Politecnico di Torino
Master's Degree in Computer Engineering
Academic Year 2023/2024

Master's Degree Thesis

Noise-Resistant Algorithms for the Optimization via Simulation of In Vitro Pharmacology Protocols in Cancer Treatment

Supervisors:
Stefano Di Carlo
Alessandro Savino
Roberta Bardini

Candidate:
Marco Masera

Abstract

Optimization via Simulation applied to biological systems is a powerful tool for conducting large-scale, automated *in silico* experiments aimed at improving real-world processes. This field presents significant challenges, as the complexity of the simulated biological systems results in complex, high-dimensional, black-box functions that are difficult to optimize algorithmically, alongside the substantial computational cost of repeated simulations.

A preliminary phase of this work addresses the algorithm decision problem for non-convex function optimization, proposing a model that estimates the performance of different optimization meta-heuristics on arbitrary functions using statistical Fitness Landscape Analysis measures.

The study then applies a multi-scale model of tumor growth and cell resistance to treatments to optimize the delivery strategy of tumor necrosis factors (TNF). The specific challenges of the multi-scale model are identified and then addressed by developing three custom algorithms. A hybrid approach combines a population-based algorithm for broad exploration of the search space with a noise-resistant single-state algorithm for refining promising solutions. Two population-based algorithms are adapted to address the specific challenges posed by the model, resulting in two noise-resistant methods that proved able to efficiently optimize the problem even with a limited computational budget.

Contents

1 Introduction

1.1 Optimization via simulation of complex biological systems

Optimization via Simulation (OvS) is a strategy used to enhance real-world processes by first developing a model or simulation of the system of interest. Once the system has been modeled, it can be represented as a function, where the independent variables correspond to potential decisions in the real-world process, and the dependent variable is a measure of the desirability of the outcome. This function is referred to as a *fitness function*, or sometimes *loss function* for minimization problems. Algorithms are then employed to optimize this function, identifying the set of decision variables that yield the most desirable result.
OvS can be seen as automated *in silico* experimentations, and it finds applications in a variety of fields including industrial manufacturing processes [75] [31] or design [42], healthcare [69], network engineering [74], and bioengineering [66] [27].

Biological systems are inherently complex, rarely understood in depth, and their study is most often empirical. In fields such as biotechnology, medicine, and drug development, optimizing processes is often a costly, time-consuming, trial-and-error endeavor. This is due to the high number of controllable parameters, their poorly understood effects, and the difficulty in predicting the overall behavior of a complex system. The introduction of simulation models enables large-scale, low-cost *in silico* experiments, allowing researchers to generate insights or hypotheses that can be then tested experimentally [4]. For these reasons, the last decade has seen a blooming in the development of computational models for biological systems, including the use of process algebras, rule-based systems, spatio-temporal models, Petri nets [12] and multi-layer Petri nets [8].
By applying optimization algorithms to these simulations, researchers can efficiently explore the parameter space and improve real-world procedures without resorting to costly large-scale *in vitro* explorations.

For instance, a notable application of OvS is in the design and optimization of bioreactors. A study on the production of Poly(3-hydroxybutyrate) from methane utilized process simulation to enhance the design of bubble column bioreactors, demonstrating how computational techniques can streamline bioprocesses and improve yield and efficiency [2]. Similarly, biofabrication, which involves highly complex processes with numerous parameters and a vast design space, benefits from OvS by enabling the automated design of protocols for cell culture and tissue engineering or the improvement of existing ones [9]. These improved protocols not only accelerate the development process but also enhance the quality of the final product [26][18].

1.2 Multi scale model of tumor growth

The real-world use case this thesis focuses on starts with the work of M. Ponce-de-Leon et al. [57].
The authors used OvS to optimize the supply strategy of in-vitro cancer treatments, with a focus on the phenomenon of cell resistance to the therapy. Models have been used previously to investigate the effectiveness of different delivery schedules for tumor necrosis factors (TNF), indicating that different sets of values for pulse period, pulse duration and concentration of TNF can lead to effective treatments or to the emergence of tumor cell resistance to TNF [16].
Cell resistance is a complex phenomenon, originating from the interplay of different processes at the molecular, cellular and multi-cellular levels [29]. The authors proposed to model the phenomenon with a multi-scale model (MSM). MSMs allow to integrate different processes happening at different scales, making them powerful tools to model complex phenomena [49][11]. With the aid of multi-scale models, it is possible to simulate complex biological phenomena that are generated from the interplay of the genotype, phenotype and multi-cellular systems.

The inherent complexity of a multi-scale model, combined with the hybrid approach that integrates continuous, discrete, and stochastic simulations, renders the model unsuitable for analytical study. Consequently, the model must be treated as a black-box, where the researcher can only test various input configurations and observe the outcomes [21].
M. Ponce-de-Leon et al. extended an existing multi-scale model of a 3T3 fibroblast spheroid by integrating a molecular level simulation of the TNF receptor's dynamics and a simulation of the downstream propagation of the signal that induces the TNF binding via a boolean model. The model is based on PhysiCell [25] and PhysiBoSS [41]. Cells are simulated as single agents, each with its intra-cellular model. The intra-cellular model accounts for the cell cycle, the death models (necrosis or apoptosis), the TNF receptor and the gene regulatory network.

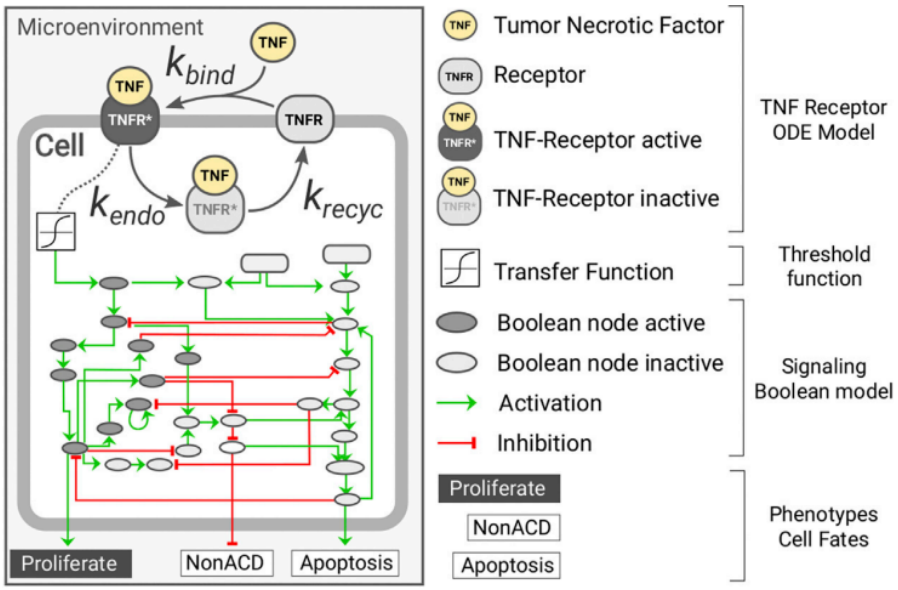

Figure 1: Diagram representing the intracellular sub-models of the multi-scale model of tumor growth. "Each individual cell agent has a kinetic model of the TNF receptor dynamics connected to the micro-environment through the presence of surrounding TNF and coupled to the Boolean network through a transfer function. The Boolean network has three readout nodes (proliferation, Non-ACD, and apoptosis), which rule the fate of the cell agent" (taken from [57]).

Using this model, the authors tested different TNF administration strategies, optimizing three different parameters: the duration and the period of the TNF pulse and the concentration of TNF, while the value used as fitness is the number of surviving cells at the end of the simulation. The authors first explored the function by means of Sweep Search, testing solutions that were randomly generated and selected from a grid covering the search space. Then they optimized it with a Genetic Algorithm and the Covariance Matrix Adaptation Evolutionary Strategy.
Given the stochastic nature of the model, the authors ran each simulation three times and averaged the fitness.

2 Background

2.1 Optimization techniques for non-convex functions

Once the fitness function is defined, mapping the inputs of the simulation to some measure of the desirability of its output, algorithms must be employed to optimize it.
The process of algorithmically optimizing a function defined by a complex, multi-scale simulator is a complex task. The function is not understood arithmetically and thus it can be only viewed as a black-box [33]: the algorithm can explore the search space and access the fitness values corresponding to any given point, but has no insights to guide the search process. Additionally, due to the inherent complexity of the modeled biological system, the fitness function itself is intricate, making the optimization problem challenging.

This problem falls within the broader field of non-convex function optimization [70], which focuses on efficiently finding the minimum or maximum of arbitrary functions that define non-convex spaces. The black-box nature of the function implies that the optimization algorithms do not rely on analytical knowledge of the function's characteristics or specific assumptions, nor can they employ problem-specific heuristics. Instead, solutions are derived through a search process rather than an analytical one. Some algorithms may utilize the function's gradient, if available, though the field is not limited to functions with known gradients.
For convex functions the optimum can be typically found with gradient descent in continuous settings and local search in discrete settings. If the gradient is not known, it can be estimated with the finite difference method, allowing to get arbitrarily close to the optimum. However, for non-convex functions, multiple local optima exist in the search space, and the aforementioned local search methods are likely to converge to different ones depending on the starting point. Furthermore, without analytical insight into the function, there is no straightforward way to determine if the local optimum found by the local search procedure is the global optimum, or to evaluate how good it is compared to other solutions that can be found from different starting points or different algorithms.
In discrete and finite search spaces, it is theoretically possible to enumerate all possible solutions and select the best one, and, for continuous functions, the global optimum can be found with a brute force approach when the *Lipschitz constant* L is known [30]. However, brute force approaches are computationally unfeasible in most cases, particularly for high-dimensional functions.

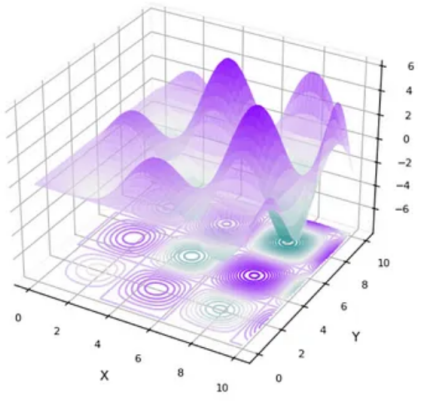

Figure 2: An example of non-convex 2D function (taken from [5])

This field addresses the inherently unsolvable problem of finding the global optimum of any non-convex, arbitrarily complex, black-box function within a feasible computational cost. Given the impossibility of solving this problem exactly, algorithms rely on heuristic methods that *typically* yield solutions that are *good enough* [32].
However, even when accepting algorithms that approximate rather than guarantee the global optimum, defining and achieving this *good enough* standard, as well as determining when it applies, presents significant challenges, as we will explore later.

Meta-Heuristics Given the unfeasible nature of exact solutions, researchers have developed numerous algorithms known as **meta-heuristics**. These are high-level procedures, or strategies, that coordinate the use of one or many heuristics to effectively explore the solution space. While meta-heuristics are often assumed to adapt to a wide range of functions, as we will see, this adaptability is not always guaranteed.
A great number of meta-heuristics have been developed and tested to tackle this problem, and they can be broadly divided into **single state** strategies and **population-based** strategies, with **evolutionary** strategies being a sub-set of the population-based.

Exploration vs Exploitation Before discussing specific meta-heuristics, it is useful to reason on the high level properties of this kind of solutions. As mentioned before, due to the black-box nature of the fitness function, each algorithm frames the problem as a search process in the function space. The key difference between these algorithms lies in the heuristic strategies they use to guide this search.
In general, all meta-heuristics exhibit two fundamental tendencies: *exploration*

and *exploitation*. Exploration involves sampling new states from the search space without prior knowledge of their fitness potential. In contrast, exploitation uses information extrapolated from the previously visited states, which is the only information about the search space available to the algorithm, to identify potentially better solutions [36].
Both exploration and exploitation are high level features, with different meta-heuristics employing various strategies to implement them. But the balance between these two plays a crucial role nevertheless. An exploration-heavy algorithm might eventually yield excellent results but typically requires significant time. Extreme examples, such as random sampling or enumeration, would theoretically find the global optimum but are extremely time-consuming. On the other hand, an algorithm focused on exploitation will converge more quickly but, because it explores a smaller portion of the search space, is more likely to settle for a local, sub-optimal minimum.
A common strategy is to modulate the balance between exploration and exploitation using a schedule. The search process might begin with a strong emphasis on exploration, sampling large areas of the search space, and then gradually shift towards exploitation as more information is gathered. Alternatively, an algorithm may focus on exploitation while it continues to find better solutions and switch back to an exploration-oriented approach when encountering fitness plateaus in the optimization process [72].

2.1.1 Single-State meta-heuristics

Hill Climbing and Tabu Search The simplest meta-heuristic is the Hill Climbing algorithm [20], a greedy, local search meta-heuristic that iteratively explores the neighborhood of its current state and moves toward new states with better fitness. Different variations can explore the entire neighborhood of a solution and move toward the best new state found (Steepest Ascent Hill Climbing) or explore just until the first improved solution is found. Other variations can dynamically adjust the distance from the current solution used to define the neighborhood, Δ, allowing for larger or smaller steps during different times of the process. All variations suffer from the same problem: being local search strategies, they generally converge toward a nearby local optimum. An algorithm built upon Hill Climbing is Hill Climbing with random restarts [20]. As the name suggests, this simple meta-heuristics runs Hill Climbing from different random starting points, allowing for a larger exploration of the search space. It is more effective than a simple Hill Climbing but not particularly efficient.
Another variant of Hill Climbing is Tabu Search [58], which introduces a Tabu list of forbidden moves, preventing it from cycling back to states already visited. Tabu Search suffers from the same main limitation as Hill Climbing.

Simulated Annealing Simulated Annealing [64] can be interpreted as an improved version of Hill Climbing. This meta-heuristic is characterized by a

probabilistic acceptance criterion for the neighborhood's states, with each state having a probability of being accepted proportional to the observed Δ fitness and to a *temperature* parameter. A high temperature increases the acceptance probability of states with worsening fitness.
The main idea is to start the algorithm with a high temperature, and let it *cool down* as iterations go by. This means the algorithm starts with a higher freedom to explore the surroundings, but with improving solutions still having a higher chance of being selected, and gradually transitions to favoring more and more improving solutions. This can allow the algorithm to escape small local minima.
Simulated Annealing is a classic example of meta-heuristics with a scheduled transition from an exploration oriented to an exploitation oriented strategy. But the algorithm requires the *temperature* as well as the *cooling schedule* to be specified as hyper-parameters, which require manual fine-tuning. It is also greatly affected by the chosen starting point, as its exploration capabilities are still limited.

2.1.2 Population-Based meta-heuristics

Population-based algorithms maintain a set of candidate solutions, or a population, at each step of the optimization process. Each algorithm employs its own strategy to evolve the population and explore the search space. Many of these population-based meta-heuristics are *bio mimetic*, inspired by the natural exploratory behavior of animal species, which is why they are often referred to as Swarm Intelligence algorithms. Evolutionary algorithms, differently, are explicitly inspired by the principles of natural evolution. They rely on defined operators for parent selection, offspring generation, mutation, and survival [53].

Particle Swarm Optimization - PSO Particle Swarm Optimization (PSO) is one of the most successful global optimization meta-heuristics, inspired by the social behavior of animals such as flocks of birds. It is known for its efficiency and performance. PSO explores the solution space by maintaining a population of candidate solutions, referred to as the *swarm of particles*, which function as a decentralized and self-organizing system. Every particle in the swarm acts as an *agent*, capable of making its own decisions. The population is initially randomly distributed across the search space, with each particle assigned a random speed and direction. During each iteration, every particle updates its speed and direction based on two factors: the best solution found by the entire swarm (global best) and the best solution found by the particle itself (local best). These influences are controlled by two hyper-parameters, the *learning factors*, which determine the weight of each direction. The two *learning factors* are also called the *cognitive coefficient* and the *social coefficient*. Additionally, random noise is introduced at each iteration to ensure that the contribution of these two directions is not deterministic. By being attracted to both their local optimum and the global optimum, PSO effectively employs both local and global search

strategies. The random initialization of particles, combined with their evolving speed and direction, facilitates a thorough exploration of the search space [53]. The hyper-parameters needed by PSO are the population size, the *cognitive* and *social coefficients* and the *inertia,* or *weights*.

Artificial Bee Colony - ABC Artificial Bee Colony (ABC) is a global optimization meta-heuristic developed in 2005, inspired by the social behavior of bees searching for food sources. It has gained attention for its strong performance and the simplicity of its hyper-parameter tuning. In ABC, the colony of bees is divided into three groups: *scout* bees, *employed* bees, and *onlooker* bees. Scout bees are responsible for the *exploration* phase, randomly searching the solution space for new promising, high-fitness areas known as *food sources*. *Employed* bees focus on *exploitation*, exploring the vicinity of a discovered food source to refine and improve the solution. *Onlooker* bees choose a food source based on a probability proportional to its perceived quality and contribute to the selection and further exploitation of promising solutions. Food sources are abandoned after a fixed number of attempts without improvements, prompting employed bees to either become scouts and search for new areas or act as onlookers to assist in the selection process [38].

ABC explicitly defines the *exploration/exploitation* trade-off, assigning different agents to the different roles. The algorithm modulates the trade-off based on whether *exploitation* is producing improved results or not. This modulation can be observed by plotting the average distance between solutions in the colony at different iterations: the population diversity shows an oscillatory behavior, with the distance decreasing when food sources are being exploited and increasing again when new sources need to be found (figure 3).

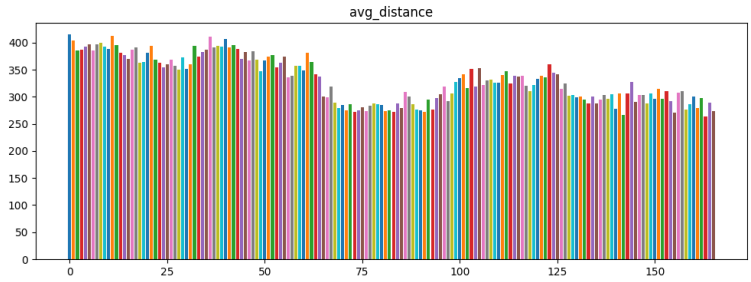

Figure 3: Average distance between individuals in the bee colony at different iterations

Genetic Algorithms - GA While the concept of artificial evolution steams from the late work of Alan Turing in the 1950s, Genetic Algorithms (GA) were formalized and popularized as optimization tools by John Holland in the early

70s [34][35]. They have since become one of the most widely used evolutionary meta-heuristics and optimization algorithms. Just as natural evolution can be seen as a search process that explores the space of possible genetic sequences to produce fitter individuals, Genetic Algorithms apply this principle to generic search spaces and fitness functions.
The first step in implementing a GA is to represent candidate solutions as a *genome*, a sequence of *genes*. Genes are defined by their *alleles*, the different states they can take. Traditionally, *genes* are represented as bits, but any data type can be used. For instance, a continuous function with N dimensions can be represented either as a sequence of N genes, defined as 32-bit floating point numbers, or as a sequence of *N*32* binary genes.

After defining the genome and alleles, the process of biological evolution is simulated using three key operators: **selection**, **crossover** and **mutation**. The **selection** operator selects the fittest individuals to serve as parents for the next generation. The **crossover** operator combines the genomes of two *parents* to create an offspring solution, and the **mutation** operator introduces random perturbations to the genome of the offspring solutions. These operators can be implemented in various ways, but the simplest approach uses a *steady-state* selection, where the fittest N individuals are chosen as parents, while the worst performing ones are removed from the population, and a *single point* crossover, where the genome of one parent is taken, and a number of sequential genes are replaced by those of the other parent. The mutation operator varies depending on the data type of the genes: bit flipping is common for binary genes, while Gaussian perturbation or uniform sampling is typically used for numeric genes [47][40].

GA are among the most theoretically rich optimization techniques, with foundational work beginning from J. Holland himself. From the perspective of the *exploration/exploitation* trade-off, GA exploit existing solutions through crossover and explore the search space via random mutations. However, the theoretical foundation of GA is rooted in the concept of *schemas*. A *schema* is a sequence of symbols drawn from the set of alleles, plus wildcards represented by the symbol *. The *order* of a schema is the number of contained symbols that are not wildcards. The *instances* of a schema are genomes that match the sequence described by the *schema*, with wildcard positions allowing for any allele value. The evolution of a population is thus interpreted as a process of selecting and amplifying the correct schemas. Given that any genome with N genes is *instance of* 2^N schemas, a population of M individuals will contain between 2^N and $M * 2^N$ schemas. As the algorithm evaluates the fitness of the M solutions, it implicitly estimates the fitness of all of the *schemas* it is instance of. Any given *schema*'s fitness is estimated as the average fitness of the individuals that are instances of it. The *schema theorem* posits that the selection and crossover operators gradually amplify schemas that produce high-fitness solutions. The *building block hypothesis* suggests that fit solutions are composed of fit high order schemas, which in turn are built from fit lower order schemas. The mu-

tation operator introduces random perturbations to the genome, enabling new and potentially unseen schemas to emerge in the population and reducing the risk of a collapse in population diversity [47][40].

GA are complex to implement and tune. They require careful decisions regarding the encoding of solutions into a genome, as well as the design of crossover, selection, and mutation operators. Its hyper-parameters, such as mutation probability, population size, the number of parents involved in mating, and strategies for population replacement or the introduction of elitism, which always preserves the N best individuals across generations, can deeply affect its performances. Different configurations can lead to a collapse in the population diversity and premature convergence, making it hard to plan the *exploration/exploitation* trade-off based on the computational budget available.

Differential Evolution - DE Differential Evolution (DE) [56] is another population-based meta-heuristic. As it uses the mutation and crossover operators, it is considered an evolutionary algorithm too, but compared to Genetic Algorithms it drifts apart from the biological concept of genetic evolution.
DE encodes individual solutions as vectors of parameters. The main operator guiding the evolution process is the *mutation* operator, which produces one *mutant* vector from three solutions randomly chosen from the population. One of them is selected as a pivotal point, to which the difference between the other two, scaled by a factor F, is summed. The *crossover* operator then mixes the *mutant vector* with the parameters of the original one, generating the *trial vector*: a hyper-parameter determines the likelihood that a component of the mutant vector will replace the corresponding component in the target vector.
Differently from GA, the *selection* operator applies only to the *trial vector* and to its parent, preserving the one with better fitness.

DE is generally considered a more *exploration-oriented* algorithm, as its crossover operator acts on randomly chosen solutions, which naturally maintains a higher population diversity.

2.2 Algorithm decision problem and no free lunch theorem

The meta-heuristics discussed so far are among the most widely used, but they represent only a small fraction of a highly active research field. Over 500 meta-heuristics exist, with more than 350 developed in the last decade [63]. This proliferation of new algorithms, often simple variations of existing ones or novel bio-mimetic algorithms inspired by various natural phenomena [14], does not ease the task of optimizing a specific problem in the absence of clear guidelines on which algorithm is most suited for the specific problem at hand. K. Rajwar et al. [63] note that the performance of the same meta-heuristic can vary greatly across different problems, with no explicit explanation of why. This variability undermines the value of comparisons based solely on benchmark function, as the effectiveness of an algorithm depends heavily on the specific fitness function being optimized. Additionally, once an algorithm is selected, its performance is significantly affected by the choice of its hyper-parameters, which can be optimized only by means of empirical experimentation.

This issue is encapsulated by the *no free lunch theorem*: no single algorithm can be universally optimal for all problems. The theorem states that every algorithm that searches for the global optimum of a fitness function performs exactly the same when averaged over all possible fitness functions [13]. This introduces the *algorithm decision problem*: determining how to choose the most appropriate meta-heuristic for a given fitness function.

2.2.1 Analysis on the characteristics of different meta-heuristics

Extensive research has been conducted to compare the performance of different meta-heuristics, and produce at least some guidelines for the algorithm decision problem.
H.R. Boveiri et al. [14] conducted a large number of experiments over 20 different meta-heuristics, producing some theoretical results. But the very authors of this paper highlighted the limitations of their approach: one of their experimental results suggested that the GA converge faster, and are thus preferable for low budget optimization, while the ABC algorithms take longer time but converge to better solutions; but the authors recognized that the behavior of GA with a larger population and a higher mutation probability would likely maintain a higher population diversity, allowing for more exploration and possibly reaching similar performances to ABC, or even better.
K. Rajwar et al. [62] focused on the structural bias of the different meta-heuristics. The authors reckon that population-based algorithms can be biased toward certain areas of the search space, such as the center. They then developed a novel methodology to detect structural bias and tested it on 6 different meta-heuristics, including GA, DE and PSO, suggesting that DE shows the smallest amount of bias.

A. Thomaser et al. [73] tried a different strategy. Observing that benchmark functions are rarely predictive of real world performances, they built custom made benchmarks, tailored to simulate characteristics of their own real world problem. The process is automatic, as benchmarks are randomly generated functions, and their similarity to the fitness function of interest is estimated from a set of statistical measures. This can be a powerful approach: if the real fitness function is expensive to evaluate, a custom benchmark function can allow for large scale experiments on the best algorithm and configuration.

Genetic Algorithms, being one of the oldest and most widely used meta-heuristics, received a lot of attention over the years. Researchers have tried to describe the features of a problem's search space that can influence the success of GA. The most treated concept is *deception*: for certain functions and certain genome encodings, *low order schemas* that are not present in the *high order schema* of the best solution will produce small fitness improvements, leading GA to preserve them and leading the search away from the global optimum. Another source of difficulty is *mutually conflicting solutions*: two or more areas of high fitness with no common *schemas* will lead GAs to produce many useless, low fitness hybrids [52]. Moreover, as mentioned earlier, GA requires an explicit decision on how to encode solutions into a genome. This decision shapes the search space that the algorithm *sees*, exerting great influence on the optimization performances.[28] While those are interesting theoretical results, they don't really help tackle the algorithm selection problem, as there is no way to easily estimate properties as *deception* in a black-box function.

2.2.2 Exploitation strategies and implicit assumptions

To better understand the algorithm decision problem, it's useful to step back from the rich, specific literature and consider it from a broader perspective. All meta-heuristics are characterized by two key features: exploration and exploitation. Meta-heuristics explore the search space to some degree randomly, due to the lack of prior information to guide the search. They then use the information gained from these random samples to heuristically direct the exploration towards more promising areas.
Different algorithms implement different heuristic strategies for exploiting the information found. For instance, the Artificial Bee Colony (ABC) algorithm explores the vicinity of promising solutions in depth and, based on certain thresholds, decides when an area is no longer worth exploring. Genetic Algorithms (GA) recombine promising solutions with their crossover operator, whereas Differential Evolution (DE) performs recombination based on the geometric distance between solutions.
When an algorithm exploits its currently available information using a given heuristics, it implicitly makes assumptions about certain properties of the search space.
It is interesting to observe that bio mimetic algorithms, inspired by the behavior of animals solving real-world problems, implicitly inherit the assumptions from

the real-world problem of the animals. For example, the Artificial Bee Colony (ABC) algorithm, which mimics the foraging behavior of bees, assumes that the vicinity of high-fitness points is likely to contain other high-fitness points [6]. This assumption works well for the natural distribution of flowers in a physical environment, which bees have evolved to exploit effectively. However, this assumption does not necessarily hold for an industrial optimization problem or a combinatorial, NP-Complete problem, because there is no guarantee that these problems will exhibit the same spatial characteristics as the natural environments that inspired the algorithm.
Genetic Algorithms make slightly more complex assumptions, described by the *building block hypothesis*, but they nevertheless are simply assumptions that are not guaranteed to hold for any problem.

2.3 Optimization via Simulation

OvS as a field introduces a great deal of complexity, with the need to build a reliable simulation to describe the real world process and quantify the desired outcome into a continuous fitness function; but it also introduces new difficulties into the optimization procedure. Most optimization meta-heuristics require a large number of samples from the search space, and the algorithm decision problem and the hyper-parameters optimization require running multiple experiments. But in most cases simulators are expensive to run, making the whole process slow and challenging [17]. Simulators can also introduce some extent of randomness, defining a stochastic fitness function. The noise in the fitness function influences the search strategies of the meta-heuristics [3].
A widespread solution for the high cost of running the simulator is introducing meta-models, or surrogate models. These meta-models are regression models that approximate the behavior of the simulator's fitness function, allowing for a lower computational cost but introducing additional noise [1]. The choice of the type of meta-model allows for any regression model to be used, but most works seem to opt for a fully connected neural network. Other possible variations are how to train the meta-models and how to introduce them in the optimization procedure. For example the meta-model can be trained before starting the optimization process, using a set of sampled points and the corresponding fitness, but it can also be trained on the solutions explored by the algorithm during the last N iterations, taking advantage of a locality assumption for which a model trained on a group of solutions will provide good results for solutions that are similar to them and reduce its performance as solutions get further away from its training set. The exact role of the meta-model in the optimization process can vary from entirely substituting the simulator during the optimization step [46] to being used to filter out unpromising solutions [75].
Other works use domain specific knowledge to guide the optimization process, biasing the initial population toward promising areas or clustering together independent variables that will have similar effects in the outcome [75]. This is another powerful approach, as the algorithms see the fitness function as a black-box, the researchers may want to exploit the knowledge they already have.

Overall, to this day the algorithm decision problem is unsolved, and optimizing complex functions, especially if defined by a computationally expensive simulator, is a challenging and time-consuming task, mostly guided by trial-and-errors.

3 Methodology

The work of this thesis is organized into two main parts.
The first part consists of a preliminary work addressing the algorithm decision problem in the general context by developing a meta-optimization model. This model utilizes statistical measures to describe the fitness functions' landscapes and predicts the performance of various optimization algorithms.
The second part focuses on the real-world optimization via simulation problem of the *multi-scale model of tumor growth*. The simulator presents specific traits and challenges; after having tested the most common optimization meta-heuristics and identified the source of their difficulty, three custom algorithms have been developed to tackle the problems posed by the model.

3.1 Meta-Optimization model

As mentioned in the Introduction, although there is a wide variety of meta-heuristic algorithms available for addressing non-convex optimization problems, no significant guidelines exist to assist in selecting the most suitable algorithm for a specific problem.
The first part of this work aims to develop a heuristic solution that guides the selection of the appropriate algorithm for a given fitness function, eliminating the need for extensive experimental testing. This solution is presented as a meta-optimization model, which takes the fitness function as input and provides estimates of how well different meta-heuristics might perform with that function.
The meta-optimization model can be described by the tuple: $(F, A, D(f), M(D(f), a))$

- **F**: universe of functions that can be optimized.
- **A**: set of meta-heuristics that can be used to optimize the functions.
- **D**(f): descriptor function. From a given function $f \in F$, $D(f) \rightarrow R^{\mathrm{n}}$ produces a set of measures that describe properties of F. For the meta-model to be practically useful, $D(f)$ must be computable with a relatively small number of samples from the function f.
- **M**$(D(f), a)$: model, for $f \in F, a \in A, M(D(f), a) \rightarrow R$ maps the descriptions produced by D on f and a given algorithm to a predicted performance score.

3.1.1 Descriptor D and Fitness Landscape Analysis

The first and most critical component of the meta-optimization model is the descriptor D. D must compute measures of the function's properties that can explain the varying performances of different meta-heuristics.
This presents a dual challenge: the measures need to be rich enough in information to effectively characterize the function and guide algorithm selection. At the same time, they must be coarse enough to be computed from a small set of samples, without requiring extensive exploration of the function's space.
A powerful set of tools to solve this problem is **Fitness Landscape Analysis**. The concept was first defined by Sewell Wright in 1932 [76] within the context of evolutionary biology. Wright posited that it is possible to define a field encompassing all possible combinations of genes. This field includes all existing biological individuals, which are located as clusters within a much larger space of possibilities, as well as any potential variation of them.
This theoretical field can be used to understand the dynamics of the evolutionary process. Some evolutionary pathways are more likely, while others, though desirable, may be nearly impossible to achieve. The landscape is characterized by gradients, peaks (local minima and maxima), and valleys (relatively flat areas with low fitness that evolution is unlikely to traverse). While Wright's work was not originally related to optimization algorithms or even computer science, it effectively describes the shapes and properties of the search space of

a non-convex fitness function. It also provides insight into how this landscape influences the trajectories that evolution, or any specified search strategy, tends to follow.

This concept has been revived and explicitly adapted to non-convex optimization decades later.
P.F. Stadler formalized the concept of fitness landscape of a function [67] in 2002, describing it as the tuple $(X, N(x), f(x))$, where

- $\mathbf{X}$: set of feasible solutions, or the search space.
- $\mathbf{N}(x)$: neighborhood function.
- $\boldsymbol{f}(x)$: fitness function.

Note that the neighborhood function $\mathbf{N}(x)$ implicitly defines how the feasible solutions of $\mathbf{X}$ are distributed in the landscape. Since we deal with continuous or discrete numeric functions, it is intuitive to reason about the distance between solutions as geometrical distance. But in the optimization process, the operator $\mathbf{N}$ depends on the algorithm used and its own notion of neighborhood. For PSO, Hill Climbing and to some extent Differential Evolution it makes sense to reason within geometrical distance, while for GA the actual distance measure depends on how the solutions are encoded in the genome. Figure 4 highlights how different distance metrics influence the shape of the search space.

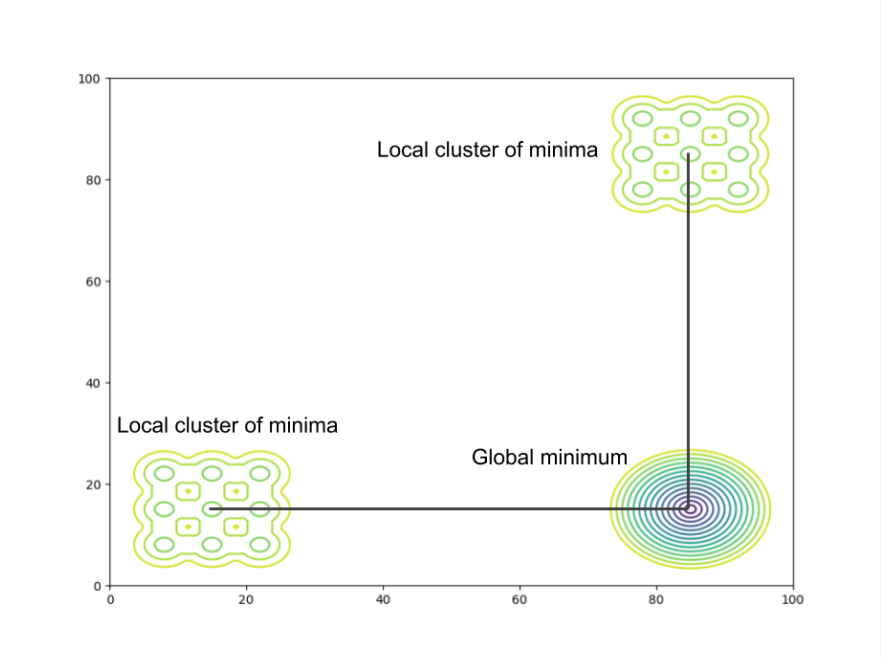

Figure 4: With a geometric distance measure, the global optimum is distant from both local clusters, similarly to a Needle in a Haystack problem. With a GA implementation that encodes the two numeric components of the vector in a two-genes genome, it is only one crossover operation away.

On the space defined by $(X, N(x), f(x))$ it is possible to define qualitative characteristics of interest:

- **Modality**: number of local optima.
- **Basins of attraction**: given a local optimum, its basin of attraction is the area surrounding it from which a local search algorithm converges into the local optima. Basins of attraction can be strong, if the local search will converge from any points in them, or weak, if it will converge for only large parts of them. Besides the size of the basin of attraction, its shape can be of interest.
- **Funnels**: presence of clusters of local optima surrounding the global optimum. They can be exploited by the search algorithm but they can also cause premature convergence to a sub-optimal solution. The protein folding problem is a fine example of an optimization landscape characterized by funnels [54].
- **Distribution of local optima**: relates to the concept of funnels. How local optima are distributed in the search space influences the algorithm

performances. A funnel around the global optimum can be exploited by an algorithm, while clusters of local optima far from the global optima can mislead it. An isolated global optimum defines the needle in a haystack problem, which is basically unsolvable.

- **Ruggedness, neutrality**: they refer to the shape of the fitness surface. A rugged surface is characterized by many small "ups and downs", while neutrality indicates areas of low or null variation. Both ruggedness and neutrality create challenges for the search algorithm. A rugged surface can introduce noise in the exploitation procedure, while a neutral landscape offers no information on promising directions. Both measures depend on some concept of scale: they can be considered on a very small, local scale or on a bigger, more global scale.

- **Deception**: it is a strongly algorithm-dependent property. It refers to characteristics of the fitness landscape that trick the algorithm into moving away from the optimal solution. The concept has been used mostly for GA [52], but a simpler way to visualize it is a local gradient carrying a Hill Climbing toward a local optimum instead of the global optimum, or an isolated cluster of local minima consuming the resources of an ABC optimization procedure.

These characteristics interact together forming unique optimization landscapes for different functions. For instance, the widely known *Travelling Salesman Problem* (TSP) shows a large basin of attraction surrounding its global optimum, together with a highly rugged, noisy surface, with the variance of the noise growing monotonously with the distance from the global optimum. High quality local optima are scattered across the search space with low correlation between each other [78].

From these qualitative descriptions of the fitness landscape, a significant number of statistical measures have been defined [44][45]. Each measure uses a set of sampled points from the function's space to estimate the properties of its fitness landscape.

The descriptor D computes a number of FLA measures, both implemented internally and included from the R package Flacco [39].

Dimensionality: the number of independent variables.

First Entropic Measure (FEM): measure for the ruggedness or the neutrality of the fitness landscape. It is based on a random walk, and it labels each consecutive three points as either rugged, smooth or neutral, based on the observed change in the fitness values. The ruggedness or smoothness of the landscape is then estimated with a measure of entropy on the sequence of labels. The operation is repeated with different step sizes on the random walk, producing different measures on different scales considered.

Fitness Distance Correlation (FDC): estimate for the distribution of local minima in the search space and the presence of funnels. It is based on a set of randomly sampled points: for each point the distance to the best known optimum is considered, and the covariance between individual fitness and its distance is considered. It is computed globally, with points sampled from the entire search space, and locally, for local subsets of the original sample. A set of measures is extracted, including the global FDC and average, maximum, minimum and std of local FDC estimates.

Dispersion: characterizes global topology and estimates the presence of funnels. It is based on a measure of *dispersion* defined as the average pair-wise distance between points. Starting from a set of randomly sampled points, the measure iteratively restricts them with more and more restrictive fitness thresholds and records the variation in the dispersion.

Jensen's Inequality Ratio: measure for non-convexity of the fitness landscape. It is based on the *Jensen's inequality*, which defines convex functions. Given a set of randomly sampled points, it computes the percentage of ones for which the inequality holds.

Meta-models: estimate of global structure properties such as separability, multi-modality or existence of plateaus [48] based on the training of a meta-model. Both a strong (Random Forest) and weak (Linear Regressor) model are trained, and both the training and validation errors are considered.

Measures by Flacco packages: on top of the aforementioned techniques, the Flacco package [39] has been included to provide additional measures:

- **Cell mapping features**: the search space is discretized in a number of cells. On these cells, the following measures are extracted:
  - Angle: considers the best and worst solutions inside the cell and measure their angle from the center.
  - Cell convexity: estimate the convexity of the search space by considering combinations of neighboring cells.
  - Gradient homogeneity: for each point in a cell and its closest neighbor, a vector is computed, directed toward the highest fitness between the two. The vectors are normalized, summed up and divided by the maximum possible vector length. In case of a randomly distributed objective values, and thus vectors pointing in random directions, the result is close to zero.
  - Generalized cell mappings: the function space is reduced to one solution for each cell, with three different approaches: *min*, *max* and *mean*. Then the transition probabilities between cells are computed as interpreted as absorbing Markov chains. Each cell is labeled either

as an *attractor* cell (which leads only to itself), *uncertain* cell (which can lead to different attractors) and *basin of attractions*, set of cells that lead to the same attractor.

- Information content-based features: set of estimates for smoothness, ruggedness and neutrality.
- Nearest-better features: set of heuristic measures to estimate the presence and size of basins of attraction.
- Principal components features: computes a covariance matrix on the sampled points and corresponding fitness and the number of components required to explain most of the observed variability.

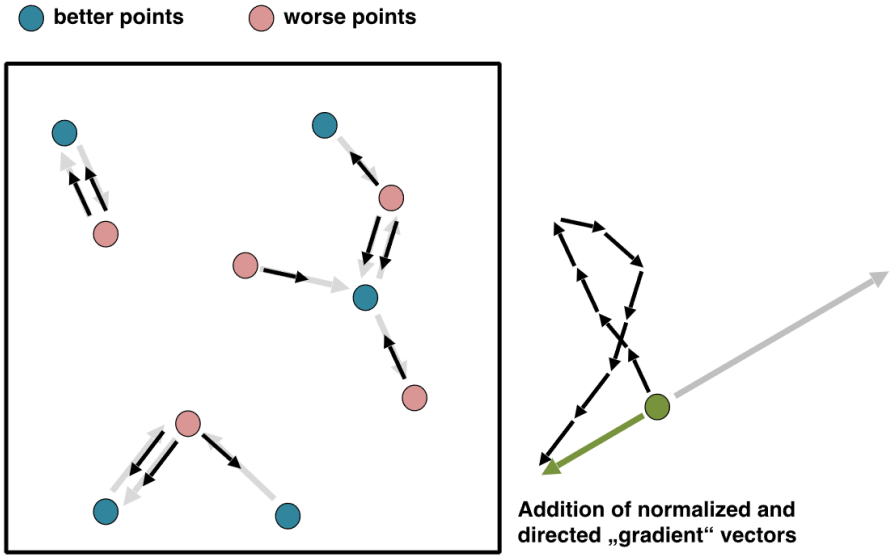

Figure 5: Example on how gradient homogeneity is computed by Flacco in a single cell. (taken from [39])

Additionally, some algorithm-specific, non FLA-related measures were computed as well:

Geometrical Evolvability Measure: simple measure that uses geometrical distance to define neighborhoods over sampled points and estimate the percentage of them showing improvements.

LPP: another evolvability measure based on geometrical distance. For different values of M, the measure computes a binary string where at each position i the value is 1 if the i^{th} closest point has better fitness, 0 otherwise. The string is then parsed as a binary number and used to score the solution. High scores indicate that the solution is immediately surrounded by improving solutions, low scores indicate that improving solutions can be found in the surroundings but small peaks must be traversed, and a value of 0 indicates a local optima.

GA and DE-specific evolvability measures: running exactly 1 iteration of the optimization algorithm with a small random population allows to extract the following measures: EAP is the difference in mean fitness between starting and end population; EPP is a binary measure indicating the presence of an improvement in the best solution of the population; PER is the percentage of the individuals showing an improvement. These measures can be taken multiple times, producing both averages and standard deviations.

3.1.2 Algorithm set A

The meta-optimization model as defined in section 3.1 is characterized by a set of optimization algorithms **A** that the model supports.
The four most used meta-heuristics have been included in the meta-optimization model, each one with a few variations in the hyper-parameters. Different configurations of the same algorithm are seen as completely different algorithms by the model.

- **GA**: based on the open-source package *PyGAD* [24], genetic algorithms are included with different crossover (single point, scattered, uniform) and parent selection (steady-state, roulette-wheel, tournament) operators. The mutation probability is set to a fixed 0.3 on a single gene, the population size to 32 and the number of parents mating to 6.

- **DE**: differential evolution is implemented by the *scipy optimize* package [65], and is included with different strategies to compute the trial vector. The mutation strategy can either start from a random solution or from the best one, and use the distance vector between two or four other random solutions. The population size is fixed at 15.

- **ABC**: artificial bee colony is implemented by the open-source package *beecolpy* [55]. The package has been forked and customized for this specific work. The meta-heuristic is included with a fixed colony size of 60, of which 30 acts as scouts.

- **PSO**: implemented by the package *PySwarms* [51], Particle Swarm Optimization has been included with a population size of 10 or 30 and a local or global-oriented strategy.

Another important aspect of any optimization algorithm is runtime. None of the algorithms considered have a default stopping criterion, and while the fitness improvement is likely to plateau at some point, the exact time required for this to happen can not be forgone. To ensure fair comparisons, each algorithm is bound to a maximum *budget* of $10,000$ samples from the fitness function, after which they are stopped.
The meta-model is not meant to estimate the efficiency of different algorithms, but is limited to estimating their performances given a fixed budget. However, its extensibility allows to easily include new algorithms, allowing to add variations with different budgets and predict whether an increment in runtime corresponds to a worthy improvement in the solution's fitness, or to re-train the model for different budget scenarios.

3.1.3 Function set F

Another component of the meta-optimization model, defined in section 3.1, is the universe of functions **F**.

To build a more reliable model, the universe of functions has been split into two sets, one for deterministic functions and one for stochastic functions. On the two sets of functions, F_{D} and F_{ND}, two corresponding models have been trained.
In both cases there are no explicit requirements for a function to be usable by the meta-model except for producing a real number as output and having known ranges and no constraints for the independent variables.
However the training set is made of functions with dimensionality between 2 and 6, and its generalization capabilities to functions with higher dimensionality haven't been tested. For functions with more than 6 independent variables it is recommended to re-train the model on a more suitable dataset. As for the ranges of the independent and dependent variables, to ensure enough diversity the model has been trained on functions with a wide variety of ranges, between 1 and 5000. For the non-deterministic dataset, randomness is introduced by adding or multiplying noise from a uniform distribution; the amount of noise introduced is always limited, and as we'll see later none of the optimization algorithms currently included in the meta-model is suitable for higher level of noise.
The set of functions used to build the training dataset of the model consists of a set of 201 benchmark functions, both implemented directly or taken from an open source benchmark functions set [71].

Name	**Dimensionalities**	**Input domain**
Rana	2, 4, 6	$[-20, 20]$
Rana	2, 4, 6	$[-60, 60]$
Rana	2, 4, 6	$[-120, 120]$

Name	**Dimensionalities**	**Input domain**
Sin	2, 4, 6	$[-10, 10]$
Sin	2, 4, 6	$[-20, 20]$
Sin	2, 4, 6	$[-30, 30]$
Schaffers	2	$[-2, 2]$
Schaffers	2	$[-3, 3]$
Schaffers	2	$[-10, 10]$
Weierstrass	2, 4, 6	$[-1, 1]$
Weierstrass	2, 4, 6	$[-4, 4]$
Weierstrass	2, 4, 6	$[-10, 10]$
DifferentPowers	2, 4, 6	$[-10, 10]$
DifferentPowers	2, 4, 6	$[-20, 20]$
BentCigar	2, 4, 6	$[-5, 5]$
Discus	2, 4, 6	$[-5, 5]$
Linear Slope	2, 4, 6	$[-50, 50]$
Ellipsoidal	2, 4, 6	$[-30, 30]$
Thevenot	2, 3, 4, 5	$[-6.28, 6.28]$
Ackley	2, 3, 4, 5	$[-32, 32]$
Ackley N. 2	2	$[-32, 32]$
Ackley N. 3	2	$[-32, 32]$
Ackley N. 4	2, 3, 4, 5	$[-35, 35]$
Adjiman	2	$[-1, 2]$
Alpine N. 1	2, 3, 4, 5	$[0, 10]$
Alpine N. 2	2, 3, 4, 5	$[0, 10]$
Bartels	2	$[-500, 500]$
Beale	2	$[-4.5, 4.5]$
Bird	2	$[-2\pi, 2\pi]$
Bohachevsky N. 1	2	$[-100, 100]$
Bohachevsky N. 2	2	$[-100, 100]$
Bohachevsky N. 3	2	$[-50, 50]$
Booth	2	$[-10, 10]$

Name	**Dimensionalities**	**Input domain**
Branin	2	$[-5, 10]$
Brent	2	$[-20, 0]$
Bukin N. 6	2	$[-15, -5]$
Colville	4	$[-10, 10]$
Cross-in-Tray	2	$[-10, 10]$
De Jong N. 5	2	$[-65.536, 65.536]$
Deckkers-Aarts	2	$[-20, 20]$
Dixon Price	2, 3, 4, 5	$[-10, 10]$
Drop-Wave	2	$[-5.2, 5.2]$
Egg Crate	2	$[-5, 5]$
Egg Holder	2	$[-512, 512]$
Exponential	2, 3, 4, 5	$[-1, 1]$
Forrester	1	$[0, 1]$
GoldsteinPrice	2	$[-2, 2]$
GramacyLee	1	$[-0.5, 2.5]$
Griewank	2, 3, 4, 5	$[-600, 600]$
HappyCat	2, 3, 4, 5	$[-2, 2]$
Himmelblau	2	$[-6, 6]$
HolderTable	2	$[-10, 10]$
Keane	2	$[-10, 10]$
Langermann	2	$[0, 10]$
Leon	2	$[0, 10]$
LevyN13	2	$[-10, 10]$
Matyas	2	$[-10, 10]$
Thevenot	2, 3, 4, 5	$[-2\pi, 2\pi]$
McCormick	2	$[-1.5, 4]$
Michalewicz	2, 3, 4, 5	$[0, \pi]$
Periodic	2, 3, 4, 5	$[-10, 10]$
Perm0dBeta	2, 3, 4, 5	$[-d, d]$
PermdBeta	2, 3, 4, 5	$[-d, d]$

Name	**Dimensionalities**	**Input domain**
Powell	2, 3, 4, 5	$[-1, 1]$
Qing	2, 3, 4, 5	$[-500, 500]$
Quartic	2, 3, 4, 5	$[-1.28, 1.28]$
Rastrigin	2, 3, 4, 5	$[-5.12, 5.12]$
Rastrigin	2, 3, 4, 5	$[-30, 30]$
Rastrigin	2, 3, 4, 5	$[-40, 40]$
Rastrigin	2, 3, 4, 5	$[-60, 60]$
Ridge	2, 3, 4, 5	$[-5, 5]$
Rosenbrock	2, 3, 4, 5	$[-5, 10]$
Rotated-Hyper-Ellipsoid	2, 3, 4, 5	$[-65.536, 65.536]$
Salomon	2, 3, 4, 5	$[-100, 100]$
Salomon	2, 3, 4, 5	$[-5, 5]$
SchafferN1	2	$[-100, 100]$
SchafferN2	2	$[-100, 100]$
SchafferN3	2	$[-100, 100]$
SchafferN4	2	$[-100, 100]$
Schwefel	2, 3, 4, 5	$[-500, 500]$
Schwefel2_20	2, 3, 4, 5	$[-100, 100]$
Schwefel 2.21	2, 3, 4, 5	$[-100, 100]$
Schwefel 2.22	2, 3, 4, 5	$[-100, 100]$
Schwefel 2.23	2, 3, 4, 5	$[-10, 10]$
Shekel	4	$[0, 10]$
Shubert	2, 3, 4, 5	$[-10, 10]$
Shubert N. 3	2, 3, 4, 5	$[-10, 10]$
Shubert N. 4	2, 3, 4, 5	$[-10, 10]$
Sphere	2, 3, 4, 5	$[-5.12, 5.12]$
Styblinski Tank	2, 3, 4, 5	$[-5, 5]$
Sum Squares	2, 3, 4, 5	$[-10, 10]$
Three-Hump	2	$[-5, 5]$
Trid	2, 3, 4, 5	$[-d^2, d^2]$

Name	Dimensionalities	Input domain
Wolfe	3	$[0, 2]$
Xin She Yang	2, 3, 4, 5	$[-5, 5]$
Xin She Yang N.2	2, 3, 4, 5	$[-2\pi, 2\pi]$
Xin She Yang N.3	2, 3, 4, 5	$[-2\pi, 2\pi]$
Xin-She Yang N.4	2, 3, 4, 5	$[-10, 10]$
Zakharov	2, 3, 4, 5	$[-5, 10]$

Table 1: Functions included in the training dataset F before data augmentation. Some functions appear with multiple allowed input domains.

From this early set, the actual dataset of functions is expanded by applying random transformations, such as rotations, scale, and traslations, to the function space, and randomly producing weighted sums or multiplications of functions, culminating in 3216 functions in total. This kind of data augmentation improves the model generalizability because optimization algorithms can be influenced by rotations and traslations of functions [62] [56], and the recursive process of defining new functions as sums or multiplications of previously defined ones (that could be themselves sums or products of other functions) produces a high variability of function landscapes. For the non-deterministic version of the model, the generated set of functions is further wrapped into stochastic functions that add or multiply noise to them. The total set of functions for the non-deterministic version is 6232.

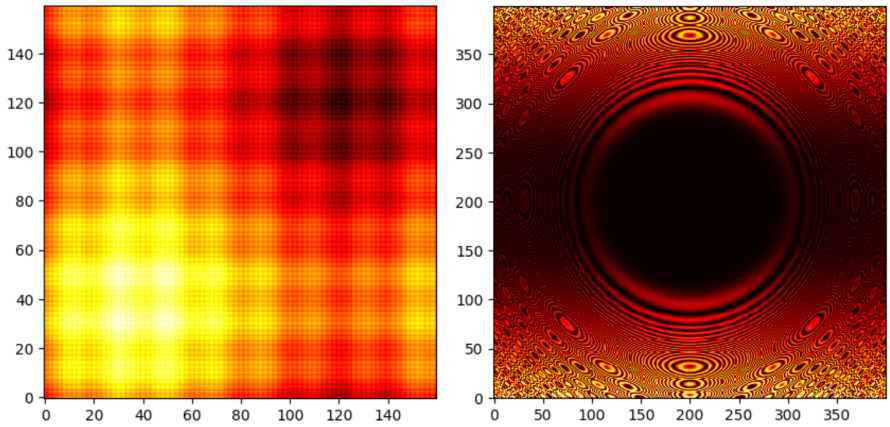

Figure 6: Examples of composite functions from the dataset. The first function is a sum between the Schwefel function on range [-10, 10] and the Weierstrass function on range [-4, 4]. The second is a multiplication between the Schaffers function on range [-10, 10] and the Weierstrass function on range [-4, 4].

3.1.4 Building the dataset and training process

Having defined the set of algorithms A, the universe of functions F and a representative subset of it F^{train} and the descriptor D, the training dataset is defined as a set of tuples $(a, D(f), Score(a, f))$ where

- a is a specific algorithm
- $D(f)$ is the FDA description of f
- $Score(a, f)$ is a performance score of algorithm a on f.

The function $Score(a, f)$ hasn't been mentioned yet. Once the function f has been optimized by a set of different algorithms A, the difference in the fitness of their respective best solutions can be observed. Building a regression model on these exact values is likely impossible, as they depend not only on the algorithms' performances but also on the ranges of the specific fitness functions, which is unlikely to be explained by the FLA measures, and not useful anyway for the purpose of the meta-optimization model.
For this reason, it is useful to convert the raw fitness values into standardized scores that are independent from the actual fitness range of the functions f. Two different *Score* functions have been tried:

- **Comparison-oriented**: linearly scales the best fitness found by each algorithm so that the best performing algorithm will score 1, and the worst will score 0. This measure maintains information on how each algorithm performs compared to the others, but discards information about the fitness range, the fitness average and the overall difference in performances between the best and the worst.
- **Performance-oriented**: linearly scales the best fitness found by each algorithm using the percentiles extracted from the fitness values. Solutions with fitness equal to the 75^{th} percentile equal to 0 and solutions with fitness equal to the 25^{th} percentile equal to 1. This measure discards information about the overall fitness ranges and the fitness average, but maintains information about individual algorithms performances.

The set of tuples$(a, D(f), Score(a, f))$ is then computed for each combination of algorithms in A and functions in F.
First the FLA measures are computed on each function in F, then each function is optimized by each algorithm, and the *scores* are computed from the results. To reduce the effect of the intrinsic randomness of these algorithms, each fitness score is averaged over 4 independent runs.

3.1.5 Model M and training process

The model has to estimate the mapping $(a, D(f)) \rightarrow Score(a, f)$ and is trained on the set of tuples $(a, D(f), Score(a, f))$.
This is a simple regression problem, and a number of models implemented in *sklearn* have been tried with increasing levels of complexity:

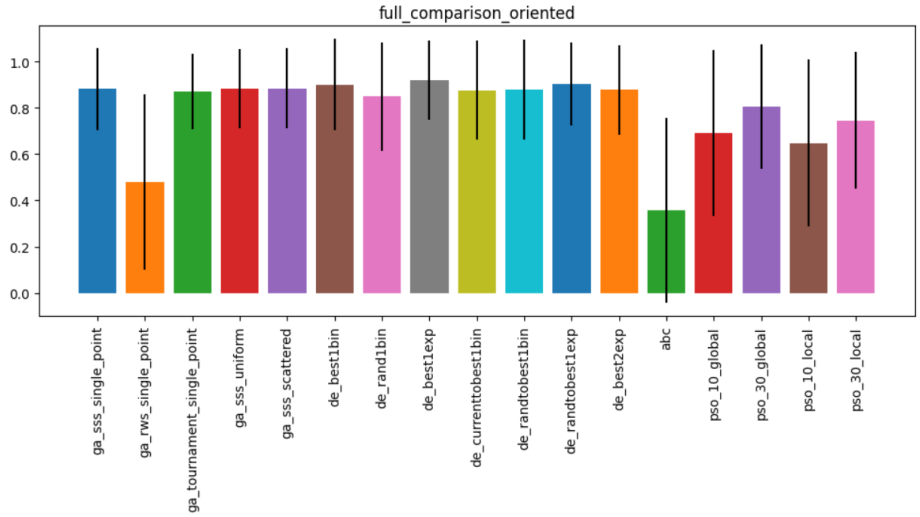

Figure 7: Average and variance of full comparison oriented scores of different algorithms over F^{train}.

- **Linear Regressor**.
- **Support Vector Regressor** with linear, polynomial and radial basis function kernels.
- **Random Forest Regressor** tested with different estimator sizes and criteria.
- **Stacking Ensemble model** consisting in 5 base models (SVR with polynomial kernel, SVR with radial basis function kernel, random forest with 50, 100 and 150 estimators) and a random forest regressor as meta-model.

A neural network regressor has been tested but later discarded because of the difficulty of using FDA measures, which show a high variability and a high number of outliers, as inputs.
The different models have been trained independently on the two sets of functions F_D and F_{ND} and for the two score functions.

3.2 Optimization of Multi-Scale Tumor Growth simulation

The second part of this thesis deals with the specific problem of optimizing the supply strategy for cancer treatments using the tumor growth model introduced in section 1.2. The fitness function is built upon the PhysiBoSs 2.0 simulator used by [57]; the independent variables to optimize are the duration and period of the TNF pulses, the concentration of TNF and the depth of the tumor at which the TNF is delivered. The fitness value used is the number of tumor cells alive at the end of the simulation, defining a minimization problem.
The meta-optimization model predicts similar results for most of the configurations of GA and DE, a slightly lower prediction for ABC and a significantly lower one for the configurations of PSO. Therefore, the first test consisted of running these three most promising algorithms on the fitness function. The results highlight the main challenges posed by this specific problem.

3.2.1 Noise

Randomness is an inherent characteristic of biological systems; moreover, it is common to adopt probabilistic phenomenological models to explain certain complex phenomena when their underlying mechanisms are not fully understood or too complicated to model [10][7].
For these reasons, the tumor growth simulator is characterized by a strong amount of noise, which presents substantial challenges for optimization. None of the algorithms included in the meta-optimization model are well-suited to handle this level of noise:

- The high variability in the sampled fitness values introduces noise into the search process, causing algorithms to mistakenly exploit solutions that appear favorable purely by chance.
- Additionally, the algorithms typically output the solution with the best observed fitness. However, due to the noise, this solution is unlikely to actually be the true optimal solution but merely one that yielded a fortunate sample.

This phenomenon creates a dual challenge: on one hand, the algorithms are misled away from converging toward the best solutions, and on the other hand, even when they output a seemingly good solution, we cannot trust that it is truly optimal.
Empirical estimates suggest the variance and standard deviation of the fitness function's noise are 511.9 and 22.6 surviving cells. To gain insight into how this level of noise impacts the evolution process of GA, the average positive difference of individual's fitness between generations, Δ_i, has been recorded while optimizing the problem, showing an average improvement of 11.6 surviving cells. This value is about half of the standard deviation of the noise. This means many of the improvements found by the algorithm, which affect the genomes passed to

later generations, are simply caused by sampling errors. When the average Δ_i is measured for individual generations the problem becomes even more concerning, as some generations show very small Δ_i, further increasing the likelihood that the algorithm is replacing individuals with worse-performing ones rather than improving the population.

Given this issue, it is useful to quantify the likelihood of replacing an individual with a worsening one when the observed Δ_i fitness is positive or zero.
A *p-value* can be defined given the following assumptions: that the noise in the fitness function can be approximated by a truncated normal distribution, the standard deviation S is constant across the search space and the fitness values are evenly distributed. Empirical tests suggest that the first two assumptions hold; regarding the third, it seems to be a valid approximation for most of the range of fitness values, while it is unlikely to hold for very low values. This means the defined *p-value* is an approximation and a lower bound, especially for low observed fitness values.
Given:

- X_i and X_{i+1}: random variables corresponding to the fitness of two solutions, in iterations i and $i+1$.
- $\tilde{x}_i$ and $\tilde{x}_{i+1}$: observed (sampled) fitness values of the two solutions.
- Assumptions: the two random variables distribute as truncated normal distributions, with unknown averages μ_i and μ_{i+1} and standard deviation $S = 22.6$.

The *p-value* can then be defined, on a minimization problem, as:

$$P(\mu_{i+1} > \mu_i | \tilde{x}_{i+1} <= \tilde{x}_i)$$

With this definition of the *p-value* and the confidence defined by (1 - p-value), the probability of the individual showing an empirical improvement of $\Delta = 11.6$, from the average observed between GA iterations, to be an actual improved individual, can be measured for different fitness values.
As can be observed for the figure 8, the *confidence* plateaus at 0.65 for solutions with a high number of surviving cells, indicating a 35% probability of accepting a worsening solution as an improvement. With better solutions, as the number of surviving cells drops to 0, the *confidence* drops to a minimum of approximately 0.3.
The truncated normal distribution is particularly unfavorable for the minimization problem because as the algorithm moves toward better solutions the disturbance caused by the noise increases to the point where the algorithm simply cannot keep improving.

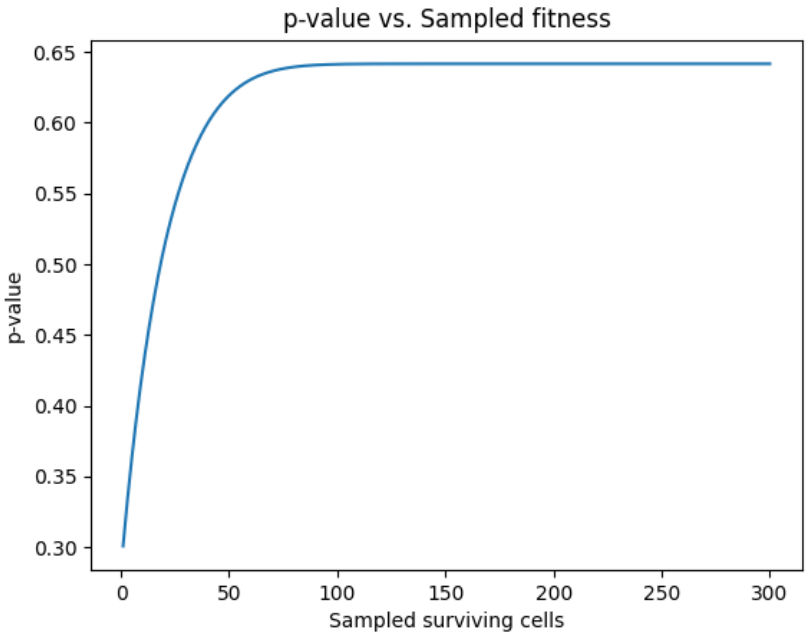

Figure 8: Confidence computed between solutions with a positive Δ_i of 11.6. With constant Δ_i, the confidence is extremely low for solutions close to 0, and plateaus below 0.65.

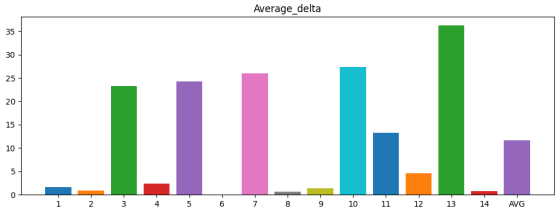

Figure 9: Average positive Δ_i of individuals between generations of GA.

Another useful measure is the confidence interval of sampled fitness measures. This allows us to understand how reliable the solutions outputted by the algorithms are. With the most desirable outcome of 0 surviving cells, a 95% confidence interval would place the value in the range $[0, 50.6]$, while a more realistic result of 10 surviving cells would result in the range $[1, 58]$. **In other words, the result of the optimization algorithm measured with a single sample is fundamentally meaningless**, as there's no way to distinguish a good solution from a lucky sampling.

The original authors repeated each simulation three times [16], reducing the standard deviation by a factor of $\sqrt{3}$. A standard deviation of approximately 13 reduces the uncertainty, but still leaves significantly large confidence intervals around the solutions found. For instance, the sampled solution with 10 surviving cells would have its confidence interval reduced only to $[0.83, 36.84]$.

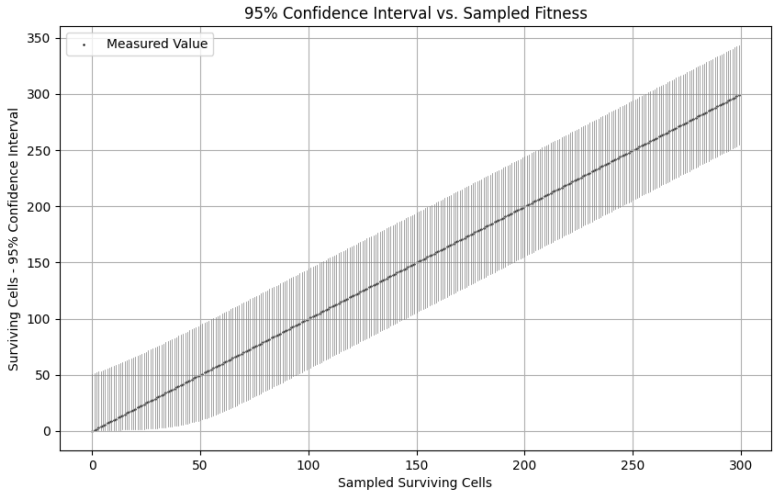

Figure 10: 95% confidence interval for solutions with different observed fitness values

This affects the predictions of the meta-optimization model on two different levels. First, since none of the algorithms included in the meta-model is suited to optimize this problem, it's clear the problem is outside of the meta-model's scope. Secondly, while the FLA measures are by nature statistical, a high level of noise in the fitness values makes them less reliable.

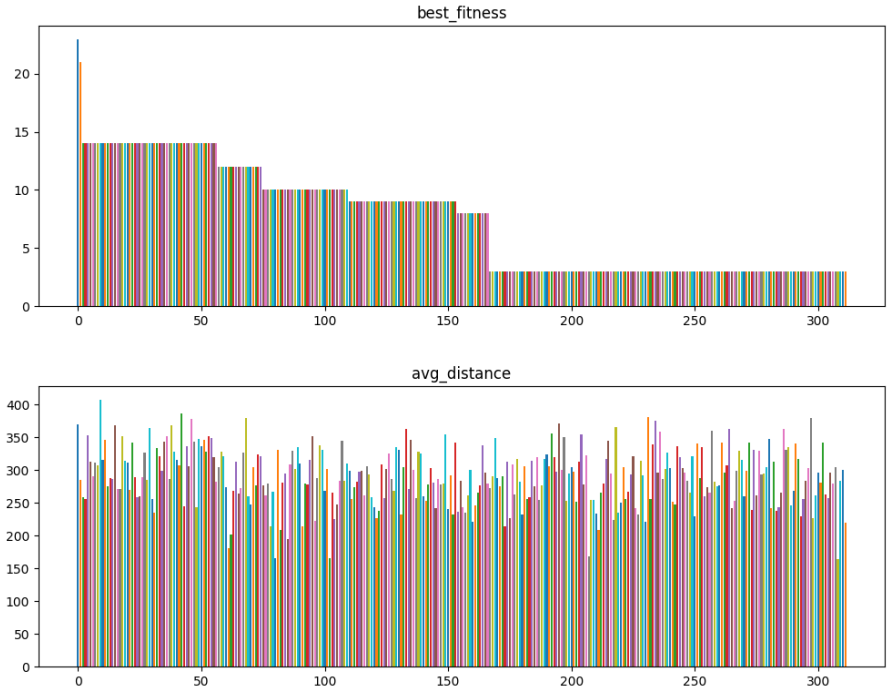

Figure 11: Best fitness and average distance of individuals in GA without explicit sampling. Fitness decreases in steps and plateaus early, while the population does not converge. The observed fitness improvements are likely to be, at least partially, fortunate samples.

3.2.2 Adapting GA for high-noise functions

Between the algorithms considered so far, GA is the one with the richest literature about the effects of noise.

J.Jin and J. Branke provide a rich survey of the effects of noisy functions on GA and possible solutions [77]. The most obvious countermeasure is explicit sampling: repeat each simulation multiple times and use the sample mean as the fitness value. Sample mean reduces the observed variance by the number of samples taken N.
Another solution is implicit sampling [77], which consists in increasing the population size: since GA tends to sample promising schemas repeatedly, with a large enough population the sampling errors from single individuals are likely to be compensated by the errors of other, similar individuals.
Both explicit and implicit sampling have the effect of linearly increasing the number of simulations run and consequently the computation time. Moreover, there is no evidence of one of the two techniques being preferable to the other.

Another common countermeasure consists of progressively increasing the sample size over the generations of GA. This approach can be interpreted as using the intrinsic noise of the fitness function to modulate the *exploration* vs *exploitation* balance, a concept that will be explored further later. Branke and Schmidt [37] follow a similar but more formalized approach, proposing to modulate the sample size to achieve the desired level of stochasticity in the Stochastic Tournament Selection operator. This strategy of progressively increasing the sample size also aligns with the previously discussed *p-values*: as the population moves toward more promising solutions, the uncertainty due to noise increases, requiring more samples.

To deal with the problem three different algorithms have been developed and tested. The first one is a hybrid algorithm consisting of a Genetic Algorithm followed by a noise-resistant implementation of Simulated Annealing. The other two are custom implementations of GA and a DE specifically adapted to work in the noisy environment.

3.2.3 Hybrid approach, GA+SANE algorithm

The hybrid algorithm developed combines two different components:

- **First Component: Genetic Algorithm (GA)** - In the initial step, a genetic algorithm is used to explore the search space, identifying promising areas and selecting a set of candidate solutions.
- **Second Component: Noise-Resistant Simulated Annealing (SANE)** - Once promising areas are identified, the algorithm switches to a noise-resistant variant of Simulated Annealing. This phase focuses on refining and evolving a set of candidate solutions.

This hybrid approach aims to balance the exploration capabilities of GA with the exploitation and noise resistance of the Simulated Annealing variant, providing a more effective optimization strategy in the noisy environment.

First Component: exploration of promising areas The goal of the first component is to explore promising areas of the search space and find a set of *candidate solutions* passed to the next component. The *candidate solutions* must cover high fitness areas of the search space and maintain a high diversity (or distance) between them. This component is implemented by a GA.

While a simple Genetic Algorithm with no explicit sampling fails to effectively optimize the problem, it can be observed by the average fitness of the population over generations that the algorithm does evolve toward more promising areas to some extent, especially in the early iterations. This is consistent with the considerations about implicit sampling: while each individual's fitness contains

noise, the whole population statistically evolves toward more promising areas. The impact of noise grows as the algorithms get closer to the most promising solutions, because the observed Δ becomes smaller and the *p-values* increase. We also observe from figure 11 that GA, when applied to the model, fails to converge toward a reduced area: the population remains spread across the search space. This is an advantage for the hybrid strategy, as having a set of promising but distant individuals increases the probability of finding the global optimum.

The applied GA implementations use steady-state parent selection, single points crossover and a relatively high mutation probability of 0.3. Implicit and explicit sampling are defined differently in three different strategies that have been tested:

1. **Low Confidence Strategy:**
    - This strategy is straightforward and selects the top N individuals with the best observed fitness across all iterations. It employs no explicit sampling and relies on implicit sampling to evolve, with a large population of 64 individuals.
    - **Advantage:** It's simple and fully utilizes all the simulation budget to evolve the population. The high noise favors a high diversity in the *candidate set*.
    - **Disadvantage:** This strategy explicitly accepts the high uncertainty of the measured fitness and consequently introduces sub-optimal individuals in the *candidate* set.
2. **Higher Confidence Strategy:**
    - This strategy uses the same population size as the previous one, but introduces an explicit sampling of 10 in its final two iterations. Only individuals from these final two iterations are considered for the candidate set, resulting in more accurate estimations of their fitness potential.
    - **Advantage:** By reducing the noise's impact, this strategy lowers the risk of including sub-optimal candidates. Applying explicit sampling only to the last two iterations limits the budget consumption.
    - **Disadvantage:** Focusing only on the last two iterations for extracting the candidate set could reduce diversity; the budget consumption caused by explicit sampling results in fewer generations, which might limit the exploration of the search space.
3. **Increasing Confidence Strategy:**
    - This strategy gradually increases the explicit sampling size as the GA progresses. The sampling size grows linearly throughout the generations, and candidates are chosen from the best individuals found during the last 50% of the generations.

- **Advantage:** This approach limits the impact of noise in the selection of the *candidate set*, while favoring a better evolution of the GA population through iterations.
- **Disadvantage:** Like the second strategy, this method reduces the number of generations used for exploration.

With each of these strategies, the *candidate solutions* are collected and filtered by removing ones too similar to others. The distance metric used is geometrical distance, as it is the one considered by the next component's algorithm. To reduce the impact of the different scales of the different dimensions, solutions are standardized by fitting each dimension in a $[0, 1]$ range; then a threshold of 0.08 minimum distance is imposed.

Second component: refining candidate solutions The second component individually optimizes each *candidate solution* with a noise resistant, single-state algorithm.
While a simple Hill Climbing with an explicit sampling size of 10 has been tested too, the main choice is a variation of the Simulated Annealing algorithm, explicitly adapted for noisy environments, called SANE (Simulated Annealing in Noisy Environments), designed by J. Branke, S. Meisel and C. Schmidt.[15]
The premise of Simulated Annealing is similar to Hill Climbing: it iteratively explores the neighborhood of the current solution, chooses a more promising one and repeats until a stopping criterion is met. Differently from Hill Climbing, Simulated Annealing adopts a stochastic acceptance criterion, where the probability of accepting a given solution is proportional to the observed fitness improvement, but worsening solutions have a non-null acceptance probability; this probability is modulated by the *temperature* T [64]. As the temperature gets close to *Infinite*, the algorithm behaves more similarly to a random walk, not considering fitness at all, while with a temperature close to 0 it would behave closely to a Hill Climbing. This should allow the algorithm to escape small local minima and keep exploring the area, while gradually reducing exploration in favor of exploitation.
This acceptance probability can be implemented by different criteria, such as the *Metropolis* criterion:

$$P^{\text{metrpolis}}(\Delta E) = \begin{cases} 1 & : \Delta E \leq 0 \\ \mathrm{e}^{-\Delta E/T} & : \Delta E > 0 \end{cases}$$

or the *Glauber* criterion

$$P^{\text{glauber}}(\Delta E) = 1/(1 + e^{\Delta E/T)}$$

In both cases the criterion computes a probability of accepting the considered solution that is a function of the Δ fitness and the temperature.

Simulated Annealing not only voluntarily introduces noise into its decision process, but formalizes it, making it the perfect candidate to employ the powerful

strategy of using the fitness function's intrinsic noise to the algorithm's advantage.
SANE is based on two main principles:

- Since SA introduces randomness into its decision process, it is possible to neutralize at least part of the fitness function's intrinsic noise by making the decision process more deterministic.
- Since the decision's probability is a function of the temperature and the observed Δ, multiple samples can be taken during the decision process until the required level of certainty is met; a concept called sequential sampling.

The work of J. Branke et. al is built upon previous works from other authors. In particular Fink [23] observed that a cumulative normal distribution with a given variance approximates the Glauber acceptance criterion with a given T. Fink observed that by making the decision process deterministic and modulating the variance of the noise by resampling until necessary, it is possible to approximate the behavior of the Gluber criterion on a deterministic fitness function. This approach has three drawbacks, as highlighted by the authors: there is an implicit maximum temperature allowed, defined by the variance of the fitness function's noise distribution, the discrete nature of the number of samples does not allow for arbitrary temperatures T and the procedure requires an impossibly high number of samples for low temperature values.
Ceperley and Dewing instead proposed an acceptance criterion that includes the noise's variance into account [19].

$$P^{\text{Ceperley Dewing}}(\hat{\delta}) = \begin{cases} 1 & : \hat{\delta} \leq -\frac{1}{2}\sigma^2/T \\ e^{-(\hat{\delta}/T + \frac{1}{2}\sigma^2/T)} & : else \end{cases}$$

This formula approximates the Metropolis criterion for small variances, but for temperatures low enough or excessive variances the probability of accepting new solutions drops significantly. Ceperley and Dewing's solution does not explicitly modulate the number of samples to take, but they observe that the number of samples required to accept a neutral move ($\delta = 0$) is proportional to the ratio σ^2/T^2.
The proposal of the authors of SANE is to use the *Ceperley and Dewing* criteria when the temperature T is ¿= T^{max}, with T^{max} being the maximum temperature allowed by the stochastic annealing defined by Fink with a sample size of 1, and to extend the work of Flink and use sequential sampling to efficiently allow for arbitrary temperatures in the remaining cases.
The threshold for determining the maximum temperature obtainable with a single sample depends on the variance of the measured δ and is given by the expression:

$$T^{\max}(\sigma^2) = 1/\sqrt{8/(\pi\sigma^2)}$$

While the probability of committing an error (accepting a worsening solution) given an observed $\hat{\delta}$ and a measured σ^2 is given by

$$P_{\text{err}}(\hat{\delta})Psi(\frac{-|\hat{\delta}|\sqrt{n}}{\sigma})$$

With Ψ being the CDF for the normal distribution.

The final algorithm takes the form of:

Algorithm 1 SANE

$state \leftarrow initial_state$
repeat
  Generate x_n in the neighborhood of **state**
  if $T >= T^{\max}$ **then**
    $acceptance \leftarrow CeperleyDewingCriteria$
  else
    repeat
      Draw 1 sample from x_n, update $\hat{\delta}$ estimate
    until $P_{\text{err}} < P^{\text{Glauber}}(|\hat{\delta}|)$
    $acceptance \leftarrow 1if\hat{\delta} \leq 0else0$
  end if
  $state \leftarrow x_n if acceptance == 1$
  update temperature
until Exit condition

Compared to the work of Fink this algorithm allows for arbitrary temperatures, not restricted by the intrinsic $T^{\max}$ or by the discrete nature of the number of samples, but also allows to reduce the number of samples needed. When Fink proposed to modulate the sample size N to regulate the noise and consequently to regulate the temperature, with no regard for the magnitude of δ, this approach allows to set a desired temperature and then use the smallest number of samples needed to maintain the correct probability of committing errors.
On top of the defined algorithm, a maximum number of samples per iteration has been added, to prevent the method from consuming all the available budget on the acceptance criterion of solutions with low δ.
The hyper-parameters have been fine-tuned using grid search on the original fitness function using real *candidate individuals* found by GA. Results indicate that the best temperature is between 50 and 65, α at 0.9. The maximum number of samples degrades the performances when is too small or too big, with the best performances found around 100.

Hybrid algorithm implementation The final implementation of the hybrid algorithm employs a Genetic Algorithm implemented by the package PyGAD [24], wrapped in a custom class that uses the callbacks offered by the implementation to record individuals at each generation. The algorithm is run with

a budget of $10,000$ simulator runs, regardless of the number of samples defined by the strategy employed. After its completion, a set of 20 *candidate solutions* is extracted.
The candidate solutions are then individually optimized by the *SANE* algorithm, with a budget of 5000 simulations each. The best result provided by the fine-tuning of the *candidate solutions* is accepted as the final solution.
To measure the performances of *SANE* as the fine-tuning algorithm, the *candidate solutions* have been fine-tuned by an implementation of Hill Climbing as well, using explicit sampling with a fixed number samples equal to 10.

3.2.4 Noise Resistant Genetic Algorithm

A custom implementation of a **Noise Resistant Genetic Algorithm (nrGA)** has been designed to address the specific challenges posed by the noisy fitness function of the multi-scale model of tumor growth. The algorithm incorporates several key design considerations:

- GA deliberately introduces randomness into its search process, through its crossover and mutation operators.
- Depending on the operator chosen, parent selection can introduce noise as well. With a simple, deterministic operator, the best N individuals can be chosen. Differently, the *roulette wheel* selection operator is probabilistic, and each individual is assigned a probability of being chosen proportional to its fitness.
- A probabilistic parent selection operator slows down convergence, but maintains a higher population diversity, favoring a deeper exploration [22].
- In the presence of a noisy fitness function, a deterministic selection process behaves like a probabilistic one. Due to noise, the perceived fitness of solutions can vary, causing sub-optimal individuals to be selected as parents occasionally.

  These considerations suggest that a noisy fitness function might enhance the exploration process in a Genetic Algorithm (GA) by preventing premature convergence and encouraging population diversity, as long as it is managed correctly. To manage it correctly, two main challenges must be dealt with:

- The amount of randomness introduced in the selection operator must be explicitly controlled. If the noise is too strong, it prevents the population from evolving in a meaningful direction, or it slows convergence down too much. At the same time, since limiting the impact of noise requires costly re-samplings, the exact number of samples needed must be computed to avoid wasting computational resources. The number of samples needed depends on the specific $\hat{\delta}$ observed in the fitness between two individuals, as a bigger $\hat{\delta}$ requires fewer samples to reach the desired level of confidence.

- Since the fitness function's noise distributes similarly to a truncated normal distribution, the uncertainty is higher with better solutions. This produces an effective ceiling to the quality that can be reached by the evolution process, as, once a certain level of fitness is reached, the parent selection operator becomes fully random, preventing further evolution. Moreover, the amount of randomness of the parent selection operator would grow as the algorithms proceed with iterations, which is the opposite of the desired behavior.

The proposed solution explicitly includes the previously defined *p-value* into the parent selection operator. The *p-value* allows us to estimate, from any two different individuals and their measured fitness difference $\hat{\delta}$, the probability of the one with better observed fitness being effectively the best one. From this, a statistically significant comparison operator can be implemented by iteratively drawing samples until the computed *p-value* reaches the desired level, and then returning the solution with the best observed fitness.

Base implementation Considering the simplest implementation of a Genetic Algorithm, with deterministic parent selection, full replacement and no elitism, the main algorithm is defined as follows:

Algorithm 2 Basic GA

```
1: population ← initial_population
2: repeat
3:     //parent selection
4:     parents ← Best_NumParents_Individuals(population)
5:     //apply crossover operator
6:     population ← Crossover(parents)
7:     //apply mutation operator
8:     population ← Mutation(population)
9: until Exit condition
```

In this procedure, the only operator in which the fitness function's noise plays a role is $Best\_NumParents\_Individuals$, the parent selection operator.
With a simple, deterministic parent selection, the operator is meant to return the best *NumParents* individual, with no constraint on the specific ordering of these selected parents, nor the need to sort the remaining individuals by fitness. This means there is no need to accurately predict each individual's fitness and not all individuals need to be compared with the others. In other words, this operator simply needs to split the population into two sets, *parents* and *not parents*.
With a deterministic fitness function, this operator would likely be implemented with a sort function followed by a slice operator. But with a noisy fitness function and the need to re-sample until the required confidence is reached, higher

efficiency can be met by following the *split into two unsorted sets* strategy.

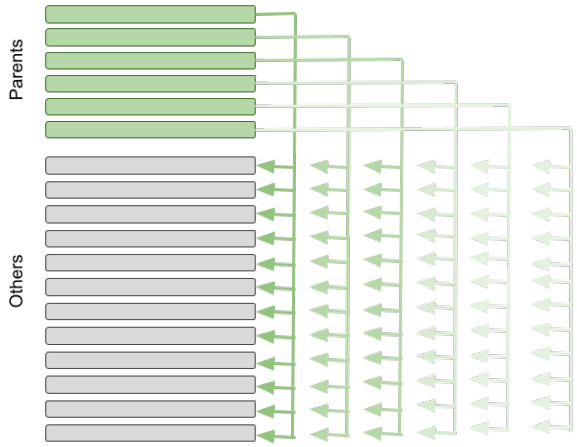

Figure 12: Required comparisons between individuals. Each arrow represents a statistically significant comparison.

An implementation of this strategy would be a variation of the *quick sort* algorithm that, compared to the standard implementation, makes only one recursive call after each *pivot* operation, depending on the *pivot* point being larger or smaller than the *NumParents*.

Algorithm 3 QuickSort parent selection

```
1: procedure ParentSelection(start, end)
2:     pivot ← NumParents − 1
3:     pivot ← SplitPopulation(start, end, pivot)
4:     //Invariant: individuals in [0, pivot] are ≤ pivotalpoint
5:     //Invariant: individuals in [pivot + 1, end) are > pivotalpoint
6:
7:     if pivot == NumParents − 1 or End == NumParents then
8:         return
9:     end if
10:
11:    if pivot >= NumParents then
12:        ParentSelection(Start, pivot+1)
13:    else
14:        ParentSelection(pivot+1, End)
15:    end if
16: end procedure
```

Where the *SplitPopulation* needs to perform N statistically significant comparisons.

While this strategy provides the lowest asymptotic complexity in the number of comparisons between individuals, it results in a far greater number of samples taken compared to the following, less asymptotical efficient algorithm:

Algorithm 4 nrGA parent selection procedure

```
1: // Sort population based on current knowledge of fitness values
2: POPULATION ← SORT(POPULATION, observedFitness)
3: i ← 0
4: while i < NumParents do
5:     j ← NumParents
6:     while j < len(POPULATION) do
7:         p_value ← COMPUTEPVALUE(POPULATION[i], POPULATION[j])
8:         if (1 − p_value) ≥ confidence then
9:             j ← j + 1
10:        else
11:            // Re-Sample from the two individuals
12:            POPULATION[I].EVALUATE(())
13:            POPULATION[J].EVALUATE(())
14:            // Re-sort population based on updated knowledge
15:            POPULATION ← SORT(POPULATION, observedFitness)
16:            // Reset i to re-evaluate the new population
17:            i ← −1
18:            break
19:        end if
20:    end while
21:    i ← i + 1
22: end while
```

While this second procedure is slower and involves a significantly greater number of comparisons, empirical results indicate that the total number of samples drawn from the fitness function is reduced by a factor of 3 to 6. This reduction can be attributed to the strategy of using sampled fitness values to sort individuals, and then comparing solutions in the *parent* set with those in the *non parent* set, which generally exhibit a greater difference in fitness, while the algorithm based on *Quicksort* compares solutions based on a random *pivot* point, requiring expensive comparisons between similar individuals. The efficiency of this approach increases as the top *NumParents* solutions exhibit a more pronounced difference in fitness compared to the rest of the population.
On this base procedure a few small optimizations have been added, such as not always re-sampling from both the individuals involved in the comparison but preferring to draw samples from the one with fewer ones taken already or skipping unnecessary *p-value* computations for solutions that have worse fitness and lower standard deviation than other solutions already compared to.
The correct implementation to use between these two depends strongly on how slow the fitness function is to compute compared to the rest of the computations done by the optimization algorithm. In the OvS scenario, the run-time of the optimization algorithm is irrelevant compared to the run-time of the simulator, thus the second procedure is preferable.

Allowing for No-Replacement and Elitism strategies The procedure described so far works only for the selection of the best *NumParents* individuals to be chosen as parents. A Genetic Algorithm with full replacement and no elitism does not need further comparisons, as each generation is made entirely of the offspring from the previous one. But other implementations introduce *Elitism*, meaning the best N individuals from the previous generation are preserved into the following, and *No-Replacement* strategies, such as *steady-state*, which add the new offsprings to the population and then filter out the worst *OffspringSize* individuals.
These two strategies require further comparisons between individuals. However the solution can be obtained using the same splitting procedure twice.

- **Elitism**: it requires three sets: the best *ElitismSize* individuals, which need to be better than all the others, the *(NumParents - ElitismSize)* individuals, which need to be better than the remaining ones, and then the remaining ones. This can be obtained by running the split operator twice, first on the entire population to find the best *ElitismSize* individuals, then on the remaining population to find the other parents.
- **Replacement**: again it requires three sets: *NumParents* that must be better than all the others, and *OffspringSize* individuals in excess that must be discarded, which need to be worse than all others. Again we can first split the population into *parents* and *others*, and then split the *others* into the sets of individuals kept and individuals removed.

Confidence schedule and hyper-parameters This parent selection operator requires specifying a desired confidence level. When a solution cannot be determined to be better than another with the level of confidence, resampling takes place. This confidence can be a simple hyper-parameter, but it makes sense to introduce a scheduled increase over the iterations, allowing for more exploration during early steps, progressively reduce it as the algorithm converges, and eventually increase it again if the optimization process reaches a plateau, meaning it hasn't been able to find new solutions for a certain number of iterations. The implementation inherits the term *temperature* from Simulated Annealing, and defines **confidence** as $1 - temperature$. The *temperature* thus represents the accepted risk of making a mistake during the comparisons between individuals.
The algorithm accepts four hyper-parameters: a starting temperature, the temperature reducing factor *alpha*, the number of iterations with no improvement found after which the temperature is reset, and the *minimum temperature* value. This last hyper-parameter is useful to prevent the algorithm from reaching excessively small temperatures, which would result in consuming the entire *budget* in a single iteration.
Hyper-parameters have been optimized using grid search and a set of benchmark functions explicitly developed to mimic the noise produced by the simulator. The results suggest that the best performance is found with a range of temper-

atures in $[0.1, 0.4]$, a relatively fast temperature decay of 0.9, and a relatively high number of parents of $1/3 * PopulationSize$. The temperature reset seems to play a small role but allows for slight improvements when set around a low number of iterations such as 20.

Standard deviation estimation In order to estimate the *p-values*, the algorithm needs an estimate of the standard deviation of the fitness function's distribution. In order to keep the estimate as accurate as possible while avoiding drawing additional samples, the following procedure is employed:

- A first *std* estimate can be provided by the user when instantiating the *nrGA* instance. If it is not provided, the algorithm produces it by taking 100 samples before starting the optimization procedure.
- The *std* estimate is updated at each iteration using the individuals from which samples have been taken during the parent selection phase. The new estimate is computed from individuals with at least 5 samples taken, and the current estimate is updated as a mean between the previous estimate and the new one.

This solution is a compromise between using all individuals from all previous generations, which would result in more samples available, but solutions would be widespread across the search space, reducing accuracy for local differences in *std*, and using only the one in the current generation, which would only estimate from points in the search space that are currently relevant but use fewer samples.

Returned solution As seen before, a noisy fitness function does not only influence the algorithms' performances, but it also reduces the confidence in the final result, as the good fitness observed in the returned solution might be a simple sampling error. The previously discussed hybrid algorithm tackles this problem by simply returning the mean from a significant number of samples. But with *nrGA*, since the whole algorithm is structured around the ability to estimate confidence intervals around solutions, a more methodical approach can be taken.
After the last generation, the algorithm estimates a 90% confidence interval for each solution, and returns the smallest set of solutions for which it's not possible, with the current sampling size, to confidently tell which one is the best. For a minimization problem, this correspond to the smallest set of solutions having a lower bound that is lower than the lowest upper bound.
From this set a single interval can be computed using the lowest lower bound and the higher upper bound. The returned solutions are wrapped into a class that allows the user to access them and to draw additional samples, reducing the final interval, until eventually only one solution is accepted as the best.

Algorithm 5 nrGA solution interval

1: **// Compute confidence intervals for the final population**
2: Intervals ← [Confidence_interval(S) for S in Population]
3: **// Compute the lowest upper bound**
4: Highest_Upper_Bound ← MIN([interval.UPPER_BOUND for interval in Intervals])
5: **// Filter intervals, keep the ones with lower bound lower than best upper bound**
6: Intervals ← Intervals.FILTER(condition: interval.LOWER_BOUND ≤ Highest_Upper_Bound)
7: **// Global interval - given current sampling size and confidence**
8: Lower, Higher ← MIN([interval.LOWER_BOUND for interval in Intervals]), MAX(MIN([interval.UPPER_BOUND for interval in Intervals])

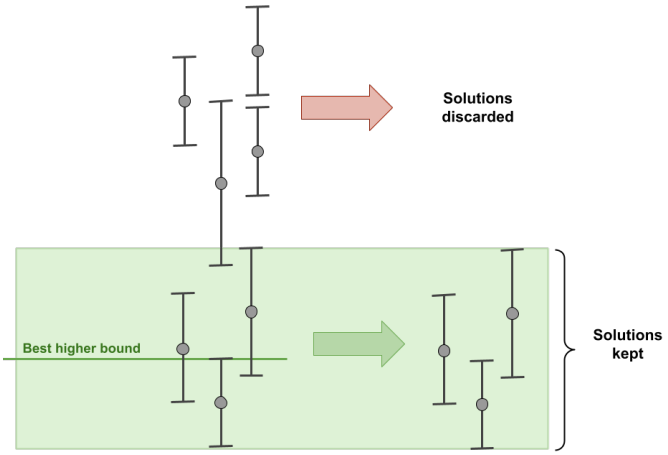

Figure 13: The population is considered as a set of intervals. Only individuals such that their lower bound is lower than the lowest higher bound are kept as solutions.

nrGA implementation Having defined the parent selection process and the hyper-parameters relative to it, the final implementation of *nrGA* is a standard Genetic Algorithm implementation with a confidence-based parent selection that performs re-sampling while necessary, a *single point* crossover, both a *normal* and a *uniform* mutation operators and that allows for both replacement and no-replacement strategies and elitism. Tests have shown that the best performance is obtained with a steady-state (no replacement) strategy, a population size of 32 and 10 parents mating at each generation.

3.2.5 Noise Resistant Differential Evolution

The same principles applied in the design of *nrGA* algorithm can be used to design a **noise resistant variant of the Differential Evolution algorithm (nrDE)**.
The original implementation of the Differential Evolution algorithm, proposed by Storn and Price in 1997 [68], is based on a randomized *crossover* operator, which selects three random individuals from the current population in order to produce one mutant vector. The mutant vector is defined as $mutant = random\_1 + F(random\_2 - random\_3)$, where the first random solution is the starting point, and the second and third random solutions are used to apply a perturbation to it. The *mutation* operator then performs a random mutation, and the *selection* operator compares each individual in the population with its own corresponding mutant vector to select the best one for the next generation.

This definition of the algorithm is particularly simple to adapt to the noisy environment as the only phase that requires comparisons, the *selection* operator, is applied exclusively to pairs of solutions. At each iteration, a constant N, equal to the population size, number of comparisons have to be made. This constant and small number of comparisons results in a low budget consumption for the algorithm compared to *nrGA*, allowing it to perform more iterations.

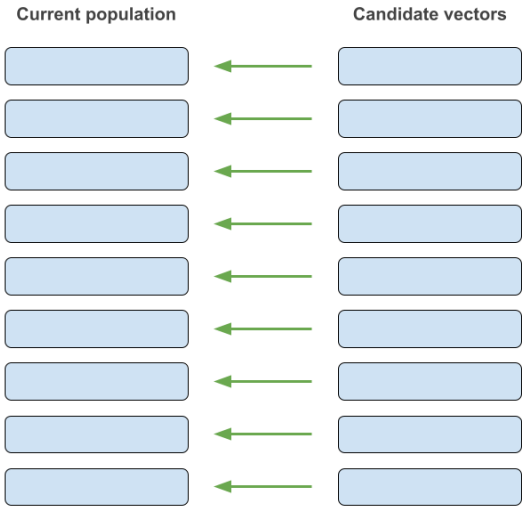

Figure 14: Required comparisons between individuals in Differential Evolution. Each arrow represents a statistically significant comparison.

The *selection* operator can be easily modified to include the *p-value* and

perform iterative re-sampling, as defined in the following pseudo-code:

Algorithm 6 nrDE selection operator

```
1: //Individual and Candidate are the two solutions to compare
2: repeat
3:     Smallest_sample_size ← MIN(Individual, Candidate, key : sample_size)
4:     Draw_Sample(Smallest_sample_size)
5:     p_value ← PValue(Individual, Candidate)
6: until (1 − p_value) ≥ Confidence
7: Return solution with smallest empirical fitness
```

Adaptive Unified Noise Resistant DE From the simple *crossover* operator defined by Storn and Prince in 1997, many variations have been proposed, based on three strategies:

- Computing crossover from two difference vectors instead of one results in better perturbations [61].
- Adding the difference vector(s) to the best known solution instead of a random one results in a faster convergence rate [50].
- Adding the difference vector(s) to the current individual instead of a random one adds the rotation-invariance property to the operator [59].

J. Qiang and C. Mitchell proposed a unified, adaptive Differential Evolution implementation [60], based on the following definition of the *crossover* operator:

$$v_{\mathrm{i}} = x_{\mathrm{i}} + F_1 * (x_{\mathrm{best}} - x_{\mathrm{i}}) + F_2 * (x_{\mathrm{r1}} - x_{\mathrm{i}}) + F_3 * (x_{\mathrm{r2}} - x_{\mathrm{r3}}) + F_4 * (x_{\mathrm{r4}} - x_{\mathrm{r5}})$$

Where v_{i} is the new mutant vector, $x_{\mathrm{r1,\ r2,\ r3,\ r4,\ r5}}$ are five different random vectors and x_{best} is the best known solution.

This operator implements ten different traditional *crossover* operators corresponding to different set of values for F_1, F_2, F_3 and F_4, as well as combinations of them.
The authors then proposed an adaptive implementation where the fours F parameters, as well as the mutation rate, are initially randomly generated from a uniform distribution, then preserved or changed depending on whether the latest iteration improved the best known solution.

This adaptive strategy cannot be directly included in the *nrDE* implementation because computing the known best solution with sufficient confidence, which J. Quiang and C. Mitchell used both in the *crossover* operator and to update the adaptive parameters, is an expensive process that would result in a fast

consumption of the optimization budget. The proposed definition of a *noise resistant adaptive unified DE* implements two changes:

1. The strategies that make use of x_{best} are removed from the adaptive *crossover* operator.
2. The parameters update operator is based on the percentage of individuals selected instead of the improvement of the best solution. This allows to use the results of the comparisons already made by the *selection* operator, without further budget consumption.

The *crossover* operator is then reduced to

$$v_{\text{i}} = x_{\text{i}} + F_1 * (x_{\text{r1}} - x_{\text{i}}) + F_2 * (x_{\text{r2}} - x_{\text{r3}}) + F_3 * (x_{\text{r4}} - x_{\text{r5}})$$

The decision to replace or keep the current parameters is probabilistic and based on the difference between the percentage of improving individuals in the current and previous generations.

- If the percentage improves, the parameters are preserved for the next generation.
- If the percentage does not improve, a random number in the range $[0, previous\_percentage]$ is generated. If the number is $\geq current\_percentage$, then the parameters are changed, otherwise they are preserved.

As in the implementation proposed by [60], when parameters are changed, they can be either sampled again from the uniform distribution or taken from a pool of previously successful parameters, with equal probability. Differently from the original implementation, which used a vector as a pool of successful parameters, where parameters that have been successfully used for N generations appear N times, this implementation uses a set, so each unique configuration that was successful at least once will have the same probability of being re-selected.

Confidence schedule and hyper-parameters The confidence level is defined as in *nrGA*, with the concept of *temperature* and the confidence defined as $1 - temperature$.
In *nrGA*, a confidence schedule was introduced, allowing to specify a temperature reducing factor *alpha* and a number of iterations without improvements after which the temperature is reset. This was meant to increase the randomicity of the *parent selection* operator, which affects the *crossover* operator, when a fitness plateau is encountered.
Compared to Genetic Algorithms, Differential Evolution is more exploration-oriented, with a *crossover* operator characterized by a high amount of randomicity. Moreover, in GA the operator affected by noise is the *parent selection* operator, and it was seen that a moderate amount of noise in this operator favors exploration, possibly allowing for unexpected and advantageous evolution directions. However, in Differential Evolution, the operator affected by noise

is the *selection* operator, which doesn't choose which individuals are used as parents but which individuals are kept or removed. The effect of randomicity in this operator is simply to risk replacing better solutions with worsening ones. With this in mind, *nrDE* implements no temperature reset and a simpler confidence schedule, where the user specifies only two hyper-parameters, *temperature_max* and *temperature_min*. The schedule linearly reduces the temperature from the maximum value to the minimum based on the percentage of the budget already consumed.

The standard Differential Evolution implementation requires only three additional hyper-parameters: the population size, the scaling factor F, which controls the differential variation in the *crossover* operator [60], and the mutation rate CR, which controls the probability of each value in the candidate vector to be replaced by the corresponding element of the parent vector. With the *adaptive unified* implementation the F hyper-parameter is removed as the algorithm itself will self regulate, and the CR hyper-parameter is defined as a range with minimum and maximum mutation rate.
Hyper-parameters have been optimized using grid search; The results indicate that best performances are found with a *temperature_min* value around 0.1 and 0.2, where higher values result in a final confidence interval too wide and lower values in worse results, likely due to the faster consumption of the budget. The population size doesn't seem to affect the performances deeply, but the results suggest that a population between 40 and 50 is preferable. For the standard configuration, CR has been found to provide the best performances around 0.8 and F around 0.5, while for the adaptive configuration CR can be assigned the wide range of $[0.5, 0.9]$, letting the algorithm choose its best value.

Standard deviation estimation and returned solutions The on-line *std* estimation and the returned solutions are implemented as in *nrGA*.

4 Results

This section presents the results of the meta-optimization model applied to generic problems and the performance of the three noise-resistant algorithms applied to the cancer growth model.

4.1 Meta-Optimization model

As previously mentioned, the meta-optimization model was trained separately for both deterministic and noisy functions. Since the results are similar, they will be presented together. Additionally, the two scoring functions correspond to separate models, resulting in a total of four distinct models.

4.1.1 Predicted scores

Before presenting the model's results, it is useful to examine the averages and variances of the two scoring functions, *comparison oriented* and *performance oriented*, on the training dataset. As illustrated in Fig. 15, the *performance oriented* scores show a low average across all included meta-heuristics, with minimal differences between algorithms, but exhibit high variance, particularly with *PSO* variants. This suggests that the *performance oriented* score is more challenging to predict, likely due to its role in not only comparing algorithms but also encoding the quality of the solutions they produce. In other words, this measure contains more information than the *performance oriented* score, which in contrast has a much smaller variance as well as clearer distinctions in mean performance across algorithms.

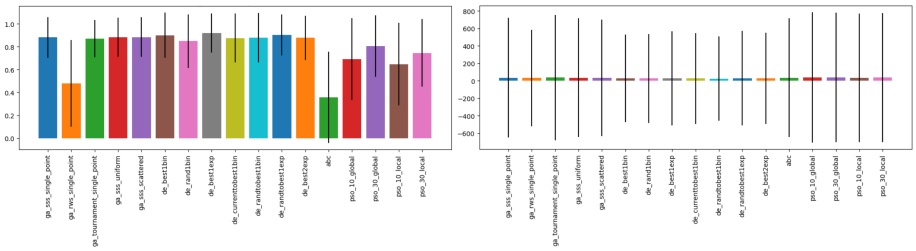

Figure 15: Averages and variances of the *comparison oriented* (left) and *performance oriented* (right) scoring functions on the training dataset. The data is presented for all meta-heuristics included in the meta-model. Variance is shown as a black line centered around the average.

4.1.2 Models' performance

The model has been tested and validated using *k-fold* with $K = 5$. Early tests tried simpler models, as a linear regressor, producing unsatisfactory results. Later, 4 different models have been trained and tested:

- Random forest model, with 100 estimators and the *squared error* criterion.
- Random forest model, with 100 estimators and the *absolute error* criterion.
- Random forest model, with 100 estimators and the *sqrt error* criterion.
- A custom developed Stacking Ensemble Model consisting of three Random Forest models, with 50, 100 and 150 estimators, a Support Vector Regressor with polynomial kernel and a Support Vector Regressor with a randiant basis kernel.

The figures 16 and 17 show the performances of the different meta-models employed on each individual meta-heuristic on the non-deterministic dataset, while figure 18 shows the result for the deterministic dataset. The measure used is the *squared error* divided by the variance observed in the score functions. A value of 1 would correspond to a model that simply predicts the score's average independently from the FLA measures. The colored bars indicate the error on the validation set, while the black dots the training errors.
Regarding the *comparison oriented* scoring, in the deterministic dataset, the best performances are achieved by the Random forest model with *sqrt* error criterion, with an average squared error of 0.690 of the original variance, very similar to the results of the Ensemble model, while for the non-deterministic datasets the Ensemble model performs the best with an average squared error of 0.586.
There are several key insights from these results:

- Some meta-heuristics exhibit significantly lower errors than others, though this trend generally aligns with the variance in their original data. Notably, *Particle Swarm Optimization* with the *comparison oriented* scoring function is somewhat harder to predict, showing similar errors to GAs and DE despite its higher initial variance.
- The relationship between FLA measures and the performance scores is intricate. Simpler models yield higher errors in both training and validation. In contrast, the ensemble model reduces validation error slightly and nearly eliminates training error, indicating that further increasing model complexity may not lead to performance gains.

This result might seem underwhelming at first sight, but considering the scale of the challenge posed by the original problem, explaining respectively 31% and 40% of the variance from the statistical FLA measures is a satisfactory result.
Another interesting validation measure is considering the difference in performances between the algorithm chosen by the meta-model on a given problem and one that would be chosen by a perfect decider. This measure is easy to compute on the validation dataset, since A^{BEST} is known and $A^{\text{META-MODEL}}$ can be estimated. The measure is then defined as $Score(A^{\text{BEST}}) - Score(A^{\text{META-MODEL}})$.
With the *comparison oriented* score function, the deterministic and non-deterministic

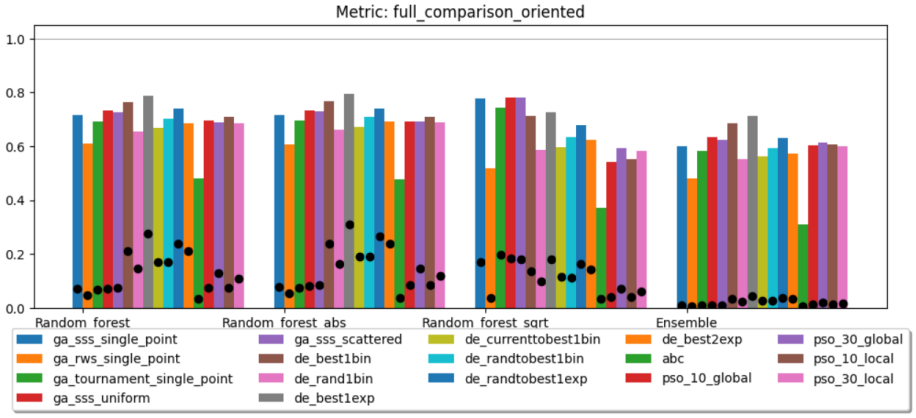

Figure 16: Results of different meta-models on the scores of each meta-heuristic - Non-deterministic dataset

.

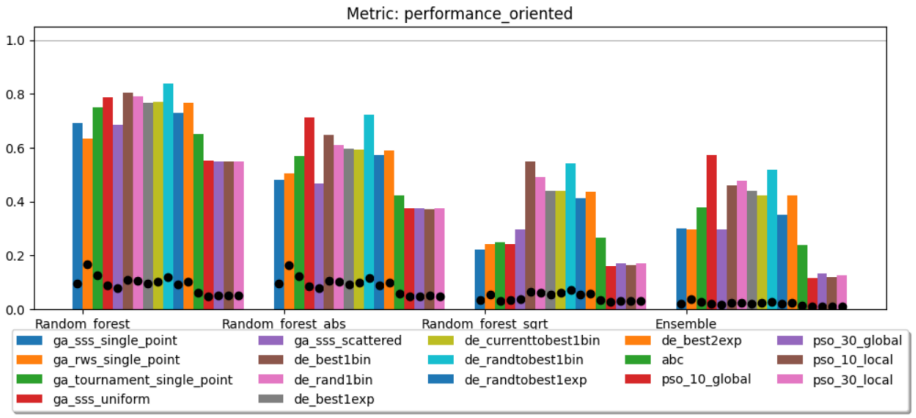

Figure 17: Errors of different meta-models on the scores of each meta-heuristic - Non-deterministic dataset

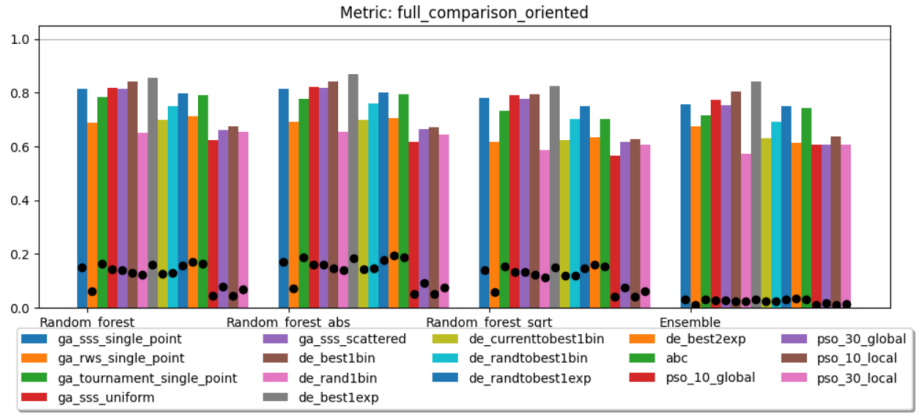

Figure 18: Results of different meta-models on the scores of each meta-heuristic - Deterministic dataset

dataset yields a value of respectively 0.094 and 0.096, indicating that while the meta-model may not always select the best algorithm, it generally chooses one that is close to optimal.

4.1.3 Computational cost of running the model

Once the meta-optimization model is trained and ready to be applied to a given fitness function, the primary computational cost to consider is the calculation of the FLA measures, which requires sampling from the fitness function.
The number of samples required for the FLA computation is specified in the meta-model configuration, and it can be freely modified. The currently used configuration requires:

- 600 samples from random points in the search space.
- 150 additional samples for the random walk required by the *FEM* measure.
- 250 additional samples required by the Jensen's Inequality Ratio measure.
- 304 additional samples required by the GA and DE specific evolvability operators.

This results in a total budget of 1304 samples from the fitness function required to compute the FLA measures.

4.2 Hybrid approach - GA + SANE

The hybrid algorithm has been tested with three different configurations to produce the candidate population, corresponding to different results.

4.2.1 Algorithm performance

Figure 19 shows the best results provided by the hybrid algorithm in the three configurations *low confidence*, *high confidence* and *increasing confidence*, as well as the average between the best five solutions. The results obtained by replacing SANE with a Hill Climbing with a fixed sampling size of 10 as the refining algorithm are provided for comparison. The *increasing confidence* configuration produces the best overall result, with the *low confidence* finding a close one. The difference reduces when considering the best 5 solutions, but the order in which the three configurations perform remains the same.
Regarding the performance of SANE compared to a simple Hill Climbing, in all of the three configurations there is a significant improvement in the results when using SANE.

Figure 20 shows the correlation between the candidate individuals' fitness and the fitness found by SANE and Hill Climbing. It can be observed that the *low confidence* configuration produces a significantly smaller correlation compared to the others: this is explained by the strong effect of noise on the candidate individual's fitness. However, in both the *low confidence* and the *high confidence* configurations, SANE exhibits a lower correlation compared to Hill Climbing, suggesting that its exploration capabilities make a difference, rendering the initial observed fitness less predictive of the final result.

Figure 19: Best result and best 5 results with SANE and with Hill Climbing.

Figure 20: Correlation between candidate individuals' fitness and refined individual fitness.

Finally, figure 21 highlights how, while SANE generally moves further away from the starting solution compared to Hill Climbing, both algorithms remain in the general proximity of the initial solution.

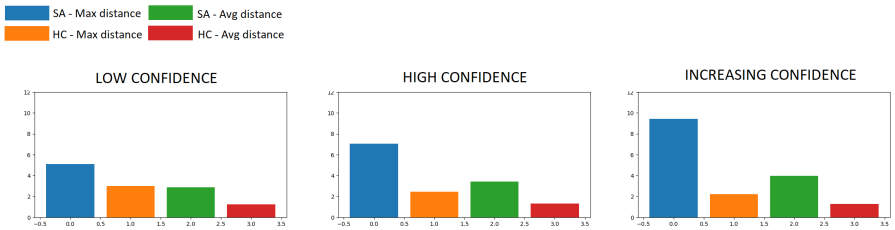

Figure 21: Average distance between candidate individuals and refined individuals.

4.2.2 Computational cost of running the hybrid algorithm

The hybrid algorithm is computationally expensive to run, because it requires running a Genetic Algorithm to produce the candidate solutions, and then running the Simulated Annealing algorithm individually on each one of them.
With the current configurations, the Genetic Algorithm component is assigned a budget of $10,000$ samples from the fitness function, and the Simulated Annealing component another $5,000$ samples. With a candidate population size of 20, the total cost amounts to $110,000$ simulations run. The average time for running one simulation on a 8-cores, 2.80GHz Intel i7 CPU with 16GB of RAM is approximately 4.25 seconds, taking the runtime of the entire algorithm to approximately 5.4 days.

4.3 Noise Resistant Genetic Algorithm

The Noise Resistant GA has been tested in two different configurations, with and without replacement, each using a budget of $8,000$ samples. Both configurations were assigned a population size of 32, with 10 parents at each generation,

a *temperature* in the range $[0.4, 0.3]$ with a reset interval of 20 generations without improvements. The configuration with *elitism* hasn't been tested on the tumor growth model as early benchmarks indicated poor performances, likely due to the additional comparison costs.
The following section presents both the best solutions found by the algorithms and the confidence intervals defined by the set of overlying solutions. An additional budget of 500 has been used to refine the set of overlying solutions by drawing additional samples.
As shown in figure 22, the cost of running the algorithm without replacement is significantly higher, leading to fewer iterations within the fixed budget. However, this reduction in iterations does not impact performance, as the configuration without replacement converges early, making subsequent iterations unnecessary.

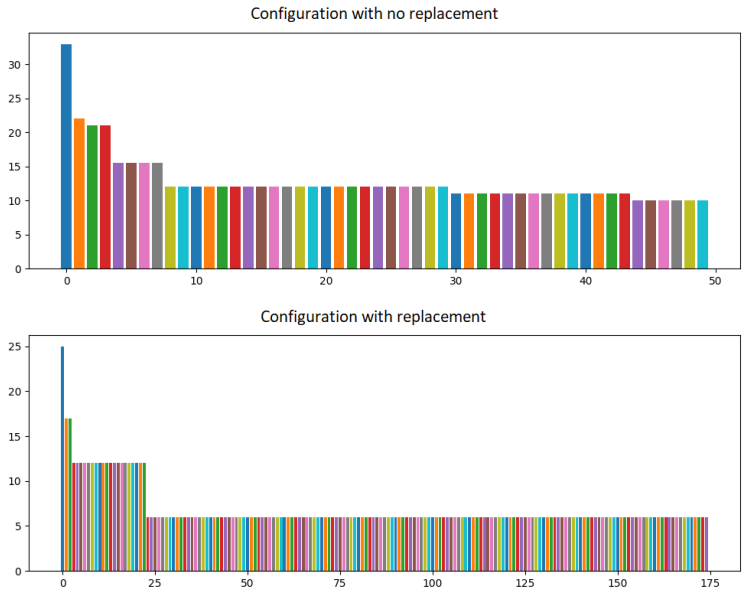

Figure 22: Best fitness by iteration of *nrGA* with and without replacement. The version with replacement is able to run longer, as it requires fewer comparisons, but reaches an early plateau.

4.3.1 Algorithm performance

The two configurations provided similar results. Both the *replacement* and *steady-state* configurations found the best fitness at 17 surviving cells, with a confidence interval of $[11.4, 22.59]$ and $[6.8, 27.3]$ respectively.
A notable difference emerged in the overlying solutions. The *steady-state* configuration returned 14 overlying solutions, which are then reduced to 1 by drawing

additional samples. In contrast, the *replacement* configuration yielded 257 overlying solution, which was narrowed to 4 using the additional 500-sample budget. This discrepancy is due to the *replacement* strategy where the entire population is replaced each iteration, resulting in solutions with only one sample by default. Conversely, the *steady-state* strategy retains fit individuals across generations, preserving all samples taken during parent selection.
However, as shown in figures 23 and 24, after additional sampling to refine the intervals, both configurations converge toward similar results, with the *replacement* configuration providing slightly better results.

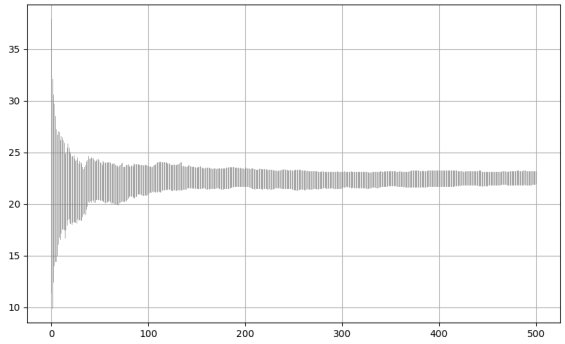

Figure 23: Confidence interval of the set of overlying solutions as additional samples are drawn. Configuration with replacement.

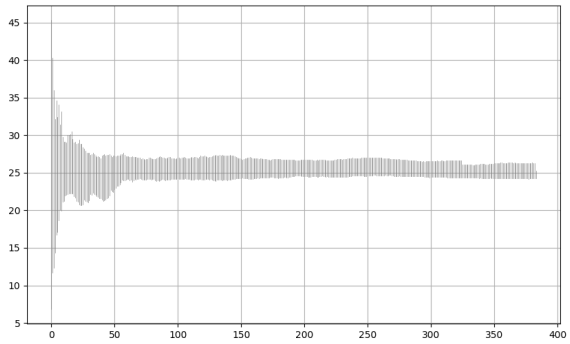

Figure 24: Confidence interval of the set of overlying solutions as additional samples are drawn. Configuration with no replacement.

4.4 Noise Resistant Differential Evolution

The Noise Resistant Differential Evolution (*nrDE*) algorithm has been tested with an assigned budget of $10,000$ samples and a population size of 48 in both its *standard* and *adaptive* configurations.
The *standard* configuration is assigned a mutation rate of 0.8 and a scaling factor of $F = 0.5$, while for the *adaptive* configuration the mutation rate range is specified at $[0.5, 0.9]$. The temperature is assigned a range of $[0.4, 0.1]$. Since the *adaptive* configuration showed a slower convergence rate, which might lead to a deeper exploration and consequently better results when given a higher budget, the algorithm has been tested with an assigned budget of $20,000$ as well.
As with the *nrGA* algorithm, both the best solution and the range characterizing the set of overlying solutions are presented, and the range is further refined with an additional budget of 500 samples.

4.4.1 Algorithm performance

Regarding the standard configuration, the algorithm's best solution presents a measured 8 surviving cells, with a confidence interval of $[1, 17]$. The set of overlying solutions presents an interval of $[8, 28]$, which reduces to $[23, 24]$ after the additional samples are drawn. The evolution of the intervals produced by *nrDE* once again highlights the likelihood of good observed solutions to be simply fortunate samples.
Figure 25 shows the best fitness found by the algorithm during iterations. Similarly to *nrGA* with replacement, despite the *exploration* oriented strategy of Differential Evolution and the large population size, the standard configuration of *nrDE* converges early, limiting its ability to make use of the budget available.

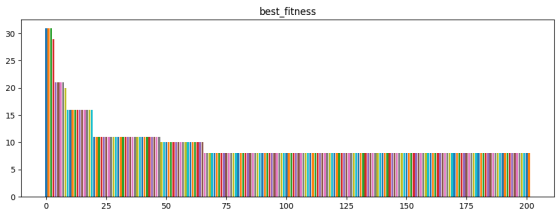

Figure 25: Best fitness during iterations of DE.

The *adaptive* implementation of *nrDE* shows a different behavior, with a slower convergence rate. With the assigned budget of $10,000$, the algorithm produces almost the same result as the standard configuration, with a sampled best fitness of 8 and a confidence interval of $[1.35, 19.35]$, but takes more than double the number of iterations to reach it. This slower convergence is easily explained by the *adaptive* nature of the algorithm. As its parameters are produced randomly and replaced or kept depending on the improvement of the

current generation, the algorithm will likely perform many iterations with sub-optimal parameters, producing little or no improvements. At the same time, this adaptive behavior might allow it to overcome fitness plateaus that the base configuration would get stuck into. To test this hypothesis, the algorithm has been run with a higher budget of $20,000$ samples.
This hypothesis seems valid, as the *adaptive* implementation with a higher budget yielded the best results, with a sampled fitness of 8 and a confidence interval of $[0.67, 14.39]$ for its best result. After refining the interval defined by the overlying solutions, the final interval is $[21.96, 22.82]$, once again the best result produced.

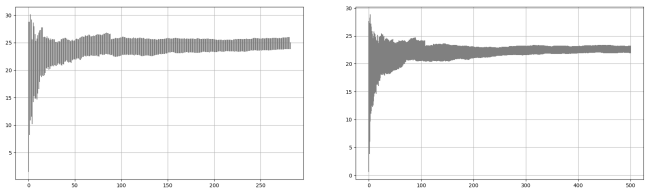

Figure 26: Confidence interval of the set of overlying solutions as additional samples are drawn.

4.5 Final considerations, limitations and future work

Meta-Optimization model : the results of the *meta-optimization model* demonstrate that it is possible, to some extent, to estimate the performance of optimization meta-heuristics using a set of statistical FLA measures; a result consistent with the work of K. M. Malan et al., who tried to apply a similar approach to *PSO* [43].
With a sampling budget of 1304 for the FLA measures computations, the meta-model predicts approximately 40% of the variance observed in the dataset and requires a complex ensemble model to achieve this. This suggests that while FLA measures provide useful information for explaining algorithm performance, the relationship between the variables is complex, and the available data can only account for part of the observed performance diversity.
It is worth considering how the budget allocated for FLA measures computations impacts meta-model performance. While naturally a higher budget will lead to more precise estimations, it is not certain that these would in turn lead to improvements in the meta-model's performance, as it's possible that the measures considered can simply not explain more than this amount of diversity. Future work could involve systematically optimizing the meta-model's hyper-parameters to evaluate performance with varying FLA measure budgets. However, this approach would be both computationally expensive and time-consuming.
Another interesting experiment would be to explore the performance of a meta-model using more complex and informative FLA measures. All the measures

currently employed are relatively inexpensive to compute, as they either share the same set of samples or require a low number of additional ones. Advanced measures, such as the *Barrier Trees* and *Local Optima Networks*, though requiring a higher budget, might provide valuable information to enhance meta-model performance. Ultimately, a balance must be struck between the accuracy of the meta-model and the computational cost of obtaining these measures.
Another limitation is the diversity of the function set F^{TRAIN}. As mentioned earlier, this set includes functions with dimensionalities between 2 and 6, and the generalizability to functions with higher dimensional functions is not assured. Additionally, the non-deterministic set is limited to functions with relatively low levels of noise. As demonstrated by the cancer growth model, the specific distribution of noise also significantly impacts performance.

The meta-model has been designed with modularity in mind, adopting an object-oriented approach for the relevant components that the user might want to extend. This design makes it easy to implement changes, such as adjusting the FLA measures' budget, adding new functions to the training dataset, incorporating additional stochastic wrappers for different noise types, or expanding the range of optimization meta-heuristics and their configurations.

Hybrid approach: The hybrid algorithm provides excellent results, but at the cost of a high run-time.
The relatively low correlation between the candidate individuals' fitness and the results of SANE indicates that reducing the candidate population's size is not a suitable method to reduce the computational cost, as there isn't a high probability of finding the best solution between the fittest candidate individuals. At the same time, empirical tests suggested that reducing the budget allocated to the individual runs of SANE results in a detriment in the results.
The high computational cost of the hybrid algorithm is likely an inevitable consequence of the hybrid approach based on multiple applications of a single-state algorithm to many starting points. Compared to a population-based algorithm, which can exploit information from all of its population to guide the search process, the single state algorithm has to iteratively explore the neighborhood of its current state to guide its next step, and not use this extracted information again.

Noise-Resistant population-based algorithms : the two noise resistant variations of Genetic Algorithms and Differential Evolution provided the most interesting results.
Both the *nrGA* and the base configuration of *nrDE* algorithms produced excellent results, comparable to those of the hybrid approach but at a significantly

lower computational cost. Compared to the 5.4 days of runtime of the hybrid algorithm, *nrGA* and *nrDE* run respectively 9.4 and 11.8 hours, while the *adaptive unified nrDE* runs for 23.6 hours. The strategy of explicitly introducing sequential sampling in the computation in the parent selection operator effectively and efficiently limits the impact of noise on the evolution process. The algorithms can be adapted to different types of problems corresponding to different noise distributions.
Both algorithms converge early, meaning they could provide the same results with an even lower computational cost. At the same time, they are not able to take advantage of the assigned budget of $10,000$ samples, since they plateau early. The *adaptive* configuration of *nrDE* on the other hand shows a slower convergence behavior, due to its inner search for better parameters for its *crossover* and *mutation* operators. This configuration reaches better results when the budget is increased to $20,000$ samples, and could provide even better ones if allowed to run for longer.

However, the evolution of the confidence interval of the overlying solutions as it is refined by drawing additional samples once again highlights the difficulty of the problem at hand. In all of the experiments, the empirical fitness of the best solution worsens as estimates become more precise and the intervals tighten toward their centers.
As mentioned in section 3.2, the p-value is computed under the assumption that fitness values are evenly distributed; an assumption that is approximated for most of the fitness range, but that becomes less and less effective when considering fitness values around the best possible.
Observing the set of overlying solutions returned by all the noise-resistant algorithms' configurations, there are a total of 96 solutions with an observed fitness value below 20, with values as low as 4. All of them are the result of the algorithms' search for the best fitness values and their measure is averaged over multiple samples. Nevertheless, none of them remains under the threshold of 20 when additional samples are taken. This suggests that near-zero solutions are rare, if existing.
If near-zero solutions actually exist, guiding the algorithm toward them is extremely difficult due to the high uncertainty around good sampled fitness values. The problem can be alleviated further by reducing the temperature to an even lower value, or by artificially requiring a minimum number of samples when the measured fitness value is below a certain empirical threshold that, observing the plots with the reducing intervals, could be placed around the value of 20. However, this would require a significantly higher computational budget, not only to run the algorithm but to test different configurations and find a satisfactory one.

Acknowledgements

I'd like to start by thanking my supervisors and Riccardo Smeriglio for giving me the opportunity to work on this thesis, for the support I received and for the freedom I was granted to explore the topics of this thesis.

A big thank you to Professor Ivan Molineris for offering me the job that supported me throughout my Master's program. The knowledge I gained and the experiences that inspired me during this time have been invaluable. I'm especially grateful for the flexibility and understanding I was shown, particularly during exam periods, without which I'm not sure I could have successfully completed this degree.

I would also like to thank my hamster, Von Neumann: your indomitable motivation to keep running on that wheel has been a true source of inspiration. Keep running, little one!

The dearest thank you goes, obviously, to my family and closest friends, for all the support over this years, for the companionship and the safe haven you always provided. When certain people are such a constant presence in your life, it's easy to take them for granted, but it's this very constancy that makes your support truly invaluable.

References

[1] M. P. Abrate, R. Smeriglio, R. Bardini, A. Savino, and S. Di Carlo. "Fast and Accurate LSTM Meta-modeling of TNF-induced Tumor Resistance In Vitro". In: *bioRxiv* (2024). DOI: 10.1101/2024.08.12.607535. eprint: https://www.biorxiv.org/content/early/2024/08/12/2024.08.12.607535.full.pdf. URL: https://www.biorxiv.org/content/early/2024/08/12/2024.08.12.607535.

[2] C. Amabile, T. Abate, C. De Crescenzo, S. Sabbarese, A. Migliaccio, S. Chianese, and D. Musmarra. "Poly(3-hydroxybutyrate) production from methane in bubble column bioreactors: Process simulation and design optimization". In: *New Biotechnology* 70 (Sept. 2022). Epub 2022 Apr 21, pp. 39–48. DOI: 10.1016/j.nbt.2022.04.004.

[3] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury. "Simulation optimization: a review of algorithms and applications". In: *Annals of Operations Research* 240 (2016), pp. 351–380.

[4] G. An. "Closing the Scientific Loop: Bridging Correlation and Causality in the Petaflop Age". In: *Science translational medicine* 2 (July 2010), 41ps34. DOI: 10.1126/scitranslmed.3000390.

[5] T. Axel. *Optimization Eye Pleasure: 78 Benchmark Test Functions for Single Objective Optimization*. https://towardsdatascience.com/optimization-eye-pleasure-78-benchmark-test-functions-for-single-objective-optimization-92e7ed1d1f12. 2020.

[6] J.C. Bansal, H. Sharma, and S.S. Jadon. "Artificial bee colony algorithm: a survey". In: *International Journal of Advanced Intelligence Paradigms* 5.1-2 (2013), pp. 123–159.

[7] Roberta Bardini et al. "A diversity-aware computational framework for systems biology." PhD thesis. Polytechnic University of Turin, Italy, 2019.

[8] Roberta Bardini, Alfredo Benso, Gianfranco Politano, and Stefano Di Carlo. "Nets-within-nets for modeling emergent patterns in ontogenetic processes". In: *Computational and Structural Biotechnology Journal* 19 (2021), pp. 5701–5721.

[9] Roberta Bardini and Stefano Di Carlo. "Computational methods for biofabrication in tissue engineering and regenerative medicine-a literature review". In: *Computational and Structural Biotechnology Journal* (2024).

[10] Roberta Bardini, Gianfranco Politano, Alfredo Benso, and Stefano Di Carlo. "Computational tools for applying multi-level models to synthetic biology". In: *Synthetic Biology: Omics Tools and Their Applications* (2018), pp. 95–112.

[11] Roberta Bardini, Gianfranco Politano, Alfredo Benso, and Stefano Di Carlo. "Multi-level and hybrid modelling approaches for systems biology". In: *Computational and structural biotechnology journal* 15 (2017), pp. 396–402.

[12] Ezio Bartocci and Pietro Lió. "Computational modeling, formal analysis, and tools for systems biology". In: *PLoS computational biology* 12.1 (2016), e1004591.

[13] B. Bilal, M. Pant, H. Zaheer, L. Garcia-Hernandez, and A. Abraham. "Differential Evolution: A review of more than two decades of research". In: *Engineering Applications of Artificial Intelligence* 90 (2020), p. 103479. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2020.103479. URL: https://www.sciencedirect.com/science/article/pii/S095219762030004X.

[14] H. R. Boveiri and R. Khayami. *On the Performance of Metaheuristics: A Different Perspective*. 2020. arXiv: 2001.08928 [cs.NE]. URL: https://arxiv.org/abs/2001.08928.

[15] J. Branke, S. Meisel, and C. Schmidt. "Simulated annealing in the presence of noise". In: *Journal of Heuristics* 14 (2008), pp. 627–654. URL: https://api.semanticscholar.org/CorpusID:13264288.

[16] L. Calzone, L. Tournier, S. Fourquet, D. Thieffry, B. Zhivotovsky, E. Barillot, and A. Zinovyev. "Mathematical modelling of cell-fate decision in response to death receptor engagement". In: *PLoS computational biology* 6.3 (Mar. 2010), e1000702. ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.1000702.

[17] Y. Carson and A. Maria. "Simulation optimization: methods and applications". In: *Proceedings of the 29th conference on Winter simulation*. 1997, pp. 118–126.

[18] A. Castrignano, R. Bardini, A. Savino, and S. Di Carlo. "A methodology combining reinforcement learning and simulation to optimize the in silico culture of epithelial sheets". In: *bioRxiv* (2023). URL: https://api.semanticscholar.org/CorpusID:266574348.

[19] D. Ceperley and M. Dewing. "The Penalty Method for Random Walks with Uncertain Energies". In: *The Journal of Chemical Physics* 110 (Dec. 1998). DOI: 10.1063/1.478034.

[20] S. Chinnasamy, M. Ramachandran, M. Amudha, and K. Ramu. "A review on hill climbing optimization methodology". In: *Recent Trends in Management and Commerce* 3.1 (2022).

[21] N. Collier and J. Ozik. "Distributed Agent-Based Simulation with Repast4Py". In: *Proceedings of the Winter Simulation Conference*. WSC '22. Singapore, Singapore: IEEE Press, 2023, pp. 192–206.

[22] A. H. Damia, M. Esnaashari, and M. Parvizimosaed. "Adaptive Genetic Algorithm Based on Mutation and Crossover and Selection Probabilities". In: *2021 7th International Conference on Web Research (ICWR)* (2021), pp. 86–90. URL: https://api.semanticscholar.org/CorpusID:235308329.

[23] T.M.A. Fink. "INVERSE PROTEIN FOLDING HIERARCHICAL OPTIMISATION AND TIE KNOTS". PhD thesis. St. John's college, University of Cambridge, 1998. URL: https://api.semanticscholar.org/CorpusID:62786994.

[24] A.F. Gad. "Pygad: An intuitive genetic algorithm python library". In: *Multimedia Tools and Applications* (2023), pp. 1–14.

[25] A. Ghaffarizadeh, R. Heiland, S. H. Friedman, S. M. Mumenthaler, and P. Macklin. "PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems". In: *PLOS Computational Biology* 14.2 (Feb. 2018), pp. 1–31. DOI: 10.1371/journal.pcbi.1005991. URL: https://doi.org/10.1371/journal.pcbi.1005991.

[26] L. Giannantoni, R. Bardini, and S. Di Carlo. "A methodology for co-simulation-based optimization of biofabrication protocols". In: *bioRxiv* (2022). URL: https://api.semanticscholar.org/CorpusID:246418793.

[27] L. Giannantoni, A. Savino, and S. Di Carlo. "Optimization of synthetic oscillatory biological networks through Reinforcement Learning". In: *bioRxiv* (2023). DOI: 10.1101/2023.11.19.567717. eprint: https://www.biorxiv.org/content/early/2023/11/19/2023.11.19.567717.full.pdf. URL: https://www.biorxiv.org/content/early/2023/11/19/2023.11.19.567717.

[28] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[29] A. Goldman, B. Majumder, A. Dhawan, S. Ravi, D. Goldman, M. Kohandel, P.K. Majumder, and S. Sengupta. "Temporally sequenced anticancer drugs overcome adaptive resistance by targeting a vulnerable chemotherapy-induced phenotypic transition". In: *Nature Communications* 6 (Feb. 2015). Published 2015 Feb 11, p. 6139. DOI: 10.1038/ncomms7139.

[30] A. A. Goldstein. "Optimization of Lipschitz Continuous Functions". In: *Mathematical Programming* 13.1 (1977), pp. 14–22. DOI: 10.1007/BF01584320. URL: https://doi.org/10.1007/BF01584320.

[31] S. M. Hadian, H. Farughi, and H. Rasay. "Development of a simulation-based optimization approach to integrate the decisions of maintenance planning and safety stock determination in deteriorating manufacturing systems". In: *Computers Industrial Engineering* 178 (2023), p. 109132. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2023.109132. URL: https://www.sciencedirect.com/science/article/pii/S0360835223001560.

[32] A. Hanif Halim, I. Ismail, and Swagatam Das. "Performance assessment of the metaheuristic optimization algorithms: an exhaustive review". In: *Artificial Intelligence Review* 54.3 (2021), pp. 2323–2409. DOI: 10.1007/s10462-020-09906-6. URL: https://doi.org/10.1007/s10462-020-09906-6.

[33] C. Hao, H. Jian-Qiang, and L. Teng. "Blackbox Simulation Optimization". In: *Journal of the Operations Research Society of China* (July 2024), pp. 2194–6698. ISSN: 1367-4803. DOI: 10.1007/s40305-024-00549-w. URL: https://doi.org/10.1007/s40305-024-00549-w.

[34] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1975.

[35] J.H. Holland. "Genetic Algorithms". In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939139 (visited on 09/20/2024).

[36] Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. "Metaheuristic research: a comprehensive survey". In: *Artificial Intelligence Review* 52.4 (2019), pp. 2191–2233. DOI: 10.1007/s10462-017-9605-z. URL: https://doi.org/10.1007/s10462-017-9605-z.

[37] J.Branke and C. Schmidt. "Selection in the Presence of Noise". In: *Genetic and Evolutionary Computation — GECCO 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 766–777.

[38] D. Karaboga, B. Gorkemli, C.Ozturk, and N. Karaboga. "A comprehensive survey: artificial bee colony (ABC) algorithm and applications." In: *Artif Intell Rev* 42 (2014). DOI: 10.1007/s10462-012-9328-0.

[39] P. Kerschke and H. Trautmann. "Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-package flacco". In: *Applications in Statistical Computing – From Music Data Analysis to Industrial Quality Improvement*. Ed. by Bauer et al. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, 2019, pp. 93–123. DOI: 10.1007/978-3-030-25147-5_7. URL: https://link.springer.com/chapter/10.1007/978-3-030-25147-5_7.

[40] Lambora, Annu, Gupta, Kunal, Chopra, and Kriti. "Genetic Algorithm-A Literature Review". In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 2019, pp. 380–384. DOI: 10.1109/COMITCon.2019.8862255.

[41] G. Letort, A. Montagud, G. Stoll, R. Heiland, E. Barillot, P. Macklin, A. Zinovyev, and L. Calzone. "PhysiBoSS: a multi-scale agent-based modelling framework integrating physical dimension and cell signalling". In: *Bioinformatics* 35.7 (Aug. 2018), pp. 1188–1196. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty766. eprint: https://academic.oup.com/bioinformatics/article-pdf/35/7/1188/48967792/bioinformatics\

_35_7_1188.pdf. URL: https://doi.org/10.1093/bioinformatics/bty766.

[42] J. Ma et al. "Active Broadband Absorber Based on Phase-Change Materials Optimized via Evolutionary Algorithm". In: *Coatings* 13.9 (2023). ISSN: 2079-6412. DOI: 10.3390/coatings13091604. URL: https://www.mdpi.com/2079-6412/13/9/1604.

[43] K. Malan and A. Engelbrecht. "Fitness Landscape Analysis for Metaheuristic Performance Prediction". In: vol. 6. Springer, Berlin, Heidelberg, Jan. 2014, pp. 103–132. ISBN: 978-3-642-41887-7. DOI: 10.1007/978-3-642-41888-4_4.

[44] K.M. Malan. "A Survey of Advances in Landscape Analysis for Optimisation". In: ().

[45] K.M. Malan and A.P. Engelbrecht. "A survey of techniques for characterising fitness landscapes and some possible ways forward". In: ().

[46] S. Mansour, A. Jalali, M. Ashjaee, and E. Houshfar. "Multi-objective optimization of a sandwich rectangular-channel liquid cooling plate battery thermal management system: A deep-learning approach". In: *Energy Conversion and Management* 290 (2023), p. 117200. ISSN: 0196-8904. DOI: https://doi.org/10.1016/j.enconman.2023.117200. URL: https://www.sciencedirect.com/science/article/pii/S0196890423005460.

[47] M. Melanie. "Genetic algorithms: An overview". In: *Complex.* 1 (1995), pp. 31–39. URL: https://api.semanticscholar.org/CorpusID:8916413.

[48] O. Mersmann, B. Bischl, H. Trautmann, P. Mike, W. Weihs, and G. Rudolph. "Exploratory landscape analysis". In: ().

[49] J. Metzcar, Y. Wang, R. Heiland, and P. Macklin. "A Review of Cell-Based Computational Modeling in Cancer Biology". In: *JCO Clinical Cancer Informatics* 3 (Feb. 2019), pp. 1–13. DOI: 10.1200/CCI.18.00069.

[50] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. "A comparative study of differential evolution variants for global optimization". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (2006). URL: https://api.semanticscholar.org/CorpusID:9027450.

[51] L.J.V. Miranda, A. Moser, and S.K. Cronin. *PySwarms*. URL: https://github.com/ljvmiranda921/pyswarms.

[52] M. Mitchell, S. Forrest, and J. H. Holland. "The royal road for genetic algorithms: Fitness landscapes and GA performance". In: 1991. URL: https://api.semanticscholar.org/CorpusID:7041271.

[53] J. Nayak, H. Swapnarekha, B. Naik, G. Dhiman, and V. Shanmuganathan. "25 Years of Particle Swarm Optimization: Flourishing Voyage of Two Decades". In: *Archives of Computational Methods in Engineering* 30 (Dec. 2022). DOI: 10.1007/s11831-022-09849-x.

[54] C. H. Norn et al. "Protein sequence design by explicit energy landscape optimization". In: *bioRxiv* (2020). URL: https://api.semanticscholar.org/CorpusID:220836140.

[55] S.C.P. Oliveira. *BeeColPy*. URL: https://github.com/renard162/BeeColPy.

[56] K. R. Opara and J. Arabas. "Differential Evolution: A survey of theoretical analyses". In: *Swarm and Evolutionary Computation* 44 (2019), pp. 546–558. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2018.06.010. URL: https://www.sciencedirect.com/science/article/pii/S2210650217304224.

[57] M. Ponce-de-Leon, A. Montagud, C. Akasiadis, J. Schreiber, T. Ntiniakou, and A. Valencia. "Optimizing Dosage-Specific Treatments in a Multi-Scale Model of a Tumor Growth". In: *Frontiers in Molecular Biosciences* 9 (2022). ISSN: 2296-889X. DOI: 10.3389/fmolb.2022.836794. URL: https://www.frontiersin.org/journals/molecular-biosciences/articles/10.3389/fmolb.2022.836794.

[58] V. K. Prajapati, M. Jain, and L. Chouhan. "Tabu search algorithm (TSA): A comprehensive survey". In: *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*. IEEE. 2020, pp. 1–8.

[59] K.V. Price. "An introduction to differential evolution". In: *New Ideas in Optimization*. 1999. URL: https://api.semanticscholar.org/CorpusID:120341790.

[60] J. Qiang and C. E. Mitchell. "A Unified Differential Evolution Algorithm for Global Optimization". In: *IEEE Transactions on Evolutionary Computation* (2014). URL: https://api.semanticscholar.org/CorpusID:55922233.

[61] K. Qin, V. Huang, and P. Suganthan. "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization". In: *Evolutionary Computation, IEEE Transactions on* 13 (May 2009), pp. 398–417. DOI: 10.1109/TEVC.2008.927706.

[62] K. Rajwar and K. Deep. "Uncovering structural bias in population-based optimization algorithms: A theoretical and simulation-based analysis of the Generalized Signature Test". In: *Expert Systems with Applications* 240 (2024), p. 122332. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.122332. URL: https://www.sciencedirect.com/science/article/pii/S0957417423028348.

[63] K. Rajwar, K. Deep, and S. Das. "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges". In: *Artificial Intelligence Review* 56 (Apr. 2023). DOI: 10.1007/s10462-023-10470-y.

[64] R.A. Rutenbar. "Simulated annealing algorithms: An overview". In: *IEEE Circuits and Devices magazine* 5.1 (1989), pp. 19–26.

[65] *SciPy*. URL: https://github.com/scipy/scipy.

[66] E. R. Soltani, H. A. Panahi, E. Moniri, N. T. Fard, I. Raeisi, J. Beik, and A. Y. Siavoshani. "Construction of a pH/Temperature dual-responsive drug delivery platform based on exfoliated MoS2 nanosheets for effective delivery of doxorubicin: Parametric optimization via central composite design". In: *Materials Chemistry and Physics* 295 (2023), p. 127159. ISSN: 0254-0584. DOI: https://doi.org/10.1016/j.matchemphys.2022.127159. URL: https://www.sciencedirect.com/science/article/pii/S0254058422014651.

[67] P.F. Stadler. "Fitness Landscapes". In: ().

[68] R. Storn and K. Price. "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11 (Jan. 1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[69] E. Tànfani and A. Testi. "Improving surgery department performance via simulation and optimization". In: *2010 IEEE Workshop on Health Care Management (WHCM)* (2010), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:35446538.

[70] E. Tekin and I. Sabuncuoglu. "Simulation optimization: A comprehensive review on theory and applications". In: *Iie Transactions* 36 (Nov. 2004), pp. 1067–1081. DOI: 10.1080/07408170490500654.

[71] A. Thevenot. $Python_Benchmark_Test_Optimization_Function_Single_Objective$. URL: https://github.com/AxelThevenot/Python_Benchmark_Test_Optimization_Function_Single_Objective.

[72] D. Thierens. "Adaptive mutation rate control schemes in genetic algorithms". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. Vol. 1. 2002, 980–985 vol.1. DOI: 10.1109/CEC.2002.1007058.

[73] A. Thomaser, M.-E. Vogt, T. Bäck, and A. V. Kononova. "Real-World Optimization Benchmark from Vehicle Dynamics: Specification of Problems in 2D and Methodology for Transferring (Meta-)Optimized Algorithm Parameters". In: *International Joint Conference on Computational Intelligence*. 2023. URL: https://api.semanticscholar.org/CorpusID:265358981.

[74] Z. Wang, L. Li, Y. Xu, H. Tian, and S. Cui. "Handover Optimization via Asynchronous Multi-User Deep Reinforcement Learning". In: *2018 IEEE International Conference on Communications (ICC)* (2018), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:51881306.

[75] B. Werth, J. Karder, A. Beham, and K. Altendorfer. "Simulation-based Optimization of Material Requirements Planning Parameters". In: *Procedia Computer Science* 217 (2023). 4th International Conference on Industry 4.0 and Smart Manufacturing, pp. 1117–1126. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.12.310. URL: https://www.sciencedirect.com/science/article/pii/S187705092202395X.

[76] S. Wright. "The roles of mutation, inbreeding, crossbreeding, and selection in evolution". In: ().

[77] J. Yaochu and J. Branke. "Evolutionary optimization in uncertain environments-a survey". In: *IEEE Transactions on Evolutionary Computation* 9.3 (2005), pp. 303–317. DOI: 10.1109/TEVC.2005.846356.

[78] H. Yoshizawa and S. Hashimoto. "Landscape analyses of search-space in travelling salesman problems". In: *Transactions of The Japanese Society for Artificial Intelligence* 16 (2001), pp. 309–315. URL: https://api.semanticscholar.org/CorpusID:67739773.