A Framework for Analysis of The Performance of a Proposed Metaheuristics

by

Remi EHOUNOU

ACTIVITY REPORT OF MTR871 PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPERIEURE IN PARTIAL FULFILLEMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTERS WITH THESIS IN SOFTWARE ENGINEERING  
M.A.Sc.

Montreal, AUGUST 10, 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE  
UniversitÉ DU quÉbec

image002Rémi Ehounou, 2020



This Creative Commons licence allows readers to download this work and share it with others as long as the author is credited. The content of this work can’t be modified in any way or used commercially

**Cadriciel D’analyse de La Performance D’une Proposition De Métaheuristiques**

Rémi Ehounou

**RÉSUMÉ**

L’évaluation des métaheuristiques joue un rôle important dans le développement de nouveaux algorithmes pour les problèmes d’optimisation. Cependant, plusieurs études dans le domaine recommandent l’implémentation de cadres d’analyses améliorés afin de les évaluer adéquatement. Ce document est une revue de la littérature du domaine des métaheuristiques dans le but d’effectuer la synthèse de cette problématique. L’importance d’utiliser des études contrôlées et bien organisées est reconnue comme étant nécessaire pour s’assurer que les résultats d’évaluation soient significatifs et publiables. Des méthodes de classification des problèmes de test en catégories basées sur des caractéristiques comme la modalité ou la séparabilité sont présentés. La sélection des paramètres d’évaluation, des instances de problèmes de test, et des méthodes statistiques sont aussi traités. Des cadriciels et des modèles de programmation non linéaires tels que Opt4J, GLOBAL, GAMS World sont ensuite étudiés dans le but d’identifier deux paradigmes de programmation qui offrent un bon potentiel de solution pour cette recherche. Finalement, la planification du projet de recherche est présentée à l’aide du cadre de Basili qui décrit les objectifs, la motivation et les activités planifiées de ce projet de recherche.

**Mots-clés :** métaheuristiques, optimisation mathématique, optimisation stochastique, cadre d’analyse

**A Framework for Analysis of The Performance of a Proposed Metaheuristics**

Rémi Ehounou

**ABSTRACT**

Metaheuristics benchmarking plays a key role in developing new algorithms for optimization problems. However, many published studies recommend that enhanced analysis frameworks be made available to benchmark algorithm proposals before publication. This research report presents a literature review of the field of metaheuristics benchmarking for the purpose of presenting an overview of the issue. The importance of well-constructed and controlled studies is recognized as a necessary step for the benchmarking results to be meaningful and publishable. Methods such as the classification of problem instances into categories based on characteristics like modality or separability are discussed. The selection of benchmarking parameters, problem instances, and statistical methods are also presented. Metaheuristic resources that could be used to improve algorithm benchmarking such as Opt4J, GLOBAL, and GAMS World are studied before two programming paradigms can be selected as the best candidate for this research. Finally, the software engineering planning framework of Victor Basili is used to clarify the motivation, objectives and the planning of the activities of this research project.

**Keywords:** metaheuristics, mathematical optimization, stochastic optimization, analytical framework

**TABLE OF CONTENTS**

[INTRODUCTION 1](#_Toc48605291)

[0.1. Problem Definition 4](#_Toc48605292)

[0.2. Contribution 5](#_Toc48605293)

[0.3. Research Objective 6](#_Toc48605294)

[0.4. Future Work 7](#_Toc48605295)

[CHAPTER 1 LITERATURE REVIEW 8](#_Toc48605296)

[1.1. Programming Paradigms 17](#_Toc48605297)

[RESEARCH PLANNING 19](#_Toc48605298)

[2.1. Phase I — Definition 20](#_Toc48605299)

[2.2. Phase II – Planning 20](#_Toc48605300)

[2.3. Phase III – Operation 21](#_Toc48605301)

[2.4. Phase IV — Interpretation 22](#_Toc48605302)

[APPENDIX I: SIMULATED ANNEALING PSEUDO CODE 25](#_Toc48605303)

[LIST OF REFERENCES 26](#_Toc48605304)

**LIST OF FIGURES**

[**Figure 1** Sample Objective Function Topology (Diego Andrés Alvarez Marín, 2010) 2](#_Toc48164777)

[**Figure 2** Structure of the Proposed Framework 7](#_Toc48164778)

[**Figure 3** Multimodal test function example: Six-Hump Camel Back (Li et al., 2013) 14](#_Toc48164779)

[**Figure 4** Unimodal test function example: Trid (A. Hedar’s, n.d.) 15](#_Toc48164780)

[**Figure 5** Example of large basin in Rosenbrock’s function 15](#_Toc48164781)

[**Figure 6** Example of the Topology of a Benchmarking Test Function (Tsang, 2018) 16](#_Toc48164782)

[**Figure 7** Work breakdown structure of the research activities 19](#_Toc48164783)

**LIST OF ABBREVIATIONS AND ACRONYMS**

FBP Flow Based Programming

NFLT No Free Lunch Theorem

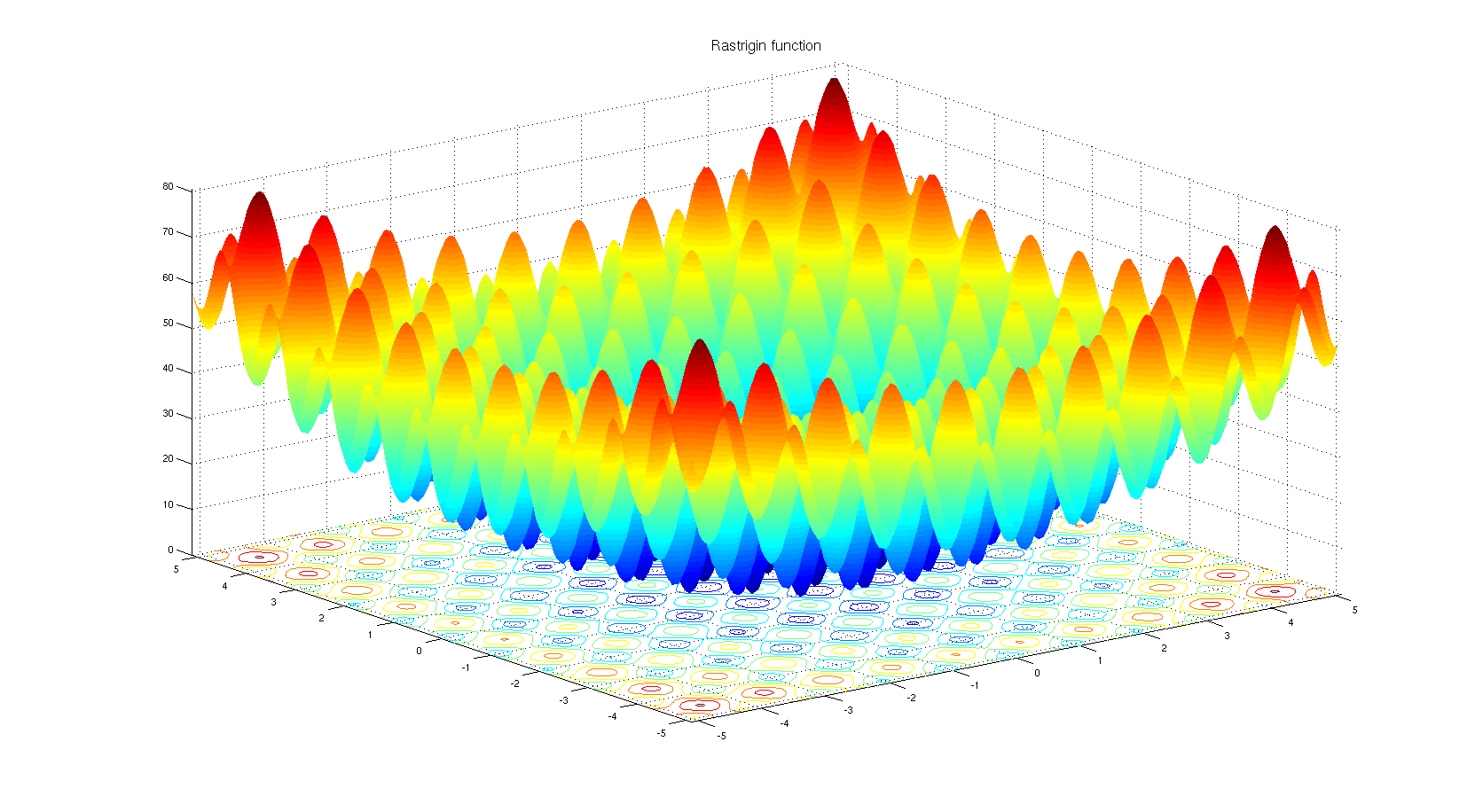
OOP Object Oriented Programming

SA Simulated Annealing

# INTRODUCTION

Several disciplines like computer science, engineering and economics require optimization problems to be solved. To accomplish this, different tools and techniques have been developed in mathematics and more recently in computer science. This research is interested in a special kind of optimization algorithm called metaheuristics. More specifically, it is concerned with the evaluation and comparative analysis of new metaheuristics proposals. The field of metaheuristics optimization has been applied to a wide range of real-world problems with great success. Such problems are often too complex to solve manually and too resource intensive for common methods such as those of linear programming (Sala & Müller, 2020). According to (Sörensen & Glover, 2013), metaheuristics define a high-level algorithmic framework that provides strategies to implement low-level heuristic optimization algorithms (Sörensen & Glover, 2013). Heuristics, from *heuriskein* in Greek which means “to find”, are algorithms that are designed using problem-specific information to find a good solution with relative ease (Bianchi et al., 2009). Therefore, metaheuristics can be seen as an abstract framework for heuristics. Metaheuristics research has grown into an established discipline and it now contains a vast body of knowledge that can be applied to almost every field that involves optimization.

Constrained optimization problems are concerned with the maximization or minimization of objective functions whose input variables are constrained for one reason or another. These constraints generally represent characteristics of the real-world problems being modeled and help ensure that the solutions found make physical sense. Objective functions model one or more variables of the studied systems that we wish to control. Constraints can also be applied on the runtime (i.e. by setting a time limit) of the optimization algorithm, although this type of constraint is considered to be of a different kind than that from the constraints on the objective function variables since they are generally not part of the problem itself. Figure 1 shows a sample objective function topology. A typical goal would be to find the coordinates for which this function has minimal value. This can be accomplished by using the ubiquitous gradient descent algorithm (Curry, 1944), but the existence of local minima almost guarantees that it will find a suboptimal solution unless the starting point is at or near the global optimum. It is therefore necessary to use a different approach, such as the use of stochastic parameters, to let the optimization escape from these regions (Luke, 2013). Optimization problems can also be categorized in different ways which determine the type of metaheuristics that will be most effective for solving it. One such distinction is made between separable and non-separable objective functions. The former is considered easier to solve due to the simple linear relationship between the components (Brownlee, 2007).



**Figure 1** Sample Objective Function Topology (Diego Andrés Alvarez Marín, 2010)

A typical metaheuristic algorithm, called Simulated Annealing (SA), follows a process similar to the annealing process in materials engineering (Luke, 2013). This is quite common with recently developed metaheuristic algorithms since many natural processes tend to have optimizing properties (Bianchi et al., 2009). As the name indicates, this algorithm optimizes objective functions by modeling the physical process by which particles inside a material rearrange themselves to achieve thermal equilibrium. It is based on the principle of local search heuristics which explores the problem space by comparing the best obtained value of the objective function to that of a neighbor and selecting the new point if it is better than the current one, or randomly selecting the new point depending on the temperature parameter: the higher the temperature parameter the more random changes are (Bianchi et al., 2009). A control parameter which effectively represents the temperature of the annealing process determines how greedily the optimization process is conducted.

This principle is presented in appendix , which outlines the simulated annealing process in pseudo code. The temperature parameter, represented by the variable , is used in the calculation of a probability factor named . The calculated value is then compared to a randomly generated number to decide if it accepts the new position or not. It is this mechanism that makes SA a stochastic algorithm. It is also this mechanism that allows SA to sometimes move to non-improving positions. The so-called temperature of the algorithm is typically set to a high value at the beginning of the process and then decreased gradually as time goes by (Luke, 2013). When the temperature is reduced, SA becomes a simple greedy algorithm that follows a consistent path toward a solution. The stochastic component is useful at the start of the process to help reduce the chances of the algorithm getting trapped in a local extreme value.

As the field keeps advancing, many algorithms are continuously being proposed and so a systematic evaluation method is needed. To solve this issue, many test problems are also being created to benchmark new metaheuristics so as to determine what types of problems they are good at solving. Benchmarking is defined as the determination of a metaheuristic’s performance when applied to a specific type of problem using a combination of theoretical and empirical approaches(Sala & Müller, 2020). In the context of this research, a framework will be developed to evaluate metaheuristics on standard test problems to determine their performance. For example, the Cross-Entropy Toolbox proposed in (Zdravko Botev et al., 2004) contains a diverse set of test functions for both constrained and unconstrained optimization that can be used to evaluate the performance of metaheuristics. This is discussed in more detail in section 1. In addition to the test functions, many principles and theories have been developed to address the challenges involved in benchmarking metaheuristics algorithms. For instance, the No Free Lunch Theorem (NFLT) stipulates that no single algorithm is appropriate for all possible types of problems (Koppen et al., 2001) (Brownlee, 2007) (Sala & Müller, 2020).

## Problem Definition

The field of metaheuristics is criticized regarding the lack of rigor in the current evaluation methods of newly proposed algorithms. The following points outline the main issues reported in the literature concerning current practices (Eiben & Jelasity, 2002):

* **Ad hoc selection of test functions**: New
  + Algorithm proposals are often tested on a set of functions without the justification of a solid experimental design, which prevents a full understanding of their performance in different contexts;
  + **Overgeneralization of obtained results**: The results of benchmark tests published for a new algorithm proposal are often generalized beyond the specific functions on which they have been tested. Better definition and classification of the problems into categories is needed to improve generalizability;
  + **Poor reproducibility**: Since the source code associated with the algorithm proposal is often not made available to the public, it is difficult or practically impossible to replicate/validate the claims and reported results;
  + **Lack of clearly stated objectives**: Proposed algorithm Results/claims are sometimes interpreted without being related to the experimental objectives and expectations.

The common practice of what esteemed Operations Research professor John Hooker called ‘competitive testing’, in which algorithms are directly compared to each other by their direct runtime performance metrics (like the time they take to converge), is highly discouraged (Hooker, 1995). He described them as non-scientific, citing two undesirable consequences when using this algorithm validation/comparison technique, and proposes better methods for comparing algorithms. In fact, directly comparing the performance numbers of metaheuristics has led to a focus on speed which distracts researcher in the field from building well designed and controlled experiments (Hooker, 1995). The author also adds that machine speed and implementation-specific details of the various metaheuristics (programming language, architecture, programming style, etc.) have too much of an impact on the runtime of algorithms for direct comparison to be scientifically meaningful.

Another issue in metaheuristics benchmarking is the classification of problems into classes as well as the selection of problem instances for testing (Brownlee, 2007). For the selection of problem instances, it is preferable to perform the study on as many classes of problems as possible to acquire a general understanding of the algorithm. When it comes to the classification of testing problem instances, the structural properties (multi-modality, isolation, needle in a haystack, collateral noise) as well as the methods of generation of the problems (i.e. real world model or artificially constructed test functions) are proposed as possible taxonomic criteria (Brownlee, 2007).

Finally, the optimizing tendency of a genetic algorithm is a characteristic that many complex systems share. Therefore, carefully tuning such algorithms is an important preliminary step to testing and special attention also needs to be paid to the selection of performance metrics (Brownlee, 2007). These recommendations must be included in future benchmark frameworks to ensure their effectiveness at comparing metaheuristics.

In short, the benchmarking of metaheuristics requires careful consideration and many recommendations for conducting this process exist in the literature (the use of a controlled experiment analytical framework, the classification of problem instances into recommended taxonomies, and the tuning of algorithms before benchmarking).

## Contribution

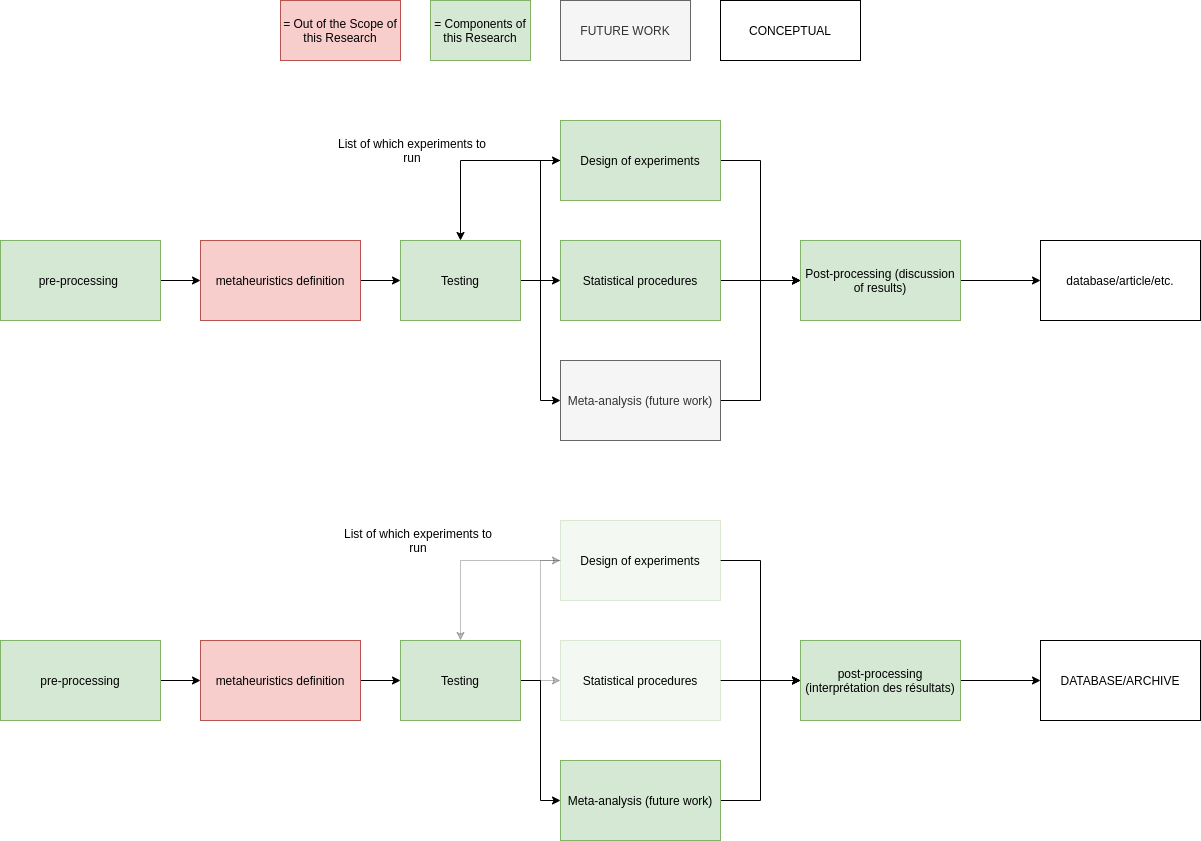
This research is part of a wider project about the design and analysis of new metaheuristics algorithms, and aims to implement and automate the recommendations from (Gagnon Iannick, 2020) about the benchmarking of metaheuristics. Recommendations from the literature which are described in the literature review chapter will also be implemented in this research. **Figure 2** shows a structure for the wider project where the elements in green are the components being addressed in this research. Therefore, this research proposes a framework for testing the metaheuristics with modules for statistical analysis as well as a preprocessing module to identify the characteristics of each study to be run through the framework. For example, the preprocessing module can be used to set up a new study for benchmarking a newly developed metaheuristic’s performance on a type of problem. The framework will then be able run the test before performing a statistical analysis. A post processing stage is also included for exporting the results data and performing some basic analysis. This framework will be implemented in the Python 3 programming language and deployed as a PyPI package.

## Research Objective

The main objective of this research project is to propose an analytical framework that helps researchers increase the quality of their work. In order to experiment with a solution proposal, an initial version of a prototype software framework for metaheuristics performance benchmarking will be designed and implemented based on the recommendations made in (Gagnon et al., 2020). This experimental framework could serve as a tool to aid the metaheuristics research community in performing more rigorous statistical analysis of newly proposed metaheuristic algorithms in the future. Before a software prototype can be designed and coded, Figure 2 shows a preliminary proposal of the architecture of the proposed framework. Because of the limited scope of a master’s degree, the boxes in green represent the components that will be addressed in this master’s degree research. Note that an important characteristic of the framework’s proposed software architecture is the ability for parallel programming to improve performance. Several secondary objectives will also be addressed:

* A modular architecture to make it easy to switch the algorithms being subjected to testing. Specifically, the framework should consider the structure of the algorithms' definition for compatibility;
* The framework will be made public on an official PyPI repository with a user guide;
* The presence of functionality to address some of the literature recommendations: modular architecture for test instances and algorithms, implementation of parametric statistical methods, implementation of the Big O notation to describe the time complexity of the algorithms.

This research, therefore, proposes to initiate the automation of the benchmarking process of new metaheuristics.

****

**Figure 2** Structure of the Proposed Framework

## Future Work

At the end of this research, results will be assessed, and potential improvements will be identified. One improvement that can already be identified aims at adapting the proposed framework to include the possibility of performing meta-analysis by consolidating the results of multiple independent studies completed by the framework itself as well as by external sources. This will not be addressed by the current research activities.

# CHAPTER 1 LITERATURE REVIEW

This section addresses the state of the art in benchmarking the performance of metaheuristics. An in-depth survey of the literature shows that this is still an unsolved problem in many ways. For example, there are no standardized sets of methods to accomplish this task (Jamil & Yang, 2013) (Brownlee, 2007). This has motivated many attempts at compiling various test functions to facilitate the evaluation of newly proposed algorithms (Brownlee, 2007) (Jamil & Yang, 2013). The literature provides many recommendations as well as theoretical notions that are important to understand for this research.

It is relevant to highlight the difference between metaheuristic optimization frameworks (MOFs) and benchmarking frameworks. MOFs like the Opt4J (Lukasiewycz et al., 2011) (*Opt4J*, 2020) and EasyLocal++ (Di Gaspero & Schaerf, 2002) are used to design metaheuristics while benchmarking frameworks are used to test and compare the performance of existing metaheuristics. OptiBench by the company CIRG@UP, is a good example of a benchmarking framework that is similar to what this research aims to accomplish (Peer et al., 2003). It comprises a library of standard problem instances and optimization algorithms, an engine for data analysis and statistical inference, and a centralized results repository. Other frameworks similar to Opt4J include jMetal, EvA2, Watchmaker framework, ECJ27 which contains benchmark functions similar to Opt4J (Sean Luke et al., 2020), JCLEC, MOEA framework, Paradiseo, and HeuristicLab. It is possible that MOFs contain benchmarking test functions like Opt4J (Lukasiewycz et al., 2011), but it is recommended to use better benchmarking methods than just the test functions provided with the MOF as is discussed in the paragraphs below. This research is focused on benchmarking frameworks and will be integrated with algorithms either generated by a MOF or taken from the literature like the bat algorithm (Gagnon et al., 2020).

Metaheuristics are used in almost all the domains of engineering (Gandomi & Yang, 2011) as well as in computer science and mathematics (Mendes et al., 2009) (*Web of Science Core Collections*, 2020). According to Web of Science, there were 404 publications with metaheuristics in the title in the last 5 years, and a growing number of citations each year, totaling 1509 citations since 2016 (*Citation Report*, 2020). Therefore, this field is growing and affects many other disciplines, making it important that the algorithms being proposed are well understood and benchmarked to ensure their appropriate use. One of the most popular techniques is to test the new algorithms on common multi-modal mathematical functions whose global min/maxima are known and to compare the results (e.g., number of objective function evaluations, first hitting times, etc.) with those of other algorithms that already exist (Hooker, 1995).This method is criticized by (Hooker, 1995) and (Brownlee, 2007) for lacking rigor and for being too simplistic considering the complexity of the algorithms being evaluated. A more mathematically rigorous approach is suggested in which the new algorithms are evaluated on a series of benchmark problems and the results are analyzed with statistical methods so as to better address their stochastic nature as well as ensuring complexities (Brownlee, 2007) (Jamil & Yang, 2013). These results may then carefully be extrapolated to other classes of problems based on how representative the sample problems used for the benchmark were (Sala & Müller, 2020). The authors also advocate for the substitution of real-life optimization problems with “computationally affordable representative benchmark problems” citing the No Free Lunch Theorem (NFLT) as a justification. This theorem is discussed in the following paragraphs. Possible problem instances are divided into classes based on their characteristics and the theory is that heuristic algorithms will have a similar performance when applied to problems of the same class. The following equation is a general mathematical description of typical optimization problems. This will be useful in standardizing the test problems in the framework as objects.

(1.1)

Where represents an objective function to be minimized or maximized depending on the specific problem being modeled.

The NFLT stipulates that all optimization algorithms have similar average performance when tested over all possible types of objective functions (Brownlee, 2007), which implies that the search for a single general-purpose algorithm is not a viable endeavor. More importantly, this theorem implies that the observed behavior of an algorithm on a specific problem class requires careful analysis when attempting to extrapolate to other problem classes. Therefore, optimization algorithms need to be matched to the specific problem classes they are suited for. This encourages the use of domain specific knowledge in optimization algorithms as a good practice since its ability to be applied to all possible optimization problems is not a meaningful characteristic of a heuristic. Even in the case of metaheuristic algorithms like particle swarm optimization (PSO) (Russell Eberhart et al., 2001), the use of domain-specific knowledge is recommended since they are designed for specific problem classes (Brownlee, 2007) (Jamil & Yang, 2013). Caution is advised when interpreting the NFLT since many do not apply it correctly due to the misunderstanding of its implications (Russell Eberhart et al., 2001). The theorem does not argue against the generalization of metaheuristics since many algorithms can solve a wide range of problems, like machine learning algorithms (Moon et al., 2019) (Lu et al., 2020) (Zhang et al., 2020). However, it does warn that the performance of metaheuristics cannot be optimal for all types of problems.

The challenges involved in getting meaningful and publishable results must be discussed. They are divided into the following three main categories as per (Brownlee, 2007):

1. Parameter selection;
2. Problem instance selection;
3. Selection of statistical methods for the analysis and interpretation of results.

The selection of algorithmic parameters is a challenge because of their non-linear correlation with the performance of the algorithm. Many approaches are proposed for this issue such as self-adaptive parameters (can be used with genetic algorithms) in which the parameters of the algorithm are encoded into the chromosomes modeled as binary strings, meta-algorithms which optimize the parameters of the algorithm in question, and sensitivity analysis which determines the sensitivity of the algorithm to changes in each parameter (Brownlee, 2007). Empirical selection by trial and error is also recommended as a good starting point, although deficiencies in this approach have been highlighted in (Francois & Lavergne, 2001). For example, they stipulate that seeking general rules for parametrization will lead to a lack of convergence and/or low efficiency. Many approaches are presented to address the selection of parameters: the Calibration and Relevance Estimation approach proposed in (Nannen, 2006) (Eiben & Jelasity, 2002); the steepest decent approach by (Coy et al., 2001); and the design of experiments (DOE) approach applied to metaheuristics research as in (Bartz, 2003). Finally, the use of Monte Carlo methods along with other statistical methods is presented for the intelligent sampling of the parameter space in (Birattari, 2002).

In addition to the selection of algorithmic parameters, a rigorous procedure for experimentation is important to ensure that the collected results will be statistically relevant. The following methods are proposed in (Brownlee, 2007):

1. Define the goals of the experiment;
2. Select measures of performance and factors to explore;
3. Design and execute the experiment;
4. Analyze the data and draw conclusions;
5. Report the experimental results.

Brownlee also reminds the reader about the usual guidelines for scientific experimentation in general. He proposes that all important factors capable of influencing the results such as computer code and the runtime environment have to be reported, that the measures be taken precisely, that the results be compared with those of other methods, that all parameters be specified, and the importance of the use of statistical experiment design (Brownlee, 2007). He reminds the reader of these scientific principles because they are often lacking in the field of meta-heuristic benchmarking. Key issues emphasized in the algorithm benchmarking literature can be identified as the duplication of efforts by the various groups working in the field due to an ineffective communication, insufficient testing, occasional failure to test using state-of-the-art techniques, poor choices of parameters, conflicting results, and sometimes invalid statistical inference (Brownlee, 2007) (Peer et al., 2003).

The issues outlined above are important to the field and addressing them is the subject this research. For example, as explained in (Brownlee, 2007), other similar fields of study, such as data science, have already passed the step of establishing standardized benchmarking methods and procedures. Those methods and procedures act as standards for the field, increasing the trustworthiness of the results produced. This enables more effective collaboration between researchers and with industry. Scientifically, the establishment of standard benchmarking and testing methods is crucial as it is at the core of the scientific method itself. Standardizing metaheuristic performance benchmarking methods will make the results more reliable and easier to reproduce, thus eliminating the risks of duplicated efforts and providing robust grounds on which the field can build upon.

The next theme concerns the selection of the problem instances and classes that are used to perform the benchmarking tests. Many of these have been created by various actors in the industry (Brownlee, 2007; Jamil & Yang, 2013). Examples of such problem instances are found in the GLOBAL library which is part of the cross-entropy toolbox (Zdravko Botev et al., 2004). This MATLAB toolbox is a collection of test problems with relevant data that can be of use to a researcher looking to benchmark a new heuristic. However, this collection of resources is challenging to implement into a study because it misses the statistical rigor alluded to in (Brownlee, 2007). This is also the case for most of the other resources available. All the components exist but there is a need to put them into a concise whole. Other resources for finding problem instances include GAMS World, which is a library of functions and test problems (*GLOBAL World - GLOBALLib*, n.d.). It was made to bridge the gap between academia and industry by providing a platform as well as resources to perform metaheuristic studies more easily. The “Cuter” testing environment was developed at the Polytechnique Montréal (Dominique Orban, 2002). It has now been superseded by a newer version named “CUTEst” which is an acronym for Constrained and Unconstrained Testing Environment with Safe Threads. This environment focuses on having a wide range of test functions numbering approximately 1150 in total (Gould et al., 2015). It is a very mature software and is optimized to be able to run the tests efficiently. It could therefore play a role in this research for the testing phase of each analysis. It is important to note that this environment does not mention any experimental framework with statistical analysis which this research aims to address. The global optimization test problems' collection by (Abdel-Rahman Hedar, n.d.) is a collection of test problems divided between a constrained and an unconstrained group. This collection is much less advanced than the “CUTEst” environment, but it is more versatile with the problems being formulated in mathematical form as well as MATLAB code compared to those of “CUTEst” which are only programmed in MATLAB. The collection of continuous global optimization test problems (*The COCONUT Benchmark*, n.d.) is a set of libraries containing test functions as well as some tools for performing basic statistics like calculating the standard unit time of an algorithm. The libraries of COCONUT use the AMPL modeling language, which is not ideal for this research since it plans to contribute to the Python library and should model its algorithms and functions using Python. Opt4J is a metaheuristics optimization framework that includes the following benchmark problem collections: knapsack, ZDT, DTLZ, WFG (Lukasiewycz et al., 2011) (*Opt4J*, 2020). It is written in Java and its test functions can be used for this research. ECJ is also written in Java and contains benchmark test functions like Opt4J despite being focused on the algorithm design aspect of generating new metaheuristics (Scott & Luke, 2019; Sean Luke et al., 2020). It is mainly used for evolutionary algorithms. The examples given above are well known, but many other such sets of test functions exist which are not necessarily well documented or recognized (Andrei, 2008; Auger & Hansen, 2005; *GEATbx - Genetic and Evolutionary Algorithms Toolbox in Matlab - Main Page*, n.d.; *Kaj Madsen - Head of Department•DTU Informatics*, n.d.; Mishra, 2006). For this research, these test functions will be useful as building blocks of the framework being implemented.

Despite the widespread use of benchmark functions, their simple use without a rigorous experiment design with statistical analysis is discouraged because they do not produce reproducible results due to the stochastic nature of the algorithms being tested (Brownlee, 2007). Brownlee advocates instead for more statistical analysis of the algorithm being tested on many test problems in a controlled setting, and a well-documented procedure is also advised to ensure the rigor of the experiment.

As proposed in (Jamil & Yang, 2013), benchmark functions can be classified in the following terms (Jamil & Yang, 2013):

* **Modality**: the number of peaks and valleys in the topology of the test function. This is relevant because the ambiguous peaks tend to trap the algorithm toward a local minimum (see Figure 3 and Figure 4).

A close up of a logo

Description automatically generated

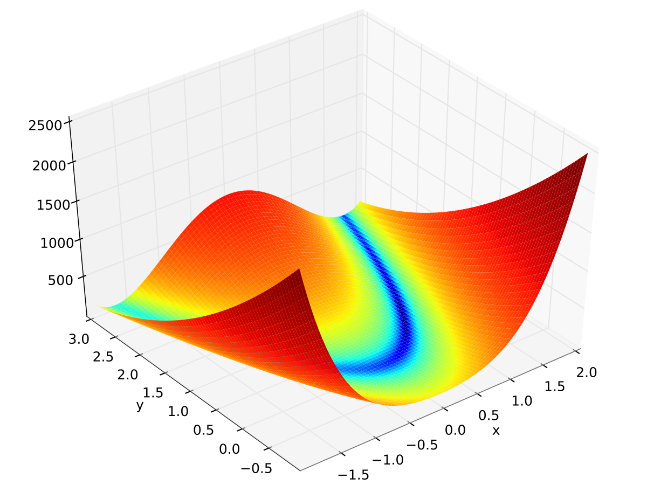
**Figure 3** Multimodal test function example: Six-Hump Camel   
Back (Li et al., 2013)

A close up of a logo

Description automatically generated

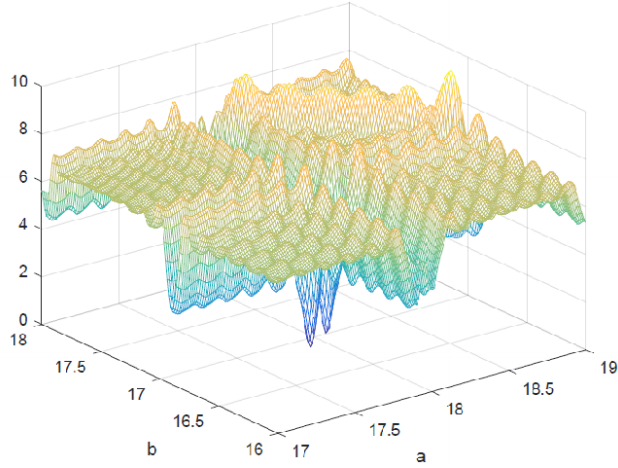
**Figure 4** Unimodal test function example: Trid (A. Hedar’s, n.d.)

* **Basins**: They are defined as a steep decline surrounding a large area. They can hamper the optimization process if the algorithm falls into a basin that leads to a local minimum. The following figure shows a graphical representation of a basin.



**Figure 5** Example of large basin in Rosenbrock’s function

* **Valleys**: Like geographical valleys, they can slow down the process when the algorithm gets to the bottom of the valley because this type of region usually does not provide local information that leads to the global solution. Figure 6 shows two valleys running parallel to the coordinate axes.



**Figure 6** Example of the Topology of a Benchmarking Test   
Function (Tsang, 2018)

* **Separability**: A measure of how difficult the test function is to solve. Separable functions are more linear than less separable ones. Equation 2.1 expresses the requirement for a function to be considered separable where represents one of the components of the vector . Separable functions can, therefore, be optimized with respect to each of the input components separately while keeping the others constant. This greatly reduces the difficulty.

(2.1)

* **Dimensionality:** The number of components of the input vector of the objective function. The difficulty of a problem generally scales with its dimensionality (Jamil and Yang, 2013).

## Programming Paradigms

To implement the framework, a programming paradigm must be selected. This section discusses object-oriented programming (OOP) and the flow-based programming (FBP) paradigms and their applicability to this research. They both have their strengths and weaknesses which make them more appropriate for modeling some programs compared to others.

OOP is a scheme in which the program is modeled as a collection of object types to be instantiated and which interact with one another via message passing (Grady Booch et al., 2007). Message passing happens when an object is passed as an argument to another object’s methods or when an object’s method directly accesses the attributes of another object for which it has access rights. This approach to programming offers the convenience of organizing the components of the solution into well-defined classes that can be interchanged or modified easily. For example, (Brownlee, 2007) proposed the creation of algorithm and problem classes to represent the components of an optimization scheme (Grady Booch et al., 2007). This makes the framework very modular; enabling the swapping of algorithms and test functions with relative ease when performing experiments. A drawback of OOP is the issue with encapsulation (Gamma et al., 1995). Encapsulation defines the accessibility of object data to the various methods of the program. A common approach is to set all the attributes of a specific class as private and creating getter as well as setter methods to access the needed ones. The problem is that this approach is not always followed correctly, and alternative encapsulation schemes are often poorly supported by even the most powerful object-oriented languages like C++ (Gamma et al., 1995). For this reason, an OOP architecture can be rendered unnecessarily complex if the design of the program is not carefully considered.

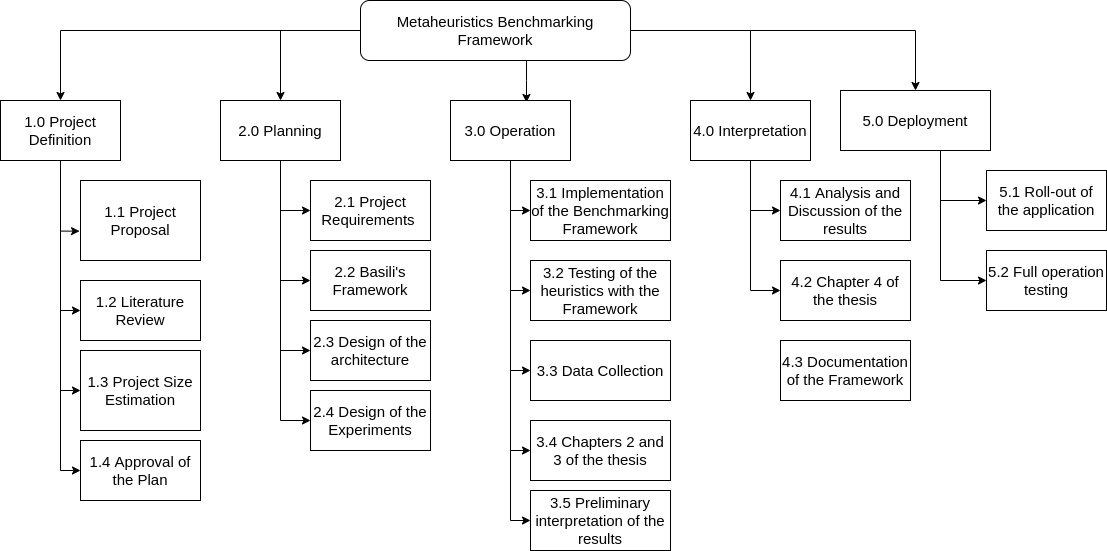
FBP is a scheme which is centered around the flow of information inside the program. In fact, this scheme organizes processes in chains with data going through them one after the other performing computation units (Grady Booch et al., 2007). This often requires the use of parallel computing (Grady Booch et al., 2007), which improves the performance of large programs compared to non-parallel computation programs by utilizing all the processing units available compared to single thread programs. Another advantage of this approach is that the computing units can be interchanged readily to improve the program or try alternative units. Weaknesses of this programming scheme is the complexity of timing the various parallel processes involved in the program so that they communicate effectively. In fact, errors in these types of programs are not easy to detect and would allow the program to keep running before failing under the right conditions. A famous example of this is the reset issue of the pathfinder Mars rover (Durkin, 1997) (Bertrand Meyer, 1997). This means that special attention needs to be paid when correcting programs based on the FBP paradigm to make sure that no errors have been made in their implementation.

This chapter presented a review of the prior articles in the field of meta-heuristic benchmarking. Common challenges in getting meaningful and publishable results are outlined and include issues of parameter selection, problem instance selection, and shortcomings in the application of solid statistical methods. The existing techniques and recommendations in the literature to improve the quality of the results obtained from metaheuristic benchmarking studies were then addressed. Some of these recommendations are to pay attention to the experiment design to make sure that it follows the guidelines of scientific control studies, that the test problems be divided into classes with similar characteristics, and that the goals of benchmarking experiments as well as the metrics to be measured be carefully considered. It is also recommended to report all factors capable of influencing the results of the experiment, like the runtime environment and the computer code. Possible programming paradigms for the implementation of the testing framework have also been presented: object-oriented and flow-oriented programming.

# RESEARCH PLANNING

This section of the report presents an overview of the proposed research using the Basili framework (Basili et al., 1986) (Bourque & Côté, 1991). This software engineering research planning framework is particularly relevant at this stage of the research because it enables a clear definition of the research subject as well as the activities that are necessary for its realization. It sets clear expectations for the activities and the objectives that will be included or excluded from the research as well as a list of activities.

The research question will be addressed by designing, implementing and testing a Python framework for metaheuristics benchmarking. The work is divided into four phases that encapsulate the activities and evolution of the research project. **Figure 7** shows a work breakdown structure for the work. It follows the structure of the Basili framework to provide a full view of the activities.



**Figure 7** Work breakdown structure of the research activities

## Phase I — Definition

This first phase presents the definition of and the audience for the research to be conducted. The motivation, subject, objectives and users of the proposed framework are identified in the table below, as recommended in Basili’s framework (Basili et al., 1986).

|  |  |  |  |
| --- | --- | --- | --- |
| Motivation | Subject | Objectives | Users |
| * Address the criticisms of the current benchmarking paradigm in metaheuristics research. with respect to benchmarking. | * The evaluation of the current methodological paradigm for metaheuristics benchmarking. | * Propose an analytical framework that helps researchers increase the quality of their work. | * Students and researchers involved or interested in mathematical optimization; * Professionals involved in projects that include or would benefit from mathematical optimization. |

**Table 1** Phase I – Definition of the research

## Phase II – Planning

This second phase of the research will focus on a literature review of the field of metaheuristic benchmarking to describe the state of the art relating to analytical frameworks that validate the claims of a proposed metaheuristic. The deliverables are:

1. Compilation and classification of currently existing metaheuristic test functions as well as frameworks;
2. Compilation and organization of tools and procedures for quantitative analysis of metaheuristics (i.e. the analytical framework);
3. Comparison between practices in the literature and the proposed framework.

|  |  |  |
| --- | --- | --- |
| Milestones | Inputs | Deliverables |
| * Literature review; * Analytical framework. | * Research articles, books, etc. | * Classification of existing benchmarking resources; * Introduction and Chapter 1 of the thesis; |

**Table 2** Phase II - Planning stage of the research

## Phase III – Operation

This third phase describes what will be accomplished by implementing the proposed framework, organizing the test functions within its structure, running the experiment following the analytical frameworks gathered in the previous phase, and performing a comparative analysis between the proposed framework and the observed practices from the literature.

The proposed framework will have elements of both object oriented and flow programming. A visual representation of its structure is shown in Figure 2. The metaheuristics definition step is outside the scope of this research. For testing a specific algorithm, the framework comes in the form of a Python software in which relevant problem instances are selected at the preprocessing stage. The algorithm is then run on the selected problem at the testing stage during which the results are collected and stored for analysis and post-processing.

|  |  |  |
| --- | --- | --- |
| Preparation | Execution | Analysis |
| * Pilot study using standard algorithms; * Design of experiments: Classification, and selection of problem instances * Collection and classification of metaheuristics test functions; | * Use of the framework to benchmark a representative set of proposed algorithms * Collection of experimental data issued from the application of the framework; | * Analysis of the gathered experimental data with tools and procedures taken from the framework; * Chapters 2 and 3 of the thesis. |

**Table 3** Phase III - Planning stage of the framework design and test

## Phase IV — Interpretation

This last phase of the research plan describes the research steps where the results of the experimentation of the new framework proposal are assessed and interpreted. It allows for reviewing the initial research objectives, organizing and discussing the results and identifying future research opportunities. Conclusions about the utility and usability of the proposed framework is also presented.

|  |  |  |
| --- | --- | --- |
| Interpretation context | Extrapolation of results | Future works |
| * Analysis of algorithms based on mathematical simplifications and statistical analysis of performance distributions on benchmark functions; * Analysis of originality based on patent law principles. | * Generalization of results based on comparative studies; * Future metaheuristics can be compared with previous results using the framework to assess originality of contribution. | * Application of framework on some of the metaheuristics that were not selected in this work; * Quantitative analysis of the trade-off between exploration and exploitation; * Automation of experimental design procedures for metaheuristics; * Chapter 4 and conclusion of the thesis. |

**Table 4** Phase IV - Interpretation and results of the research

This section briefly returned on the initial goals of the proposed research. The overall planning proposed, based on Victor Basili’s software engineering research method, helps in providing a well laid out project structure.

.

# Design of the Framework

The framework’s requirements were specified using diagrams similar to the unified modelling language (UML). A set of logical diagrams which are presented in the subsequent figures have been produced and serve as the requirement specifications for the project. This visual form has been preferred to producing a written requirements document in the form of the system requirement specification format (SRS document) because this report plays the role of such of document already. In addition, UML is widely used in the software engineering discipline and is conducive to better requirement engineering as in less ambiguity between the parties and easier requirements tracking processes (Borges & Mota, 2007). More specifically, the 4+1 views model introduced by (Kruchten, 1995) will be implemented. It is composed of four views dedicated to the functionalities of the software

1. A use case view: This view outlines the workflows of the software as well as the resources involved
2. The process view defines the interactions between the various execution threads as well as tasks and how they are synchronized.
3. The logical view describes the processes of the algorithm as well as the data structures involved. Descriptions of the objects are also included
4. The realisation view organizes the components of the software in the development environment.
5. A view dedicated to the deployment and production environment of the software is also included in the model.

Diagram

Description automatically generated

Figure 8 A Diagram of the "4+1" View Model (Kruchten, 1995)

The following figures outline the design that was produced for this research on the framework for the analysis of the performance of proposed metaheuristics.

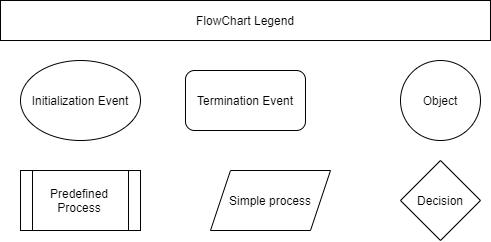


Figure 9 Legend of the Components in the Flowchart

This diagram shows the convention used in the design of the various flowcharts. This convention was used instead of the formal UML shapes for reasons of keeping the toolchain used for the design process as streamlined as possible. In fact, the use of the formal UML shapes would have required the introduction of a new software as well as the modification of the development workflow. The solution that was finally chosen focuses on implementing the UML format despite the use of a custom convention for the elements.

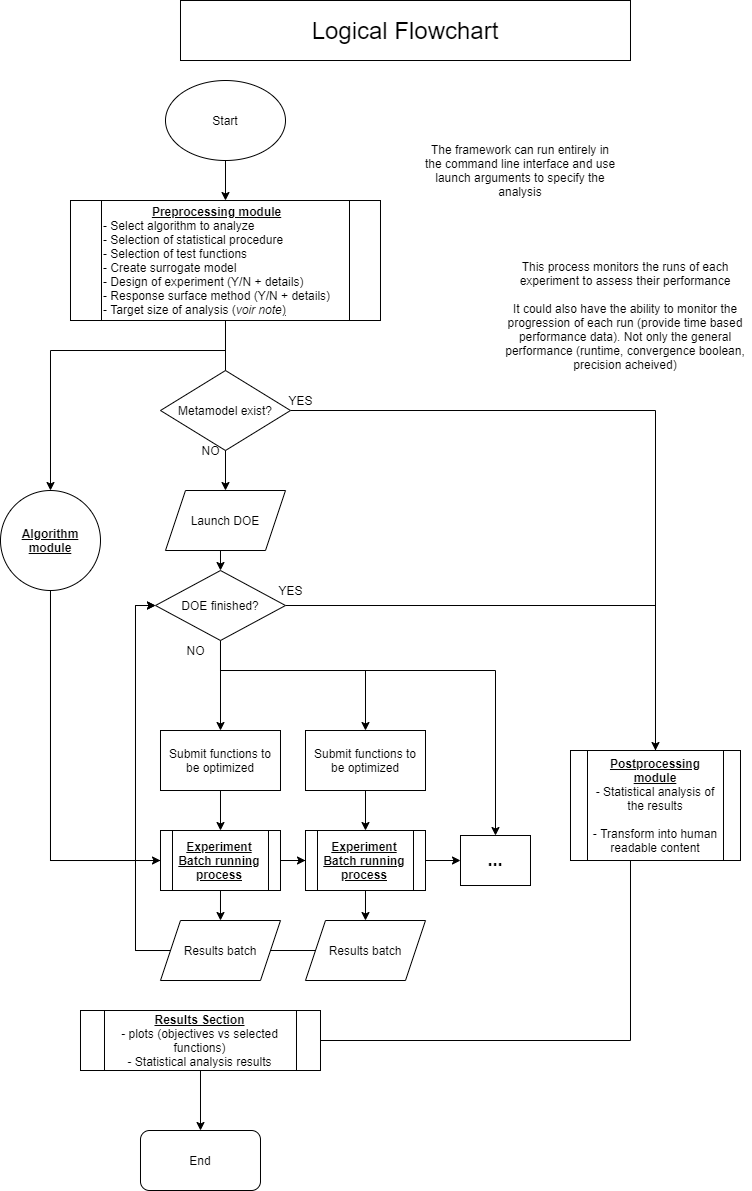


Figure 10 Main Logical View of the FrameWork

The flowchart above displays the main logical view of the framework to specify its requirements. The most important aspects to notice are the need for an algorithm module and the preprocessing module. The importance of these two elements was realised while producing the diagram which reinforces the importance of producing a good architecture. These two modules are specified in more details in the following figures. Another point to note is the fact that the actual runs of the proposed algorithm on the various test problems will be executed simultaneously in batches using parallel computing. The framework is therefore using a hybrid between the object oriented and flow-based programming paradigms. Example objects are the algorithm module as well as the test problems. They indicate the object-oriented nature of the framework. The flow-oriented nature of the program can be observed by looking closely at the flow by which the experiments are run. All the parallel processes can be modelled as sets of operations that perform calculations on data structures containing the attributes of each experiment. These attributes are:

1. The metaheuristic object being tested
2. The tests to apply for the specific run
3. The results obtained after the run
4. The performance data collected about the specific run

To accomplish this, multiple processes and threads will be accessing the experiment data structure quasi simultaneously. Semaphores (which can essentially be understood as system level variables) are to be used to synchronize these activities.

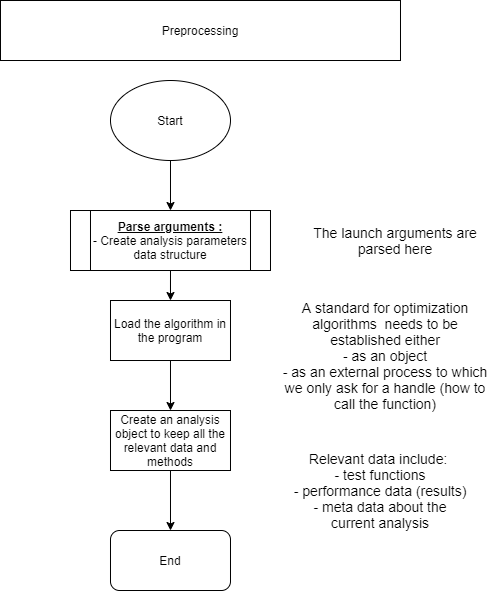


Figure 11 The Preprocessing Module

The preprocessing module plays an important role in the framework as it sets up the environment required for the experiments to be run correctly and parses the inputs given by the user. For this reason, its development involves the user more than the other modules and the design decisions taken at this level determine in many ways that of the rest of the framework.

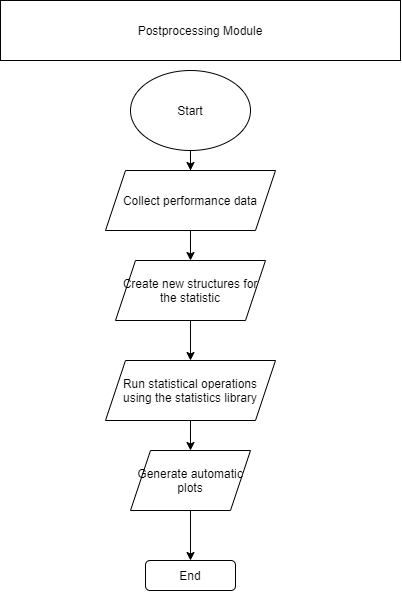


Figure 12 The Postprocessing Module

The post processing module, like the preprocessing module is important because it communicates directly with the user.

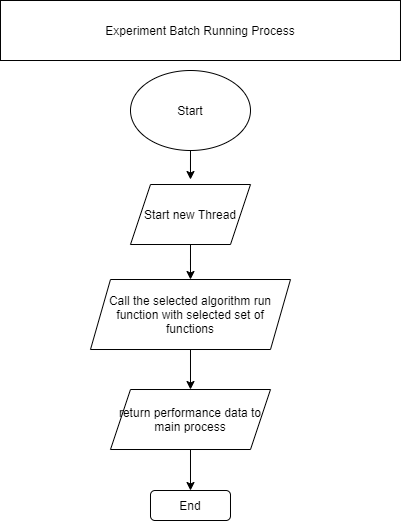


Figure 13 The Experiment Running Process

The diagram above showcases the experiment batch running process for one experiment. As displayed in the main flowchart, all the runs will be performed in different threads to leverage the power of parallel computing and reduce the execution time of the whole process. The various threads will be synchronized using semaphores and the data will be stored in custom data structures containing all the required information as specified above.

The Algorithm module below (Figure 14) specifies the contents and format of the algorithm object that the user has to provide as argument to the framework when calling it.

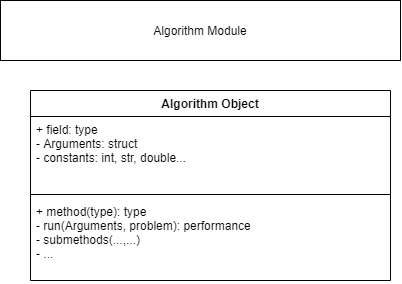


Figure 14 The Algorithm Module

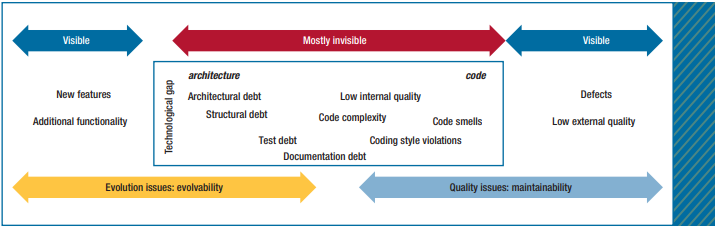
This architecture is a hybrid between the flow oriented and object-oriented paradigms. This is to leverage the advantages of both paradigms as outlined in section 1.1.

Beyond the architecture of the program itself, the lifecycle used by the development team is also an important consideration. In the case of this research, an agile lifecycle was preferred since the requirements were not all completely defined from the start and the ability to refine the architecture based on the results of the various experiments in the testing phase is important.

Therefore, the life cycle chosen was Kanban because it fulfills the agile requirement and provides the required flexibility of a research project. It also makes it possible to reassess the priorities of the various tasks and functionalities being treated to make sure that the stages of the project are fulfilled correctly.

The importance of a well designed and documented architecture is emphasized in this project as it makes the software more accessible to future contributions and facilitates collaboration. It is also associated with reduced development cost in industry as well as improved quality and reliability. In fact, it is much easier to avoid breaking a program when its architecture is known compared to trying to modify an obscure software. an added benefit of well designed and documented architectures is therefore the reduction of the maintenance cost associated with it. For example, less time is invested fixing unattended consequences of code changes. The following paragraph expands on this by discussing the notion of technical debt.

In their paper titled, “Technical Debt: from Metaphor to Theory and Practice”, Philippe et al. explain the impact that skipping the good practices of software development can have on the product (2012). The following figure outlines the technical debt landscape as defined in the paper and organized by how “visible” they are. The visibility characteristics relates to how readily detectable they are by usual identification tools like static code analysers.



**Figure 15** The Technical Debt Landscape as proposed by Philippe (2012)

To address this issue of technical debt, the practice of refactoring is recommended. The article also recommends paying attention to the design phase of the development process and to use iterative design lifecycles as they provide the opportunity to fix non ideal processes and weakly implemented standards and protocols. The article also brings to light the fact that the use of iterative lifecycles does not automatically improve technical debt and that there is a need to specifically dedicate some activities for refactoring.

## Requirements tracking

For tracking the requirements of the framework, many tools have been explored to select the right solution that would conform to the lifecycle of the project as well as its level of complexity. Following is a list of notable tools that have been considered:

* + - 1. Jira with Confluence. This tool enables the integration of the entire development environment from the version control resources like GitHub to the task management utilities and specification and scheduling documents. A communication functionality is also included. (Atlassian, n.d.)
      2. Forecast PM as in project management is a management solution that integrates project management tools and integrates an artificial intelligence for process automation purposes. It is important to note that this is a commercial software(Forecast, n.d.).
      3. Visure is a very comprehensive solution that offers many features for project management as well as requirements tracking (Hewitt, 2014). It uses a process driven approach by pushing the users to define the processes by which the requirements are supposed to be managed and enforces them in the workflow presented to access the requirements. It also comes with a library of standards that can be applied to the requirements management process. This has the benefit of simplifying the process control procedures of the users.

From the tools presented above, Jira is most interesting and accessible as it is particularly well suited for the lifecycle used in this research namely Kanban. It will therefore be used to integrate the tools that are already in use like Slack and Draw.io

## Requirements of the Framework

This section lists the most critical requirements of the framework. It will therefore be a resume of the recommendations provided from (Gagnon Iannick, 2020)

1. The framework must be compatible with all widespread computer platforms
2. The framework should make use of performance improvement techniques to reduce the execution as much as possible.

# APPENDIX I: SIMULATED ANNEALING PSEUDO CODE

/\*

Borrowed Code: Simulated Annealing pseudo code

The following lines have been borrowed from (Luke, 2013)

\*/

/\* End of the Borrowed Code \*/

# LIST OF REFERENCES

A. Hedar’s. (n.d.). *Trid Function*. Retrieved August 7, 2020, from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar\_files/TestGO\_files/Page2904.htm

Abdel-Rahman Hedar. (n.d.). *GLOBAL OPTIMIZATION TEST PROBLEMS*. Retrieved July 31, 2020, from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar\_files/TestGO.htm

Andrei, N. (2008). An unconstrained optimization test functions collection. *Adv. Model. Optim.*, *10*.

Atlassian. (n.d.). *A brief overview of Confluence*. Atlassian. Retrieved November 29, 2020, from https://www.atlassian.com/software/confluence/guides/get-started/confluence-overview

Auger, A., & Hansen, N. (2005). *Performance evaluation of an advanced local search evolutionary algorithm*. *2*, 1777–1784. https://doi.org/10.1109/CEC.2005.1554903

Bartz, T. (2003). *Experimental Analysis of Evolution Strategies – Overview and Comprehensive Introduction*. 36.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Transactions on Software Engineering*, *SE-12*(7), 733–743. https://doi.org/10.1109/TSE.1986.6312975

Bertrand Meyer. (1997). *Object-Oriented Software Construction, Second Edition*. https://archive.eiffel.com/doc/oosc/

Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, *8*(2), 239–287. https://doi.org/10.1007/s11047-008-9098-4

Birattari, M. (2002). A racing algorithm for configuring metaheuristics. *Proceedings of the Genetic and Evolutionary Computation Conference*, 11–18.

Borges, R. M., & Mota, A. C. (2007). Integrating UML and Formal Methods. *Electronic Notes in Theoretical Computer Science*, *184*, 97–112. https://doi.org/10.1016/j.entcs.2007.03.017

Bourque, P., & Côté, V. (1991). An experiment in software sizing with structured analysis metrics. *Journal of Systems and Software*, *15*(2), 159–172. https://doi.org/10.1016/0164-1212(91)90053-9

Brownlee, J. (2007). *A Note on Research Methodology and Benchmarking Optimization Algorithms*.

*Citation Report*. (2020). Web of Science. https://apps.webofknowledge.com/CitationReport.do?action=home&product=WOS&search\_mode=CitationReport&cr\_pqid=5&qid=5&isCRHidden=&SID=7EOfvJq9WTtEVdtZTx4

Coy, S. P., Golden, B. L., Runger, G. C., & Wasil, E. A. (2001). Using Experimental Design to Find Effective Parameter Settings for Heuristics. *Journal of Heuristics*, *7*(1), 77–97. https://doi.org/10.1023/A:1026569813391

Di Gaspero, L., & Schaerf, A. (2002). Writing Local Search Algorithms Using Easylocal++. In S. Voß & D. L. Woodruff (Eds.), *Optimization Software Class Libraries* (pp. 155–175). Springer US. https://doi.org/10.1007/0-306-48126-X\_5

Diego Andrés Alvarez Marín. (2010). *Rastrigin Function*.

Dominique Orban. (2002). *CUTEr Website*. http://www.cuter.rl.ac.uk//problems.html

Durkin, T. (1997). *What the Media Couldn’t Tell You About Mars Pathfinder*. 3.

Eiben, A. E., & Jelasity, M. (2002). A critical note on experimental research methodology in EC. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600)*, *1*, 582–587. https://doi.org/10.1109/CEC.2002.1006991

Forecast. (n.d.). *Help Center | The Basics*. Retrieved November 29, 2020, from https://help.forecast.it/the-basics

Francois, O., & Lavergne, C. (2001). Design of evolutionary algorithms-A statistical perspective. *IEEE Transactions on Evolutionary Computation*, *5*(2), 129–148. https://doi.org/10.1109/4235.918434

Gagnon, I., April, A., & Abran, A. (2020). A critical analysis of the bat algorithm. *Engineering Reports*, *n/a*(n/a), e12212. https://doi.org/10.1002/eng2.12212

Gagnon Iannick. (2020). *A Proposed Framework for the Design and Analysis of Metaheuristics (Unpublished doctoral dissertation)*. École de technologie supérieure.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. 431.

Gandomi, A. H., & Yang, X.-S. (2011). Benchmark Problems in Structural Optimization. In S. Koziel & X.-S. Yang (Eds.), *Computational Optimization, Methods and Algorithms* (pp. 259–281). Springer. https://doi.org/10.1007/978-3-642-20859-1\_12

*GEATbx—Genetic and Evolutionary Algorithms Toolbox in Matlab—Main Page*. (n.d.). Retrieved July 31, 2020, from http://www.geatbx.com/

*GLOBAL World—GLOBALLib*. (n.d.). Retrieved July 31, 2020, from http://www.gamsworld.org/global/globallib.htm

Gould, N. I. M., Orban, D., & Toint, P. L. (2015). CUTEst: A Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, *60*(3), 545–557. https://doi.org/10.1007/s10589-014-9687-3

Grady Booch, Robert Maksimchuk, Michael Engle, Jim Conallen, Kelli Houston, & Bobbi Young. (2007). *Object-Oriented Analysis and Design with Applications*. https://www.pearson.com/us/higher-education/product/Booch-Object-Oriented-Analysis-and-Design-with-Applications-3rd-Edition/9780132797443.html

Hewitt, P. (2014). *BUSINESS BENEFITS OF EFFECTIVE REQUIREMENTS MANAGEMENT*. 11.

Hooker, J. (1995). Testing Heuristics: We Have It All Wrong. Journal of Heuristics 1(1): 33-42. *Journal of Heuristics*, *1*, 33–42. https://doi.org/10.1007/BF02430364

Jamil, M., & Yang, X.-S. (2013). A Literature Survey of Benchmark Functions For Global Optimization Problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, *4*(2), 150. https://doi.org/10.1504/IJMMNO.2013.055204

*Kaj Madsen—Head of Department•DTU Informatics*. (n.d.). Retrieved July 31, 2020, from http://www2.imm.dtu.dk/~kajm/Test\_ex\_forms/test\_ex.html

Koppen, M., Wolpert, D. H., & Macready, W. G. (2001). Remarks on a recent paper on the “no free lunch” theorems. *IEEE Transactions on Evolutionary Computation*, *5*(3), 295–296. https://doi.org/10.1109/4235.930318

Kruchten, P. (1995). The 4+1 View Model of Architecture. *IEEE Software*, *12*, 45–50. https://doi.org/10.1109/52.469759

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*, *29*(6), 18–21. https://doi.org/10.1109/MS.2012.167

Li, X., Engelbrecht, A., & Epitropakis, M. G. (2013). *Benchmark Functions for CEC’2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization*. 10.

Lu, W., Fu, D., Kong, X., Huang, Z., Hwang, M., Zhu, Y., Chen, L., Jiang, K., Li, X., Wu, Y., Li, J., Yuan, Y., & Ding, K. (2020). FOLFOX treatment response prediction in metastatic or recurrent colorectal cancer patients via machine learning algorithms. *Cancer Medicine*, *9*(4), 1419–1429. https://doi.org/10.1002/cam4.2786

Lukasiewycz, M., Glaß, M., Reimann, F., & Teich, J. (2011). Opt4J: A modular framework for meta-heuristic optimization. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation - GECCO ’11*, 1723. https://doi.org/10.1145/2001576.2001808

Luke, S. (2013). *Essentials of metaheuristics: A set of undergraduate lecture notes* (Second edition, online version 2.0). lulu.com.

Mendes, Sí. P., Molina, G., Vega-Rodriguez, M. A., Gomez-Pulido, J. A., Saez, Y., Miranda, G., Segura, C., Alba, E., Isasi, P., Leon, C., & Sanchez-Perez, J. M. (2009). Benchmarking a Wide Spectrum of Metaheuristic Techniques for the Radio Network Design Problem. *IEEE Transactions on Evolutionary Computation*, *13*(5), 1133–1150. https://doi.org/10.1109/TEVC.2009.2023448

Mishra, S. K. (2006). *Performance of the Barter, the Differential Evolution and the Simulated Annealing Methods of Global Optimization on Some New and Some Old Test Functions* (SSRN Scholarly Paper ID 941630). Social Science Research Network. https://doi.org/10.2139/ssrn.941630

Moon, S. J., Hwang, J., Kana, R., Torous, J., & Kim, J. W. (2019). Accuracy of Machine Learning Algorithms for the Diagnosis of Autism Spectrum Disorder: Systematic Review and Meta-Analysis of Brain Magnetic Resonance Imaging Studies. *Jmir Mental Health*, *6*(12), e14108. https://doi.org/10.2196/14108

Nannen, V. (2006). A Method for Parameter Calibration and Relevance Estimation in Evolutionary Algorithms. *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings*, 8–12.

*Opt4J*. (2020). Opt4J - A Modular Framework for Meta-Heuristic Optimization. http://opt4j.sourceforge.net/

Peer, E. S., Engelbrecht, A. P., & van den Bergh, F. (2003). CIRG@UP OptiBench: A statistically sound framework for benchmarking optimisation algorithms. *The 2003 Congress on Evolutionary Computation, 2003. CEC ’03.*, *4*, 2386-2392 Vol.4. https://doi.org/10.1109/CEC.2003.1299386

Russell Eberhart, Yuhui Shi, & James Kennedy. (2001). *Swarm Intelligence—1st Edition*. https://www.elsevier.com/books/swarm-intelligence/eberhart/978-1-55860-595-4

Sala, R., & Müller, R. (2020). Benchmarking for Metaheuristic Black-Box Optimization: Perspectives and Open Challenges. *ArXiv:2007.00541 [Cs, Math]*. http://arxiv.org/abs/2007.00541

Scott, E. O., & Luke, S. (2019). ECJ at 20: Toward a general metaheuristics toolkit. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1391–1398. https://doi.org/10.1145/3319619.3326865

Sean Luke, Eric O. Scott, Liviu Panait, Gabriel Balan, & Sean Paus. (2020). *ECJ*. https://cs.gmu.edu/~eclab/projects/ecj/

Sörensen, K., & Glover, F. (2013). *A History of Metaheuristics* (pp. 960–970). https://doi.org/10.1007/978-1-4419-1153-7\_1167

*The COCONUT Benchmark*. (n.d.). Retrieved July 31, 2020, from https://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark.html

Tsang, K. K. T. (2018). Basin of Attraction as a measure of robustness of an optimization algorithm. *2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 133–137. https://doi.org/10.1109/FSKD.2018.8686850

*Web of Science Core Collections*. (2020). Web of Science. https://wcs.webofknowledge.com/RA/analyze.do?product=WOS&SID=7EOfvJq9WTtEVdtZTx4&field=TASCA\_JCRCategories\_JCRCategories\_en&yearSort=false

Zdravko Botev, Dirk Kroese, Thomas Taimre, Jenny Liu, & Sho Nariai. (2004). *Main*. The Cross-Entropy Toolbox. https://www.maths.uq.edu.au/CEToolBox/

Zhang, P., Wu, H.-N., Chen, R.-P., & Chan, T. H. T. (2020). Hybrid meta-heuristic and machine learning algorithms for tunneling-induced settlement prediction: A comparative study. *Tunnelling and Underground Space Technology*, *99*, 103383. https://doi.org/10.1016/j.tust.2020.103383