as the minimizer.

This method has the advantage of having meaningful penalties and not requiring the setting of penalty parameters but has the disadvantage of requiring parameters to update the Lagrange Multipliers. If the update schedule is too *aggressive* the algorithm won't be able to converge because the fitness landscape will constantly change, if on the other hand the update schedule is to *soft* the search will be too slow.

Chapter 5

Memetic Algorithms

Most of what in unusual about man can be summed up in one word:"culture".

- Richard D.

5.1 Prisoners Dilemma

Imagine that two people,A and B, are arrested. They are being held in custody and interrogated in two different rooms with no way of communicating with each other. They are offered the same conditions: if no one speaks (the prisoners **cooperate** with each other) then they both serve 2 years in prison, if one of them **defects** then he is free (the best reward) and the others serves 10 years in prison(the worst outcome), if both speak then they will both serve 5 years.

The rules are set, then, the question is "What is the best move?"

To answer this question we will take A's point of view, the answer is the same regardless of the prisoner we choose as they were given the same conditions.

If B doesn't speak than our 2 options are:

- remain silent and serve 2 years
- defect and be free

So if B remains silent our best move is to defect.

If B defects our 2 options are:

- remain silent and serve 10 years
- defect and serve 5 years

So regardless of B's move our best move is to defect.

		What the other prisoner does: Cooperate	What the other prisoner does: Defect
What I do	Cooperate	**Good Outcome** (We both serve 2 years)	**Worst Outcome** (I serve 10 years and the other prisoner is free)
	Defect	**Best Outcome** (I'm free and the other prisoner serves 10 years)	**Bad Outcome** (We both serve 5 years)

Figure 5.1: Prisoner's Dilemma Payoffs

Therefore the best move is for both prisoners to defect and serve 5 years. This comes as a paradox, because if they both remained silent they would only serve 2 years, a better outcome for both.

This is the standard Prisoner's Dilemma, but there is variation of this problem: the Iterated or Repeated Prisoner's Dilemma.

In this second version the players play the game an indefinite amount of times.

Axelrod[38, 39] showed experimentally that the best strategy in this situation would be one of "Tit for Tat", a strategy that reciprocates whatever the other players last move was. This kind of strategy encourages other players to cooperate and penalizes the ones that defect.

A parallel can be drawn with our optimization algorithms, where a strategy that includes not only competition but also cooperation should yield better results.

5.2 No free lunch theorem

Any two [optimization] algorithms are equivalent
when their performance is averaged over all possible problems.

This theorem was deduced by Wolpert and Macready in 1997[1]. In a simple manner, we cannot say that a single search heuristic is better than any other(such as random sampling) when they are given the same information. We already had a hint of this in 4.4.1 when we saw that changing our representation doesn't ensure a better outcome.

Two reactions can arise when reading the NFL[40]:

1. Some researchers argue that the NFL concerns to the set of all possible discrete problems but not to real-world problems. Others point out that the set of all possible discrete functions is infinitely large and most functions are incompressible in the sense that there is not a representation whose size is significantly less than the size of the function when fully enumerated. This is the case for two reasons:

   - There are more random functions than non-random functions.
   - The set of all possible programs is countable infinite while the set of all possible cost functions is uncountable infinite. Therefore the set of possible functions that can be implemented in a program is much smaller than the set of all possible cost functions.

2. The other reaction is to accept the NFL, and that there isn't an efficient black-box solution for all problems. Instead problem specific knowledge must be used to improve the performance when solving such problem.

Both points of view hold some truth.

One one hand, the NFL looks at search as a blind process, that is, the only information we get when evaluating a point is the value of the function at that point. In fact, we can only be sure that we reached the global optimum after evaluating all the possible solutions.

On the other hand search is often not a blind process, we do not sample a function randomly until we find a good enough solution. When we build an algorithm to solve a particular type of problem the operators and encoding used take that problem into account.

In engineering we are concerned about real-world problems. In engineering problems we can make some assumptions such as non-randomness and the presence of some kind of continuity, i.e, we expect that two points close by each other in space have a similar cost. Plus, if we include whatever knowledge we have about that problem we can simplify the search procedure.

In summary, instead of stifling the search for better algorithms this theorem showed that if we wanted to improve the optimization process we must find more ways to include information about the problem in the process.

5.3 Double optimization problem

Anyone that has ever tried to use an optimization algorithm has faced at least one (hyper) parameter that affects the convergence of the algorithm but can't be calculated *a priori* as a consequence of the NFL. The choice of this parameter(s) is usually based on past experience or, in the case of an inexperienced designer, some kind of average value.

In Machine Learning this is called Hyperparameter Optimization[41]. In this area many steps have been taken in order to use data/knowledge from previous learning runs to improve the speed and robustness of future runs.

Methods for Hyperparameter Optimization include[42]:

- Grid Search
- Random search
- Bayesian optimization
- Gradient-based optimization
- Evolutionary optimization
- Population-based training
- Early stopping-based training

5.4 Cultural Evolution

Killer whales (Orcinus Orca) of Patagonia, and their behavior of intentional stranding while hunting nearshore[43]

The killer whales of Punta Norte, Penisula Valdes, Argentina have developed a unique behaviour.

Typically, the individual preparing to beach itself swam purposefully towards the shore, actively guiding itself towards the prey, occasionally riding on an advancing wave. At times, other whales congregated beside the beaching whale, potentially to prevent the prey from escaping in their direction. The whale seized the prey within the tumultuous surf zone or within the forward motion of a wave on the beach, often while the prey was still in the water. Due to the killer whale's substantial size in comparison to the prey, the killer whale would become "beached" or grounded with water still flowing around it, and the prey still capable of swimming. Once grounded, the whale contorted its body, raising the head and tail, and swayed laterally. This movement typically aligned it parallel to the beach, with subsequent waves aiding in lifting it off the seabed. It would then swim back into deeper waters, carrying the prey in its mouth if it had successfully captured it.

Sea lions and elephant seals are uncoordinated on land and struggle during the transition from swimming to walking in shallow, particularly turbulent waters. This hunting strategy takes advantage of this weakness.

This is a great example of cultural evolution within Nature where the hunting strategy is the meme. This behaviour isn't intrinsic to the genetic material, but is allowed by it, that is, killer whales are some of the smartest beings in the animal kingdom granting them the ability to learn new hunting strategies.

Juvenile orcas mimic adults, stranding themselves but often missing their target. As they get older they refine their hunting skills, becoming more effective.

It's reasonable to assume the first killer whales that tried this approach weren't very effective either and over time their procedure was improved. It is also reasonable to assume that if the juvenile whales didn't have adult examples to teach them they either wouldn't use this strategy or would have to develop it from scratch.

5.5 Foundations

5.5.1 Memes

The term *meme* was first coined by Richard Dawkins in his book "The Selfish Gene"[44]. It meant "a unit of cultural transmission, or a unit of *imitation*", drawing inspiration from the Greek word *mīmēma* (μιμημα) which means "imitated thing".

Dawkins gives examples of memes as "tunes, ideas, catch-phrases,clothes fashions, ways of making pots or of building arches. "He draws a parallelism between cultural evolution and genetic evolution, showing that this kind of "ideas" also undertake evolution, adapting to the region they are inserted and the passage of time.

5.5.2 Classification of Memetic Algorithms

Hyper-heuristics are the way we decide which low-level heuristics to use to find the optimum solution, i.e, they try to improve the performance of the algorithm itself rather than improving the solutions.

MAs can be interpreted as evolutionary algorithms that includes in the evolutionary cycle of crossover-mutation-selection a local search stage.

Multimeme Memetic Algorithms (MMAs) distinguish themselves by the use of multiple local searchers.

AMAs use multiple memes in the search which are chosen at the same time as the search.

Algorithm 7 Generic AMA

```
Generate an initial GA population
while Stoping criteria not met do
    Evaluate all individuals
    for Each individual do
        Select memes
        Local search
    end forApply regular GA operators - crossover, mutation, selection
end while
```

Ong *et al.* [45] classified AMAs based on their taxonomy:

1. Hyperheuristic Adaptive Memetic Algorithm (HAMA) Multiple memes are considered during the search process. Cowling *et al.* [46] subdivide HAMAs into:

   (a) random - the memes used in local search are chosen at random and the probability is kept constant during the whole search process.

   (b) greedy - every memes is tried during local search and the one that produces the biggest improvement is chosen.

   (c) choice-function - the memes are chosen based on their previous performance during the search. The probability is calculated with a choice function that uses goodness metrics that access how well the meme is doing. Usually these choice functions have three components: f_1 indicates the recent performance of the meme; f_2 relates to the improvement brought about by the subsequent pairings of memes; f_3 indicates the time since the meme was used.

2. Multimemes and Co-Evolving MAs As proposed by Krasnogor, each individual carries genetic and memetic material. The genetic material relates to the design solution and the memetic material indicates what meme to use during local search.

3. Meta-Lamarckian Learning The meme used in local search is chosen based on past experiences when searching in that region. This way the memes work together by improving the solutions in the regions they are the most effective.

They also propose a classification based on the adaptation type:

1. External - The knowledge obtained during the search process is not used to chose the meme, i.e, there is no online knowledge about what memes are the most effective. Instead the memes are chosen *a priori* by user based on previous experience when solving similar problems.

2. Local - These AMA use part of the historical knowledge (usually the most recent one) to decide which meme to use.

3. Global - They take the entire historical knowledge into consideration.

Meuth et al.[47] propose a generational classification of MAs, see Table 5.1. The authors did so because they identified that MAs followed a trend of increasing complexity, and in the end they propose a new generation that included more knowledge about the problem than all the previous generations.

Table 5.1: Generation description of MAs

Classes	**Characteristics**	**Examples**
1^{st} generation	Global search combined with local search.	Canonical MA Adaptive global/local search Evolutionary search with gradient information
2^{nd} generation	Global search with multiple local optimizers. The memetic information used in the choice of the optimizer is passed down to the offspring.	Hyper-heuristic MA Meta-Lamarckian MA Multimeme MA
3^{rd} generation	All of the above. Mapping between the evolutionary trajectory and the choice of the local optimizer is learned.	Co-evolution MA Self-generation MA
4^{th} generation	Mechanisms of recognition, generalization, optimization and memory.	Unknown

5.5.3 Local Search

In this work the genetic algorithm will be hybridized with local search. Hybridizations of GAs and LS operators were the first attempt of MAs pioneers. Their objective was to refine decent solutions and nudge the search into better regions. Usually, the operator where chosen/created based on the landscape of the problem.

What is local search?

Local search, or hill-climbing, refers to the group of methods that are used together with evolutionary algorithms to perform an intensification of the search in a set of promising solutions.[1]

The search can be exhaustive or random in the neighborhood (we will talk about this concept later) or directed based on some kind of information about the problem (restriction violation, gradients, etc.).

Why use local search?

Pure GAs are very good at identifying promising regions and dealing with multimodal landscapes but struggle to refine good solutions and reach the optimum solution at a high precision. That's were local search methods excel.

Local search methods are often based on classical methods that make assumptions about the convexity and unimodality of the objective function which are not true for real world problems. This problem can be circumvented if we restrict ourselves to a small neighborhood.

How do *I* apply local search?

When performing local search the first thing to define is where to look. [When blindly searching for the tallest building you would expect it to be close to other tall buildings, let's say 100m away, and not 10km away]. This leads to the definition of the first parameter of local search: **the neighborhood function**.

Typically defined as a n-ball or a n-cube, the radius of the neighborhood depends on the problem and the phase of search leading the parameter to be a perfect case of value that can be controlled by a meme. If you are close to the optimum solution you need smaller steps to refine your solution, on the other hand, if you are in an initial phase of the search you can afford to take bigger steps in order to push your solution to a more favourable search zone.

[1] Typically, local search methods minimize the function while we are trying to maximize the fitness function, this can be easily circumvented by passing in the symmetric of the function (and it's respective derivatives) to the local search methods.

The other parameter to define is how many points do we want to evaluate. If we search all the points we would be spending a lot of CPU time searching a zone that might not be the most promising. Therefore we must choose, *a priori* , **the depth**. The depth of search will also increase the exploitation and, if it's done on multiple points, tend to lead them to the same region decreasing exploration.

There is another important concept about local search, **the pivot rule**. This rule defines the criteria to accept a new point. *Is it enough to simply find a better solution or must we find the best?* If we simply accept a better solution, *how much better should it be?*

These three concept are intertwined: if we decide to find the best solution in the neighborhood we need a deeper search but in that case the search might be unfeasible for a bigger neighborhood.

Besides these three concepts there is another question that remains to answer. *What individuals should be improved?* Houck *et al.*[48] have shown that we shouldn't apply local search to all individuals, instead we should choose a small, random fraction. In the case of their paper the operator was applied to the offspring.

Lamarckian Learning and the Baldwin Effect

Two approaches can be taken when performing local search. Either the *improved* version replaces the original one or only the fitness is affected. To the former we refer as Lamarckism and to the latter we refer as Baldwinism (Baldwin Effect).

In practice as been shown that a Lamarckian approach can be faster, specially at earlier stages of evolution, but has a propensity to fall to local optima while a Baldwinian approach can be much more stable despite being slower at earlier stages[49]. Recent studies have a tendency to take pure Lamarckian approach or a partial Lamarckian approach where only a fraction of the population undertakes local search.

In a Baldwinian approach individuals that exist in a promising region, i.e, have a high innate fitness, can more easily have their lifetime fitness improved making it more likely that they will survive and be selected for crossover, and thus spread the genes that have the potential to improve. This will drive the search to those regions.

The Baldwin effect has been demonstrated to happen mathematically and in Nature. The notion of Baldwin effect is intertwined with the notion of phenotypic plasticity.

In Nature, the phenotype of an individual depends on other factors beyond it's genotype, such as the environment, random events, learning, adaptation, etc. How much the phenotype is affected by this external factor is defined as **phenotypic plasticity**.

We can say that plasticity changes the way genotype maps into the phenotype. Therefore we can conclude that a change on plasticity may induce a change in this mapping. This change may magnify the effect of the genotype in the phenotype, thus increasing the evolutionary pressure, or do the opposite.

Baldwin argued that learning could "guide" evolution. If a change in the environment were to occur, individuals with the potential to learn and adapt to the new environment could do so within their lifetimes, becoming fitter than those who weren't able to adapt. This would cause an evolutionary pressure towards individuals that can more easily adapt to such situation. This means that eventually, the only individuals to survive (or a big part of them) would be the ones that had the genes that allowed them to adapt to that environment[50, 51].

Baldwin hypothesised that this process would lead to a new behaviour faster than natural selection based only in genetic material.

Mery and Kawecki [52] showed that learning can accelerate or slow down evolution. They did so by selecting which medium fruit flies were allowed to lay their eggs, and comparing the results when they were allowed to learn and when they weren't. They concluded that learning can have any of the two effects depending on the medium.

Paenke, Sendhoff and Kawecki [53] developed a general framework to study the effect of plasticity and other similar effect in the speed of evolution under directional selection.

When discussing the Baldwin effect we should keep in our heads four metrics:

1. lifetime fitness - fitness calculated after the learning process
2. innate fitness - potential fitness before learning
3. phenotypic plasticity - proportion of plastic phenotypes in the population
4. phenotypic variation - absolute difference in phenotypic values before and after the learning process

Suziki and Arita [54] show how the Baldwin Effect affects the evolutionary process. In the paper they describe the "Three-Step Evolution through the Baldwin Effect" as follows:

- Step 1 - There is an increase in lifetime fitness and phenotypic plasticity, caused by the benefit of learning.
- Step 2 - Decrease in plasticity, most likely due to the implicit cost of learning caused epistatic relations.[2]

[2]Epistatesis - When the effect of a gene depends on the presence/absence of other genes.

- Step 3 - Genetic assimilation in the remaining phenotypes.

Table 5.2: Three step evolution through the Baldwin effect characterized by four the indices[54].

Step	**Lifetime fitness**	**Innate fitness**	**Phenotypic plasticity**	**Phenotypic variation**
1^{st}	Increasing	Steady	Increasing	Steady
2^{nd}	Increasing	Steady	Decreasing	Increasing
3^{rd}	Increasing slowly	Increasing	Steady	Decreasing

Table 5.3: The benefits and costs of learning that caused three-step evolution through the Baldwin effect[54].

Step	Benefit of learning	Cost of learning
1^{st}	Search for plastic individuals in every direction by adjusting many phenotypic values	-
2^{nd}	Directional and long-distance search by adjusting a small number of phenotypic values	Epistatic interactions among plastic phenotypes and limitation of learning ability
3^{rd}	Genetic robustness against mutations	Limitation of learning ability
Equilibrium	Genetic robustness against mutations	-

In the first stages individuals that aren't innately fit and can't adapt are eliminated and the evolutionary pressure occurs mostly in the direction of creating individuals that can adapt to the new situation because that is *cheaper* than evolution towards individuals that are innately fit. As the the average fitness of the population rises the evolutionary pressure shifts towards individuals that are innately fit in detriment of individuals that have the ability to learn because of the innate cost of learning.

When a population reaches it's maximum potential the individuals still keep some ability to learn that ensures that any deviation of the optimum can be corrected.

5.5.4 Preserving Diversity

A crucial point about applying GAs is to keep diversity, and the same applies to MAs.

If local search is applied to all individuals in an exhaustive manner they will all converge to local optima leading to a loss in diversity.

Additionally, even if the local search is stopped before arriving at the exact local optima, if it was applied to individuals present in a big bay of attraction, the process may also converge too early to a sub-optimal solution. To circumvent this problem a few strategies can be undertaken:

1. when creating an initial population, include only a small percentage of known individuals with good fitness
2. apply local search only to a small fraction of the population
3. use crossover operator built to preserve diversity, for example take one parent from the worst performing individuals already present in the population
4. use multiple local search operators, each one with a different search space with different local optima
5. modify the selection operator to prevent duplication of individuals
6. use a criteria that controls diversity explicitly as a pivot rule, for example a LS operator than stops if the solution generated is too similar to a solution already present in the population
7. modify the selection operator and the stooping criteria of local search based on Boltzmann's method in order to preserve diversity

5.5.5 Choosing Candidate Operators

We have talked a lot about the importance of the operator chosen how it can significantly affect the search speed, how it can help or disrupt evolution, but we haven't discussed how.

It has been shown both formally and empirically that different operators should be chosen in the local and global search[55, 56]. Doing so ensures a distinct search at both search levels, this leads to a more robust algorithm.

By using multiple different operators each one of them introduces a new search area , with it's own local minima, this helps the algorithm avoiding getting stuck in a local minimum.

At this point we should remenber what was discussed in 4.4.3.

The way the operator is chosen may depend on how elaborate the MA is. If we take it in it's the most basic form we might just choose *a priori* a local search algorithm with a different *move operator*[3] than global search. If on the other hand we use a Multimeme Algorithm the fact that we are using different local search operators in parallel might be enough to avoid the problems of having the same move operator in the two phases of the algorithm.

As described in section 5.5.2, the probability each operator is chosen may be static along all the search process, change according to some predefined schedule/function of the search state or even be tied to evolution as a meme.

5.5.6 Hybridization and use of previous knowledge

The only way to get a free lunch is to bring your own.

When designing a MA for a particular problem one should take in consideration previous experience when solving similar problems. This is one of the most important aspects of MAs.

As we have discussed previously if we try to build an *black-box* algorithm capable of solving every optimization problem quickly we will fail, but if we have specific knowledge about the problem: symmetry, derivatives, previous known solutions obtained by experience, efficient operators, particular ways to deal with constraints,etc. we can "cheat" the NFL by giving that knowledge to our algorithm that wouldn't be available to a *black-box* algorithm.

The inclusion of known good solutions in the initial population has it's benefits:

- It avoids *reinventing the wheel* by avoiding terrible solutions and replacing them with decent ones, reducing computational power that would be wasted in those individuals.
- By generating the population in a non random process we can direct the search to the most promising regions, accelerating the convergence, thus increasing the efficacy of the algorithm or the quality of the final solution.

In short, for a given computational effort, the combination of an initialization heuristic with the evolutionary search can provide better results than a pure evolutionary search.

[3]Move operator refers to the step of generating a point or set of points from another. By type move operator we mean what points may be generated. Each type of move operator has has it's own definition of neighbourhood(for example Hamming space vs Euclidean space). By using diferente types of move operator we can search diferente regions.

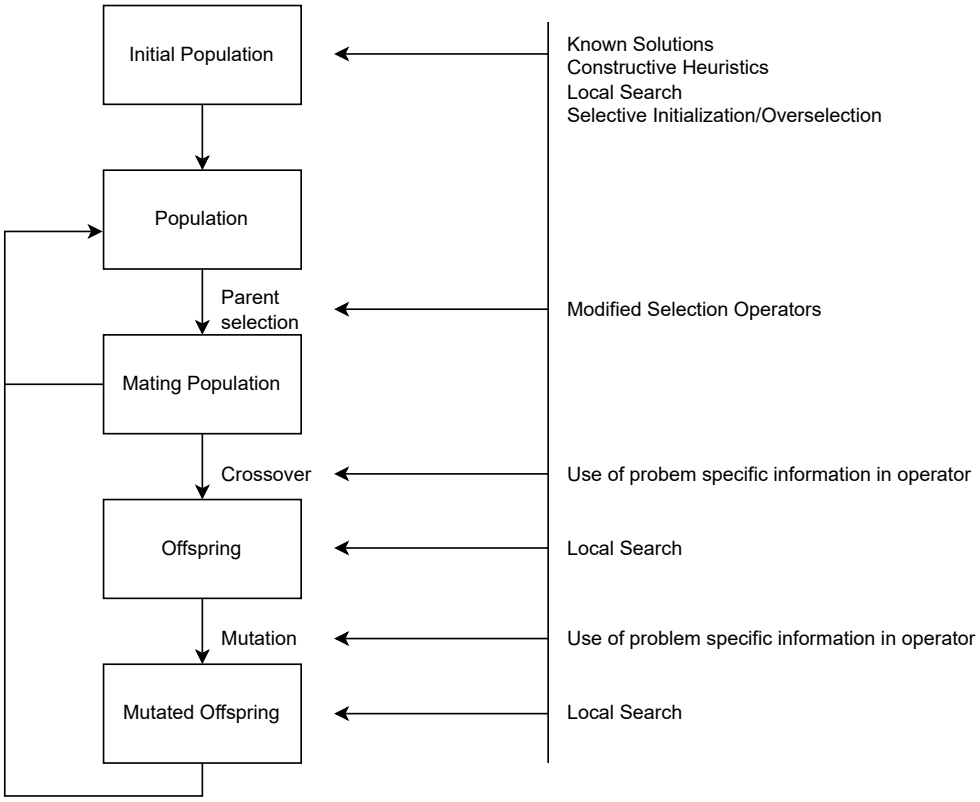

Figure 5.2: Possible ways to incorporate knowledge or other operators

Some alternatives to initialize the population are:

- Introducing seeds in the population that relate to good solutions previously known or obtained from other techniques
- Creating a big set of random solutions and make a selection from these, based in the fitness and/or other metrics such as diversity
- Applying a local search operator to each one of the initial solutions
- Using one or more of the described above and select one or more good solutions. Then clone and mutate them.

Once again we must pay attention to the importance of diversity, if the solutions produced by the methods above are too similar all that computational power might have been wasted.

Ideally at the end of the search procedure the MA would give information about the most successful memes so they can be reused in future runs of similar problems. This is yet to be widely applied in MAs. Figure 5.3 shows the intended flow of information.

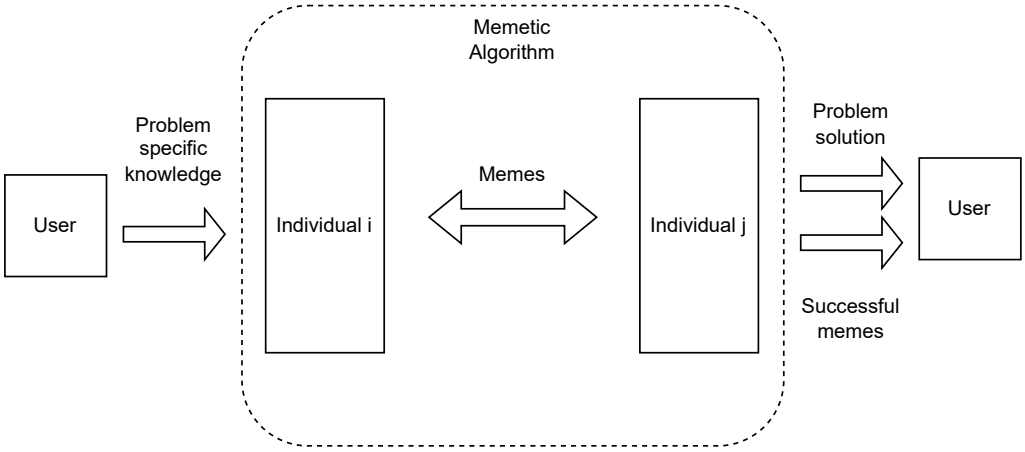

Figure 5.3: Information flow

5.5.7 Controlled crossover and mutation

Controlled crossover and mutation are a way of including knowledge about the problem. This is the one of best ways for the user to put his expertise about the problem in action.

Multiple authors have proposed and incorporated such mechanisms[57, 58].

Gutkowski , Iwanow and Bauer [59], in the minimization of the weight of a truss structure, propose a mutation operator that takes into account the cross section area and the constraint violation to decide the direction and in which member should the mutation be made.

5.6 Local search operators

When performing LS different operators can be applied. We are not restricted to Classical Methods or using geometric distance to establish the neighborhood, we could instead use a 0 Order method that uses the Hamming Distance to establish the neighborhood.

Classical Methods

If the objective function allows so, we can use classical methods in the local search. As was said previously we shouldn't spend to much CPU time performing local search leading to Order 1 classical methods (Pure Gradient and Conjugated Gradient) or Quasi-Newton methods to be the most common methods when performing this kind of search. Some researchers have used Newton's Method and Sequential Quadratic Programming[22].

Bio-inspired algorithms

We can also apply evolutionary and swarm intelligence based algorithms at the local search level. The initial population of this level can be a part of the population form the global search or generated randomly from a solution. It must be emphasized again that the operator used at the local search must be different than the ones used at the global level[55].

0 Order Methods

This methods sample random points in the neighborhood of an individual and accept/reject them given an heuristic. Some examples are Random Walk, Simmulated Annealing, Licthenberg Algorithm, Hill Climbing and the Hooke-Jeeves Method.

5.7 Selection Hyper-heuristics

Naturally when using an AMA one of the first questions to arise is *How do I decide what heuristics to use at each moment?*. Hyper heuristics are the way we decide what low-level heuristic we choose.

Recently, Drake, Kheiri and Özca[60] compiled and classified selection hyper-heuristics based on multiple characteristics:

- Feedback
- Low-Level Heuristics
- Solutions
- Objectives
- Move acceptance
- Parameter Setting

5.7.1 Feedback

When discussing the classification of MAs in 5.5.2 we have talked a little bit of the nature of feedback.

Simple operators, metaheuristics or even simple hyper-heuristics might be considered low-level heuristics when designing an hyper-heuristic approach[60]. They were initially categorized according to the type of search-related feedback that was given.

In order to process input throughout the search process selection hyper-heuristics integrate either no learning mechanism or online learning mechanisms influencing subsequent judgments made at the hyper-heuristic level.

Additionally, there are hyper-heuristics for offline learning that are frequently applied to generational problems. This kind of method receives feedback before the search in order to develop new heuristics that are applicable to previously unknown problem situations. The method is trained on sample problem instances.

5.7.2 Low-level Heuristics

When discussing analyzing the low-level heuristics we can further divide our classification into the type/structure of low-level heuristics, the kind of set and how the heuristics are grouped chosen and applied.

Search structure

Two categories of low-level heuristics inside selection hyper-heuristics can be distinguished based on the search structure used: construction and perturbation heuristics[60].

Constructive heuristics start with nothing and work their way up to a complete solution by choosing which of a number of low-level heuristics to use at each stage.

Perturbation heuristics execute local search operations utilizing pre-defined neighborhood structures on complete solutions.

Low-level heuristic set

When building a MA the designer must establish a finite set of low-level heuristics: different types of mutation, crossover, local search, etc.

The designer must then select which low-level heuristics the hyper-heuristic can manage.

The hyper-heuristic can operate with the entire set of low-level heuristics, a reduced set or even an augmented set via hybridization of different low-level heuristics.

How heuristics are grouped, chosen and applied

A common way to approach is to select and apply low-level heuristics without grouping them. Other approaches group them and select them based on this grouping.

Other approaches group them and select them based on this grouping. One possibility is to fix them and then applied them according to a predefined sequence. To select the heuristic types used during the search, one may employ one or more selection hyper-heuristics.

Another alternative is to use hyper-heuristics that operate in a stage-based manner, selecting a group of low-level heuristics to apply at each stage, either with an fixed or an adaptive schedule[60].

5.7.3 Solutions and Objectives

Hyper-heuristics can use a single solution, multiple solutions or a mixed approach in a phased manner.

The hyper-heuristics can have one objective or multiple objectives(diversity preservation, expected fitness gain, etc.)

5.7.4 Move acceptance

The move acceptance can be stochastic, if there is a probability of accepting/rejecting bad/good moves based on a probabilistic framework, or non-stochastic. The latter can be further divided into accepting all moves, accepting equal or improving moves, accepting only improving moves and accepting moves based on a threshold.

5.7.5 Parameter setting

Both the overall search algorithm and the the low-level heuristics require parameters. Durillo et al.[61] divide parameter control into three groups: deterministic, adaptive and self-adaptive.

The parameter can be defined by the user at the beginning of the search process, dynamically(following a predefined rule) or adaptively(reacting to the search state). The latter can be stated as search of the optimum solution and the optimal parameter, i.e, **solving the double optimization problem**.

Chapter 6

Implementation of a Memetic Algorithm

Like the early MAs, a GA will be taken as a base algorithm and we will try improve upon it.

6.1 Genetic Algorithm

The base algorithm is inspired by the GA proposed by António, C.[62]It has a strong elitist component coupled with a similarity control, this characteristics make it so it can converge to a good solution even with small populations, while at the same time avoiding early convergence to a local minimum. It has been tested and validated against multiple optimization problems.

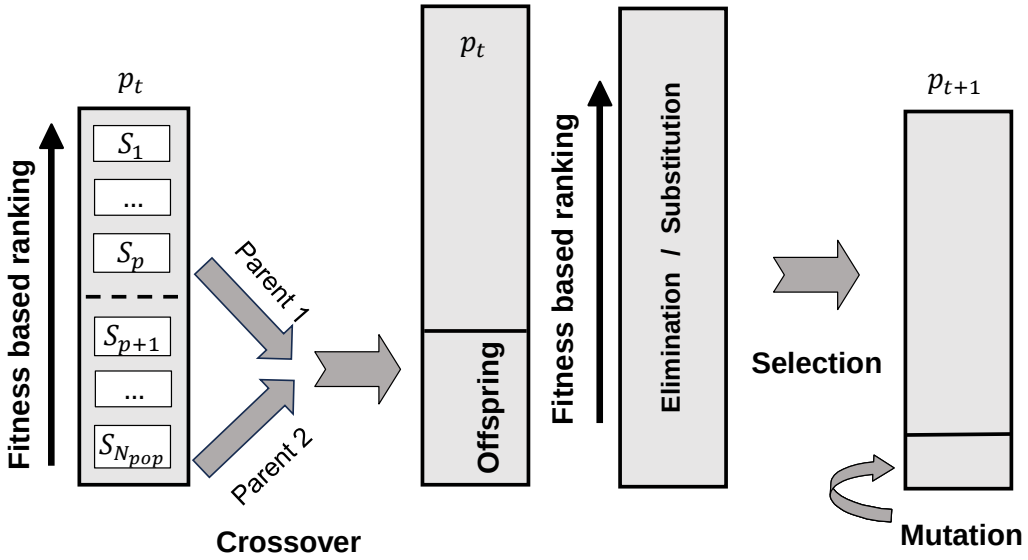

Figure 6.1: GA implemented. Adapted from [62].

The operators applied are as follows:

Step 1 Initialization

The initial population is randomly generated.

Step 2 Selection

The population is ranked based on a fitness function. The elite group is a fraction of the population with the most fit individuals. Pairs of parents from the elite group and the remaining of the population are selected for the next step. The probability any individual being selected(inside each group) is proportional to the fitness.

Step 3 Crossover

An offspring is generated for each pair of parents. To generate the genetic material a parameterized uniform crossover operator is used. The operator has a higher probability of selecting genetic material from the best parent with a given bias[62].

Step 4 Elimination & Replacement

The offspring group is added to the population generating a new enlarged population. This new population is ranked based on the fitness function. Clones are replaced by randomly generated individuals and the solutions are ranked once again. The worst solutions are discarded in order to recover the initial population size.

Step 5 Mutation

A small fraction of the worst individuals are replaced by randomly generated ones(implicit mutation).

Step 6

The stopping criteria used is the maximum number of generations. This can be justified by the fact that in practical problems users doesn't have a clear idea of what the optimum solution looks like or how much time it takes to achieve, instead they are limited to how much time(or money) we can dedicate to the search.

The parameter used in all examples the algorithm were:[1]

Table 6.1: Base GA's parameters

Parameter	Value
Population Size	30
Offspring Size	18
Crossover bias	0.7
Nr. of individuals in the elite	10
Nr. of individuals mutated	2
Maximum generations	20,000
Nr. of bit per variable	8

6.2 Memetic model

The crossover and mutation operator will be acted upon and a local search operator will be implemented. In the following sections the general ideas behind the specific operators developed will be discussed.[2]

6.2.1 Local Optimizer

In problems that allow so, local search using gradient based methods can be quite powerful.

As a generic algorithm, the local search was implemented as follows:

Algorithm 8 Local Optimizer

Select an individual($\boldsymbol{x}_0$). $k \leftarrow 0$.

repeat

Compute $\boldsymbol{d_k}$ using a **Gradient Based Method**.

Compute $\boldsymbol{x}_{k+1}$

$k \leftarrow k + 1$

until $||\boldsymbol{x}_0 - \boldsymbol{x}_k|| \geq \varepsilon$ or $k = k_{max}$

The **neighbourhood function**($||\boldsymbol{x}_0 - \boldsymbol{x}_k|| \geq \varepsilon$) and the **search depth**($k = k_{max}$) are included in the stopping criteria. We should notice that $\boldsymbol{x}_k$ is accepted even if it falls outside of the radius ε.

[1] One could do many a study to find the best parameters for a given problem by testing them across multiple seeds, but this has very little practical use because any problem has it's own optimal parameters. Instead, parameters that have shown to perform well across multiple problem should be used, and a bigger emphasis should be given to problem specific operators.

[2] At this point might be valuable to go back to section 5.5.6 and review where information can be included.

The computation of $\boldsymbol{x}_{k+1}$ was done using the following algorithm:

Algorithm 9 Computing $\boldsymbol{x}_{k+1}$

Given $\delta \in (0,1).\gamma \in (0,0.5), c \in (0,1), \boldsymbol{x}_k$ and $\boldsymbol{d}_k$
Compute an initial α_k such that: $\alpha_k \geq c\frac{\nabla f(\boldsymbol{x}_k) \cdot \boldsymbol{d}_k}{||\boldsymbol{d}_k||^2}$
while $\boldsymbol{x}_k + \alpha_k \cdot \boldsymbol{d}_k \notin \Omega_{Side}$ **do**
    $\alpha_k \leftarrow \delta \cdot \alpha_k$
end while
while $f(\boldsymbol{x}_k + \alpha_k \cdot \boldsymbol{d}_k) > f(\boldsymbol{x}_k) + \gamma\alpha_k \nabla f(\boldsymbol{x}_k) \cdot \boldsymbol{d}_k$ **do**
    $\alpha_k \leftarrow \delta \cdot \alpha_k$
end while
$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \alpha_k \cdot \boldsymbol{d}_k$

The previous algorithm can be summarized to an Armijo line search coupled with a condition to make sure the point never violates the side constraints.

When derivatives aren't available it's still possible to apply Local Search using 0^{th} Order Methods.

The 0^{th} Order Method applied was a Hookes-Jeeves method (see Algorithm 10) in a similar way to how Moser and Chiong did[63].

Notice that the last successful search direction is preserved across runs of the local optimizer, therefore there is cooperation between individuals that undergo local search.

The step size is chosen in such a way that the point generated always falls in the grid.

After performing local search using the local optimizer we have to look for the closest point in the grid (the local search is done on a continuous domain while the global search is done on a discrete domain).

The new individual is accepted if it's fitness is is greater or equal to the original individual - **pivot rule**.

The individual can be selected from various groups:

- the offspring
- individuals generated by mutation operators
- the worst in the elite

Algorithm 10 Hookes-Jeeves Memetic Algorithm

Step 1 Select an individual from the population

Step 2 Set the base point as the selected individual

if There is a saved last successful sample direction **then**

Set the search direction as the last successful direction.

else

Set the search direction as a random direction.

end if

Step 3 Move the base point across the search direction.

if The trial point is inside the search space and isn't already in the population **then**

Step 4 Evaluate the trial point.

if The trial point is better than the base point **then**

Step 5 Set trial point as the base point, save the current direction as the last successful direction and go to **Step 3**

else

Go to **Step 5**

end if

else

Step 5 Move in the opposite direction.

if The trial point is inside the search space and isn't already in the population **then**

Step 6 Evaluate the trial point.

if The trial point is better than the base point **then**

Step 7 Set trial point as the base point, save the current direction as the last successful direction and go to **Step 3**

end if

end if

end if

if The stopping criteria haven't been met **then**

if All directions have been depleated for the current step size **then**

Reduce the step size and go to **Step 3**.

else

Set the search direction as random direction that hasn't been tried and go to **Step 3**.

end if

else

Stop

end if

- the best in the elite
- the best outside the elite
- the entire population

from these we can improve all or a fraction.

In this thesis LS will be applied to a small amount of solutions, the three best individuals outside the elite, this way it is intended that it won't drive the search, instead it will nudge it into a slightly better regions.

6.2.2 Controlled Mutation

Controlled Mutation (CM) is one way to include knowledge about the problem in the search process.

In mechanical engineering problems it's expect that increasing the size/thickness/diameter of a part it's stiffness will increase, therefore decreasing displacements, strains and stresses. So, when we act in the direction of reducing the violation of one constraint we also expect a decrease on the violation of the others that depend on the same state variables.

On the other hand, when no stress or displacement constraint is violated we can afford to reduce the stiffness of the part by reducing it's dimensions, thus reducing the cost.

Algorithm 11 Controlled Mutation Algorithm

Select a random individual, u, from the elite group.
Rank the constraints based on their values by decreasing order. $\bar{g}(u)$ is the most violated and $\underline{g}(u)$ is the least.
Identify variables that affect the $\bar{g}(u)$, x_{sup}, and $\underline{g}(u)$, x_{inf}.
if $\bar{g}(u) > 0$ **then**
    Mutate u in x_{sup} in the direction the reduces $\bar{g}(u)$ according to the Design Rules.
else
    Do not implement mutation in x_{sup}.
end if
if $\underline{g}(u) > 0$ **then**
    Do not implement mutation in x_{inf}.
else
    Mutate u in x_{inf} in the direction the reduces $f(u)$ according to the Design Rules.
end if

The Design Rules for Controlled Mutation come from a preliminary study of how each variable affects the constraints and the objective function. This can be done based on previous experience or, in the case of this work, based on the equations that govern the problem.

In the implementation done the bit-string relative to the each of the variables is affected by 1 bit in whatever direction it's intended to change that variable.

We have to take 4 cases into account: adding/subtracting and the bit in question has value 0/1.

The following schemes summarize what happens:

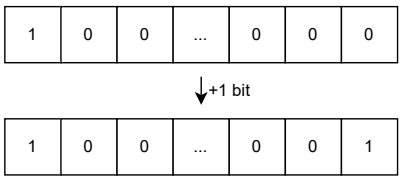

Figure 6.2: Add 1 bit when the last bit is 0

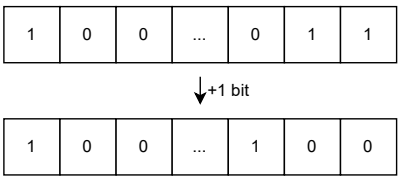

Figure 6.3: Add 1 bit when the last bit is 1

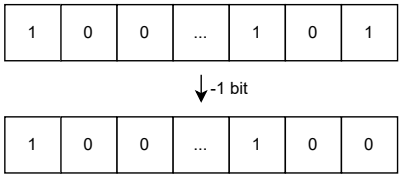

Figure 6.4: Subtract 1 bit when the last bit is 1

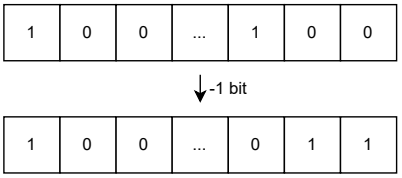

Figure 6.5: Subtract 1 bit when the last bit is 0

6.2.3 Hybrid Crossover

To improve the crossover operator we can allow it to do specific moves based on knowledge about the problem.

Typically, a crossover operator blindly generates a child(or a set of children) in the space generated by the parents. The objective of the hybrid crossover implemented in this work [57] is to approximate the region around the parents and apply a greedy, but quick, local search using that approximation.

The search is done via Random Walk in the Hamming space.

The operation can be summarized in **4** steps:

1. Select one parent from the elite group and one from the remaining group
2. Create n trial solutions inside the Hamming space generated by the two parent using a traditional crossover operator
3. Estimate the trial solution's fitness
4. Select the best candidate

Parent Selection

Ranking based fitness of the population

Divide the population into two groups: the elite group comprising of the n_{elite} best individuals and the rest grouping comprising of the remaining solutions.

Select a parent form each of the groups using an uniform distribution.

Hybrid crossover's scheme

The concept of Hamming distance[3] is used in the local optimization process to define the search space and the an approximated local fitness is defined to rank the offspring candidates.

The local fitness function, $\Phi(\boldsymbol{x})$, is defined as[57]:

$$\Phi(\boldsymbol{x}) = C(\boldsymbol{x}) + \beta \cdot \Psi(\bar{\Delta}_s) \tag{6.2.1}$$

where $C(\boldsymbol{x})$ is the cost calculated exactly and $\Psi(\bar{\Delta}_s)$ is a constraint related component that is calculated approximately using the Hamming distance between the candidate offspring and the parents.

The use of an exact component and an approximated one can be justified by the fact that, usually, the cost/weight of a structure doesn't need a model evaluation. On the other hand, the constraints usually require the calculation of stress states that are only obtainable with model evaluations.

[3] At this point might be convenient to remind ourselves of the definition distance in the Hamming Space present in 4.4.1.

The estimation of the constraint violation based on the distance between the candidate and the parents is given by the formula[57]:

$$\bar{\Delta}_s = (1 - \frac{d_1}{d_1 + d_2} \cdot \Delta_{p1} + \frac{d_1}{d_1 + d_2} \cdot \Delta_{p2}) \tag{6.2.2}$$

where Δ_{p1} and Δ_{p2} are the constraint violations associated with the parents p_1 and p_2 respectively. With these constraint estimation we can obtain the penalty component:

$$\Psi(\bar{\Delta}_s) = \begin{cases} 0 & \text{, if } \bar{\Delta}_s \leq 0 \\ K_s \cdot (\bar{\Delta}_s)^{q_s} & \text{, if } \bar{\Delta}_s > 0 \end{cases} \tag{6.2.3}$$

where K_s is problem dependent constant and $q_s = 2$

Chapter 7

Applications

7.1 Welded Beam

7.1.1 Problem description

The welded beam problem is a classic test problem for optimization algorithms, the objective is to minimize the cost of a welding subject to constraints on shear stress on the weld, bending stress on the beam, buckling load on the bar, maximum deflection and size constrains . It's very non-linear both in it's objective function and constraints.

7.1.2 Mathematical formulation

The cost of the welding is given by:

$$f(\boldsymbol{x}) = (C_1 + C_3) \cdot V_{weld} + C_2 \cdot V_{bar} \,[\$] \tag{7.1.1}$$

where:

$$V_{weld} = x_1^2 \cdot x_2 \,[in^3] \tag{7.1.2}$$

$$V_{bar} = x_3 \cdot x_4 \cdot (L + x_2) \,[in^3] \tag{7.1.3}$$

The objective function is subject to:

$$\begin{cases} g_1(\boldsymbol{x}) = \tau(\boldsymbol{x}) - \tau_{max} \leq 0 \\ g_2(\boldsymbol{x}) = \sigma(\boldsymbol{x}) - \sigma_{max} \leq 0 \\ g_3(\boldsymbol{x}) = x_1 - x_4 \leq 0 \\ g_4(\boldsymbol{x}) = C_1 \cdot x_1^2 + C_2 \cdot x_3 \cdot x_4 \cdot (L + x_2) - 5 \leq 0 \\ g_5(\boldsymbol{x}) = x_1 - 0.125 \leq 0 \\ g_6(\boldsymbol{x}) = \delta(\boldsymbol{x}) - \delta_{max} \leq 0 \\ g_7(\boldsymbol{x}) = P - P_c(\boldsymbol{x}) \leq 0 \\ g_8(\boldsymbol{x}) \text{ to } g_{11}(\boldsymbol{x}) : 0.1 \leq x_i \leq 2.0, i = 1, 4 \\ g_{12}(\boldsymbol{x}) \text{ to } g_{15}(\boldsymbol{x}) : 0.1 \leq x_i \leq 10.0, i = 2, 3 \end{cases} \tag{7.1.4}$$

where:

$$\tau(\boldsymbol{x}) = \sqrt{(\tau'(\boldsymbol{x}))^2 + 2 \cdot \tau'(\boldsymbol{x}) \cdot \tau''(\boldsymbol{x}) \cdot \frac{x_2}{2R} + (\tau''(\boldsymbol{x}))^2} \tag{7.1.5}$$

$$\tau'(\boldsymbol{x}) = \frac{P}{\sqrt{2} \cdot x_1 \cdot x_2} \tag{7.1.6}$$

$$\tau''(\boldsymbol{x}) = \frac{M(\boldsymbol{x}) \cdot R(\boldsymbol{x})}{J(\boldsymbol{x})} \tag{7.1.7}$$

$$M(\boldsymbol{x}) = P \cdot (L + \frac{x_2}{2}) \tag{7.1.8}$$

$$R(\boldsymbol{x}) = \sqrt{\frac{x_2^2}{4} + (\frac{x_1 + x_3}{2})^2} \tag{7.1.9}$$

$$J(\boldsymbol{x}) = 2(x_1 \cdot x_2 \cdot \sqrt{2} \cdot (\frac{x_2^2}{12} + (\frac{x_1 + x_3}{2})^2)) \tag{7.1.10}$$

$$\sigma(\boldsymbol{x}) = \frac{6 \cdot P \cdot L}{x_4 \cdot x_3^2} \tag{7.1.11}$$

$$\delta(\boldsymbol{x}) = \frac{4 \cdot P \cdot L^3}{E \cdot x_4 \cdot x_3^2} \tag{7.1.12}$$

$$P_c(\boldsymbol{x}) = \frac{4.013 \cdot E \cdot \sqrt{\frac{x_3^2 \cdot x_4^6}{36}}}{L^2} \cdot (1 - \frac{x_3}{2 \cdot L} \cdot \sqrt{\frac{E}{4 \cdot G}}) \tag{7.1.13}$$

and

$$E = 30 \cdot 10^6 \,\text{psi}$$

$$G = 12 \cdot 10^6 \,\text{psi}$$

$$L = 14 \,\text{in}$$

$$\tau_{max} = 13600 \,\text{psi}$$

$$\sigma_{max} = 30000 \,\text{psi}$$

$$\delta_{max} = 0.25 \,\text{in}$$

$$P = 6000 \,\text{lb}$$

$$C_1 = 0.10471 \,\$/\text{in}^3$$

$$C_2 = 0.04811 \,\$/\text{in}^3$$

$$C_3 = 1$$

It's well know that the solution to this problem is[64]:
$\boldsymbol{x} = (0.205729639770726, 3.47048866582864, 9.03662391069483, 0.205729639770726)$
$f(\boldsymbol{x}) = 1.72485230854216631$

7.1.3 Optimization model

It was decided to use a penalty function as mean to include the constraints in the optimization model:

$$fitness(\boldsymbol{x}) = C - (f(\boldsymbol{x}) + \lambda \Gamma(\boldsymbol{x})) \tag{7.1.14}$$

where $f(\boldsymbol{x})$ is the cost, $\Gamma(\boldsymbol{x})$ is a quadratic penalty function and C, λ are constants. After some tuning it was established that C, λ should be 10 and 100 respectively.

It is easy to understand that restrictions $g_8(\boldsymbol{x})$ to $g_{15}(\boldsymbol{x})$ are side constraints, leaving us with 7 constraints and 8 side constraints. We could even combine $g_5(\boldsymbol{x})$ and $g_8((\boldsymbol{x}))$ to get rid of one constraint.

Design Rules for Controlled Mutation

From the equations that govern the mathematical model we can derive the search direction we have to take to reduce the value of each restriction and the objective function.

The following table 7.1 summarizes the rules: 1 means that in order to reduce the value of the restriction we have to increase the value of the variable, -1 means we have to decrease and 0 means the restriction is unaffected by the variable. The same rules apply to the objective function. It doesn't take into account how much the restriction is violated neither how much the variable influences that constraint.

Table 7.1: Design rules - Mutation Directions, Welded Beam

		x_1	x_2	x_3	x_4
Constraints	g_1	1	1	1	0
	g_2	0	0	1	1
	g_3	-1	0	0	1
	g_4	1	0	0	0
	g_5	0	0	1	1
	g_6	0	0	-1	-1
	g_7	-1	-1	-1	-1
Objective Function		-1	-1	-1	-1

7.1.4 Results

Mutation was the first operator to be modified. The following figures show the evolution of the most fit individual over the generations. Both regular mutation and controlled mutation occur.

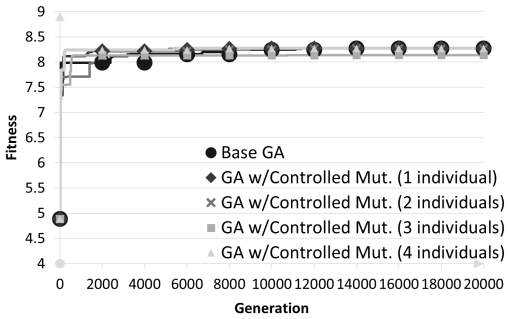

Figure 7.1: Fitness of the fittest individual, Controlled Mutation - Welded Beam.

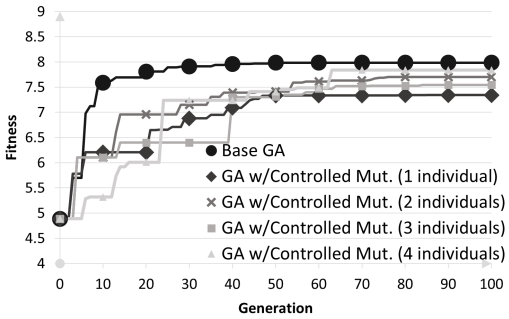

Figure 7.2: Fitness of the fittest individual in the first 100 generations, Controlled Mutation - Welded Beam.

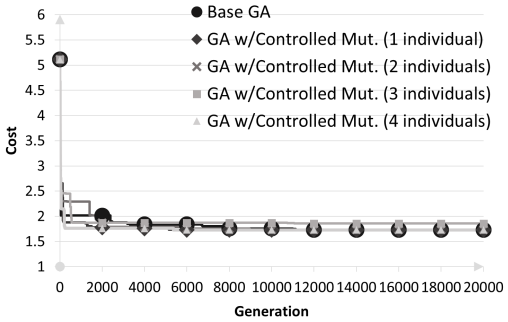

Figure 7.3: Cost of the fittest individual, Controlled Mutation - Welded Beam.

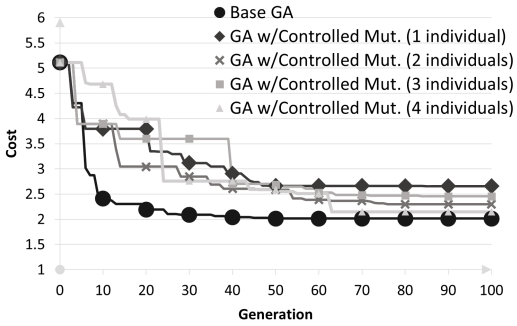

Figure 7.4: Cost of the fittest individual in the first 100 generations, Controlled Mutation, - Welded Beam.

The best individual obtained by each run was:

Table 7.2: Resulting points from controlled mutation applied to the welded beam problem.

Run	x_1 [in]	x_2 [in]	x_3 [in]	x_4 [in]
Base GA	0.205882	3.47765	9.06824	0.205882
GA w/Controlled Mutation(1 individual)	0.205882	3.47765	9.06824	0.205882
GA w/Controlled Mutation(2 individuals)	0.205882	3.47765	9.06824	0.205882
GA w/Controlled Mutation(3 individuals)	0.242647	3.05059	8.33059	0.242647
GA w/Controlled Mutation(4 individuals)	0.205882	3.47765	9.06824	0.205882

Table 7.3: Results from controlled mutation applied to the welded beam problem.

Run	Best fitness	Cost [$]	Generation
Base GA	8.26730	1.73270	11991
GA w/Controlled Mutation(1 individual)	8.26730	1.73270	6180
GA w/Controlled Mutation(2 individuals)	8.26730	1.73270	9631
GA w/Controlled Mutation(3 individuals)	8.13866	1.85657	11066
GA w/Controlled Mutation(4 individuals)	8.26730	1.73270	5415

With these preliminary results, we can conclude that Controlled Mutation has a great effect in the performance and should be done carefully.

Hybrid crossover was tested alone and in conjunction with regular crossover:

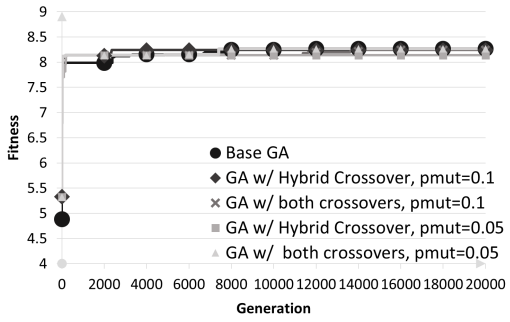

Figure 7.5: Fitness of the fittest individual, Hybrid Crossover - Welded Beam.

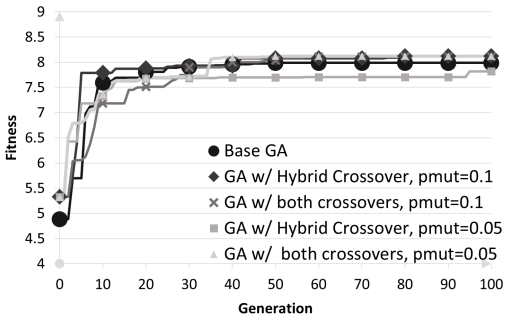

Figure 7.6: Fitness of the fittest individual in the first 100 generations, Hybrid Crossover - Welded Beam.

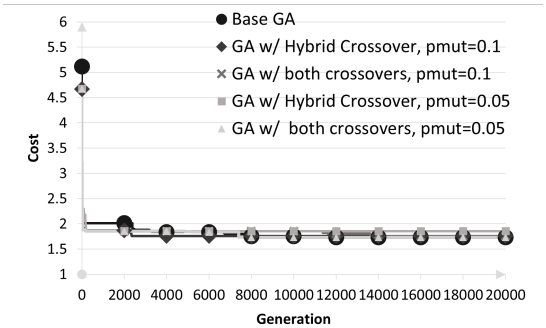

Figure 7.7: Cost of the fittest individual, Hybrid Crossover - Welded Beam.

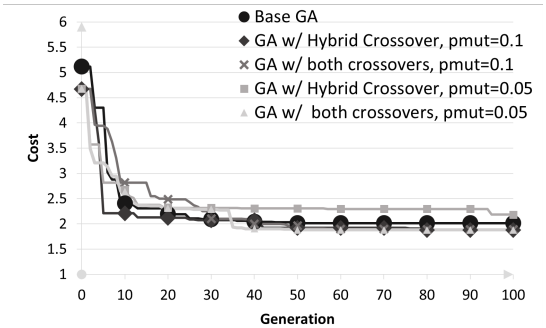

Figure 7.8: Cost of the fittest individual in the first 100 generations, Hybrid Crossover - Welded Beam.

Table 7.4: Resulting points from controlled mutation applied to the welded beam problem.

Run	x_1 [in]	x_2 [in]	x_3 [in]	x_4 [in]
Base GA	0.205882	3.47765	9.06824	0.205882
GA w/ Hybrid Crossover, $p_{mut} = 0.1$	0.213235	3.36118	8.91294	0.213235
GA w/ both crossovers, $p_{mut} = 0.1$	0.213235	3.36118	8.91294	0.213235
GA w/ Hybrid Crossover, $p_{mut} = 0.05$	0.242647	3.05059	8.33059	0.242647
GA w/ both crossovers, $p_{mut} = 0.05$	0.205882	3.47765	9.06824	0.205882

Table 7.5: Results from hybrid crossover applied to the welded beam problem.

Run	Best fitness	Cost [$]	Generation
Base GA	8.26730	1.73270	11991
GA w/ Hybrid Crossover, $p_{mut} = 0.1$	8.24358	1.75626	2345
GA w/ both crossovers, $p_{mut} = 0.1$	8.24358	1.76032	14276
GA w/ Hybrid Crossover, $p_{mut} = 0.05$	8.13866	1.85657	247
GA w/ both crossovers, $p_{mut} = 0.05$	8.26730	1.73270	7720

Finally both modified operators were combined:

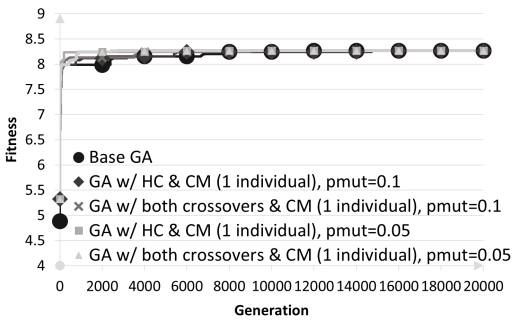

Figure 7.9: Fitness of the fittest individual, Mixed Operators - Welded beam.

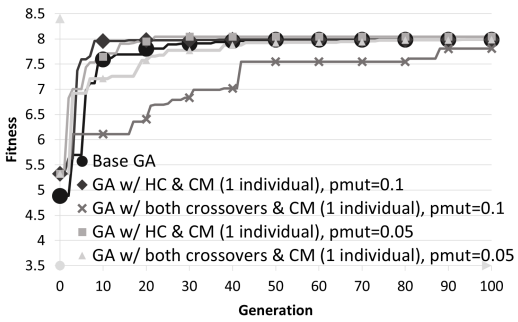

Figure 7.10: Fitness of the fittest individual in the first 100 generations, Mixed Operators - Welded beam.

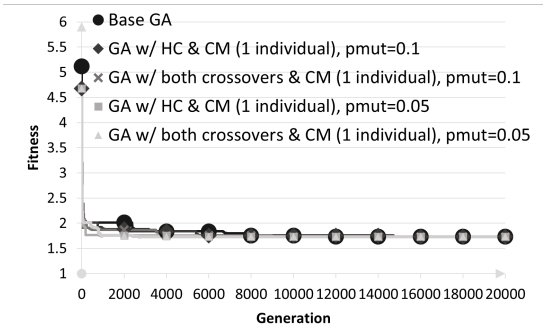

Figure 7.11: Cost of the fittest individual, Mixed Operators - Welded beam.

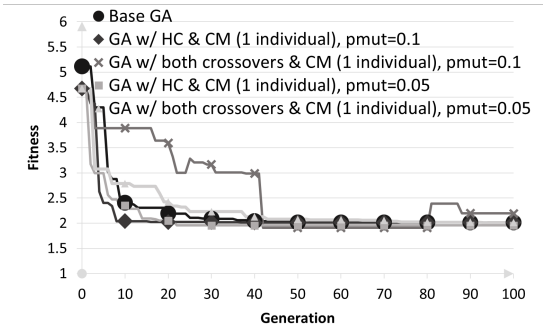

Figure 7.12: Cost of the fittest individual in the first 100 generations, Mixed Operators - Welded beam.

Table 7.6: Resulting points from both modified operators applied to the welded beam problem.

Run	x_1 [in]	x_2 [in]	x_3 [in]	x_4 [in]
Base GA	0.205882	3.47765	9.06824	0.205882
GA w/ Hybrid Crossover & Controlled Mutation, $p_{mut} = 0.1$	0.205882	3.47765	9.06824	0.205882
GA w/ both crossovers & Controlled Mutation, $p_{mut} = 0.1$	0.205882	3.47765	9.06824	0.205882
GA w/ Hybrid Crossover & Controlled Mutation, $p_{mut} = 0.05$	0.242647	3.05059	8.33059	0.242647
GA w/ both crossovers & Controlled Mutation, $p_{mut} = 0.05$	0.205882	3.47765	9.06824	0.205882

Table 7.7: Results from both modified operators applied to the welded beam problem.

Run	Best fitness	Cost [$]	Generation
Base GA	8.26730	1.73270	11991
GA w/ Hybrid Crossover & Controlled Mutation, $p_{mut} = 0.1$	8.26730	1.73270	14725
GA w/ both crossovers & Controlled Mutation, $p_{mut} = 0.1$	8.26730	1.73270	9809
GA w/ Hybrid Crossover & Controlled Mutation, $p_{mut} = 0.05$	8.26730	1.73270	5946
GA w/ both crossovers & Controlled Mutation, $p_{mut} = 0.05$	8.26730	1.73270	2774

In this last plot there is an increase there are increases in cost of the best solution between a generation an the next. This happens because the solutions with low cost found at early did so at the cost of violating constraints, showing the importance of looking not only at the cost but also the fitness of the solutions found.

The previous "upgrades" introduce very little computational cost at any generation when compared to the base GA, but, when analyzing the results of implementing local search it's important

to take into account the number of function evaluations. This is particularly important when the amount of variables is high because the maximum number of function evaluations when applying the Hokees-Jeeves method implemented is proportional to the number of variables(search directions).

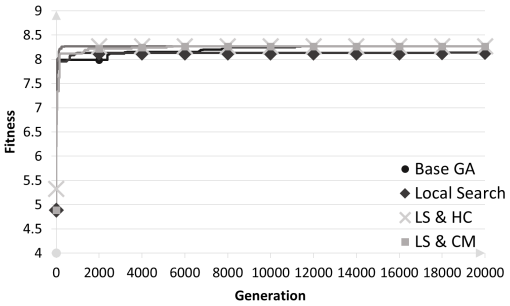

Figure 7.13: Fitness of the fittest individual, new operators & LS - Welded Beam.

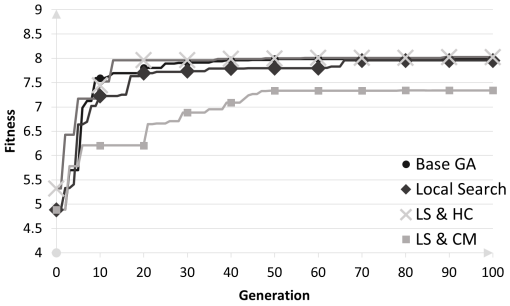

Figure 7.14: Fitness of the fittest individual in the first 100 generations, new operators & LS - Welded Beam.

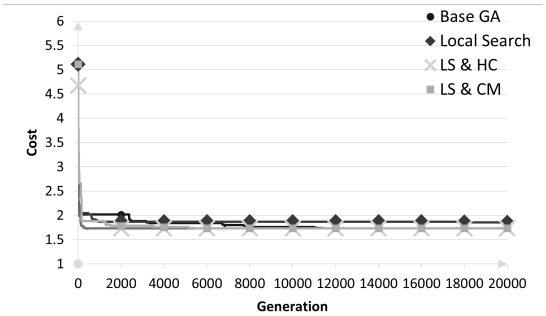

Figure 7.15: Cost of the fittest individual, new operators & LS - Welded Beam.

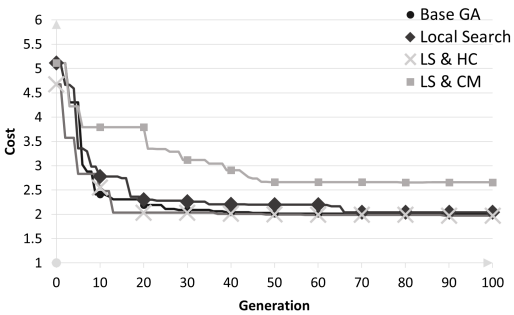

Figure 7.16: Cost of the fittest individual in the first 100 generations, new operators & LS - Welded Beam.

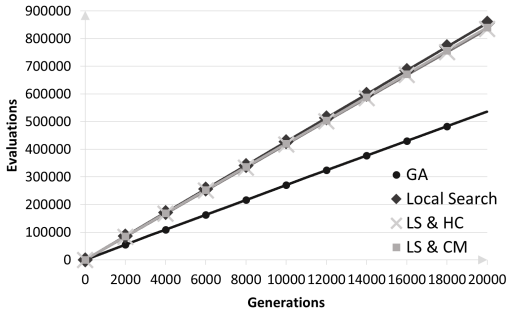

Figure 7.17: Number of function evaluations over the generations, new operators & LS - Welded Beam.

With 4 variables, LS leads to an increase of about 50% function evaluations.

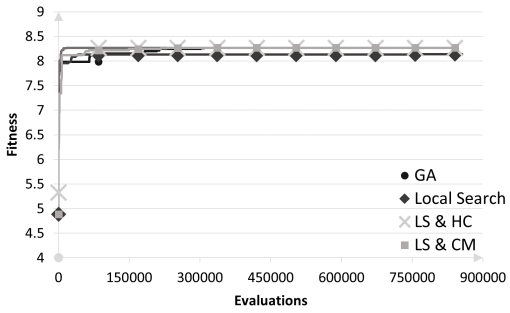

Figure 7.18: Fitness of the fittest individual versus the number of evaluations, new operators & LS - Welded Beam.

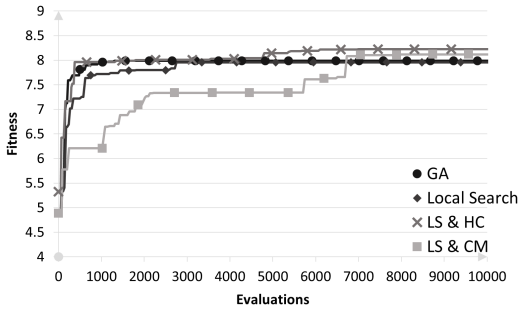

Figure 7.19: Fitness of the fittest individual in 1000 evaluations, new operators & LS - Welded Beam.

Table 7.8: Resulting points from both modified operators applied to the welded beam problem.

Run	x_1 [in]	x_2 [in]	x_3 [in]	x_4 [in]
Base GA	0.205882	3.47765	9.06824	0.205882
Local Search	0.242647	3.05059	8.33059	0.242647
LS & HC	0.205882	3.47765	9.06824	0.205882
LS & CM	0.205882	3.47765	9.06824	0.205882

Table 7.9: Results from both modified operators applied to the welded beam problem.

Run	Best fitness	Cost [$]	Generation	Evaluations
Base GA	8.26730	1.73270	11991	323251
Local Search	8.1357	1.8643	17882	765638
LS & HC	8.26730	1.73270	468	11068
LS & CM	8.26730	1.73270	6179	162833

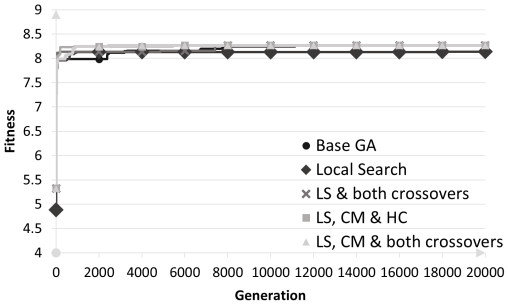

Figure 7.20: Fitness of the fittest individual, new operators & LS - Welded Beam.

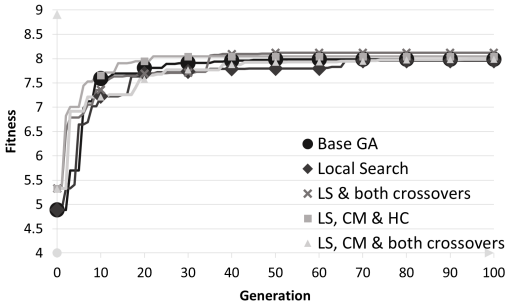

Figure 7.21: Fitness of the fittest individual in the first 100 generations, mixed operators & LS - Welded Beam.

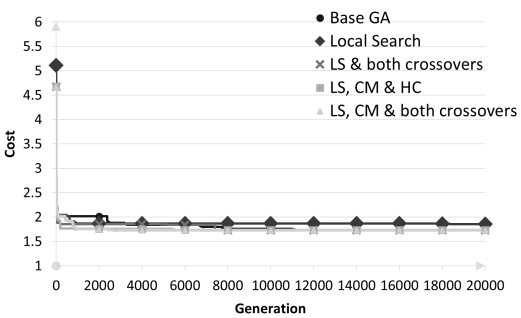

Figure 7.22: Cost of the fittest individual, mixed operators & LS - Welded Beam.

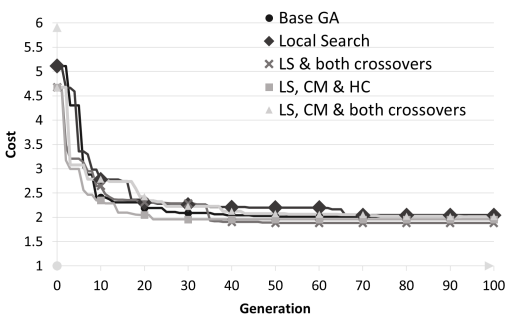

Figure 7.23: Cost of the fittest individual in the first 100 generations, mixed operators & LS - Welded Beam.

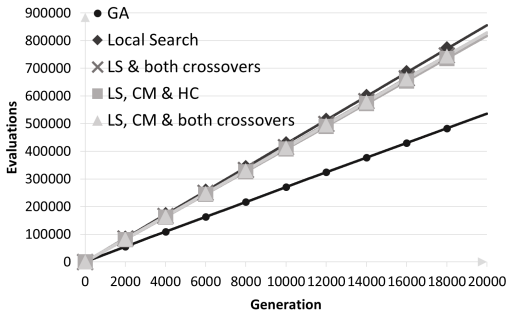

Figure 7.24: Number of function evaluations over the generations, mixed operators & LS - Welded Beam.

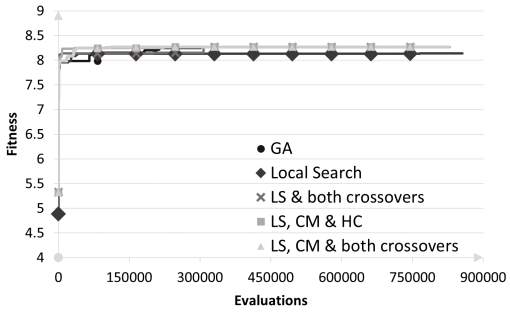

Figure 7.25: Fitness of the fittest individual versus the number of evaluations, mixed operators & LS - Welded Beam.

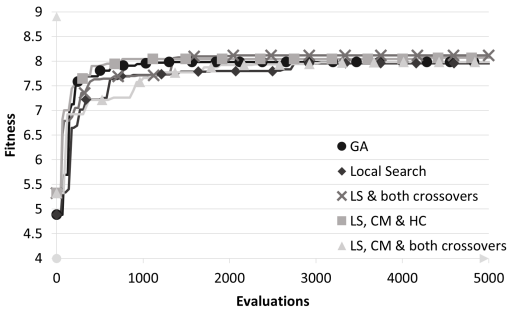

Figure 7.26: Fitness of the fittest individual in 1000 evaluations, mixed operators & LS - Welded Beam.

Table 7.10: Resulting points from both modified operators applied to the welded beam problem.

Run	x_1 [in]	x_2 [in]	x_3 [in]	x_4 [in]
Base GA	0.205882	3.47765	9.06824	0.205882
Local Search	0.242647	3.05059	8.33059	0.242647
LS, HC & CM	0.205882	3.47765	9.06824	0.205882
LS & both crossovers	0.205882	3.47765	9.06824	0.205882
LS, CM & both crossovers	0.205882	3.47765	9.06824	0.205882

Table 7.11: Results from both modified operators applied to the welded beam problem.

Run	Best fitness	Cost [$]	Generation	Evaluations
Base GA	8.26730	1.73270	11991	323251
Local Search	8.1357	1.8643	17882	765638
LS, HC & CM	8.26730	1.73270	468	11068
LS & both crossovers	8.26730	1.73270	7720	320392
LS, CM & both crossovers	8.26730	1.73270	2774	72544

When LS was applied by itself it happened to not achieve the global optimum, this might be justified by a vary early convergence to a local minimum.

7.2 Welded Gusset

7.2.1 Problem description

An L 100 X 65 X 11 [mm] profile with a section of $A = 1.7 \cdot 10^{-3}$ m^2, must be welded to a $1.4 \cdot 10^{-3}$ m thick plate so that the section of the profile can be subject to tension in it's totality in Figure 7.27.To do that, each profile wing will be welded according to the fraction of the load it should support. The material used for the beam is a St37 steel with $\sigma_{adm} \leq 137.34$ MPa and the flux of the weld has an admissible stress of $\sigma_{adm,weld} \leq 107.1$.

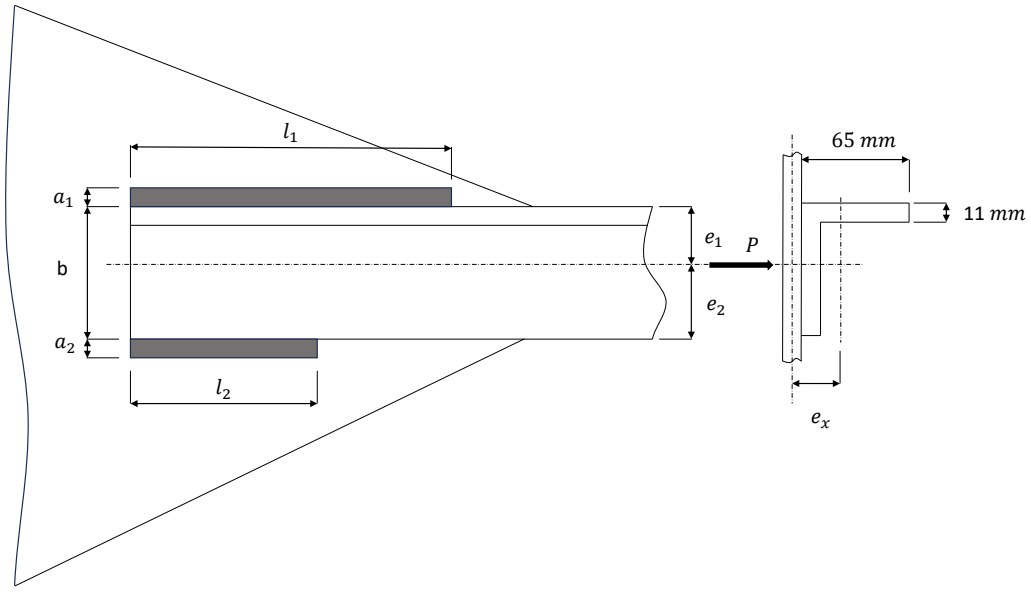

Figure 7.27: Welded Gusset design problem

were $b = 100$ mm and $e_1 = 34mm$.

The welding will be analyzed according to the Portuguese Regulation of 1986 (REAE)[65]. The objective is to minimize the cost of the welding.

7.2.2 Mathematical formulation

The cost is given by the volume of the welding multiplied by a constant that represents costs with materials, manpower, etc. $f(x) = k \cdot (a_1 \cdot l_1 + a_2 \cdot l_2)[CU]$(Cost Units). k doesn't affect the optimal point so we might as well set it to 1 for simplicity sake.

Calculation of the maximum admissible force P:

$$A = 1.7 \cdot 10^{-3} \text{ m}^2 \tag{7.2.1}$$

$$\sigma_{adm} = 137.31 \text{ MPa} \tag{7.2.2}$$

$$P = A\sigma_{adm} = 1.7 \cdot 10^{-3} \cdot 137.34 \cdot 10^6 = 233.478 \text{ kN} \tag{7.2.3}$$

Calculation of N_1 and N_2 using the moment equilibrium:

$$N_1 \cdot b - P \cdot e_2 = 0 \equiv N_1 = \frac{P \cdot e_2}{b} \tag{7.2.4}$$

$$N_2 \cdot b - P \cdot e_1 = 0 \equiv N_2 = \frac{P \cdot e_1}{b} \tag{7.2.5}$$

$$e_2 = b - e_1 = 0.066 \text{ m} \tag{7.2.6}$$

$$N_1 = 154.095 \text{ kN} \tag{7.2.7}$$

$$N_2 = 79.383 \text{ kN} \tag{7.2.8}$$

Maximum moment,M_{max}:

$$M_{max} = P \cdot e_x = 233.478 \cdot 10^3 \cdot 0.0237 = 5533.428 \,\text{N} \cdot \text{m} \tag{7.2.9}$$

According to REAE the flexural strength modulus, Z_f, is given by:

$$Z_{f1} = \frac{1}{6} l_1^2 \cdot a_1 \tag{7.2.10}$$

$$Z_{f2} = \frac{1}{6} l_2^2 \cdot a_2 \tag{7.2.11}$$

$$Z_f = Z_{f1} + Z_{f2} = \frac{1}{6}(l_1^2 \cdot a_1 + l_2^2 \cdot a_2) \tag{7.2.12}$$

and the normal stress is given by:

$$n = \frac{M_{max}}{Z_f} = \frac{M_{max}}{\frac{1}{6}(l_1^2 \cdot a_1 + l_2^2 \cdot a_2)} \tag{7.2.13}$$

The stress in the fillet welds, parallel to their axis, is given by:

$$t_{//} = \frac{P}{l_1 \cdot a_1 + l_2 \cdot a_2} \tag{7.2.14}$$

The equivalent stress, σ_{eq}, can then be calculated using the two previous stresses:

$$\sigma_{eq} = \sqrt{1.4n^2 + 1.8t_{//}^2} \tag{7.2.15}$$

The equivalent stress must be less or equal than the admissible stress of the weld material, $\sigma_{adm,weld}$, affected by a safety factor, $\bar{\alpha}$:

$$\sigma_{eq} \leq \bar{\alpha} \cdot \sigma_{adm,weld} \tag{7.2.16}$$

REAE tells us this safety factor should be calculated using the following equations:

$$\alpha_i = 0.8(1 + \frac{1}{a_i}), i = 1, 2 \bar{\alpha} = \text{Min}(\alpha_i, i = 1, 2) \tag{7.2.17}$$

where $a_i \geq 3$[mm],$i = 1, 2$

Lastly, we must establish an upper and lower bound for the variables:

$$3 \leq a_1 \leq 20 \text{ [mm]} \tag{7.2.18}$$

$$3 \leq a_2 \leq 20 \text{ [mm]} \tag{7.2.19}$$

$$20 \leq l_1 \leq 400 \text{ [mm]} \tag{7.2.20}$$

$$20 \leq l_2 \leq 250 \text{ [mm]} \tag{7.2.21}$$

In short, the optimization problem is:

$$\text{minimize}(l_1 \cdot a_1 + l_2 \cdot a_2) \tag{7.2.22}$$

subject to:

$$\begin{cases} g_1(\boldsymbol{x}) = \sigma_{eq} - \bar{\alpha} \cdot \sigma_{adm,weld} \geq 0 \\ g_2(\boldsymbol{x}) \text{ to } g_4(\boldsymbol{x}) : 3 \leq a_i \leq 20, i = 1, 2 \\ g_5(\boldsymbol{x}) \text{ and } g_6(\boldsymbol{x}) : 20 \leq l_1 \leq 400 \\ g_7(\boldsymbol{x}) \text{ and } g_8(\boldsymbol{x}) : 20 \leq l_2 \leq 250 \end{cases} \tag{7.2.23}$$

7.2.3 Optimization model

Similarly to what was done with the previous example the constraints were included in the optimization model through a quadratic penalty factor. A preliminary study to access the value of the constants led to $C = 10^4$ and $\lambda = 10^6$ after normalizing the constraint. For this values the optimal point is around[1]:

$$\boldsymbol{x} = (4.93, 4.93, 400.0, 250.0) \, [mm] \tag{7.2.24}$$

$$f(\boldsymbol{x}) = 3210.7 \, [CU] \tag{7.2.25}$$

$$fitness(\boldsymbol{x}) = 6789.3 \tag{7.2.26}$$

[1]These values were obtained doing an intensive search using the Steepest Descent Method starting from a good solution obtained from a GA

Design rules for Controlled Mutation

In a similar fashion to what was done in the previous application, here the mutation direction can also be obtained by analyzing the cost function and the constraint.

Table 7.12: Design rules - Mutation Directions, Welded Gusset.

		a_1	a_2	l_1	l_2
Constraint	g_1	1	1	1	1
Objective Function		-1	-1	-1	-1

7.2.4 Results

The results are presented below.

First the outcomes of including the modified operators by themselves:

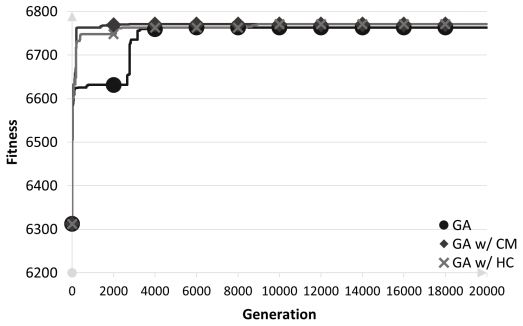

Figure 7.28: Fitness of the fittest individual, new operators - Welded Gusset.

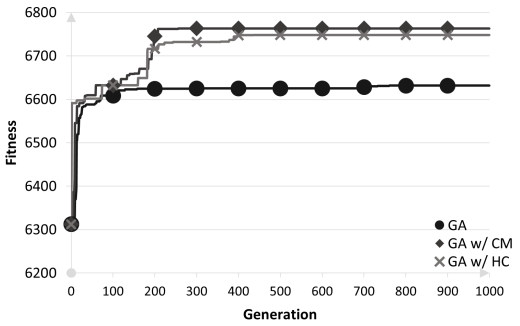

Figure 7.29: Fitness of the fittest individual in the first 1000 generations, new operators - Welded Gusset.

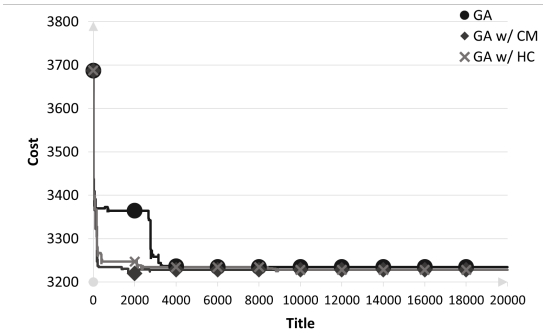

Figure 7.30: Cost of the fittest individual, new operators & LS - Welded Gusset.

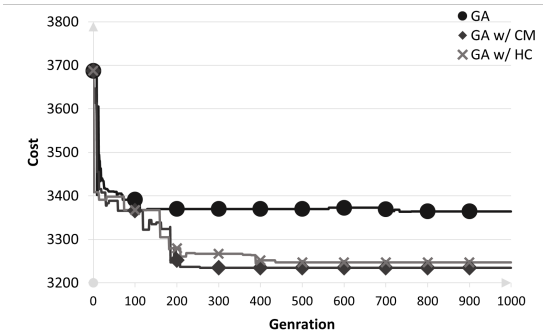

Figure 7.31: Cost of the fittest individual in the first 1000 generations, new operators - Welded Gusset.

Table 7.13: Resulting points from modified operators applied to the welded gusset problem.

Run	$a_1\,[mm]$	$a_2\,[mm]$	$l_1\,[mm]$	$l_2\,[mm]$
Base GA	5.13333	5.13333	400.0	230.157
GA w/Controlled Mutation	5.13333	4.73333	400.0	248.196
GA w/Hybrid Crossover	5.13333	4.73333	400.0	248.196

Table 7.14: Results from modified operators applied to the welded gusset problem.

Run	Best fitness	Cost $[CU]$	Generation
Base GA	6763.30	3234.81	4180
GA w/Controlled Mutation	6770.71	3228.13	2738
GA w/Hybrid Crossover	6770.71	3228.13	8936

And the effects from using multiple operators:

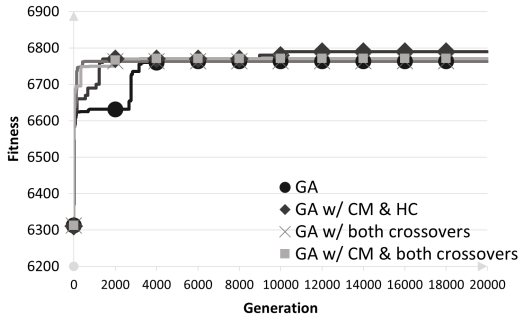

Figure 7.32: Fitness of the fittest individual, mixed operators - Welded Gusset.

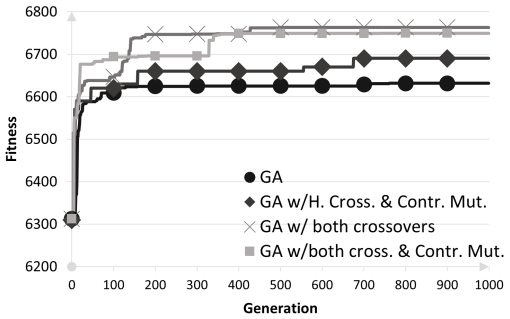

Figure 7.33: Fitness of the fittest individual in the first 1000 generations, mixed operators - Welded Gusset.

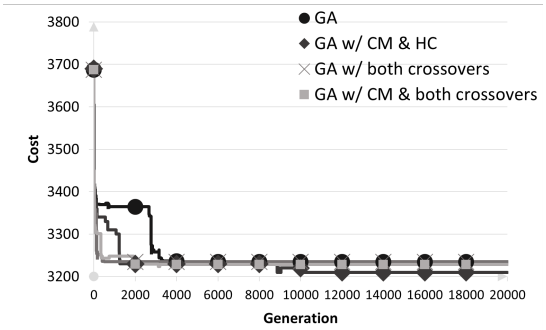

Figure 7.34: Cost of the fittest individual, mixed operators - Welded Gusset.

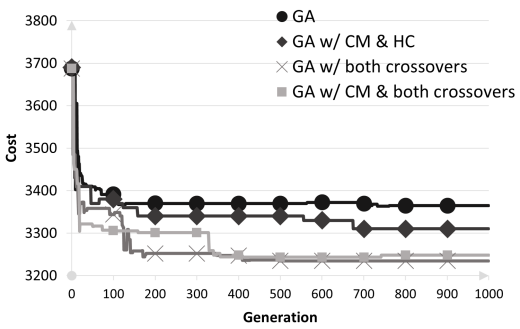

Figure 7.35: Cost of the fittest individual in the first 1000 generations, mixed operators - Welded Gusset.

Table 7.15: Resulting points from multiple modified operators applied to the welded gusset problem.

Run	$a_1\,[mm]$	$a_2\,[mm]$	$l_1\,[mm]$	$l_2\,[mm]$
Base GA	5.13333	5.13333	400.0	230.157
GA w/ both crossovers	5.13333	5.13333	400.0	230.157
GA w/ H. Cross. & Contr. Mut.	4.93333	4.93333	400.0	250.0
GA w/ both cross. & Contr. Mut.	5.13333	4.73333	400.0	248.196

Table 7.16: Results from multiple modified operators applied to the welded gusset problem.

Run	Fitness	Cost [CU]	Generation
Base GA	6763.30	3234.81	4180
GA w/ both crossovers	6763.30	3234.81	497
GA w/ H. Cross. & Contr. Mut.	6789.34	3206.67	10122
GA w/ both cross. & Contr. Mut.	6770.71	3228.13	3178

Variables a_1 and a_2 lack precision. Raising the number of bits that encode this variables would increase precision but also increment the complexity and therefor slow down convergence. Alternatively we could systematically reduce the search space by decreasing the amplitude of values they can take. In practice however, it's unlikely the welder is able to achieve such precision.

When introducing the LS operator the following results were obtained:

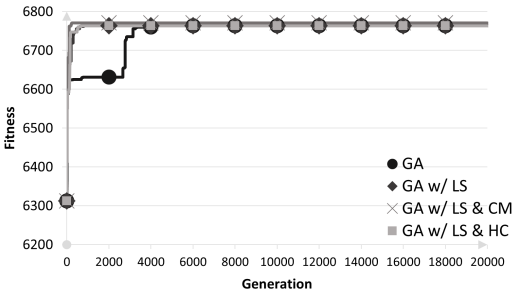

Figure 7.36: Fitness of the fittest individual, new operators & LS - Welded Gusset.

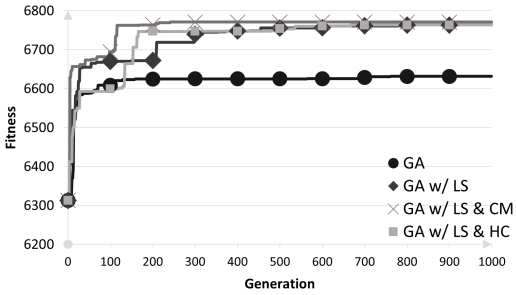

Figure 7.37: Fitness of the fittest individual in the first 1000 generations, new operators & LS - Welded Gusset.

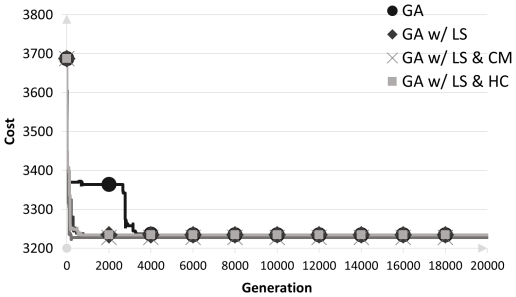

Figure 7.38: Number of evaluations over the generations, new operators & LS - Welded Gusset.

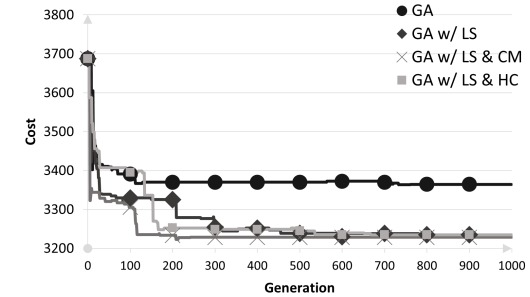

Figure 7.39: Cost of the fittest individual in the first 1000 generations, new operators & LS - Welded Gusset.

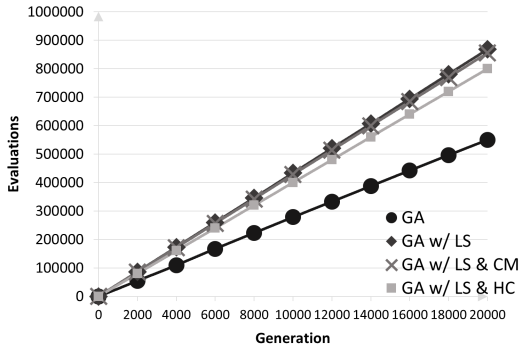

Figure 7.40: Cost of the fittest individual
in the first 1000 generations,
new operators & LS
- Welded Gusset.

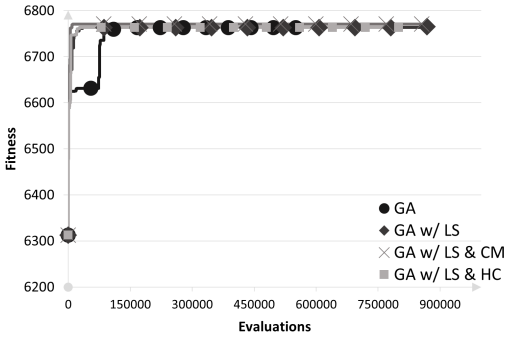

Figure 7.41: Fitness of the
fittest individual,
new operators & LS
- Welded Gusset.

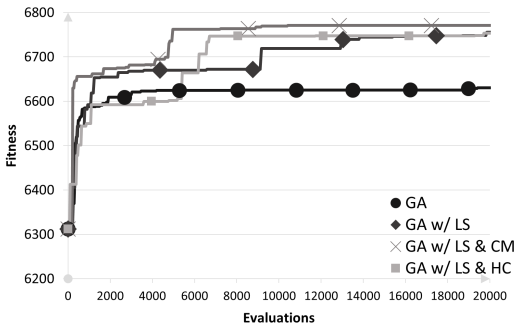

Figure 7.42: Fitness of the fittest individual
in the first 1000 generations,
new operators & LS
- Welded Gusset.

Table 7.17: Resulting points from local search and modified operators
applied to the welded gusset problem.

Run	$a_1\ [mm]$	$a_2\ [mm]$	$l_1\ [mm]$	$l_2\ [mm]$
Base GA	5.13333	5.13333	400.0	230.157
GA w/ Local Search	5.13333	4.86667	395.529	246.392
GA w/ LS & CM	5.13333	4.73333	400.0	248.196
GA w/ LS HC	5.13333	5.13333	400.0	230.157

Table 7.18: Results from local search and modified operators applied to the welded gusset problem.

Run	Best fitness	Cost $[CU]$	Generation	Evaluations
Base GA	6763.30	3234.81	4180	114936
GA w/ Local Search	6765.47	3229.49	19963	866604
GA w/ LS & CM	6770.71	3228.13	242	10425
GA w/ LS HC	6760.56	3239.44	605	24370

When applying LS and mixing operators the following was obtained:

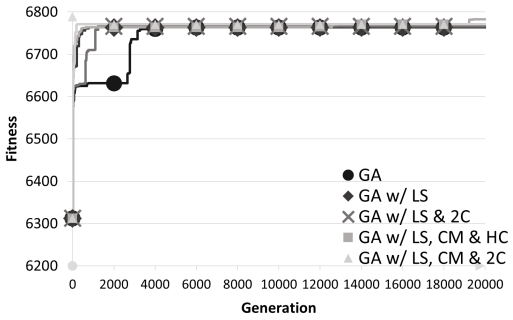

Figure 7.43: Fitness of the fittest individual, mixed operators & LS - Welded Gusset.

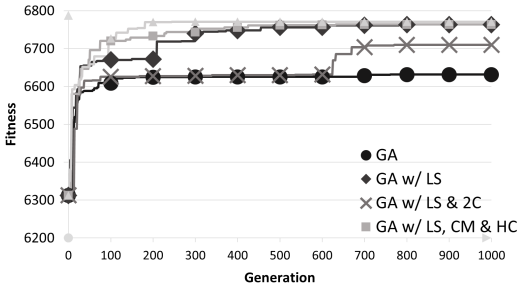

Figure 7.44: Fitness of the fittest individual in the first 1000 generations, mixed operators & LS - Welded Gusset.

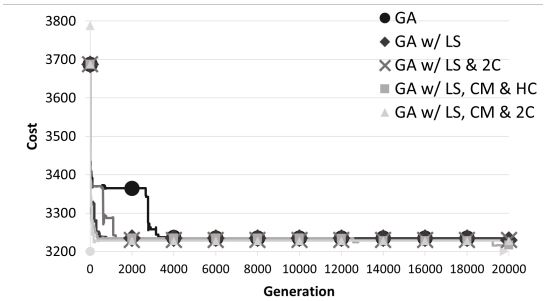

Figure 7.45: Cost of the fittest individual, mixed operators & LS - Welded Gusset.

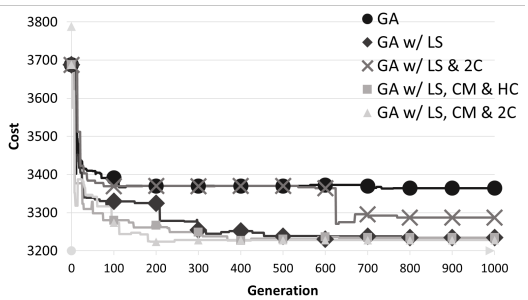

Figure 7.46: Cost of the fittest individual in the first 1000 generations, mixed operators & LS - Welded Gusset.

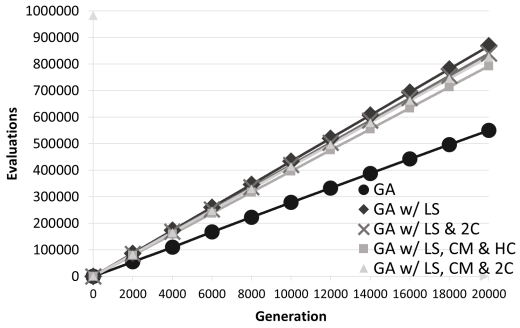

Figure 7.47: Number of function evaluations, mixed operators & LS - Welded Gusset.

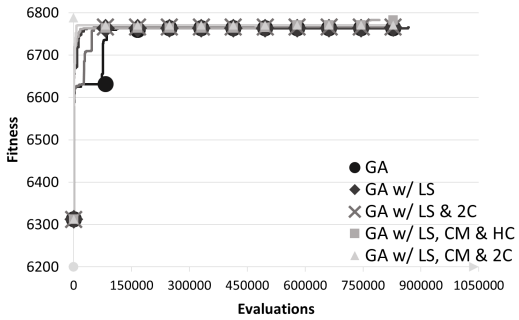

Figure 7.48: Fitness of the fittest individual, mixed operators & LS - Welded Gusset.

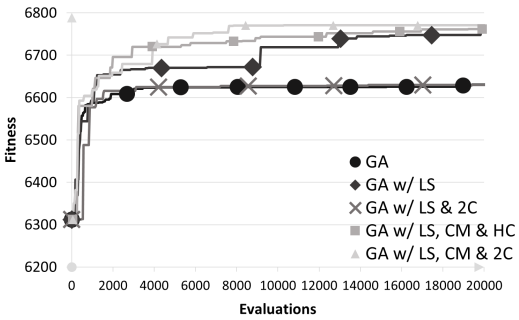

Figure 7.49: Fitness of the fittest individual in the first 20000 evaluations, mixed operators & LS - Welded Gusset.

Table 7.19: Resulting points from local search and multiple operators applied to the welded gusset problem.

Run	a_1 $[mm]$	a_2 $[mm]$	l_1 $[mm]$	l_2 $[mm]$
Base GA	5.13333	5.13333	400.0	230.157
GA w/ Local Search	5.13333	4.86667	395.529	246.392
GA w/ LS & 2C	4.86667	5.13333	400.0	250.0
GA w/ LS, CM & HC	5.13333	4.86667	400.0	250.0
GA w/ LS, CM & 2C	5.13333	5.13333	400.0	248.196

Table 7.20: Results from local search and multiple operators applied to the welded gusset problem.

Run	Best fitness	Cost $[CU]$	Generation	Evaluations
Base GA	6763.30	3234.81	4180	114936
GA w/ Local Search	6765.47	3229.49	19963	866604
GA w/ LS & 2C	6766.00	3230.00	1237	52262
GA w/ LS, CM & HC	6782.69	3216.67	19407	769576
GA w/ LS, CM & 2C	6770.71	3228.13	244	10289

7.3 Truss structure with 11 elements

A truss structure as depicted in fig 7.50 is subject to three loads at the same time. The objective is to minimize it's weight with respect to the cross sectional areas($\boldsymbol{x} = \{A_1, A_2, ..., A_{11}\}$) while making sure the displacements aren't too excessive.

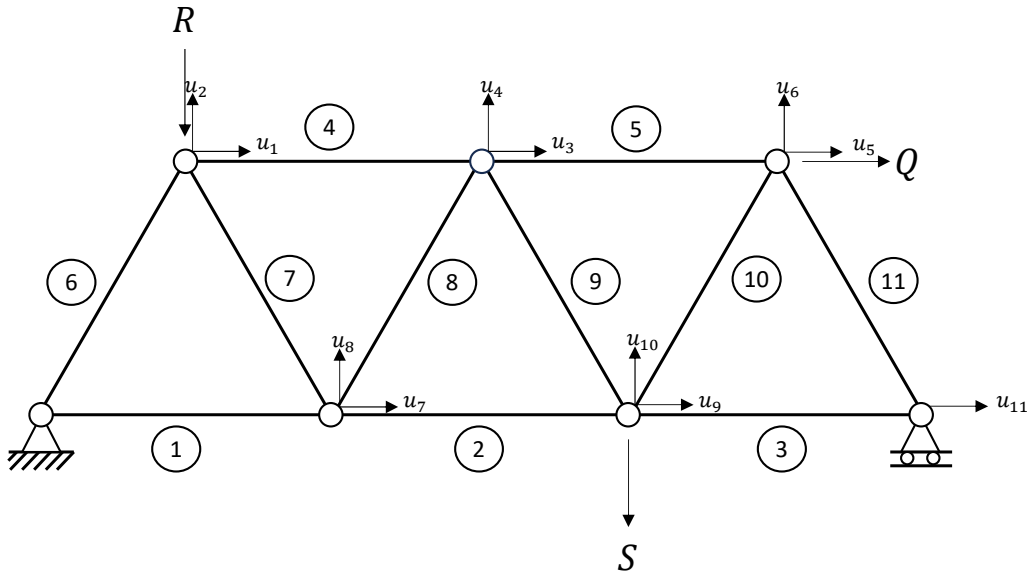

Figure 7.50: Truss structure.

The problem data is:

Table 7.21: Data for the truss structure problem.

E	σ_{yeald}	Q	R	S	l
210 GPa	210 MPa	300 kN	200 kN	50 kN	3 m

Even tho a structure can be modeled by a set of linear equations with the use of finite elements, obtaining explicit equations for the displacements (and consequentially the constraints) is nigh impossible. So they must be obtained numerically for a given $\boldsymbol{x}$.

7.3.1 Mathematical model

The displacements can be obtained by solving a system of linear equations. In the FEM this equations are typically written in the form:

$$[\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{7.3.1}$$

where $[\boldsymbol{K}]$ is the stiffness matrix, $\{\boldsymbol{u}\}$ the vector of nodal displacements and $\{\boldsymbol{f}\}$ the vector of external forces. The stiffness matrix and the vector of external forces are shown in the Appendix. To solve the linear system, Gaussian elimination with partial pivoting was implemented.

The weight of the structure can be obtained simply by adding the weight of each element:

$$W(\boldsymbol{x}) = \sum_{i=1}^{11} \rho_i x_i l_i \tag{7.3.2}$$

By asserting that all the elements have the same length and are made of the same material the objective function con be simplified to:

$$f(\boldsymbol{x}) = \sum_{i=1}^{11} x_i \, [CU] \tag{7.3.3}$$

The displacement constraints are:

$$g_i(\boldsymbol{x}) = |u_i| - \delta \leq 0 \, [\text{m}], \; i = 1, 2, ..., 10 \tag{7.3.4}$$

on top of these we should also include maximum stress constraints:

$$g_{11} \text{ to } g_{21}(\boldsymbol{x}) = |\sigma_i| - \sigma_{yeald} \leq 0 \, [\text{MPa}], \; i = 1, 2, ..., 11 \tag{7.3.5}$$

where σ_i is the normal stress at a bar, i, given by:

$$\sigma_i = \frac{x_{i,2} - x_{i,1}}{l_i} E \tag{7.3.6}$$

As for size constraints we have:

$$g_{11} \text{ to } g_{22} : 0.0001 \, m^2 \geq x_i \geq 0.1 \, m^2, \; i = 1, 2, ..., 11 \tag{7.3.7}$$

7.3.2 Optimization model

To ensure no constraint is violated the constants used in fitness function may be $C = 1$ and $\lambda = 100$.

In this problem there is a physical meaning to controlled mutation. If a structure doesn't violate any constraint it's possible to reduce the stiffness of at least one of it's members, to do so we can change the cross section of the members connected to the node with the lowest displacement and therefor reduce the weight of the structure.

If, on the other hand there is a node with too much displacement we must increase the stiffness of the members connected to such node.

A similar logic can be applied to the stress constraints: if a member is too stressed then the cross section must be increased, if instead the stress is very low then the cross section can be reduced.

7.3.3 Results

Once again, the results obtained after applying the modified operators by themselves were:

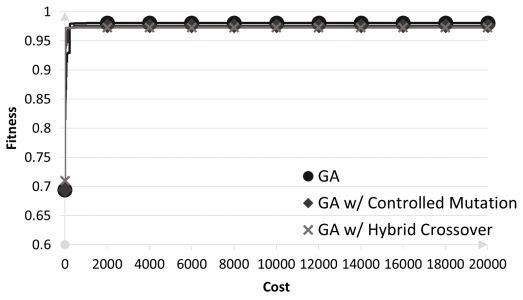

Figure 7.51: Fitness of the fittest individual, new operators - Truss Structure.

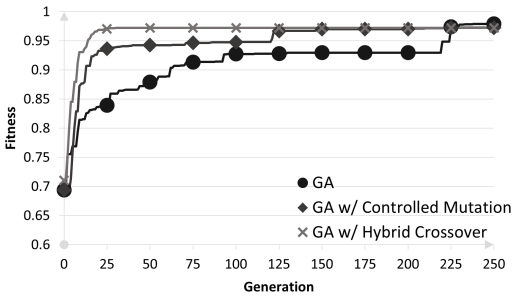

Figure 7.52: Fitness of the fittest individual in the first 250 generations, new operators - Truss Structure.

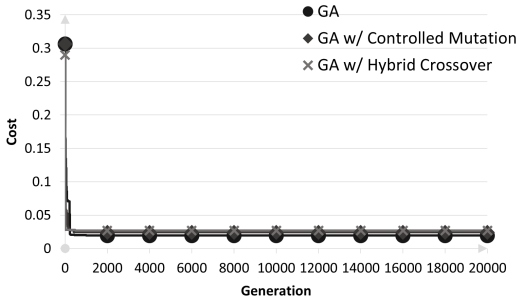

Figure 7.53: Cost of the fittest individual, new operators - Truss Structure.

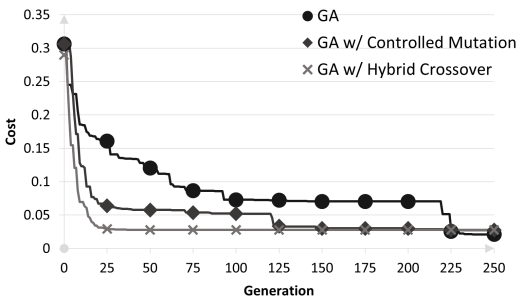

Figure 7.54: Cost of the fittest individual in the first 250 generations, new operators - Truss Structure.

Table 7.22: Resulting points from modified operators applied to the truss structure problem.

Variable	Base GA	GA w/ Controlled Mutation	GA w/ Hybrid Crossover
x_1 $[m^2]$	$3.23412 * 10^{-3}$	$3.23412 * 10^{-3}$	$3.62588 * 10^{-3}$
x_2 $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-3}$
x_3 $[m^2]$	$4.91765 * 10^{-4}$	$4.91765 * 10^{-4}$	$1.66706 * 10^{-3}$
x_4 $[m^2]$	$1.00000 * 10^{-4}$	$1.27529 * 10^{-3}$	$1.27529 * 10^{-3}$
x_5 $[m^2]$	$4.80118 * 10^{-3}$	$6.36824 * 10^{-3}$	$5.19294 * 10^{-3}$
x_6 $[m^2]$	$2.05882 * 10^{-3}$	$6.36824 * 10^{-3}$	$6.36824 * 10^{-3}$
x_7 $[m^2]$	$3.62588 * 10^{-3}$	$3.23412 * 10^{-3}$	$3.23412 * 10^{-3}$
x_8 $[m^2]$	$3.23412 * 10^{-3}$	$1.27529 * 10^{-3}$	$2.05882 * 10^{-3}$
x_9 $[m^2]$	$1.66706 * 10^{-3}$	$1.66706 * 10^{-3}$	$3.23412 * 10^{-3}$
x_{10} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$
x_{11} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$

Table 7.23: Results from modified operators applied to the truss structure problem.

Run	Best fitness	Cost $[CU]$	Generation
Base GA	0.980487	0.0195129	1005
GA w/ Controlled Mutation	0.975786	0.0242141	421
GA w/ Hybrid Crossover	0.973044	0.0269565	515

And when mixing operators:

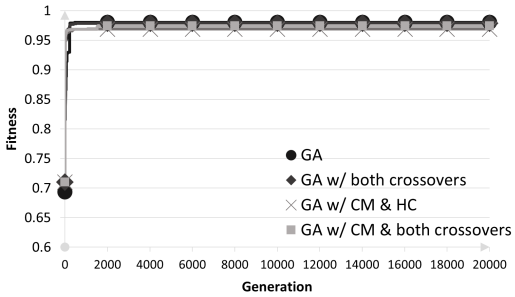

Figure 7.55: Fitness of the fittest individual, mixed operators - Truss Structure.

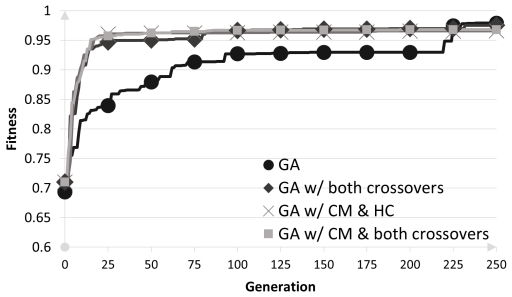

Figure 7.56: Fitness of the fittest individual in the first 250 generations, mixed operators - Truss Structure.

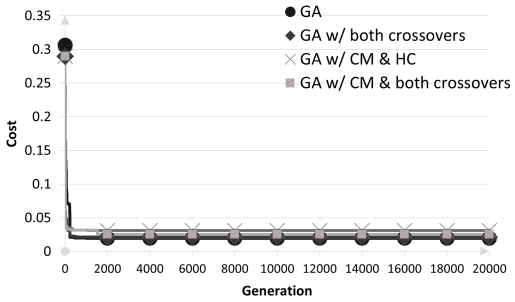

Figure 7.57: Fitness of the fittest individual, mixed operators - Welded Gusset.

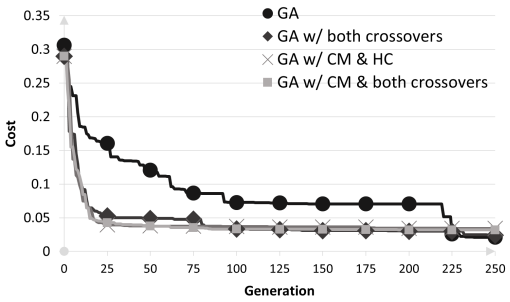

Figure 7.58: Fitness of the fittest individual in the first 250 generations, mixed operators - Truss Structure.

Table 7.24: Resulting points from multiple operators applied to the truss structure problem.

Variable	Base GA	GA w/ both cross.	GA w/ CM & HC	GA w/ CM & both cross.
x_1 $[m^2]$	$3.23412 * 10^{-3}$	$3.23412 * 10^{-3}$	$8.83529 * 10^{-4}$	$3.23412 * 10^{-3}$
x_2 $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-3}$	$1.00000 * 10^{-3}$
x_3 $[m^2]$	$4.91765 * 10^{-4}$	$4.91765 * 10^{-4}$	$1.66706 * 10^{-3}$	$4.91765 * 10^{-4}$
x_4 $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$2.45059 * 10^{-3}$	$1.00000 * 10^{-4}$
x_5 $[m^2]$	$4.80118 * 10^{-3}$	$6.36824 * 10^{-3}$	$1.26365 * 10^{-2}$	$6.36824 * 10^{-3}$
x_6 $[m^2]$	$2.05882 * 10^{-3}$	$2.45059 * 10^{-3}$	$7.15176 * 10^{-3}$	$2.05882 * 10^{-3}$
x_7 $[m^2]$	$3.62588 * 10^{-3}$	$3.23412 * 10^{-3}$	$8.83529 * 10^{-4}$	$6.36824 * 10^{-3}$
x_8 $[m^2]$	$3.23412 * 10^{-3}$	$3.23412 * 10^{-3}$	$1.66706 * 10^{-3}$	$3.23412 * 10^{-3}$
x_9 $[m^2]$	$1.66706 * 10^{-3}$	$1.66706 * 10^{-3}$	$3.23412 * 10^{-3}$	$3.23412 * 10^{-3}$
x_{10} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$
x_{11} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$

Table 7.25: Results from multiple operators applied to the truss structure problem.

Run	Best fitness	Cost $[CU]$	Generation
Base GA	0.980487	0.0195129	1005
GA w/ both crossovers	0.978920	0.0210800	1568
GA w/ CM & HC	0.969126	0.0308741	1054
GA w/ CM & both crossovers	0.974611	0.0253894	1548

We can see that the pure GA yielded the best results both when using the new operators by themselves and when mixing multiple operators.

When applying local search these results were found:

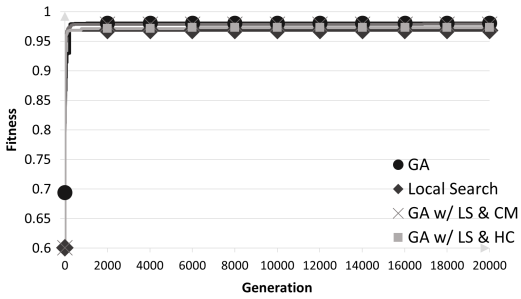

Figure 7.59: Fitness of the fittest individual, new operators & ls operators - Truss Structure.

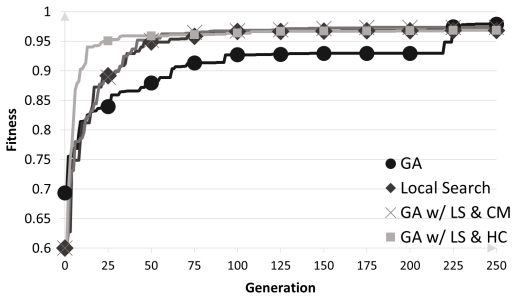

Figure 7.60: Fitness of the fittest individual in the first 250 generations, new operators & ls operators - Truss Structure.

ls mix

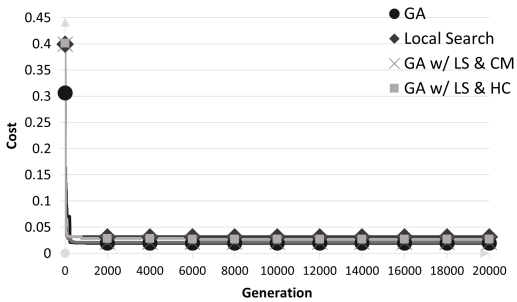

Figure 7.61: Fitness of the fittest individual, new operators & ls operators - Welded Gusset.

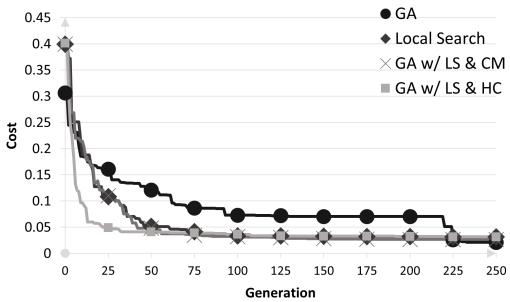

Figure 7.62: Fitness of the fittest individual in the first 250 generations, new operators & ls operators - Truss Structure.

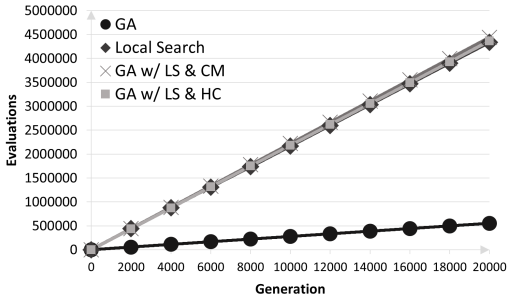

Figure 7.63: Fitness of the fittest individual
in the first 250 generations,
new operators & ls operators
- Truss Structure.

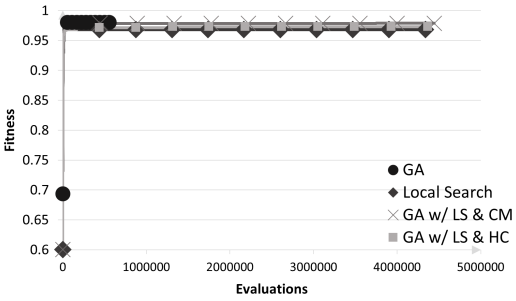

Figure 7.64: Fitness of the
fittest individual,
new operators & ls operators
- Welded Gusset.

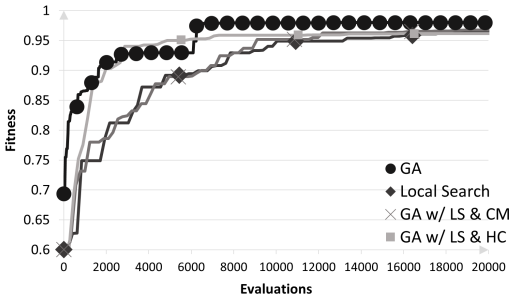

Figure 7.65: Fitness of the fittest individual
in the first 250 generations,
new operators & ls operators
- Truss Structure.

When applying the same operators with LS the results were:

Table 7.26: Resulting points from local search and modified operators applied to the truss structure problem.

Variable	Base GA	GA w/ LS	GA w/ LS & CM	GA w/ LS & HC
x_1 $[m^2]$	$3.23412 * 10^{-3}$	$1.00000 * 10^{-4}$	$2.45059 * 10^{-3}$	2.05882^{-3}
x_2 $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	1.00000^{-3}
x_3 $[m^2]$	$4.91765 * 10^{-4}$	$8.83529 * 10^{-4}$	$4.91765 * 10^{-4}$	1.66706^{-3}
x_4 $[m^2]$	$1.00000 * 10^{-4}$	$1.30282 * 10^{-2}$	$1.00000 * 10^{-4}$	1.66706^{-3}
x_5 $[m^2]$	$4.80118 * 10^{-3}$	$3.23412 * 10^{-3}$	$6.36824 * 10^{-3}$	6.36824^{-3}
x_6 $[m^2]$	$2.05882 * 10^{-3}$	$3.23412 * 10^{-3}$	$3.23412 * 10^{-3}$	6.36824^{-3}
x_7 $[m^2]$	$3.62588 * 10^{-3}$	$1.00000 * 10^{-4}$	$3.23412 * 10^{-3}$	3.23412^{-3}
x_8 $[m^2]$	$3.23412 * 10^{-3}$	$1.00000 * 10^{-4}$	$2.84235 * 10^{-3}$	2.05882^{-3}
x_9 $[m^2]$	$1.66706 * 10^{-3}$	$8.83529 * 10^{-4}$	$1.66706 * 10^{-3}$	3.23412^{-3}
x_{10} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	1.00000^{-4}
x_{11} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	1.00000^{-4}

Table 7.27: Results from local search and modified operators applied to the truss structure problem.

Run	Best fitness	Cost $[CU]$	Generation	N. evaluations
Base GA	0.980487	0.0195129	1005	28578
GA w/ Local Search	0.968342	0.0316576	210	46387
GA w/ LS & Controlled Mutation	0.978903	0.0210800	16870	3747194
GA w/ LS $ Hybrid Crossover	0.973435	0.0265647	5806	1271932

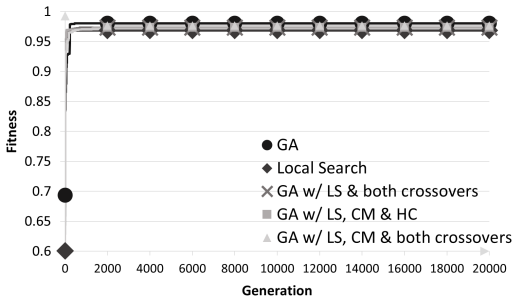

Figure 7.66: Fitness of the fittest individual, mixed operators & ls operators - Truss Structure.

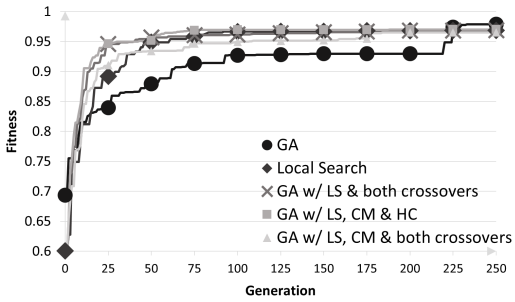

Figure 7.67: Fitness of the fittest individual in the first 250 generations, mixed operators & ls operators - Truss Structure.

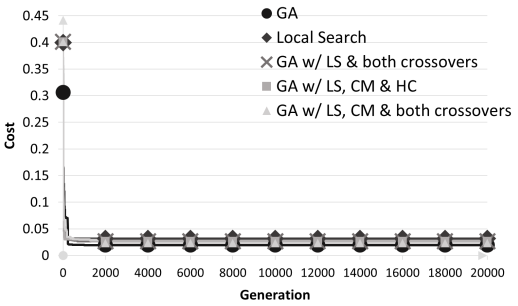

Figure 7.68: Fitness of the fittest individual, mixed operators & ls operators - Welded Gusset.

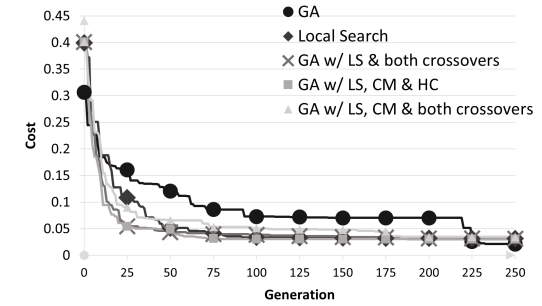

Figure 7.69: Fitness of the fittest individual in the first 250 generations, mixed operators & ls operators - Truss Structure.

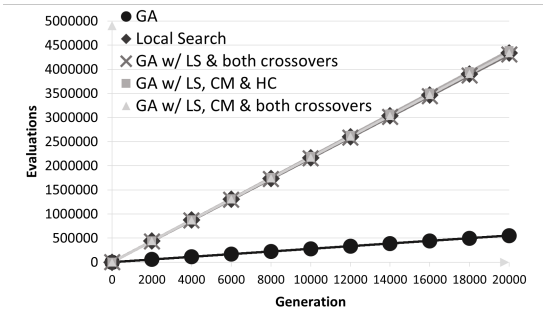

Figure 7.70: Fitness of the fittest individual in the first 250 generations, mixed operators & ls operators - Truss Structure.

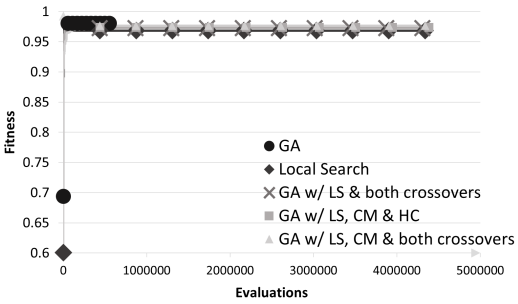

Figure 7.71: Fitness of the fittest individual, mixed operators & ls operators - Welded Gusset.

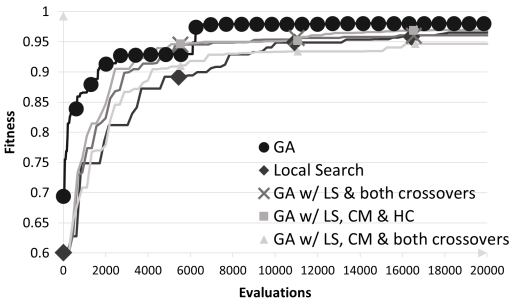

Figure 7.72: Fitness of the fittest individual in the first 250 generations, mixed operators & ls operators - Truss Structure.

When applying LS and multiple operators the following was found:

Table 7.28: Resulting points from local search and multiple operators applied to the truss structure problem.

Variable	Base GA	LS	LS & both cross.	LS, CM & HC	LS, CM & both cross.
x_1 $[m^2]$	$3.23412 * 10^{-3}$	$1.00000 * 10^{-4}$	$3.23412 * 10^{-3}$	$2.45059 * 10^{-3}$	2.05882^{-3}
x_2 $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$4.91765 * 10^{-4}$	$1.00000 * 10^{-4}$	1.00000^{-4}
x_3 $[m^2]$	$4.91765 * 10^{-4}$	$8.83529 * 10^{-4}$	$4.91765 * 10^{-4}$	$1.66706 * 10^{-3}$	8.83529^{-4}
x_4 $[m^2]$	$1.00000 * 10^{-4}$	$1.30282 * 10^{-2}$	$1.00000 * 10^{-4}$	$1.66706 * 10^{-3}$	1.00000^{-4}
x_5 $[m^2]$	$4.80118 * 10^{-3}$	$3.23412 * 10^{-3}$	$6.36824 * 10^{-3}$	$6.36824 * 10^{-3}$	6.36824^{-3}
x_6 $[m^2]$	$2.05882 * 10^{-3}$	$3.23412 * 10^{-3}$	$1.66706 * 10^{-3}$	$6.36824 * 10^{-3}$	3.23412^{-3}
x_7 $[m^2]$	$3.62588 * 10^{-3}$	$1.00000 * 10^{-4}$	$6.36824 * 10^{-3}$	$2.05882 * 10^{-3}$	2.45059^{-3}
x_8 $[m^2]$	$3.23412 * 10^{-3}$	$1.00000 * 10^{-4}$	$6.36824 * 10^{-3}$	$2.45059 * 10^{-3}$	6.36824^{-3}
x_9 $[m^2]$	$1.66706 * 10^{-3}$	$8.83529 * 10^{-4}$	$1.66706 * 10^{-3}$	$3.23412 * 10^{-3}$	1.66706^{-3}
x_{10} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	1.00000^{-4}
x_{11} $[m^2]$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	$1.00000 * 10^{-4}$	1.00000^{-4}

Table 7.29: Results from multiple modified operators applied to the truss structure problem.

Run	Fitness	Cost $[CU]$	Generation	N. evaluations
Base GA	0.980487	0.0195129	1005	28578
GA w/ Local Search	0.968342	0.0316576	210	46387
GA w/ LS & both crossovers	0.973044	0.0269565	678	148536
GA w/ LS, H. Cross. & Contr. Mut.	0.979704	0.0202965	2514	58107
GA w/ LS, both cross. & Contr. Mut.	0.976569	0.0234306	1912	418667

In the past GAs have proven to be very successful at finding optimal solutions to truss structures, so it's no surprise that, even with an significant increase in the number of variables relative to the previous problems explored, the algorithm converges very quickly. By analysing the first 250 generations we can see that the modified operators increase the speed of convergence, it just so happens the basic GA was able to randomly generate a very good solution, as we can see by the sudden jump in fitness. But even then, the best solution was found when using LS and both crossovers at the same time.

Because there are 11 degrees of freedom the number of function evaluations increases drastically, almost eight times.

Chapter 8

Conclusions

The expected number of points sampled ,without resampling, until we find the optimal solution for 32 $(4 * 8)$ bits is $2^{31} + 0.5 = 2,147,483,648.5 > 2 \cdot 10^9$ and for 88 $(11 * 8)$ bits is $2^{87} + 0.5 = 154,742,504,910,672,534,362,390,528.5 > 1.5 \cdot 10^{26}$, so it's very likely that a GA is better than random search for the problems analyzed.

To make sure the loss in exploration wasn't too excessive, the number of individuals generated by controlled mutation wasn't discounted in the number of individuals that suffered implicit mutation.

Controlled mutation introduces a new individual that is similar to the elite one that gave it origin. This appears to have an effect that reduces the speed of convergence in the initial generations but allows for finer improvements in the later stages of the search.

The increase in the number of individuals that undergo controlled mutation must be done carefully as the marginal gains(increase in exploitation) gained with each new individual added decrease while the marginal cost(decrease in exploration) increases. The increase can lead to an acceleration in convergence but might also lead to too early convergence(see Controlled Mutation in 3 individuals).

The inclusion of Controlled Mutation in one individual always led to improvements in the performance. Taking this is consideration it's recommend that, for this base GA's parameters, only one individual should be generated by this operator.

Controlled Mutation showed to be the best way to include knowledge obtained when solving past problems.

The use of just hybrid crossover seems to have negative effects. Likely because it prevents the possibility of crossover to happen between very similar parents, something that is necessary for convergence to happen at the final stages of the search. The hybrid crossover is a *greedy* operator as always takes the individual that it expects to be the best, this seams to lead to more elitist algorithm than a regular crossover operator as seen by the typical fast convergence in the early stages and slowing down later on.

When hybrid crossover and regular crossover were used together the results were better than when they were used in separate.

Sometimes the new operators yielded worst results than the original ones, but they showed good synergy when used together, highlighting the benefits of operators that perform different kind of moves.

The best results were achieved when multiple operators were employed at various stages of each generation.

The Hooke-Jeeves method implemented had good effects when applied to four dimensions both speeding up the search in terms of generations and number of evaluations. When applied to eleven dimensions however, the number of evaluations increased too much. Local search overtook mutation and crossover as driving forces of search which decreases convergence speed and increases the risk of falling into local minima, the algorithm became too elitist.

Sometimes local search lead to premature convergence, future work must be done to better balance out that aspect.

Still, the fact that there is a share of information of the last successful search direction when performing LS seems to be beneficial.

Summarizing: hybrid crossover should not be the only crossover operator, the Hooke-Jeeves methods isn't appropriate to problems with high dimensions and controlled mutation should always be included in MAs because it's the best way to include problem specific knowledge.

The knowledge obtained in this thesis can be used when solving future optimization problems, we are in the presence of offline learning.

8.1 Future works

This work explores a small part of MAs, in the future the following tasks can be done:

- Implement an adaptive scheme
- Quantify landscape properties
- Study the effectiveness of operators across different classes of problems
- Modify the Hooke-Jeeves method to handle high-dimension problems
- Use MA on surrogate models of complex problems

Bibliography

All the websites were accessed between the February and September of 2023

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

[2] Ferrante Neri, Carlos Cotta, and Pablo Moscato, eds. *Handbook of Memetic Algorithms*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-23247-3.

[3] Carlos C. António. *Otimização de Sistemas em Engenharia - Fundamentos e algoritmos para o projeto ótimo*. Ed. by Quântica Editora. 1st ed. New York, NY: Booki, Dec. 2020. ISBN: 978-989-901-734-4.

[4] Singiresu S. Rao. *Engineering Optimization, Theory and Practice*. Ed. by Inc. John Wiley & Sons. Wiley-Interscience, 1996.

[5] In: *Introduction to Optimum Design*. Ed. by Jasbir Singh Arora. Fourth Edition. Boston: Academic Press, 2017. ISBN: 978-0-12-800806-5. DOI: https://doi.org/10.1016/B978-0-12-800806-5.00025-1.

[6] Panos Y. Papalambros and Douglass J. Wilde. *Principles of Optimal Design*. Cambridge University Press, July 2000. ISBN: 9780511626418. DOI: 10.1017/cbo9780511626418.

[7] Garret N. Vanderplaats. "Numerical Optimization Techniques". In: *Computer Aided Optimal Design: Structural and Mechanical Systems*. Springer Berlin Heidelberg, 1987, pp. 197–239. DOI: 10.1007/978-3-642-83051-8_5.

[8] Yew Soon Ong and A.J. Keane. "Meta-Lamarckian learning in memetic algorithms". In: *IEEE Transactions on Evolutionary Computation* 8.2 (2004), pp. 99–110. DOI: 10.1109/TEVC.2003.819944.

[9] Natalio Krasnogor and Steven Gustafson. "A Study on the use of ''self-generation'' in memetic algorithms". In: *Natural Computing* 3.1 (2004), pp. 53–76. DOI: 10.1023/b:naco.0000023419.83147.67.

[10] Quang Huy Nguyen, Yew Soon Ong, and Meng Hiot Lim. "Non-Genetic Transmission of Memes by Diffusion". In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. GECCO '08. Atlanta, GA, USA: Association for Computing Machinery, 2008, pp. 1017–1024. ISBN: 9781605581309. DOI: 10.1145/1389095.1389285.

[11] Pablo Moscato, La Plata, and Michael Norman. "A "Memetic" Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems". In: 1 (July 1999).

[12] Z. Fawaz, Y.G. Xu, and K. Behdinan. "Hybrid evolutionary algorithm and application to structural optimization". In: *Structural and Multidisciplinary Optimization* 30.3 (May 2005), pp. 219–226. DOI: 10.1007/s00158-005-0523-3.

[13] YoungSu Yun. "Hybrid genetic algorithm with adaptive local search scheme". In: *Computers & Industrial Engineering* 51.1 (Sept. 2006), pp. 128–141. DOI: 10.1016/j.cie.2006.07.005.

[14] Minh Nghia Le et al. "Lamarckian memetic algorithms: local optimum and connectivity structure analysis". In: *Memetic Computing* 1.3 (Sept. 2009), pp. 175–190. DOI: 10.1007/s12293-009-0016-9.

[15] Daniel Molina et al. "Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains". In: *Soft Computing* 15.11 (Sept. 2010), pp. 2201–2220. DOI: 10.1007/s00500-010-0647-2.

[16] Daniel Molina et al. "Memetic Algorithms for Continuous Optimisation Based on Local Search Chains". In: *Evolutionary Computation* 18.1 (Mar. 2010), pp. 27–63. DOI: 10.1162/evco.2010.18.1.18102.

[17] Chunmei Zhang, Jie Chen, and Bin Xin. "Distributed memetic differential evolution with the synergy of Lamarckian and Baldwinian learning". In: *Applied Soft Computing* 13.5 (May 2013), pp. 2947–2959. DOI: 10.1016/j.asoc.2012.02.028. URL: https://doi.org/10.1016/j.asoc.2012.02.028.

[18] Carlos C. António. "A memetic algorithm based on multiple learning procedures for global optimal design of composite structures". In: *Memetic Computing* 6.2 (Apr. 2014), pp. 113–131. DOI: 10.1007/s12293-014-0132-z.

[19] Liang Feng et al. "Memes as building blocks: a case study on evolutionary optimization + transfer learning for routing problems". In: *Memetic Computing* 7.3 (Aug. 2015), pp. 159–180. DOI: 10.1007/s12293-015-0166-x.

[20] Jingfeng Yan, Meilian Li, and Jin Xu. "An adaptive strategy applied to memetic algorithms". In: *IAENG International Journal of Computer Science* 42.2 (2015), pp. 73–84.

[21] Yutong Zhang et al. "A multi-objective memetic algorithm based on decomposition for big optimization problems". In: *Memetic Computing* 8.1 (Feb. 2016), pp. 45–61. DOI: 10.1007/s12293-015-0175-9.

[22] Yuanlong Wang et al. "Optimization of an auxetic jounce bumper based on Gaussian process metamodel and series hybrid GA-SQP algorithm". In: *Structural and Multidisciplinary Optimization* 57.6 (Dec. 2017), pp. 2515–2525. DOI: 10.1007/s00158-017-1869-z.

[23] Jialong Shi, Qingfu Zhang, and Edward Tsang. "EB-GLS: an improved guided local search based on the big valley structure". In: *Memetic Computing* 10.3 (July 2017), pp. 333–350. DOI: 10.1007/s12293-017-0242-5.

[24] Beatrice Luca and Mitica Craus. "Local search algorithms for memetic algorithms: understanding behaviors using biological intelligence". In: *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*. 2018, pp. 553–558. DOI: 10.1109/ICSTCC.2018.8540690.

[25] Wenwen Liu, Shiu Yin Yuen, and Chi Wan Sung. "A generic method to compose an algorithm portfolio with a problem set of unknown distribution". In: *Memetic Computing* 14.3 (May 2022), pp. 287–304. DOI: 10.1007/s12293-022-00367-8.

[26] Hongling Chen et al. "Optimal operator selection based on the hybrid operator selection strategy". In: *Memetic Computing* 14.4 (Oct. 2022), pp. 461–476. DOI: 10.1007/s12293-022-00382-9.

[27] Valeria Simoncini. *Krylov Subspaces*. Ed. by J. Nicholas and Higham. Princeton University Press, 2015, pp. 113–114.

[28] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. New York, NY: Springer, July 2006. Chap. 16-18.

[29] Mark M. Millonas. *Swarms, Phase Transitions, and Collective Intelligence*. 1993. arXiv: adap-org/9306002 [adap-org].

[30] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence*. Oxford University Press, Oct. 1999. DOI: 10.1093/oso/9780195131581.001.0001.

[31] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. "Distributed Optimization by Ant Colonies". In: Jan. 1991.

[32] M. Dorigo. "Optimization, Learning and Natural Algorithms". PhD thesis. Politecnico di Milano, 1992.

[33] Rainer Storn and Kenneth Price. In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. DOI: 10.1023/a:1008202821328.

[34] Pablo Moscato. "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms". In: *Caltech Concurrent Computation Program* (Oct. 1989).

[35] Terry Jones. "One Operator, One Landscape". In: (Mar. 1995). "Last accessed 7 july 2023". URL: https://www.researchgate.net/publication/2439915_One_Operator_One_Landscape.

[36] Terry Jones. "Evolutionary Algorithms, Fitness Landscapes and Search". In: (June 1995). "Last accessed 7 july 2023". URL: https://www.researchgate.net/publication/23740259_Evolutionary_Algorithms_Fitness_Landscapes_and_Search.

[37] Hojjat Adeli and Nai-Tsang Cheng. "Augmented Lagrangian Genetic Algorithm for Structural Optimization". In: *Journal of Aerospace Engineering* 7.1 (1994), pp. 104–118. DOI: 10.1061/(ASCE)0893-1321(1994)7:1(104).

[38] Robert Axelrod. "Effective Choice in the Prisoner's Dilemma". In: *The Journal of Conflict Resolution* 24.1 (1980), pp. 3–25. ISSN: 00220027, 15528766. URL: http://www.jstor.org/stable/173932 (visited on 08/30/2023).

[39] Robert Axelrod. "More Effective Choice in the Prisoner's Dilemma". In: *The Journal of Conflict Resolution* 24.3 (1980), pp. 379–403. URL: https://www.jstor.org/stable/173638 (visited on 08/29/2023).

[40] Darrell" "Whitley. ""Sharpened and Focused No Free Lunch and Complexity Theory"". In: *"Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques"*. Ed. by Edmund K. "Burke and Graham" Kendall. "Boston, MA": "Springer US", 2014. Chap. 11, "451–476". ISBN: "978-1-4614-6940-7". DOI: "10.1007/978-1-4614-6940-7_16".

[41] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automated Machine Learning*. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-05318-5.

[42] Wikipedia contributors. *"Hyperparameter optimization — Wikipedia, The Free Encyclopedia"*. "Last accessed 12 June 2023". 2023. URL: https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=1156035022.

[43] Juan Carlos Lopez and Diana Lopez. "Killer Whales (Orcinus orca) of Patagonia, and Their Behavior of Intentional Stranding While Hunting Nearshore". In: *Journal of Mammalogy* 66.1 (1985), pp. 181–183. ISSN: 00222372, 15451542. DOI: 10.2307/1380981. (Visited on 07/11/2023).

[44] R Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, UK, 1976.

[45] Yew-Soon Ong et al. "Classification of adaptive memetic algorithms: a comparative study". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 36.1 (2006), pp. 141–152. DOI: 10.1109/TSMCB.2005.856143.

[46] Peter Cowling, Graham Kendall, and Eric Soubeiga. "A Hyperheuristic Approach to Scheduling a Sales Summit". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001, pp. 176–190. DOI: 10.1007/3-540-44629-x_11.

[47] Ryan Meuth et al. "A proposition on memes and meta-memes in computing for higher-order learning". In: *Memetic Computing* 1 (June 2009), pp. 85–100. DOI: 10.1007/s12293-009-0011-1.

[48] Christopher R. Houck et al. "Empirical Investigation of the Benefits of Partial Lamarckianism". In: *Evolutionary Computation* 5.1 (Mar. 1997), pp. 31–60. DOI: 10.1162/evco.1997.5.1.31.

[49] Darrell Whitley, V. Scott Gordon, and Keith Mathias. "Lamarckian evolution, the Baldwin effect and function optimization". In: *Parallel Problem Solving from Nature — PPSN III*. Ed. by Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 5–15. ISBN: 978-3-540-49001-2.

[50] J. Mark Baldwin. "A New Factor in Evolution". In: *The American Naturalist* 30.354 (June 1896), pp. 441–451. DOI: 10.1086/276408.

[51] J. Mark Baldwin. "Organic Selection". In: *Science* 5.121 (Apr. 1897), pp. 634–636. DOI: 10.1126/science.5.121.634.

[52] Frederic Mery and Tadeusz J. Kawecki. "The effect of learning on experimental evolution of resource preference in drisophila melanogaster". In: *Evolution* 58.4 (Apr. 2004), pp. 757–767. DOI: 10.1111/j.0014-3820.2004.tb00409.x.

[53] Ingo Paenke, Bernhard Sendhoff, and Tadeusz J. Kawecki. "Influence of Plasticity and Learning on Evolution under Directional Selection". In: *The American Naturalist* 170.2 (Aug. 2007), E47–E58. DOI: 10.1086/518952.

[54] Reiji Suzuki and Takaya Arita. "The Dynamic Changes in Roles of Learning through the Baldwin Effect". In: *Artificial Life* 13.1 (Jan. 2007), pp. 31–43. DOI: 10.1162/artl.2007.13.1.31.

[55] N. Krasnogor. "Studies on the Theory and Design Space of Memetic Algorithms". Supervisor: Dr. J.E. Smith. PhD thesis. University of the West of England, 2002. URL: http://www.cs.nott.ac.uk/~nxk/PAPERS/thesis.pdf.

[56] Carlos C. António. "A study on synergy of multiple crossover operators in a hierarchical genetic algorithm applied to structural optimisation". In: *Structural and Multidisciplinary Optimization* 38.2 (May 2008), pp. 117–135. DOI: 10.1007/s00158-008-0268-x.

[57] Carlos C. António and Inácio Lhate. "A hybrid crossover operator for structural optimisation based on commonality in genetic search". In: *Engineering Computations* 20 (June 2003), pp. 390–408. DOI: 10.1108/02644400310476315.

[58] C. A. Conceição António. "A hierarchical genetic algorithm with age structure for multimodal optimal design of hybrid composites". In: *Structural and Multidisciplinary Optimization* 31.4 (Jan. 2006), pp. 280–294. DOI: 10.1007/s00158-005-0570-9.

[59] W. Gutkowski, Z. Iwanow, and J. Bauer. "Controlled mutation in evolutionary structural optimization". In: *Structural and Multidisciplinary Optimization* 21.5 (July 2001), pp. 355–360. DOI: 10.1007/s001580100114.

[60] John H. Drake et al. “Recent advances in selection hyper-heuristics”. In: *European Journal of Operational Research* 285.2 (Sept. 2020), pp. 405–428. DOI: 10.1016/j.ejor.2019.07.073.

[61] Juan J. Durillo and Antonio J. Nebro. “jMetal: A Java framework for multi-objective optimization”. In: *Advances in Engineering Software* 42.10 (Oct. 2011), pp. 760–771. DOI: 10.1016/j.advengsoft.2011.05.014.

[62] Carlos C. António, Catarina F. Castro, and Luisa C. Sousa. “Optimization of metal forming processes”. In: 82.17-19 (July 2004), pp. 1425–1433. DOI: 10.1016/j.compstruc.2004.03.038.

[63] Irene Moser and Raymond Chiong. “A Hooke-Jeeves Based Memetic Algorithm for Solving Dynamic Optimisation Problems”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 301–309. DOI: 10.1007/978-3-642-02319-4_36.

[64] *Welded Beam Design Optimization - Maple Help*. URL: https://www.maplesoft.com/support/help/maple/view.aspx?path=applications%2FWeldedBeam.

[65] *Regulamento de Estruturas de Aço para Edifícios, 1986*. Decreto Lei nº 211/86 de 31 julho do Ministério das Obras Públicas, Transportes e Comunicações, Last accessed 26 june 2023. URL: https://files.dre.pt/1s/1986/07/17400/18841906.pdf.

8.2 Appendix

$$
[\boldsymbol{K}] = \begin{bmatrix}
4x_4+x_6+x_7 & \sqrt{3}(x_6-x_7) & -4x_4 & 0 & 0 & 0 & -x_7 & \sqrt{3}x_7 & 0 & 0 \\
 & 3(x_6+x_7) & 0 & 0 & 0 & 0 & \sqrt{3}x_7 & -3x_7 & 0 & 0 \\
 & & 4x_4+4x_5+x_8+x_9 & \sqrt{3}(x_8-x_9) & -4x_5 & 0 & -x_8 & -\sqrt{3}x_8 & -x_9 & \sqrt{3}x_9 \\
 & & & 3(x_8+x_9) & 0 & 0 & -\sqrt{3}x_8 & -3x_8 & -\sqrt{3}x_9 & -3x_9 \\
 & & & & 4x_5+x_{10}+x_{11} & \sqrt{3}(x_{10}-x_{11}) & 0 & 0 & -x_{10} & -\sqrt{3}x_{10} \\
 & & & & & 3(x_{10}+x_{11}) & 0 & 0 & -\sqrt{3}x_{10} & -x_{10} \\
 & & & & & & 4x_1+4x_2+x_7+x_8 & -\sqrt{3}(x_7-x_8) & -4x_2 & 0 \\
 & & & & & & & 3(x_7+x_8) & 0 & 0 \\
 & & & & & & & & 4x_2+4x_3+x_9+x_{10} & -\sqrt{3}(x_9-x_{10}) \\
 & & & & & & & & & 3(x_9+x_{10})
\end{bmatrix}
$$

sym.