Parameter Calibration of a System Dynamics Model

A Comparison of Three Evolutionary Algorithms

by Felipe Haro – July 2013

ABSTRACT

This research seeks to improve the parameter calibration process of a System Dynamics model. A "movie release strategies" model has been developed in 2012 using a gradient-based optimization algorithm to estimate all the parameters. On this research, three modern optimization algorithms are initially compared using mathematical benchmark functions and then tested with the model to compare results. The tested algorithms are modifications of the Artificial Bee Colony algorithm, the Cuckoo Search and the Genetic Sampler. The results show that by using the Artificial Bee Colony algorithm, better performance is achieved in terms of speed and fitness. It is also shown how the optimization problem definition was improved resulting from a better optimization process.

Acknowledgments

This project and would not have been possible without the support of some people. First of all, thanks to my family for supporting each step of the process, thanks to Ryan Hughes for building the bridge between PwC and the Erasmus Mundus System Dynamics program and PwC itself for giving support to this project. Also thanks to Rushil Mistry for giving me access to a real research database. Thanks to Pål Davidsen for being my supervisor and to all the people that I met during these two years that made a difference in my life.

Contents

Abbreviations

ABC: Artificial Bee Colony

AI: Artificial Intelligence

CS: Cuckoo Search

DE: Differential Evolution

EA: Evolutionary Algorithm

ES: Evolution Strategy

FA: Firefly Algorithm

GA: Genetic Algorithms

GML: Gauss-Marquardt-Levenberg

GS: Genetic Sampler

MPAA: Motion Picture Association of America

PSO: Particle Swarm Optimization

PwC: PricewaterhouseCoopers

SD: System Dynamics

TS: Tabu Search

vSBX: version of Simulated Binary Crossover

List of Figures

List of Tables

1 Introduction

Building System Dynamic (SD) Models comes along with the process of searching and analyzing data to support the development. And it can be said out of common sense that the more precise and detailed the collected information is, the higher is the potential quality of the quantitative model built. But sometimes the data is not measurable, not observable or simply too difficult or expensive to get. What is to be done with the unavailable information then? If the unavailable data is seen as a group of parameters with uncertain value, one way to deal with this issue is to make for instance an educated guess of each parameter in order to fulfill the requirements of the model. A second way is to define a range in which each of these parameters can move and calibrate them so the model replicates the observed data as close as possible. This is called "parameter calibration" and it's a common practice in SD.

It is known that for the human mind it's very difficult to understand and learn about the simpler of the models (Kopainsky et al. 2009). Hence the process of calibrating the parameters is very difficult for a human to be done in an effective way because of the infinite amount of possible combination of parameters that lead to an infinity of different outcomes. It is possible in certain occasions from a mix of luck, expertise and a trial and error process to choose a set of parameters that represents an acceptable solution depending on the modeler's needs. Nevertheless it is unthinkable that among all the combinations, a manual calibration can result in the best of all possible outcomes.

The set of parameters $\vec{x}$, with their particular possible range defined by the lower boundary x_i^{min} and the higher boundary x_i^{max} for each parameter i can be called the parameter search space, which is an N-Dimensional cube. But on this search space there are certain combinations that are not feasible and thereby not a solution for the problem. The combinations of parameters that are suitable and feasible belong to the feasibility region. This defines a typical constrained optimization problem with an objective function defined by how well the data is represented by the model and a set of restrictions defined by the parameters

search space and the feasibility region. This can be portrayed as the following standard constrained optimization problem with n parameters and m restrictions

$$\begin{array}{lll} max & f(x) & \\ subject\ to & l_i < x_i < u_i & for\ i = 1, \dots, n \\ & h_j(\vec{x}) \leq 0 & for\ j = 1, \dots, m \end{array}$$

Solving an optimization problem can be done in many ways and several different techniques exist in the literature such as the very basic simplex algorithm. But in the last years, a number of new modern methods meant to solve non-linear problems, among them the meta-heuristic optimization, have appeared. Bianchi et al (2009) defines a meta-heuristic as a procedure designed to find a good solution to a difficult optimization problem. Meta-heuristics are indeed an approach to deal with non-linear systems and to search for near-optimal solutions and have been proven to work in many areas and for many optimization problems. Luke (2009) defines it as a stochastic optimization and describes a large number of meta-heuristic methods among which evolutionary algorithms for optimization problems can be found.

In Artificial Intelligence (AI), an evolutionary algorithm (EA) is a population-based meta-heuristic optimization process where biological mechanisms such as reproduction, mutation, recombination and selection are used. Candidate solutions to the optimization problem play the role of individuals of the population and the fitness or objective function draws the world where the population lives. There is a good amount of evolutionary algorithms in the literature that have been applied in a large number of applications for different kind of optimization problems.

Modern algorithms inspired by nature use a trade off between randomization and local search (exploration and exploitation). A good combination of these components improves the opportunity to get global optimality, but usually, near-optimal solutions are found and there is no guarantee that the optimal solution will be reached.

PwC as a company was concerned about the parameter calibration options available and proposed to use one of their models (a “movie release strategies”

model that will be treated later in this document) to test a number of new optimization algorithms to not only gain some knowledge about the ones that are out there and are applicable to SD and Agent based modeling, but also to try to improve this model in terms of speed, results and optimization problem definition.

1.1.1 Research Objective

This research seeks to compare three nature-inspired optimization algorithms for the parameter calibration of an SD model provided by PwC. This model has previously been calibrated with the objective of obtaining the best possible representation of the data using a gradient-based algorithm. To make the comparison, the EA's are tested against mathematical benchmark functions with characteristics similar to that of the objective function of the tested model. This test allows the finding of the best candidate algorithm to be applied to the model measuring its performance in terms of speed and fitness and helps to find possible improvements of the optimization problem definition.

1.1.2 Research Questions

The following questions are important to evaluate the improvement of the results of the new algorithms applied to the SD model:

- Which algorithm performs better with benchmark functions that are similar to the one of the SD model?
    - Which algorithm requires less iterations?
    - Which algorithm finds a closer solution to the optimal value?
- How well do the new algorithms perform compared to the previous calibration work?
    - What is the difference of the results in terms of mean, median and minimum in the "movie release strategies" model?
    - What are the improvements in terms of feasibility of the solution?
    - How well is the optimization problem definition improved resulting from the comparison of the EA's?

1.1.3 Relevance

Experimenting with new ways to solve optimization problems for particular applications is a way to improve the potential results that can be achieved in the

future. In particular for the field of SD, the major part of the existent modeling Software currently available such as Vensim™ and Powersim™ deal with the parameter calibration problem as a black box with the objective function and restrictions characteristics as an input and with the solution as an output. For commercial reasons, the methodology is not publically available. In particular OptQuest™, an optimization tool used by AnyLogic™, used to combined the meta-heuristics of Tabu Search, Neural Networks, and Scatter Search into a single search heuristic (Kleijnen & Wan 2007).

This research shows transparent procedures to reach suitable solutions for an SD model and it is also an overview of different methods explained with detail to be used by future developers who need to solve optimization problems. The research also gives a comparison of three different evolutionary algorithm variations that have never been compared before for global or constrained optimization problems. This adds value to the existent algorithms comparison research and helps to ease the decision when it comes to choose an algorithm to apply it in similar applications.

This work is also relevant for developers who want a detailed cookbook on how to implement some optimization algorithms and for AnyLogic™ users who want to use the Java code or the pseudo-code used and developed on this research to optimize not only SD models with the aim of calibrating parameters, but also for other applications including Agent-based Modeling, Discrete Event Modeling or hybrids, without the limitations of black box tools with no transparency.

This research also shows in a practical way the process of improvement of the optimization definition problem and its importance to effectively perform the parameter calibration of a model. It is well known by the optimization community that one of the most difficult and most important things to take into consideration to get good results is the definition of the fitness function and the definition of the constraints and parameter boundaries. This requires a very good understanding of the problem that is being faced and this work shows the process of obtaining good solutions.

1.1.4 Thesis Outline

This thesis is divided in the following chapters. The first chapter gives a general overview of the concept of optimization and the algorithms available along with a clear definition of the research questions and objective. The second chapter gives a review of some of the most popular meta-heuristic methods available to solve different kinds of optimization problems and shows some of the existent literature that compares them. The third chapter explains in general terms the movie model used to make the analysis, how the optimization problem was defined initially for this model and what data is used. Chapter 4 gives a detailed explanation of the algorithms used on this research and the algorithm used by PwC to generate the parameters of the model. Chapter 5 explains what kind of benchmark functions were used to test the different algorithms and show the results obtained. It also portrays the process lived through meetings with PwC to improve the optimization problem definition and finally it shows the final results and the comparison between the EA's and the gradient-based algorithm used by PwC. Chapter 6 contains the final conclusions with its limitations and the discussion of the results. Added to this, a seventh chapter contains the references and chapter 8 contains the instructions to use the some of the files that were developed during this research.

2 Literature Review

To understand the extent of the existent research on EA's it is necessary to give an overview of all the most popular algorithms, the existent comparison work and the use of them in SD applications.

2.1 Evolutionary Algorithms Overview

The power of all modern meta-heuristics comes from the fact that they imitate the best characteristics of nature. Algorithms that imitate those nature features are becoming increasingly popular. Two characteristics of the nature behavior are the selection of the fittest (intensification) and adaptation to the environment (diversification). Intensification seeks to select the best candidates for next generations and diversification ensures that the landscape is explored efficiently.

Some of the most popular meta-heuristic algorithms are briefly reviewed as part of the research effectuated to choose the suitable algorithms for the application where the optimization process is applied. More emphasis is put on the three algorithms that are finally chosen to run the comparisons.

2.1.1 Ant Colony

The ant colony optimization (ACO) was first proposed by Dorigo et al. (1991) and it is a probabilistic technique for solving optimization problems that can be reduced to finding good paths through graphs, for instance for path planning in robotics (Haro & Torres 2006).

In nature the ants wander initially randomly and when food is found return to their colony leaving pheromone trails for other ants to follow the trail. Over time the pheromone trail starts to evaporate reducing its attractive strength. For ants, short trails tend to be more attractive than long paths because of the pheromone evaporation time. The ACO mimics this behavior with virtual ants walking on the problem landscape.

ACO has a number of variations that can be used to solve different kind of optimization problems and for some of them it has been proved that the algorithm

can find a global optimum in finite time (Negulescu et al. 2008, Stützle & Hoos 2000, Bullnheimer et al. 1997, Hu et al. 2008, Katteda et al. 2011).

2.1.2 Artificial Bee Colony

As defined by Karaboga & Basturk (2007), in the Artificial Bee Colony (ABC) model, the colony consists of three groups of bees: employed bees, onlookers and scouts. It is assumed that there is only one artificial employed bee for each food source. Employees bees go to their food source and come back to hive and dance on this area. The employed bee whose food source has been abandoned becomes a scout and starts searching to find a new food source. Onlookers watch the dances of employed bees and choose food sources depending on dances.

The classic ABC pseudo-code of the algorithm is as follows:

1. Initialize the population of solutions $x_{i,j}, i = 1 \dots SN, j = 1 \dots D$
2. Evaluate the population
3. cycle=1
4. REPEAT
    a. Produce new solutions $v_{i,j}$ for the employed bees by using $v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{i,k})$ where $k \in \{1,2, \dots, SN\}$
    b. Apply the greedy selection process.
    c. Calculate the probability values $P_{i,j}$ for the solutions $x_{i,j}$ by using $p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n}$
    d. Produce the new solutions $v_{i,j}$ for the onlookers from the solutions $x_{i,j}$ selected depending on $P_{i,j}$ and evaluate them
    e. Apply the greedy selection process
    f. Determine the abandoned solution for the scout, if exists, and replace it with a new randomly produced solution $x_{i,j}$ using $x_i^j = x_{min}^j + rand(0,1)(x_{max}^j - x_{min}^j)$
    g. Memorize the best solution achieved so far
    h. cycle = cycle + 1
5. UNTIL (requirements are met)

Table 1: Pseudocode for standard ABC algorithm

Where SN is the number of food sources, which is equal to the number of employed or onlooker bees, $\phi_{i,j}$ is a random number between [-1,1], fit_i is the fitness value of the solution i which is proportional to the nectar amount of the food source in the position i. The greedy selection process is defined by the fact that each candidate source position v_{ij} is produced and then evaluated by the artificial bee and its performance is compared with the old one. If the new food has an equal or better nectar level than the old source, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory.

In ABC, the position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the fitness of the associated solution. The number of the employed bees is equal to the number of solutions in the population. At the first step, a randomly distributed initial population (food source positions) is generated. After initialization, the population is subjected to repeat the cycles of the search processes of the employed, onlooker and scout bees, respectively. An employed bee produces a modification on the source position in its memory and discovers a new food source position. Provided that the nectar amount of the new one is higher than that of the previous source, the bee memorizes the new source position and forgets the old one. Otherwise it keeps the position of the one in its memory. After all employed bees complete the search process they share the position information of the sources with the onlookers on the dance area. Each onlooker evaluates the nectar information taken from all employed bees and then chooses a food source depending on the nectar level of the source. As in the case of the employed bee, it produces a modification on the source position in its memory and checks its nectar amount. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one. The sources abandoned are defined and new sources are randomly produced to be replaced by the abandoned ones by artificial scouts.

Bolaji et Al. (2013) have done a survey on the ABC algorithm, giving an overview of all the existent variants and applications since its creation in 2005. A lot of modifications of this algorithm have been developed during the years mostly to solve constrained and unconstrained optimization problems. But there is no conclusion on which modification is better and depending on the chosen

parameters of the algorithm and the characteristic of the problem, different modifications can lead to better results. This work will be done with the latest version of the ABC algorithm for constrained problems proposed by Karaboga & Akay (2011), which has been compared with some benchmark functions against some other modern algorithms.

2.1.3 Cuckoo Search

An obligate parasite is a parasitic organism that cannot complete its life cycle without exploiting a suitable host. If an obligate parasite cannot obtain a host it will fail to reproduce. Obligate parasites have evolved a variety of parasitic strategies to exploit their hosts. The cuckoos are birds that are included in the "obligate brood parasites" type and require nests and parental care of other passerines in order for their young to grow into a stage of development where they can fly. Cuckoo Search is an optimization algorithm inspired by some cuckoo species that lay eggs in the nests of other host birds. These hosts can identify these eggs as parasites and take an action such as throwing them away or moving to a new place building there a new nest. In particular, the striped cuckoo, which is a brood-parasitic cuckoo specie present in the American continent have evolved in such a way that the females are specialists in choosing hosts with very similar color and pattern (Payne et al. 2005). Other species remove others' eggs to put their own. Cuckoo Search, developed by Yang & Deb (2009) has taken this behavior and idealized it to apply it in optimization problems.

To idealize it, Cuckoo Search works in the following way: each egg (from any bird) in a nest represents a solution for the optimization problem, and a cuckoo egg represents a new solution. The objective is to replace with cuckoo eggs (that are potentially better solutions) the other eggs in the different nests. The algorithm can be extended to more complicated cases in which each nest has multiple eggs representing a set of solutions. Alkallak (2012) uses this extension to solve a combinatory problem.

Cuckoo Search is based on three idealized rules:

1. Each cuckoo lays one egg at a time and dumps its egg in a randomly chosen nest.

2. The best nests with high quality of eggs will carry over to the next generation.
3. The number of available hosts nests is fixed, and the host bird discovers the egg laid by a cuckoo with a certain probability. In this case the host bird can either throw the egg away or abandon the nest, and build a completely new nest.

In comparison to other meta-heuristic algorithms, Cuckoo Search is very simple since there are only two parameters to adjust: the fraction of the worse nests that are abandoned for new ones to be built and the population size. The choice of the setting of that fraction does not affect the convergence rate and the authors of CS propose to use 0.25.

A random walk is a mathematical formalization of a path that consists of a succession of random steps. CS uses Lévy flight, which is a random walk where the step length has a probability distribution that is heavy tailed. According to Asmusssen & Søren (2003) the distribution of a random variable X with distribution function F is said to have heavy right tail if: $\lim_{x\to\infty} e^{\lambda x} \Pr[X > x] = \infty$ for all $\lambda > 0$

The basic principle of this kind of random walk is that it consists of long travels to different regions of the landscape, meaning that there is a long movement to a random area, and then small movements at that area. According to Yang & Deb (2009), the random walk via Lévy flights is more efficient in exploring the search space as its step length is much longer in the long run.

The following is the pseudo code for Cuckoo Search:

1. Initialize a population of n host nests $x_i, i = 1,2, \dots, n$
2. **for** $i = 1$ to n
    a. Calculate fitness $F_i = f(x_i)$
3. **end for**
4. **while** $NumberObjectiveEvaluations < MaxNumberEvaluations$
    a. Generate a cuckoo egg (x_j) by taking a Lévy flight from random nest
    b. $F_j = f(x_j)$
    c. Choose a random nest i
    d. **if** $F_j > F_i$ **then**

i. the nest x_i is replaced by x_j with its associated fitness

e. end if

f. Abandon a praction p_a of the worst nests

g. Build new nests at new locations via Lévy Flights to replace nests lost

h. Evaluate fitness of new nests and rank all solutions

5. **end while**

Table 2: Pseudo-code for Cuckoo Search

The algorithm works as follows as shown in Yang & Deb (2009):

When generating new solutions $X(t+1)$ for a cuckoo i, a Lévy flight is performed using the stochastic equation for random walk.

$X_i(t+1) = X_i(t) + \alpha \oplus Lévy(\lambda)$, where $\alpha > 0$ is the step size which should be related to the scales of the problem of interests (α =1 can be used in most cases). The product $\oplus$ is a Hadamard product, which is a binary operation that takes two matrices of the same dimensions and produces a new matrix where each element ij is the product of elements ij of the original two matrices. The Lévy flight provides a random walk while the random step length is drawn from a Lévy distribution with an infinite variance and an infinite mean.

$$Lévy \sim u = t - \lambda, (1 < \lambda \leq 3)$$

Here the steps form a random walk process with a power-law step-length distribution with a heavy tail. Some of the new solutions should be generated by Lévy walk around the best solution obtained so far speeding the local search. However a good fraction of the new solutions should be generated by far field randomization and whose locations should be far enough from the current best solution, avoiding being trapped in a local optimum.

2.1.4 Differential Evolution

Differential Evolution (DE) is another meta-heuristic optimization method born in 1997 that uses agents as the population (Storn & Price 1997). The agents are subjected to recombination, evaluation and selection. The recombination approach involves the creation of new candidate solution components based on the weighted difference between two randomly selected population members

relative to the spread of the broader population (Brownlee 2011). DE has a nomenclature to define the configuration/variation of the algorithm of the form DE/x/y/z where x represents the solution to be perturbed, y represents the number of difference vectors used in the perturbation x and z represents the recombination operator.

2.1.5 Evolution Strategy

Evolution Strategy (ES) was developed in 1973 and is one of the oldest evolutionary algorithms. It uses recombination, mutation, evaluation and selection as the methodology to produce new and improved population (Rechenberg 1973). The different selection and mutation methodologies define the different variations of this method (Karaboga & Akay 2009).

2.1.6 Firefly Algorithm

The Firefly Algorithm (FA) is a meta-heuristic algorithm inspired by the flashing behavior of fireflies. The firefly's flash works as a signal system to attract other fireflies. Yang (2011) formulates the algorithm by assuming:

1. All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex
2. Attractiveness is proportional to their brightness, thus for any two flashing fireflies, the less brighter one will move towards the brighter one. The attractiveness is proportional to the brightness and they both decrease as their distance increases. If there is no brighter one than a particular firefly, it will move randomly
3. The brightness of a firefly is affected or determined by the landscape of the objective function.

There are a number of variations of this algorithm depending on the particular application.

2.1.7 Genetic Algorithms

This is the most popular type of EA. As described by Brownlee (2011), Genetic Algorithms (GA) is inspired by population genetics and evolution at the population level, as well as the Mendelian understanding of the structure and mechanisms. Individuals of a population contribute their genetic material (called

the genotype) proportional to their suitability of their expressed genome (called their phenotype) to their environment, in the form of offspring. The next generation is created through a process of mating that involves recombination of two individuals genomes in the population with the introduction of random copying errors (called mutation). This iterative process may result in an improved adaptive-fit between the phenotypes of individuals in a population and the environment.

The objective of the Genetic Algorithm is to maximize the payoff of candidate solutions in the population against a cost function from the problem domain.

The classic pseudo-code for minimizing a cost function is as follows, even though in the literature a big number of variations exist:

1. Define inputs: $Population_{size}, Problem_{size}, P_{crossover}, P_{mutation}$
2. InitializePopulation($Population_{size}, Problem_{size}$)
3. EvaluatePopulation(*Population)* and get best solution S_{best} from *Population*
4. **While** (Stop condition)
    a. Select *Parents*
    b. **foreach** $Parent_1, Parent_2 \in Parents$
        i. $Get\ Child_1, Child_2\ through\ crossover(Parent_1, Parent_2, P_{cross}$
        ii. $Get\ Children\ through\ Mutation(Child_1, P_{mutation})$
        iii. $Get\ Children\ through\ Mutation(Child_2, P_{mutation})$
    c. End foreach
    d. EvaluatePopulation**(***Children*)
    e. Get best solution S_{best}
    f. Get new *Population* through Replace*(Population, Children)*
5. **End while**
6. **Return** S_{best}

Table 3: Pseudo-code for Genetic Algorithms

2.1.8 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is one of the most popular meta-heuristic optimization algorithms and for that reason a large number of variants exist in the literature. The most basic variant works by having a swarm of particles that move around the search space guided by their own best position and the

entire swarm best position. This algorithm is inspired by the social foraging behavior some animals such as flocking behavior of birds and the schooling behavior of fish (Brownlee 2011).

2.1.9 Scatter Search

Scatter Search is a meta-heuristic and a global optimization algorithm associated with the field of evolutionary computation given the use of population and recombination (Brownlee 2011). It is a flexible method where all its elements can be implemented in a variety of ways and degrees of sophistication. It contains a diversification generation method to generate an initial collection of solutions, an improvement method to enhance the initial solution, a reference set update method to guild and maintain a reference set of best solutions, a subset generation method to create a base to create combined solutions and a solution combination as a recombination procedure (Marti et al. 2006).

2.1.10 Tabu Search

Tabu Search (TS) is a meta-heuristic local optimization method that was formalized by Glover (1989) and Glover (1990). It uses a local search procedure to move to better solutions in a defined neighborhood. The objective for the TS is to constrain an embedded heuristic from returning to recently visited areas of the search space, referred as cycling. It maintains a short-term memory of the specific changes of recent moves within the search space and preventing future moves from undoing those changes. Additional intermediate-term memory structures may be introduced to bias moves toward promising areas of the search space, as well as longer-term memory structures that promote a diversification of the solutions. TS has been mostly applied to discrete domains (Brownlee 2011).

2.2 Comparison

In general in the literature (Civicioglu & Besdok 2011, Karaboga & Akay 2009), meta-heuristic optimization methods are compared in simple landscapes that present different kind of shapes with known optimal solution. This is good to test general behavior of algorithms and prove how well they work, and it has also been done during this research.

Karaboga & Akay (2009) tested the performance of the standard ABC algorithm with the standard versions of GA, PSO, DE and ES optimization algorithms on a large set of unconstrained test functions. It is discussed how new candidate solutions are generated and how difficult it is to tune the parameters of the algorithms. It is concluded on this research that the performance of the ABC algorithm is better than the others or at least similar.

Civicioglu & Besdok (2011) compares the algorithmic concepts of CS, PSO, DE and ABC algorithms using 50 different benchmark functions. It concludes that the problem solving success of CS is very close to the DE algorithm, but the DE requires less function evaluations. The CS and DE algorithms supply more robust and precise results than the PSO and ABC algorithms.

Hegerty et al. (2009) compares DE and GA algorithms for combinatorial problems and it concludes that even though DE has a higher level of computational complexity, it provides more stability while GA gets stuck in local optima.

Jones (2005) compares GA and PSO concluding that GA is superior because it is faster, gives more accurate results and it arrives to its final parameter values in fewer generations than the PSO.

Sayadi et al. (2010) explores the FA to solve an NP-Hard optimization problem and compares it with a version of the ACO obtaining better results in almost all the benchmark functions tested.

For this work, PwC, as the client, requested using a variation of GA's called Genetic Sampler, and according to the comparison research review, the other two algorithms chosen to develop for this thesis are variations of CS and ABC. These algorithms are chosen because of their potential even though there is not enough research to conclude that one algorithm is better that all the others for any case and it depends highly on the author of the research and his biases towards a preferred algorithm, the tunable parameters used and the application.

2.3 Applications

It is quite trivial to notice that since parameter calibration is an optimization problem, any meta-heuristic algorithm previously explained is suitable for the task if a good variation exists or is developed. In particular for the field of SD, most of

the research on this topic is old, for instance Graham (1976) and there is little new research for the parameter calibration process with modern algorithms and it generally works as a black box in private Software such as Vensim™ or AnyLogic™.

Oliva (2003) gives a model calibration strategy, saying that usually the calibration of the model is done "by hand" in an iterative process He proposes two approaches for parameter estimation: full-information maximum-likelihood via optima filtering (FIMLOF) and model reference optimization (MRO), discarding the first one because of the need of system linearization. The use of a module from Vensim developed in 1995 is proposed.

Yücel & Barlas (2011) study an automated approach for parameter search based on behavior patterns. To generate parameters it uses a standard GA. The approach is somehow different in this case because it seeks to match pattern behaviors and the intention is to use it even if historical data doesn't exist.

오덕교 (2012) compares three optimization methods for a system dynamics model with the objective of obtaining the best fit of the model to the reference data. The three methods are: Excel optimization, manual optimization and Vensim™ with the Euler optimization method. It is concluded that Vensim™ gives better results.

3 Methodological Approach

3.1 Movie Release Strategies Model

This work has been requested by the company PwC and seeks to explore new algorithms for the parameter optimization process for not only SD models, but also for others such as agent-based and discrete events modeling. Using AnyLogic™ as the developing Software, some JAVA functions were programmed and applied to the "movie release strategies" model as a step forward to use these techniques in different projects and models in the future.

3.1.1 The model

A "movie release strategies" model has been developed by Hughes (2012) aiming to explain the variance in box-office revenue between movies and to forecast the expected revenue as a function of a dynamic diffusion structure, the movie's intrinsic attributes, the movie's release strategy, and the competitive environment the movie is released on.

This model includes an optimization process that aims to maximize the R^2 value of the box-office revenue, where a number of parameters have to be calibrated. The author specifies that the optimization method used on his work was not the right one because the algorithm gets stuck on local minima throughout the mathematical landscape, making it difficult to find an optimal solution and in certain cases, making the manual calibration more effective than the result given by the algorithm. And because of the complexity of an SD model, a manual parameter calibration process to find an optimal or a near-optimal solution is far from being as effective as a good optimization method. Moreover, complex non-linear systems, among which SD models can be found, cannot be solved with Simplex or other linear programming methods.

3.1.2 Optimization Problem

As stated by Steel & Torrie (1960), in statistics, the coefficient of determination, denoted by R^2 or R-Squared, indicates how well data points fit a

curve or in other words, it gives information about how well the model fits a set of observations. It is a statistic used in the context of statistical models with the purpose of testing hypotheses. This index provides a measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model.

R^2 is defined by the data set of observed values y_i each of which is associated to a modeled value f_i. The variability of the data set is measured with $SS_{tot} = \sum_i (y_i - \bar{y})^2$, where $\bar{y}$ corresponds to the mean of the n observed data values as $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. The discrepancy between the data and the estimated values thrown by the model are measures by the residual sum of squares as $SS_{res} = \sum_i (y_i - f_i)^2$. The most general definition of the coefficient of determination is $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$. The closer this value is to one, the better the model explains the observed data.

In some cases it is possible that the coefficient of determination as the fitness function is enough, but for this model in particular it leaded to unfeasible solutions. For this reason Hughes (2012) adds a new characteristic in the objective function, which is that the movie awareness peaked after the movie's initial release. With this, the final optimization problem was defined as follows, where $peakAwTest$ is a binary variable that defines if the mentioned condition is fulfilled.

$$Max \left\{ \left(1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i \left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i \right)^2} \right) + 10(peakAwTest - 1) \right.$$

Subject to model constraints defined as:

$$l_i \leq parameter_i \leq u_i$$

Where y_i is the revenue i, f_i is the revenue obtained through the model, n is the number of observations of y, l_i is the lower boundary of parameter i and u_i is the upper boundary of parameter i. The detailed parameters are not shown here because of privacy requirements from PwC.

3.1.3 Data and Parameters

All the data for all the analyzed movies were provided directly by PwC. It was proposed to perform an accurate comparison to work exclusively with all the movies from 2009 because the model was optimized only for that year in the previous work using the gradient-based algorithm. There was no additional data research during this project and PwC chose all the parameter boundaries in all the steps of the process. New changes on the optimization problem definition were discussed in meetings but most of the analysis work to come up with new constraints or new parameter boundaries to improve the feasibility of the solutions was done by PwC. This because some of the data was not possible to share and most of the expertise with movie release strategies was there.

4 The Algorithms

Beyond what has already been explained by the definition of various meta-heuristic algorithms and the comparison of some of them, it is necessary to use modifications to correctly apply them into a particular problem. The following lines explain how the different algorithms have been implemented.

4.1 PEST

PEST is a model-independent Software specially designed for parameter calibration. It was the optimization tool used in the "movie release strategies" model to estimate the parameters for each movie using the Gauss-Marquardt-Levenberg (GML) algorithm. Opposed to meta-heuristic methods, the GML algorithm uses gradients to obtain results.

As described by Skahill & Doherty (2006), and more generally, X being the action of a linear model, p its m parameters, h the n observations and the vector ε a representation of the noise associated with h, the relationship of these variables can be represented by $Xp = h + \varepsilon$. And knowing an objective function defined by $\phi = (h - Xp)^t Q (h - Xp)$, it can be shown that the minimization of that objective function for p can be calculated as $p = (X^t Q X)^{-1} X^t Q h$. When a model is non-linear, the implementation of this function becomes an iterative process, which starts with a defined set of initial parameters requiring linearization and upgrading the parameters for each iteration.

The best advantage of the GML algorithm is its speed, but it is very susceptible to find local optimality instead of global optimality.

4.2 Modified Artificial Bee Colony Algorithm for Constrained Optimization problems

The algorithm developed by Karaboga & Akay (2011) is explained as follows:

The parameters to adjust for this algorithm are the maximum number of cycles for the algorithm MCN, the size of the population sn, MR as a value in the range [0,1] which controls the number of parameters to be modified, the cost function ϕ defined here as a uniformly distributed random real number in the range of $[-1,1]$, the penalty function $penalty$ defined in this case as the number of violations to the constraints, the scout production period SPP and $limit$ as the maximum allowed number of cycles for the Scout bees phase.

The solution is presented as $\overrightarrow{x_i}$ with $i = 1,2 \dots sn$, and each solution has associated a value $failure_i$ as a counter of the times the solution x_i has not been improved, a value $violation_i$ associated with the penalty function $penalty$, a fitness value $fitness_i$ defined as the objective function of the optimization problem, $change_i$ as an auxiliary variable assigned to a solution i for the employed bees phase with initial value 0,

The number of optimization parameters is D. For every x_i, x_{min}^j is the lower bound of the parameter j and x_{max}^j the upper bound of the parameter j, with $j = 1,2 \dots D$.

With $cycle$ as the number of the current iteration number, the structure of the pseudo-code is as follows:

1. **Initialization**
2. $cycle = 1$
3. **while** $cycle \leq MCN$
    a. **Employed Bees Phase**
    b. **Calculate Probabilities for Onlookers**
    c. **Onlooker Bees Phase**
    d. **Scout Bees Phase**
    e. Memorize the best solution achieved so far
    f. $cycle = cycle + 1$
4. **end while**

4.2.1 Initialization

Being $rand\ (0,1)$ a random number between 0 and 1:

1. **for** $i = 1\ to\ sn$
    a. Generate a solution $\vec{x}_i$ as follows:
    b. **for** $j = 1\ to\ D$

i. Generate a parameter $x_i^j = x_{min}^j + rand(0,1)\left(x_{max}^j - x_{min}^j\right)$

c. end for

d. $failure_i = 0$

e. Evaluate $fitness_i$ *and* $violation_i$

2. end for

4.2.2 Employed bees phase

With R_j a uniformly distributed random real number in the range [0,1]

1. **for** $i = 1$ *to* sn

   a. $change_i$=0

   b. Produce a new food source $\vec{v}_i$ for the employed bee of the food source $\vec{x}_i$ as follows:

   c. Choose $k \neq i$ randomly from $\{1,2 \dots sn\}$

   d. **for** $j = 1$ *to* D

      i. $v_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) & if R_j < MR \\ x_{ij} & otherwise \end{cases}$

      ii. if $R_j < MR$, make $change_i = 1$

   e. end for

   f. **if** $change_i = 0$

      i. Choose j randomly from $\{1,2 \dots D\}$

      ii. $v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$

   g. end if

   h. Apply the selection process between $\vec{v}_i$ and $\vec{x}_i$ through Deb's Method.

   i. If solution $\vec{x}_i$ does not improve $failure_i = failure_i + 1$, otherwise $failure_i = 0$.

2. end for

4.2.3 Deb's Method

Deb's method works as follows:

- Any feasible solution ($violation_i \leq 0$) is preferred to any infeasible solution ($violation_i > 0$) (solution i is dominant)

- Between two feasible solutions ($violation_i \leq 0, violation_i \leq 0$), the one having better objective function value is preferred ($fitness_i \geq fitness_j$, solution i is dominant)
- Between two infeasible solutions ($violation_i > 0, violation_i > 0$), the one having smaller constraint violation is preferred ($violation_i < violation_j$, solution i is dominant).

4.2.4 Calculate Probabilities for Onlookers

With the probability values for the solutions p_i:

1. **for** $i = 1$ *to* sn
   a. Calculate $p_i = \begin{cases} 0.5 + 0.5\left(\frac{fitness_i}{\sum_{j=1}^{sn} fitness_i}\right) & if\ solution\ is\ feasible \\ 0.5 - 0.5\left(\frac{violation_i}{\sum_{j=1}^{sn} violation_i}\right) & if\ solution\ is\ infeasible \end{cases}$
2. **endfor**

4.2.5 Onlooker bees Phase

1. $t = 0, i = 1$
2. **while** $t \leq sn$
   a. **if** $rand(0,1) < p_i$ **then**
      i. $t = t + 1$
      ii. Produce a new food source $\vec{v}_i$ for the onlooker bee of the food source $\vec{x}_i$ as follows:
      iii. Choose $k \neq i$ randomly from $\{1,2, \dots, sn\}$
      iv. **for** $j = 1$ *to* D
         1. Calculate $v_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) & if R_j < MR \\ x_{ij} & otherwise \end{cases}$
      v. end for
      vi. Apply the selection process between $\vec{v}_i$ and $\vec{x}_i$ through Deb's Method.
      vii. If solution $\vec{x}_i$ does not improve $failure_i = failure_i + 1$, otherwise $failure_i = 0$.
   b. **end if**
   c. $i = i + 1$
   d. $i = i\ mod\ (sn + 1)$
3. **end while**

4.2.6 Scout bees phase

1. **if** $cycle\ mod\ SPP = 0\ And\ \max(failure_i) > limit$ **then**
    a. Replace $\vec{x}_i$ with a new randomly produced solution $x_i^j = x_{min}^j + rand(0,1)\left(x_{max}^j - x_{min}^j\right)$
2. **end if**

4.3 Genetic Sampler

Genetic Sampler (GS) is a new Real-parameter Genetic Algorithm that successfully characterizes the parameter space by locating multiple unconnected optimal regions. Ballester & Carter (2006) use this algorithm to characterize the parameter space of an inverse problem.

In GS, two parents are selected from the current population of size N to produce λ children through the crossover operator. The objective function value associated with each child is thereafter evaluated. Offspring and current population are then combined so that the population remains at a constant size through the replacement operator. Selection, crossover, fitness evaluation and replacement form a GA iteration (as seen in the table 3). Even though genetic algorithms are generally related to binary coding, it is possible to use real coding to solve continuous search space problems and this is what GS does. More specifically, the algorithm works as follows:

The GS contains five tunable parameters: the population size N (which remains constant always), the amount of children to produce through the crossover operator λ, the number of individuals that competes against the offspring for a place in the population NREP, the maximum number of cycles for the algorithm MCN and η is a tunable parameter that defines how concentrated the search must be around the parents (with a high η, more concentration). In particular for GS, there is no mutation operation, and only the crossover operation is performed to obtain a new generation of the population.

As presented in Deb & Agrawal (1995), in real-coded GAs, the variables are directly used. In binary-coded GAs, the string length must be chosen to achieve certain precision. The more precision required, the larger the string length increasing then the population, which increases the computational complexity. For

this particular algorithm, real-coded GAs is used so the string is composed by the solution itself. The pseudo-code presented here is based on the global optimization because of its characteristics, it is not able to effectively find optimal solutions when constraints are added.

The pseudo-code:

1. Initialization
2. **for** $i = 1\ to\ MCN$
    a. Selection
    b. Crossovers
    c. Replacement
3. **end for**

4.3.1 Initialization

Being D the number of optimization parameters and $rand(0,1)$ a random value in the range [0,1]:

1. Create the initial population as follows:
2. **for** $i = 1\ to\ N$
    a. Generate a chromosome $\vec{x}_i$ as follows:
    b. **for** $j = 1\ to\ D$
        i. Generate a parameter $x_i^j = x_{min}^j + rand(0,1)\left(x_{max}^j - x_{min}^j\right)$
    c. end for
3. **end for**

4.3.2 Selection

1. Select k,l as random integer numbers from the range $[1,2,\dots,N]$ with $k \neq l$
2. $Parent_k, Parent_l$ are selected from the population.

4.3.3 Crossovers

A version of the Simulated Binary Crossover (vSBX) is used as stated in Ballester & Carter (2004), with $x_j^{(i)}$is the j^{th} component of the i^{th} parent. $\vec{x}^{ofp}$is the offspring.

1. **for** i = 1 to λ
    a. Calculate $w = rand(0,1)$
    b. **for** $j = 1\ to\ D$

i. Calculate $u_j = rand(0,1)$

ii. Calculate $\beta_j = \begin{cases} \left(\frac{1}{2u_j}\right)^{\frac{1}{\eta+1}} & 0 < u_j \leq 0.5 \\ \left(\frac{1}{2(1-u_j)}\right)^{\frac{1}{\eta+1}} & 0.5 < u_j \leq 1 \end{cases}$

iii. $y_j = \begin{cases} \frac{1}{2}\left(\left(1+\beta_j\right)x_j^{(1)} + \left(1-\beta_j\right)x_j^{(2)}\right) & 0 < u_j \leq 0.5 \\ \frac{1}{2}\left(\left(3-\beta_j\right)x_j^{(1)} + \left(1-\beta_j\right)x_j^{(2)}\right) & 0.5 < u_j \leq 1 \end{cases} \quad w < 0.5$

$y_j =$

$\begin{cases} \frac{1}{2}\left(\left(1-\beta_j\right)x_j^{(1)} + \left(1+\beta_j\right)x_j^{(2)}\right) & 0 < u_j \leq 0.5 \\ \frac{1}{2}\left(-\left(1-\beta_j\right)x_j^{(1)} + \left(3-\beta_j\right)x_j^{(2)}\right) & 0.5 < u_j \leq 1 \end{cases} \quad w \geq 0.5$

c. **end for**

d. $\vec{x}^{ofp} = \vec{y}$

2. **end for**

4.3.4 Replacement

1. Select *NREP* individuals $\vec{x}^{cst}$ randomly from the current population
2. Define f_{best} as the best function value of the best individual in the offspring and the NREP individuals
3. $\vec{x}^{cst}$ and $\vec{x}^{ofp}$ compete for a place in the population according to the following surviving likelihoods: $p\left(\vec{x}^{ofp}\right) = \frac{f\left(\vec{x}^{ofp}\right) - f_{best}}{f\left(\vec{x}^{ofp}\right) + f\left(\vec{x}^{cst}\right) - 2f_{best}}$, $p\left(\vec{x}^{cst}\right) = \frac{f\left(\vec{x}^{cst}\right) - f_{best}}{f\left(\vec{x}^{ofp}\right) + f\left(\vec{x}^{cst}\right) - 2f_{best}}$

4.4 Modified Cuckoo Search

Walton et al. (2011) proposes a modified cuckoo search algorithm, which according to classical benchmark functions gives overall better results. This is what is going to be used to apply it in the movie model.

The modification presents two modifications

1. The size of the step size in CS is constant and in this modification the value of the step size decreases when the number of generations increases, allowing more localized searching, as the eggs get closer to the solution.

The initial value is a step size A, and each generation, a new Lévy flight step is calculated as $A/\sqrt{G}$, where G is the generation number.

2. Adding information exchange between eggs to speed up convergence. A fraction of the eggs with best fitness are put into a group of top eggs. For each of the top eggs, a second egg in this group is picked at random and a new egg is then generated on the line connecting these two top eggs. The new egg is located at a distance equal to the inverse of the golden ratio $\frac{1+\sqrt{5}}{2}$, so it is closer to the egg with best fitness. If both eggs have same fitness, the new egg is place in the midpoint. If the same egg is picked twice, a local Lévy flight with step size A/G^2 starting from the picked nest.

There are two parameters: the fraction of nests to be abandoned and the fraction that are put in the top nests. The authors propose to use 0.75 and 0.25 respectively. Deb's rules for constraint violations are added to this modification.

4.4.1 The algorithm

A nest x_i is represented by the string of solution parameters. The first step is to define the tunable parameters of the algorithm. These are the maximum Lévy step size A, the fraction of nests put in the top nests p_t, the fraction of nests to be abandoned p_a, the maximum number of evaluations MNE and the population size n (number of nests). The golden ratio is defined as $\varphi = \frac{1+\sqrt{5}}{2}$ and the penalty function $violation_i$ is defined as the number of constraint violations of the solution i.

The pseudo-code is constructed as follows:

Being G the number of the iteration

1. Initialization
2. **for** $Evaluation = 1\ to\ MNE$
    a. $G = G + 1$
    b. Sort nests by order of fitness
    c. Abandoning Phase
    d. Top Nests Phase
3. **end for**

4.4.2 Initialization

Being D the number of parameters of the solution, x_{min}^{j} the lower bound of the parameter j and x_{max}^{j} the upper bound of the parameter j

1. $G = 0$
2. **for** $i = 1\ to\ n$
    a. **for** $j = 1\ to\ D$
        i. $x_i^j = x_{min}^j + rand(0,1)(x_{max}^j - x_{min}^j)$
    b. end for
    c. Calculate fitness $F_i = f(x_i)$
    d. Calculate $violation_i$
3. **end for**

4.4.3 Abandoning Phase

1. **for** $i = 1\ to\ n * p_a$
    a. Calculate Lévy Flight step size $\propto = {}^{A}/_{\sqrt{G}}$
    b. Perform Lévy flight from x_i to generate new egg x_k
    c. $x_i = x_k$
    d. Calculate fitness $F_i = f(x_i)$ and $violation_i$
2. **end for**

4.4.4 Top nests phase

1. **for** $i = n - n * p_t\ to\ n$
    a. Pick j as a random integer value in the range $[n - n * p_t, \dots, n]$
    b. **if** $(x_i = x_j)$
        i. Calculate Lévy Flight step size $\propto = {}^{A}/_{G^2}$
        ii. Perform Lévy flight from x_i to generate new egg x_k
        iii. $F_k = f(x_k)$, Calculate $violation_k$
        iv. Pick l as a random integer value in the range [0,…,n]
        v. **if** ($(violation_k = violation_l\ and\ F_k > F_l)\ or\ violation_k < violation_l$)
            1. $x_l = x_k$
            2. Calculate fitness $F_l = f(x_l)$ and violation_l

vi. **end if**

c. **else**

i. $dx = \frac{|x_i - x_j|}{\varphi}$

ii. Move distance dx from the worst nest to the best nest to find x_k considering Deb's rules.

iii. $F_k = f(x_k)$, Calculate $violation_k$

iv. Pick l as a random integer value in the range [0,…,n]

v. **if** ($(violation_k = violation_l \; and \; F_k > F_l) \; or \; violation_k < violation_l$)

1. $x_l = x_k$
2. Calculate fitness $F_l = f(x_l) \; and \; violation_l$

vi. **end if**

d. **end if**

2. **end for**

4.4.5 Lévy Flight

The modified CS uses a simplification of the Lévy Flight. Being $randn$ a Gaussian distributed value in the range [0 1], the function returns $newx_i$ with x_i and $\propto$ as inputs.

1. **for** $i = 1 \; to \; D$

a. $newx_{ij} = x_{ij} + \propto randn$

2. **end for**

5 Research Results

The optimization process had to be applied to more than 150 movies. And it is only after applying the parameter calibration to all the movies that it is possible to make new conclusions about the model and to define new characteristics of the optimization problem. It is also a matter of common sense to understand that being this an SD model, the parameter values that optimize the R-Squared value are not necessarily the parameter values that exist in real life, and for that reason it is reasonable to assume that getting a parameter set that defines a very high R-Square (if possible) is enough to accept that the model effectively represents the data. This means that it is not necessary to look for the best existent solution, but for a solution that is good enough. This allows some flexibility in terms of reducing the amount of iterations the optimization algorithm needs and strengthens the the fact that meta-heuristic algorithms only assure the finding of near-optimal solutions.

If the idea is to learn about the model and to improve the optimization problem to get better and more feasible solutions, and knowing that more than 150 movies have to be optimized, it is needed that the algorithms run fairly fast, getting good solutions in a small amount of time. The first step was then, after the literature review and selection of 3 suitable algorithms for the problem, to run these algorithms with a maximum of iterations using mathematical functions similar to the movie model problem with known optimal solution:

$$\begin{aligned} &max && f(x) \\ &subject\ to && l_i < x_i < u_i \quad for\ i = 1, \dots, n \end{aligned}$$

Validation of the algorithms and the optimization are done through testing with mathematical benchmark functions, meetings with the client where the problem definition was improved and data analysis covered by PwC.

The functions are taken from Molga & Smutnicki (2005) and the results are shown in this work.

5.1 Benchmark Functions

The functions tested are limited to a maximum of 2000 iterations because the objective is to find good results in a very short time. It has to be taken into consideration that an evaluation of the mathematical function is much faster than the evaluation of an SD model. 2000 iterations assure that all the movies can be optimized in less than 24 hours. It has to be noted also that all the algorithms get very near to the optimal solution in all cases at some point over 2000 iterations. The following results don't show that, but they show how the algorithms evolve in terms of fitness for each iteration.

5.1.1 De Jong's Function

This function was tested with two parameters, with the test area restricted to $-5.12 \leq x \leq 5.12, -5.12 \leq y \leq 5.12$. The global minimum $f(x) = 0$ is obtainable for $x = 0, y = 0$. Figure 1 presents the results for all the algorithms.

$$f(x, y) = x^2 + y^2$$

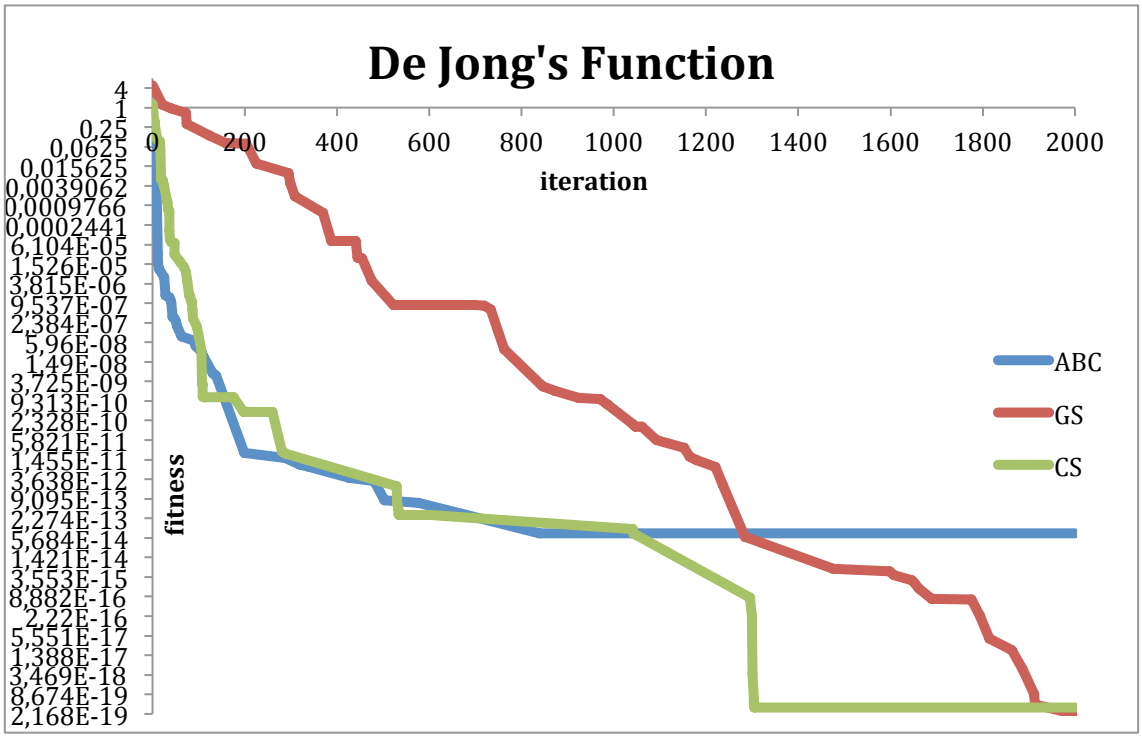

Figure 1: De Jong's Function optimization

5.1.2 Axis Parallel Hyper-Ellipsoid Function

This function was tested with two parameters, with the test area restricted to $-5.12 \leq x \leq 5.12, -5.12 \leq y \leq 5.12$. The global minimum $f(x) = 0$ is obtainable for $x = 0, y = 0$. Figure 2 presents the results for all the algorithms.

$$f(x, y) = x^2 + 2y^2$$

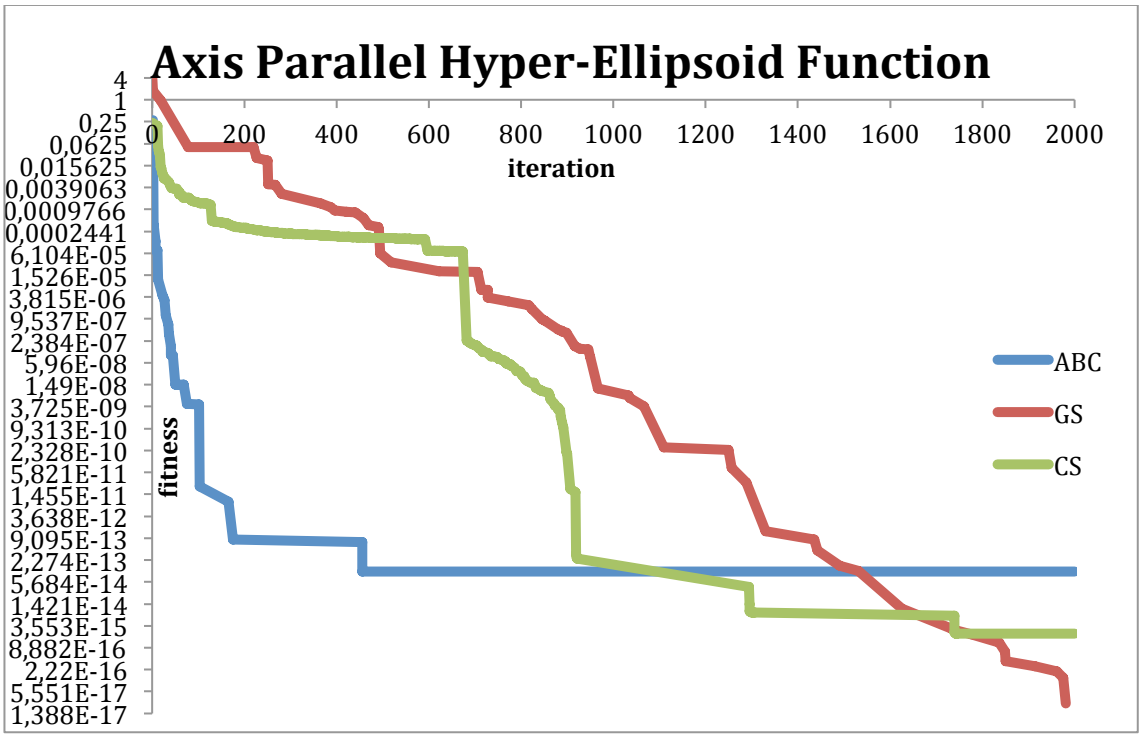

Figure 2: Axis Parallel Hyper-Ellipsoid function optimization

5.1.3 Rosenbrock's Valley

This function was tested with two parameters, with the test area restricted to $-2 \leq x \leq 2, -2 \leq y \leq 2$. The global minimum $f(x) = 0$ is obtainable for $x = 1, y = 1$. Figure 3 presents the results for all the algorithms.

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

5.1.4 Rastringin's Function

This function was tested with two parameters, with the test area restricted to $-5.12 \leq x \leq 5.12, -5.12 \leq y \leq 5.12$. The global minimum $f(x) = 0$ is obtainable for $x = 0, y = 0$. Figure 4 presents the results for all the algorithms.

$$f(x, y) = 20 + [x^2 - 10\cos(2\pi x)] + [y^2 - 10\cos(2\pi y)]$$

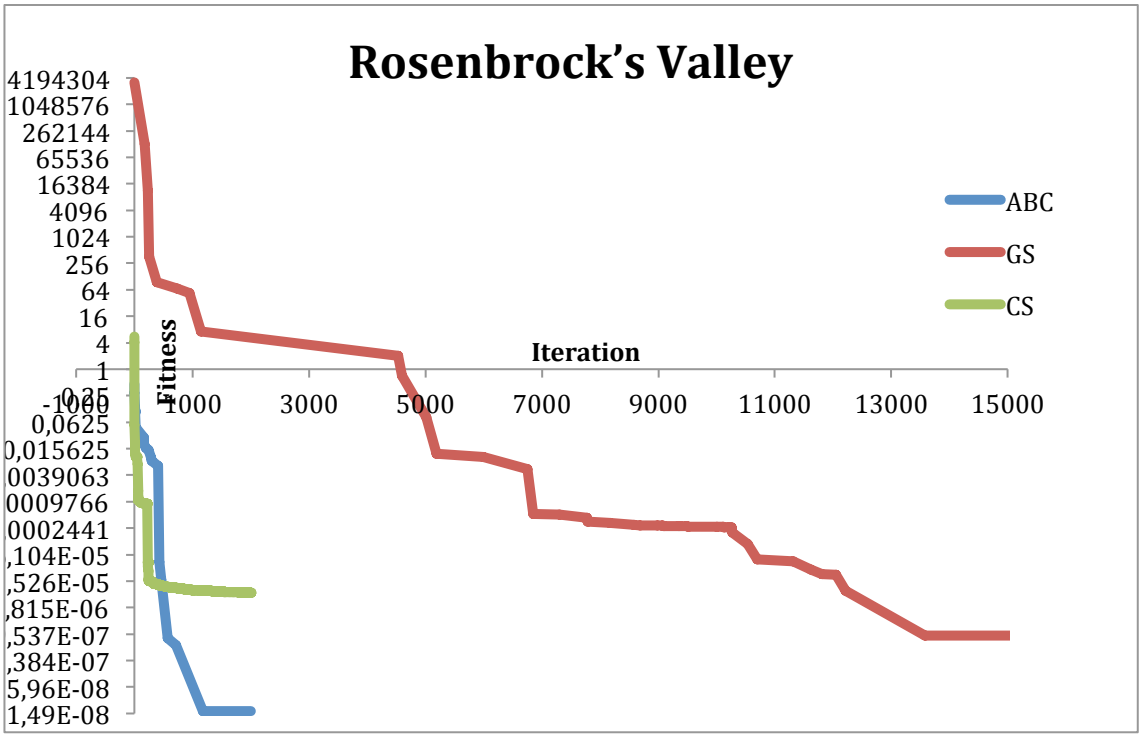

Figure 3: Rosenbrock's Valley optimization

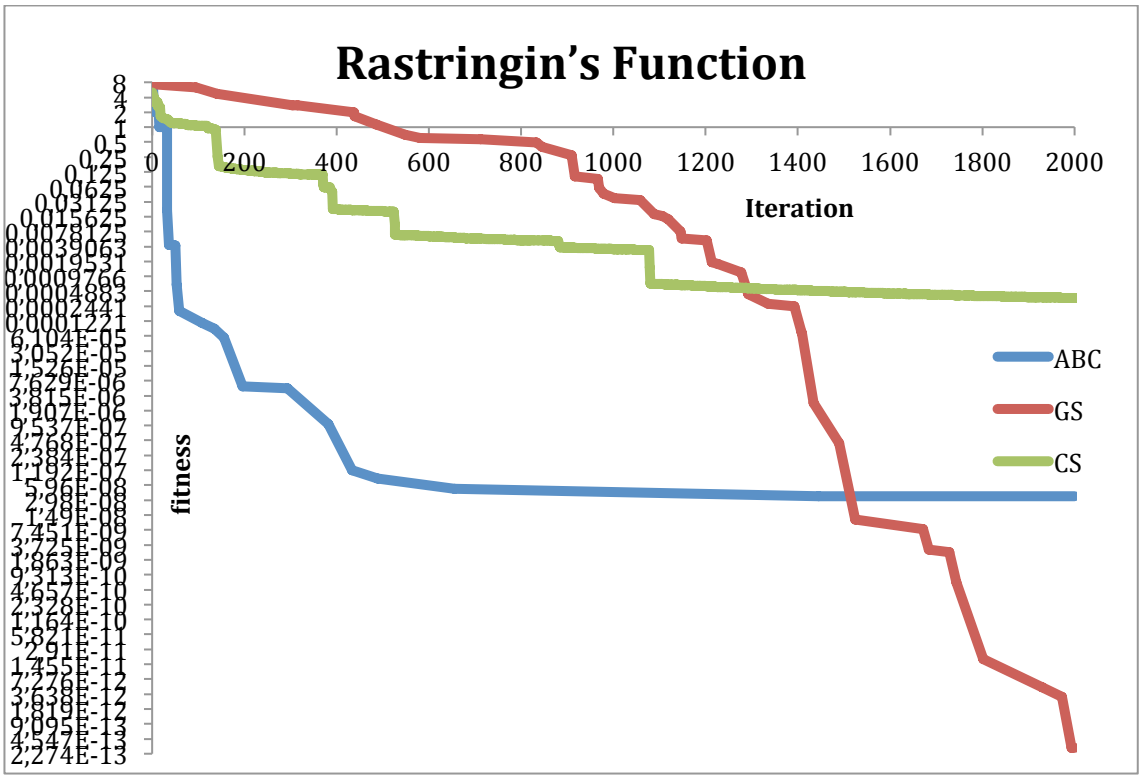

Figure 4: Rastringin's Function optimization

5.1.5 Schwefel's Function

This function was tested with two parameters, with the test area restricted to $-500 \leq x \leq 500, -500 \leq y \leq 500$. The global minimum $f(x) = -837.9658$ is

obtainable for $x = 420.9687$ $y = 420.9687$. Figure 5 presents the results for all the algorithms.

$$f(x,y) = \sum_{i=1}^{n} \left[-x_i \sin\left(\sqrt{|x_i|}\right) \right]$$

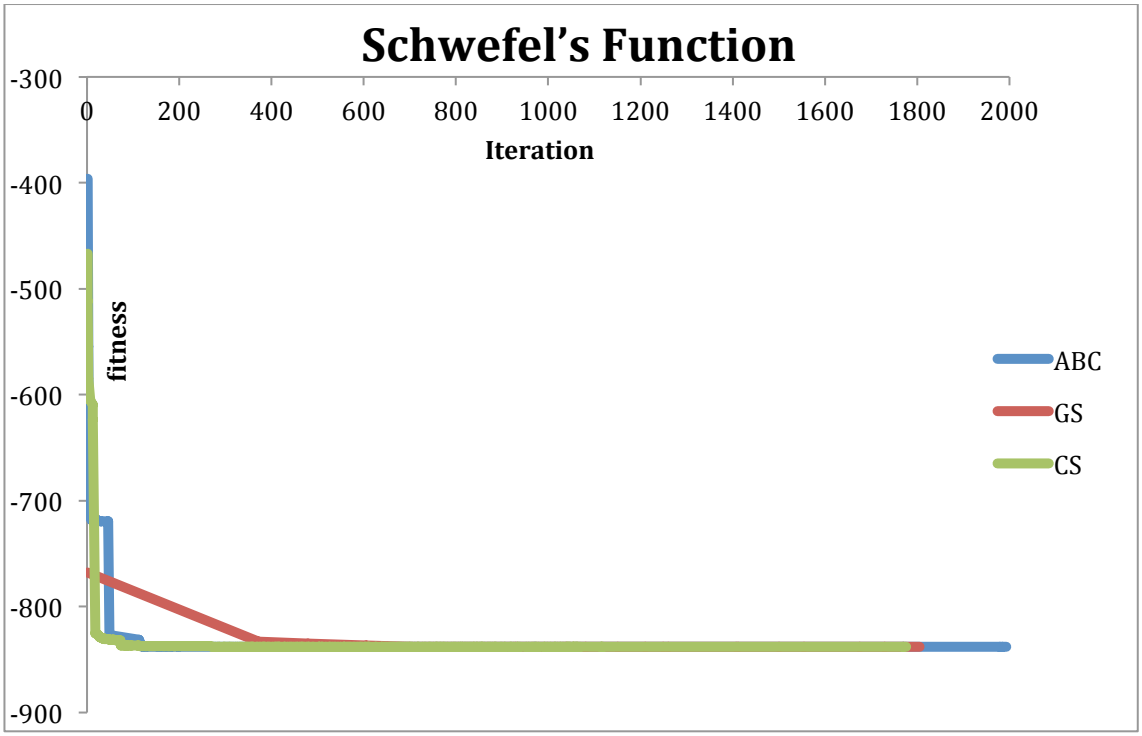

Figure 5: Schwefel's Function optimization

5.1.6 Analysis

All the algorithms give very good results in the first 2000 iterations. It can be seen that in general GS has a slower convergence but arrives to a better result in the long term. In particular for Rosenbrock's Valley, it doesn't converge well in the first 2000 iterations, but it converges to the solution in a longer time. ABC and CS are quite competitive and at this point, it seems like a good idea to test one of the algorithms with the movie model to check the results. The GS was chosen to perform the first movie model analysis using the original optimization problem definition.

5.2 Genetic Sampler into the movie model

The GS was tested for all the movies of 2009 with the following results compared to PEST (Table 4):

	Genetic Sampler	PEST
median	1,00	0,98
mean	0,99	0,91
min	0,80	0,05
Movies with r-squared over 0.75	1,00	91%

Table 4: GS vs PEST for the original optimization problem

These results show an amazing improvement over the PEST implementation, meaning that the GS was able to explore the search space more and find better solutions for 100% of the movies. It has to be considered that Hughes (2012) has artificially chosen an initial condition for the algorithm that forced it to get stuck in local optima, but with feasible results. With the GS and with this simulation run, a feasibility dilemma was shown. For example, according to the optimization, 100% of the US audience thought that the "Fast and Furious" movie was an appropriate movie for their tastes (theme acceptability parameter).

New restrictions were then needed to solve this problem, changing the optimization problem to the following one with d parameters and m restrictions:

$$\begin{aligned} &max && f(\vec{x}) \\ &subject\ to && l_i < x_i < u_i \quad for\ i = 1, \dots, d \\ & && h_j(\vec{x}) \geq 0 \quad for\ j = 1, \dots, m \end{aligned}$$

PwC expressed the fact that due to these results, and according to the existent data, it was important to add the concept that the intention to view peaks after the movie's initial release, often almost doubled between the first and second week. To add this idea, the following restriction was added where pctIWV is the percentage of people who intended to view that actually went to see the movie in a defined week:

$$h_1(\vec{x}) = pctIWV(week\ 1) - pctIWV(week\ 2) \geq 0$$

This change required modifications on the algorithms to be able to handle restrictions of the form $h_j(\vec{x}) \geq 0$. The algorithms were changed adding the Deb's rule for constraints violations mentioned in the algorithms descriptions. This rule was tested on ABC in previous research (Karaboga & Basturk 2007), but not in CS, which could be an interesting test for a new modification for that algorithm. The GS doesn't have a version for constraint optimization so the application of Deb's

rule is also new.

5.3 Benchmark Functions for Constrained Optimization

All the benchmark functions tested are taken from Runarsson & Yao (2000)

5.3.1 Function 1

The following functions has the bounds $0 \le x_i \le 1 (i = 1, \dots, 9), 0 \le x_i \le 100$ $(i = 10,11,12)$ and $0 \le x_{13} \le 1$. The global minimum is at f(x)=-15 with x=(1,1,1,1,1,1,1,1,1,3,3,3,1). Figure 6 presents the results for all the algorithms.

Minimize:

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

Subject to:

$$\begin{gathered} g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0 \\ g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0 \\ g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0 \\ -8x_1 + x_{10} \le 0 \\ -8x_2 + x_{11} \le 0 \\ -8x_3 + x_{12} \le 0 \\ -2x_4 - x_5 + x_{10} \le 0 \\ -2x_6 - x_7 + x_{11} \le 0 \\ -2x_8 - x_9 + x_{12} \le 0 \end{gathered}$$

5.3.2 Function 2

The following function has the bounds in $78 \le x_1 \le 102, 33 \le x_2 \le 45$ and $27 \le x_i \le 45$ $(i = 3,4,5)$. The optimum solution is $x = (78, 33, 29.995256025682, 45, 36.775812905788)$ with $f(x) = -30665.539$. Figure 7 presents the results for all the algorithms.

Minimize:

$$f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

Subject to:

$$\begin{gathered} g_1(x) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \le 0 \\ g_2(x) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \le 0 \\ g_3(x) = -80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \le 0 \\ g_4(x) = 80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \le 0 \\ g_5(x) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \le 0 \\ g_6(x) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \le 0 \end{gathered}$$

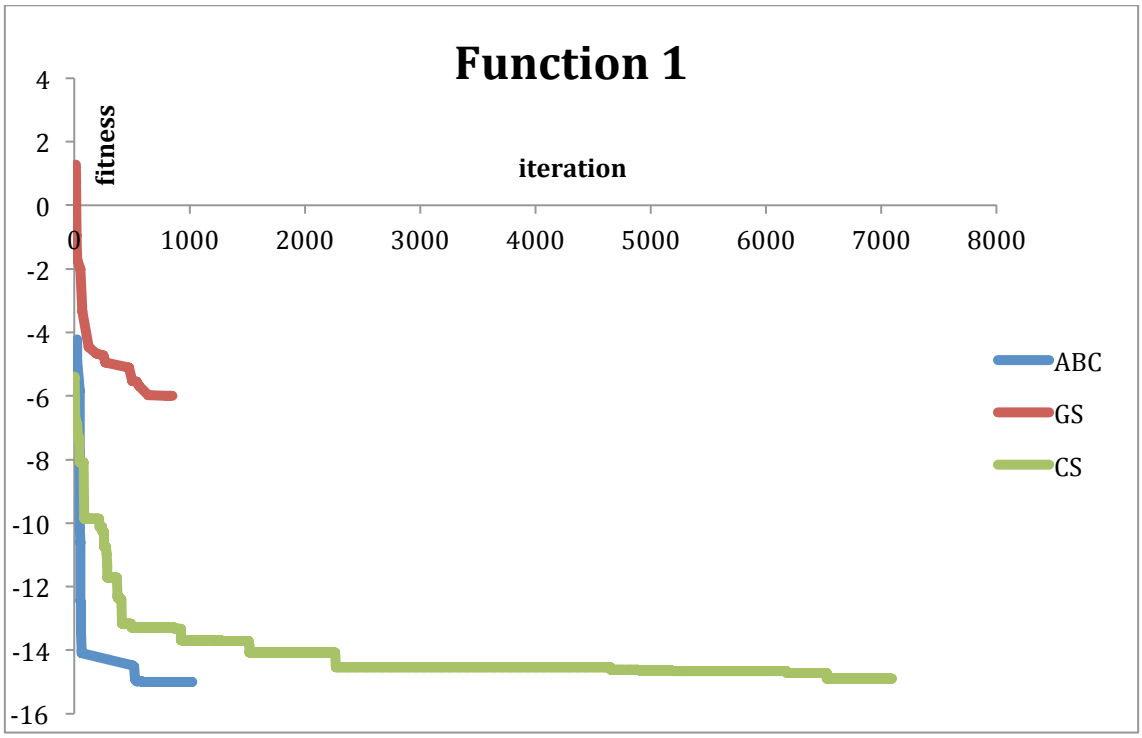

Figure 6: Constrained function optimization

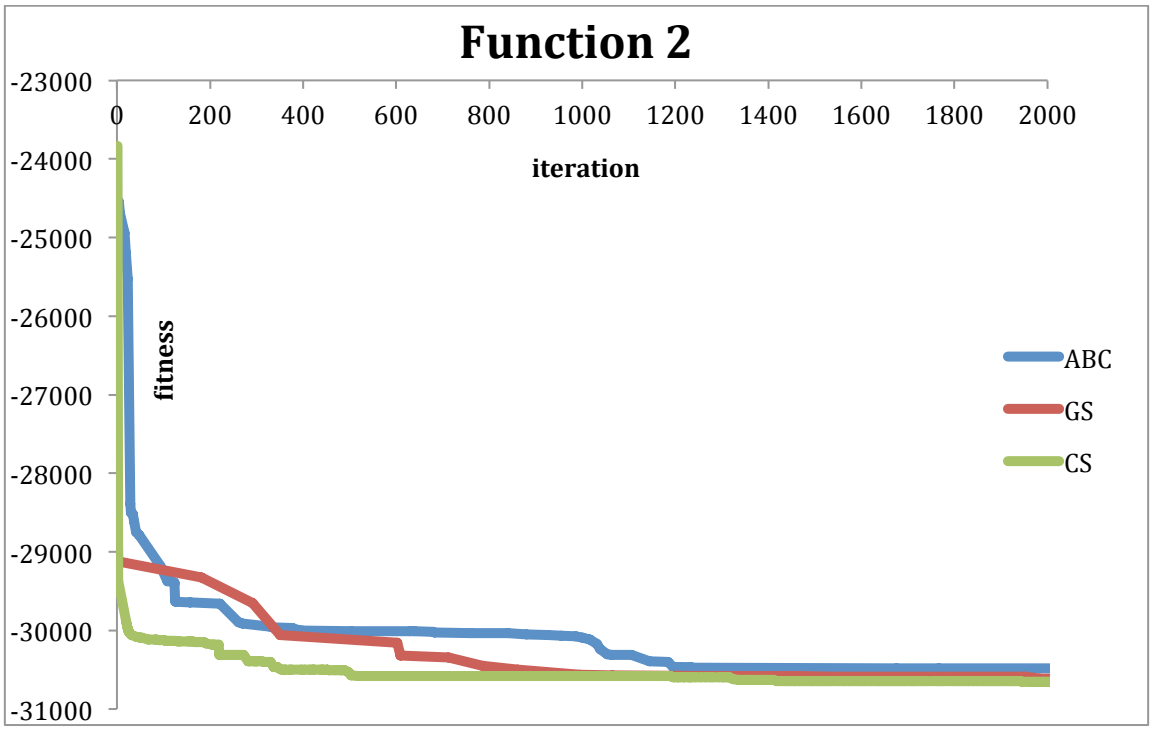

Figure 7: Constrained Function Optimization

5.3.3 Function 3

The following equation has the boundaries in $13 \leq x_1 \leq 100, 0 \leq x_2 \leq 100$, with the optimum at x=(14.095, 0.84296) where f(x)=-6961.81388. Figure 8

presents the results for all the algorithms. GS fails to find a feasible solution unless a feasible solution is randomly chosen in the initialization.

Minimize

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$

subject to:

$$g_1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$
$$g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$$

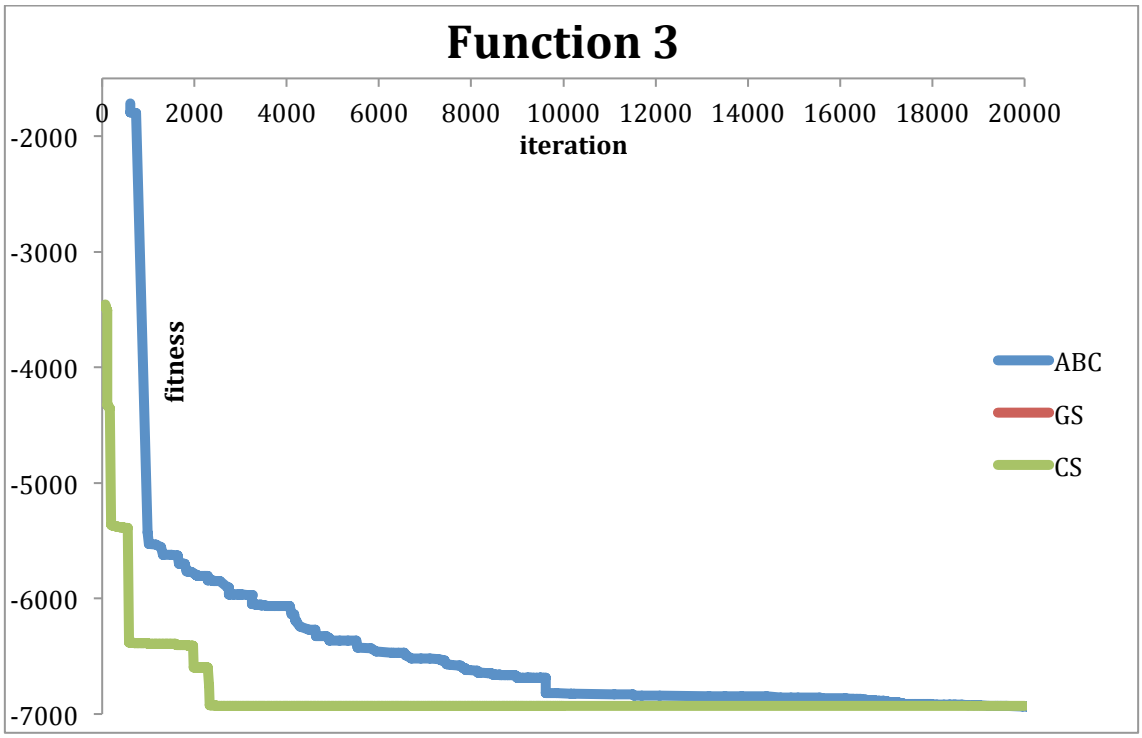

Figure 8: Constrained Function Optimization

5.3.4 Analysis

GS fails to solve the optimization problems. This is understandable due to the fact that the algorithm was initially not meant to handle restrictions and was designed to solve global optimization problems. The addition of Deb's rules to handle restrictions doesn't work well with this algorithm mainly because the crossover operation doesn't take into consideration the fitness function, hence it can't take into consideration the restrictions. This leads to unfeasible results most of the time. More research is needed if there is any interest in making this algorithm work for constrained optimization problems, but it's not the goal of this research to do that.

On the other hand, ABC and CS are able to converge to a solution and look quite competitive.

5.4 New Algorithms in the movie model

5.4.1 First run

The ABC algorithm was applied to the movie with the new restriction and run again for all 2009 movies. The optimization resulted in extremely high values of $pctIWV$ for week one, with values of 100% in some cases. Also, things didn't change with the other parameters in relation with the previous run. For that reason a new analysis was made to improve the restrictions.

In the first place, because of lack of data, the audience flow patterns were restricted instead of directly restricting the playability (the level of interest the movie generates after it has been released) and the marketability (the level of interest the movie generates before it is released). Figure 9 shows this pattern where it can be seen how the percentage of awareness (meaning the percentage of the population that is aware of the existence of the movie) changes with the money spent in marketing. The restriction associated to this was chose as:

$$h_2(\vec{x}) = 0.08\left(\frac{marketingBudget}{10000000}\right) + 0.25 - percentage\ of\ awareness \geq 0$$

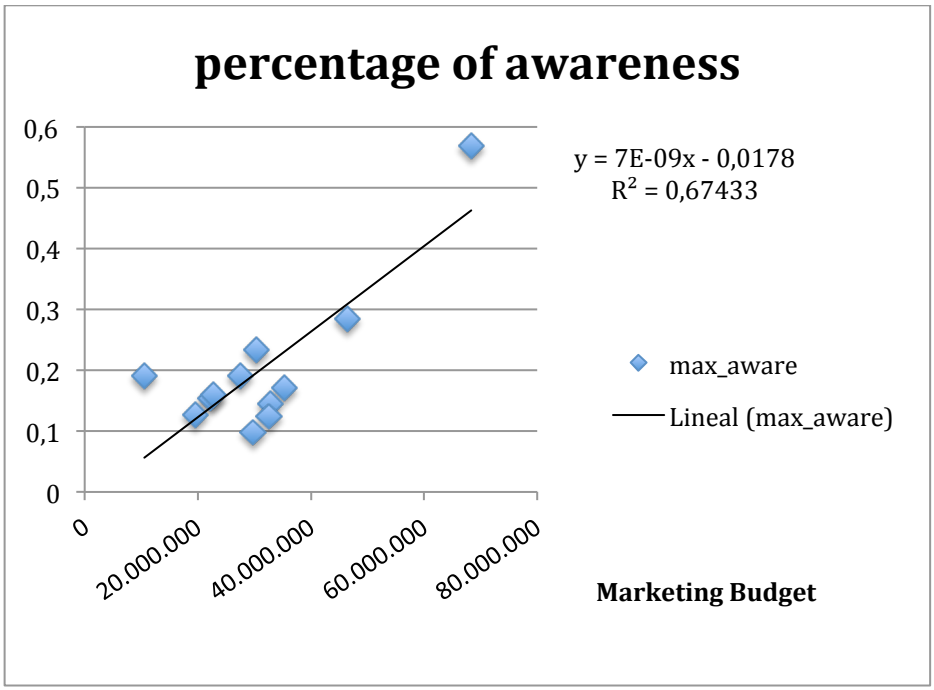

Figure 9: Percentage of awareness vs Marketing Budget

Due to the high values of theme acceptability resulting from the optimization, the value was restricted to a maximum 0.65. Leaving the following new constraint boundaries:

$$0.65 \geq theme\ acceptability \geq 0$$

5.4.2 Second Run

The results obtained with the new restrictions where described by the client as "dramatically better" than PEST, but the playability, marketability and theme acceptance parameters were still unrealistic. The following results were obtained:

	box_rSq	Peak Aware	Wide Release	Platform Release	Limited Release
MIN	0.5586	0.0296	0.5586	0.9223	0.8879
MEAN	0.9855	0.1748	0.9881	0.9781	0.8920
MAX	0.9996	0.6566	0.9996	0.9991	0.8960
COUNT	154	154	131	21	2
rSq over 0.9	0.9675		0.9771	1	0
rSq over 0.8	0.9870		0.9847	1	1

Table 5: Optimization run, ABC algorithm

It was proposed after this batch to add new characteristics to the optimization problem. To avoid marketability and playability to have unfeasible values, it was decided to use the rotten tomato values available at http://www.rottentomatoes.com/. Meaning that marketability was fixed to the rotten tomato critics and playability to the rotten tomato user. Also some of the parameters with innocuous effect were fixed and others had their range increased. Also the $pctIWV$ values were limited to 0.65 for the first week and 0.4 for the second week resulting in the following new restrictions:

$$h_3(\vec{x}) = 0.65 - pctIWV(week\ 1) \geq 0$$
$$h_4(\vec{x}) = 0.4 - pctIWV(week\ 2) \geq 0$$

5.4.3 Third Run

ABC algorithm was run with the described restrictions and had better results than PEST and also was able to achieve total feasible results. The modified Cuckoo Search algorithm was also run to compare ABC, CS and PEST. GS was not tried because of its failure trying to solve the benchmark functions with restrictions. The comparison of CS, ABC and PEST algorithms are shown in table 6 and 7. CS and ABC were run for the 154 movies from 2009 obtaining clear better results for the ABC algorithm.

	CS	ABC	PEST
median	0,97	0,99	0,98
mean	0,90	0,93	0,91
min	-0,13	-0,24	0,05
Movies with r-squared over 0.75	90%	92,81%	91%

Table 6: Final results and final optimization problem definition: CS, ABC and PEST comparison

	Number of Movies
CS has better performance:	11
ABC has better performance:	131
Both have a performance over 0.999	12
Total number of movies	154

Table 7: CS and ABC comparison over 154 movies

6 Discussions / Conclusions

This work begun as an attempt to find new valuable methods to solve the parameter calibration problem usually present in SD models, using modern evolutionary algorithms in a transparent way and compare them to other optimization packages present in the market. The use of self-made Software allows a deeper analysis and a customization for the particular problem; meaning that there are no limits in the ways it can be implemented.

Previous work on the movie model was done using AnyLogic™ as the modeling Software and PEST as the optimization tool. AnyLogic™ is and advanced Software that contains as a possible additional package an optimization tool called "OptQuest". This tool, as powerful as it can be, works as a black box and for that reason it has certain limitations. The movie model uses three different types of movie releases within 154 movies for the year 2009 and the test required reading the data and optimizing all those movies to assure that the model effectively works for any kind of release. According to the PwC, the limitations of OptQuest made impossible to work with this model because the execution speed was extremely low and it didn't allow a single procedure in which all the movies could be optimized together. And for that reason, PEST was chosen. It didn't solve the speed issue but it solved the fact that it was able to run all the movies together. Nevertheless, the low speed using OptQuest or PEST was due to the fact that AnyLogic needed to load various Excel files each time the model was executed and it wasn't necessarily a problem with the algorithms themselves. This issue was solved on this work increasing this way the optimization speed dramatically.

During the implementation of the evolutionary algorithms, the model itself was modified in order to run more efficiently and as fast as possible, allowing the optimization of all 154 movies in less than one day, which was tenths times faster than the PEST implementation. This allowed to not only find a better and more suitable algorithm for the problem at hand, but also to make several runs to be able to analyze the outputs and to refine the optimization problem definition with its fitness function and its restrictions.

6.1 The Optimization

One of the most difficult things of any optimization problem is to appropriately define it, including the fitness function, the restrictions and the boundaries of the parameters that have to be tuned or estimated. During this work, the process of defining the problem is shown with a successful outcome. With PEST optimization, analyzing the results was very difficult because of the time it took for the optimization to run, and the author of that work used additional objective functions to test results instead of using constraints. According to PEST documentation and the GML algorithm, the only constraints allowed are the parameter's lower and upper boundaries. It was not possible to add more sophisticated constraints. The EA's applied in this work are an improvement also in that sense.

This research explored three modern optimization algorithms for the parameter calibration of an SD model, and during this work it was also possible to test these algorithms for global and constrained optimization problems.

GS showed very good results for global optimization but failed when Deb's rules for constrained optimization were added. To handle constraints, other versions of Genetic Algorithms should be therefore used. The CS and the ABC algorithms showed a very competitive performance for the optimization of not only the benchmark functions, but also for the movie model, meaning that they can be suitable to use in other models when high levels of customization are required. Overall, ABC showed the best performance in most of the movies analyzed.

6.2 Limitations and Future Research

Unfortunately, due to the limitations of the *OptQuest* tool, it was not possible to test it against the ABC. It remains to be studied if the OptQuest tool is better than the ABC or other algorithms when it comes to calibrate the parameters of a model. AnyLogic limits the development slightly, not allowing the creation of user interfaces for custom experiments, meaning that the algorithms cannot be used directly in other models and have to be reprogrammed and customized for any new problem.

7 References

Alkallak, I. (2012). *A Hybrid Algorithm from Cuckoo Search Method with N-Queens Problem. Raf. J. of Comp. & Math's, Vol. 9 No. 2, 2012*

Asmussen, S. (2003). *Applied probability and queues* (Vol. 2). New York: Springer.

Ballester, P. J., & Carter, J. N. (2004, January). An effective real-parameter genetic algorithm with parent centric normal crossover for multimodal optimisation. In *Genetic and Evolutionary Computation–GECCO 2004* (pp. 901-913). Springer Berlin Heidelberg.

Ballester, P. J., & Carter, J. N. (2006). Characterising the parameter space of a highly nonlinear inverse problem. *Inverse Problems in Science and Engineering*, *14*(2), 171-191.

Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an international journal*, *8*(2), 239-287.

Bolaji, A. L., Khader, A. T., Al-Betar, M. A., & Awadallah, M. A. *Artificial bee colony algorithm, its variants and applications: A survey,"Journal of Theoretical and Applied Information Technology, 47(2) Pages 434-459, January 2013.*

Brownlee, J. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. Jason Brownlee.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1997). A new rank based version of the Ant System. A computational study.

Civicioglu, P., & Besdok, E. (2011). A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review*, 1-32.

Deb, K., & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems* 9 115-148.

Dorigo, M., Colorni, A., & Maniezzo, V. (1991, December). Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life* (Vol. 142, pp. 134-142).

Glover, F. (1989). Tabu search—part I. *ORSA Journal on computing*, *1*(3), 190-206.

Glover, F. (1990). Tabu search—part II. *ORSA Journal on computing*, *2*(1), 4-32.

Graham, A. K. (1976). Parameter formulation and estimation in system dynamics models. In *The System Dynamics Method, The Proceedings of the 1976 International Conference on System Dynamics, Geilo, Norway, August*(Vol. 8, No. 15, pp. 541-580).

Haro, F., & Torres, M. (2006, October). A comparison of path planning algorithms for omni-directional robots in dynamic environments. In *Robotics Symposium, 2006. LARS'06. IEEE 3rd Latin American* (pp. 18-25). IEEE.

Hegerty, B., Hung, C. C., & Kasprak, K. A (2009). Comparative Study on Differential Evolution and Genetic Algorithms for Some Combinatorial Problems. Available for free at: http://www.micai.org/2009/proceedings/complementary/cd/ws-imso/88/paper88.micai09.pdf

Hu, X. M., Zhang, J., & Li, Y. (2008). Orthogonal methods based ant colony search for solving continuous optimization problems. *Journal of Computer Science and Technology*, *23*(1), 2-18.

Hughes, R. (2012), *Movie Model: an SD/ABM model of box-office performance", Proceedings of the 30th International Conference, St. Gallen, Switzerland*.

Jones, K. O. (2005). Comparison of genetic algorithm and particle swarm optimization. In *Proceedings of the International Conference on Computer Systems and Technologies*.

Karaboga, D., & Basturk, B. (2007). Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems. In *Foundations of Fuzzy Logic and Soft Computing* (pp. 789-798). Springer Berlin Heidelberg.

Karaboga, D., & Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, *214*(1), 108-132.

Karaboga, D., & Akay, B. (2011). A modified artificial bee colony (ABC) algorithm for constrained optimization problems. *Applied Soft Computing*,*11*(3), 3021-3031.

Katteda, S. R., Raju, C. N., & Bai, M. L. (2011). Feature Extraction for Image Classification and Analysis with Ant Colony Optimization Using Fuzzy Logic Approach. *Signal and image processing, An International Journal (SIPIJ)*, *2*(4).

Kleijnen, J. P., & Wan, J. (2007). Optimization of simulated systems: OptQuest and alternatives. *Simulation Modelling Practice and Theory*, *15*(3), 354-362.

Kopainsky, B., Alessi, S. M., Pedercini, M., & Davidsen, P. I. (2009, July). Exploratory strategies for simulation-based learning about national development. In *27th International Conference of the System Dynamics Society, Albuquerque, NM*.

Luke, S. (2011). Essentials of Metaheuristics. Lulu (2009). *Available for free at http://cs. gmu. edu/_sean/book/metaheuristics*.

Marti, R., Laguna, M., & Glover, F. (2006). Principles of scatter search.*European Journal of Operational Research*, *169*(2), 359-372.

Molga, M., & Smutnicki, C. (2005). Test functions for optimization needs. *Available for free at www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf*

Negulescu, S. C., Oprean, C., Kifor, C. V., & Carabulea, I. (2008, August). Elitist ant system for route allocation problem. In *Proceedings of the 8th Conference on Applied Informatics and Communications* (pp. 62-67).

Oliva, R. (2003). Model calibration as a testing strategy for system dynamics models. *European Journal of Operational Research*, *151*(3), 552-568.

Payne, R. B., & Sorensen, M. D. (2005). *The cuckoos* (Vol. 15). OUP Oxford.

Rechenberg , I., (1973). *Evolution strategie: optimierung technisher systeme nach prinzipien der biologischen evolution.*

Runarsson, T. P., & Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*,*4*(3), 284-294.

Sayadi, M. K., Ramezanian, R., & Ghaffari-Nasab, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, *1*(1), 1-10.

Skahill, B. E., & Doherty, J. (2006). Efficient accommodation of local minima in watershed model calibration. *Journal of Hydrology*, *329*(1), 122-139.

Steel, R. GD, and JH Torrie. 1960. Principles and procedures of statistics: with special reference to the biological sciences. *McGraw Hill, 1960, pp.187, 287.*

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341-359.

Stützle, T., & Hoos, H. H. (2000). MAX–MIN ant system. *Future Generation Computer Systems*, *16*(8), 889-914.

Walton, S., Hassan, O., Morgan, K., & Brown, M. R. (2011). Modified cuckoo search: A new gradient free optimisation algorithm. *Chaos, Solitons & Fractals*,*44*(9), 710-718.

Yang, X. S., & Deb, S. (2009, December). Cuckoo search via Lévy flights. In*Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on* (pp. 210-214). IEEE.

Yang, X. S. (2011). *Nature-inspired metaheuristic algorithms*. Luniver Press.

Yücel, G., & Barlas, Y. (2011). Automated parameter specification in dynamic feedback models based on behavior pattern features. *System Dynamics Review*, *27*(2), 195-215.

오덕교. (2012). Comparison of Three Optimization Methods Using Korean Population Data. *한국 시스템다이내믹스 연구*, *13*(2), 7.

8 Appendix

This appendix explains how to use all the AnyLogic files. Complicated procedures are needed because for custom experiments AnyLogic doesn't allow visual interfaces. For that reason many changes have to be done by hand, changing the JAVA code directly.

Privacy issues do not allow the presence of the model source code or detailed data. This is because of previous agreements between PwC and the Erasmus Mundus Masters Program on the "movie release strategies" model.

8.1 Benchmark Functions Instructions

The files attached contain a folder named "benchmark functions" where the file "training2.alp" can be found. This file is the AnyLogic model that has all the classes and experiments of the benchmark functions. The file "resultbook.xlsx" contains the output of the simulations of the functions without constraints and "resultbookC.xlsx" contains the output of the simulations of the functions with constraints. The data has to be erased manually on the Excel files before simulating if there is interest in seeing the results graphically. The program does not erase the files automatically.

The "training2.alp" file has classes and experiments that are related with the particular algorithm and the optimization modality. The class ABC corresponds to the ABC algorithm without constraints and the class ABCC corresponds to the ABC algorithm with constraints. The class Fitness contains the unconstrained functions and the class FitnessC contains the constrained functions. The same format works for CuckooSearch and GeneticSampler.

Bees experiment uses the ABC algorithm to solve the fitness functions without constraints whiles the BeesC experiments does the same using constraints. The same format works for Cuckoo and Genetic experiments.

8.1.1 How to run the simulations

For unconstrained simulations, the Fitness class has to be open to choose which function to choose:

```
//0://De Jong's function
//case 1://Axis parallel hyper-ellipsoid function
//case 2://Rotated hyper-ellipsoid function NOT USED
//case 3://Rosenbrock's valley
//case 4://Rastrigin's function
//case 5://Schwefel's function
int function=5;
```

All the simulations were tested only with two parameters for the unconstrained functions, even though there is potential to use up to 17 parameters. To modify the two parameters boundaries the following lines have to be modified (where in this example the lower and upper boundaries are -500 and 500 respectively for each parameter:

```
params_lb.put("theme",-500.0); //lower boundary of parameter 0
params_ub.put("theme",500.0); // upper boundary of parameter 0
params_lb.put("mktg",-500.0); // lower boundary of parameter 1
params_ub.put("mktg",500.0); // upper boundary of parameter 1
```

To run the algorithm, the associated experiment without constraints has to be chosen. The console will show the evolution of the best solutions obtained and those solutions will be written in the output file "resultbook.xlsx". This Excel file contains the results in the tab with the name of the algorithm followed by the number of the function. For instance, the ABC algorithm with the function 4 will have its results shown in the tab ABC4.

For constrained simulations the FitnessC class has to be modified in the same way choosing the function. For instance `public int function =2;` will run the simulations using function 2. (Where function 0 in the code corresponds to function 1 in this document, function 1 to function 2 and function 2 to function 3). The boundaries are set as defined by the literature previously mentioned and therefore shouldn't be changed.