Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: chemie

Masterthesis

Automatic Optimization of Reaction Engineering Processes

Iliyas Melnikov

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: chemie

PROMOTOR :
Prof. dr. ir. Mumin Enis LEBLEBICI

BEGELEIDER :
ing. Ulderico DI CAPRIO

Gezamenlijke opleiding UHasselt en KU Leuven

Universiteit Hasselt | Campus Diepenbeek | Faculteit Industriële Ingenieurswetenschappen | Agoralaan Gebouw H - Gebouw B | BE 3590 Diepenbeek

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE 3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE 3500 Hasselt

2024
2025

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: chemie

Masterthesis

Automatic Optimization of Reaction Engineering Processes

Iliyas Melnikov

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: chemie

PROMOTOR :

Prof. dr. ir. Mumin Enis LEBLEBICI

BEGELEIDER :

ing. Ulderico DI CAPRIO

UHASSELT KU LEUVEN

PREFACE

This work marks the completion of a significant chapter in my academic journey. It has been a challenging yet rewarding experience, and I am deeply grateful to those who have supported me along the way.

First and foremost, I want to express my heartfelt gratitude to my family and friends. Their unwavering support, encouragement, and belief in me have been a constant source of motivation throughout this process. They stood by me during moments of doubt and celebrated my progress every step of the way.

I would like to express my heartfelt gratitude to my teachers for their invaluable guidance and expertise, as their feedback, patience, and encouragement have been essential in helping me reach this important milestone. The lessons I have learned from them extend beyond this work and will stay with me as I continue on my journey. Additionally, I must thank my promoter, Prof. Leblebici, for the opportunity to work under his supervision, which has significantly broadened my perspective on research and provided me with new knowledge that will greatly benefit my career. Finally, I want to sincerely acknowledge my supervisor, Dr. Di Caprio, for your unwavering guidance and support throughout this entire process.

This preface is an opportunity to acknowledge the collective effort that made this journey possible. While my words may fall short of expressing my full gratitude, I hope they convey how much I appreciate everyone who has supported me, whether directly or indirectly.

Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis investigates the application of Genetic Algorithms (GAs) in the field of reaction engineering, focusing on determining the effectiveness of GAs in solving ideal reactor chain configuration problems. Utilizing customized genetic operations, including selection, crossover, and mutation, alongside established optimization techniques such as NSGA-II, SLSQP, and DE. These algorithms are used to iteratively search for the optimal arrangement of reactor sequences, with the goal of minimizing residence time or maximizing product yield while minimizing the number of operational units. Two GAs were developed. The first presented in this thesis is for reactions involving one main reaction, the Single Reaction Genetic Algorithm (SRGA). The second GA is developed for applications within parallel reaction schemes, the Parallel Reaction Genetic Algorithm (PRGA). These GAs are optimized by utilizing hypothetical chemical reaction kinetics and complex, enhancing their ability to handle difficult cases. The optimization revealed that PRGA's framework performs faster than SRGA due to the employment of SLSQP instead of DE. It also revealed that both GAs profit more from gene chain growth mechanisms rather than gene mutation mechanisms. The optimization resulted in two robust GAs that both engineers and non-engineers can utilize to optimize ideal reactor setups efficiently.

DUTCH ABSTRACT

Deze scriptie onderzoekt de toepassing van genetische algoritmen (GAs) op het gebied van reactortechniek, met een focus op het bepalen van de effectiviteit van GAs bij het oplossen van configuratieproblemen voor ideale reactorreeksen. Hierbij worden aangepaste genetische operatoren gebruikt, waaronder selectie, crossover en mutatie, samen met optimalisatietechnieken zoals NSGA-II, SLSQP en DE. Deze algoritmen worden ingezet om iteratief te zoeken naar de optimale rangschikking van reactorreeksen, met als doel het minimaliseren van de verblijftijd of het maximaliseren van de productopbrengst, terwijl het aantal operationele eenheden wordt geminimaliseerd. Er zijn twee GAs ontwikkeld. De eerste, gepresenteerd in deze scriptie, is bedoeld voor reacties met één hoofd reactie en wordt de Single Reaction Genetic Algorithm (SRGA) genoemd. De tweede GA is ontwikkeld voor toepassingen binnen parallelle reactieschema's, genaamd Parallel Reaction Genetic Algorithm (PRGA). Deze GAs zijn geoptimaliseerd door gebruik te maken van hypothetische chemische reactiekinetiek, waardoor het vermogen om moeilijke gevallen op te lossen verbeterd. Uit de optimalisatie bleek dat het PRGA-framework sneller presteert dan SRGA, dankzij het gebruik van SLSQP in plaats van DE. Bovendien werd duidelijk dat genketen groei mechanismen beter zijn voor de GAs dan gen mutatie mechanismen. De optimalisatie resulteerde in twee robuuste GAs die zowel ingenieurs als niet-ingenieurs kunnen gebruiken om ideale reactorconfiguraties efficiënt te optimaliseren.

1 INTRODUCTION

Process optimization in the chemical industry is important for enhancing efficiency and reducing costs while improving sustainability and lowering environmental impact [1]. This involves improving various operational processes, such as identifying the optimal reactor chain setup, to achieve better yields, minimize waste, and use resources more effectively. As the chemical sector is known for its intensive energy consumption and environmental footprint [2], optimization plays a crucial role in mitigating this while maintaining profitability. Optimizing processes can be a time-consuming task and presents significant challenges due to the multitude of parameters that must be considered. Each variable can greatly affect the efficiency, safety, and cost-effectiveness of the operation. Additionally, these variables often interact in complex ways, meaning that changes to one can lead to cascading effects on others. For example, increasing the temperature may accelerate the reaction and enhance yield, but it can also raise the risk of producing unwanted byproducts or even create safety hazards such as explosions or fires [3, 4].

To handle the complexities of process optimization, chemical engineers increasingly rely on advanced modeling tools and newer methods like machine learning. These tools enable engineers to understand the interactions between various parameters and predict the behavior of a process [5]. Even with these sophisticated and innovative methods, achieving an ideal balance through traditional approaches often requires iterative testing and adjustments, which can be time-consuming and resource-intensive. However, a non-conventional tool called Genetic Algorithms (GAs) shows promise for streamlining this iterative process [6]. GAs are well-suited for dealing with complex, multi-dimensional, and nonlinear problems where traditional optimization methods show limitations [7]. It operates through mechanisms such as mutation, crossover, and selection, inspired by natural selection and genetics. This approach allows GAs to explore a vast solution space more efficiently than traditional methods without requiring gradient or derivative information, which can be difficult to obtain for complex chemical processes [6, 8].

GAs have been successfully applied across various domains within the chemical industry for optimization. For instance, researchers have used GAs to optimize autothermal ammonia synthesis reactors by adjusting key parameters such as reactor length and operating temperatures [9]. These parameters are crucial for the rate of ammonia production and energy efficiency. By exploring a wide range of potential configurations, GAs enabled researchers to identify the most effective settings to maximize output while minimizing energy consumption. This approach not only improves the profitability of the process but also reduces operational costs by ensuring the reactor operates under optimal conditions suited to the specific characteristics of the reaction.

In addition, membrane-based separation processes benefitted from the optimization of variables such as membrane thickness, surface area, and operating pressures using GAs. These variables significantly influence the effectiveness of separation by determining how well substances are filtered. Ultimately improving product quality and extending the lifespan of the membranes. By balancing these variables, researchers have maximized separation efficiency while minimizing energy consumption [10].

A different study of solid oxide fuel cells, GAs have been applied to optimize the performance of a cathode. The algorithm helped model the cell performance by varying cathode preparation temperatures, operating temperatures, and current densities. Determining the optimal conditions to maximize power output efficiently and being able to handle complex, non-linear relationships between multiple variables [11].

In another study on the optimization of multiproduct batch chemical processes, GAs have streamlined the production parameters, effectively balancing cost reduction with increased output. The research utilized GAs to address a complex multiobjective optimization problem, integrating various performance measures such as cost, production time, and revenue maximization. This approach allowed for the exploration of a wide range of possible solutions, providing a clear overview of the best options available for optimizing batch plant operations [12].

GAs have also been successfully applied in the optimization of a large-scale industrial reactor for the hydrogenation of o-cresol to produce 2-methyl-cyclohexanol. This process, characterized by its complex, high-dimensional non-linear model, posed significant challenges for conventional optimization methods. However, GAs allowed for effective navigation and optimization of this complexity, enabling the maximization of 2-methyl-cyclohexanol production while maintaining adherence to environmental constraints. This application highlights again the versatility and effectiveness of GAs in tackling complex optimization problems in chemical engineering, proving them as robust tools for enhancing process efficiency and productivity [13].

While GAs are commonly used in chemical process optimization, there has been no research focusing on their application for reactor chain configuration. The use of such algorithms is advantageous because it can shorten the process design time and eliminate the bias often associated with human-driven design. Thus this thesis will focus on developing GAs for optimizing reactor setups in complex reaction kinetics, particularly autocatalytic reactions and reactions with parallel pathways. Autocatalytic reactions, where products catalyze further reactant conversion, complicate optimal setup configurations due to their accelerated reaction rates [14]. Reactions with parallel pathways pose additional challenges as multiple pathways compete for the same reactants, necessitating designs that favor desired products while minimizing byproducts [3, 15]. Optimizing these setups requires understanding the kinetic behavior of each pathway to improve product yields and reduce waste, thereby enhancing overall chemical process efficiency and sustainability [16]. These complex reactions are prevalent across various industries, including polymerization, pharmaceuticals, and petrochemicals [17-20]. By leveraging the adaptability of GAs, this thesis aims to navigate through setups containing different reactor types, such as continuous stirred tank and plug flow reactors, with varying intermediate conversion grades or reactant concentrations. The thesis will cover a comprehensive literature review on reaction engineering and GAs, followed by the development and optimization of two GA-based tools for different scenarios. Concluding with an analysis and optimization of these tools' effectiveness and propose further research prospects.

2 LITERATURE STUDY

Chemical reactors are essential to the chemical industry. The design and optimization of reactors is critical for enhancing efficiency, safety, and economic viability [21]. Ideal reactors, such as Continuous Stirred Tank Reactors (CSTR) and Plug Flow Reactors (PFR), are fundamental models for understanding reactor behavior and guiding the design of real-world reactor setups. The fundamental difference between CSTRs and PFRs lies in their flow and mixing patterns, which affect their performance and suitability for different reactions [3, 4]. Understanding each reactor's performance equations aids in selecting the appropriate reactor type or setup for a given process.

2.1 Continuous Stirred Tank Reactor

A Continuous Stirred Tank reactor (CSTR) is an ideal reactor model where the contents are perfectly mixed, resulting in a uniform composition throughout the reactor [3, 4]. A stream of reactants is continuously fed into the reactor, and a product stream is continually removed, maintaining steady-state operation, schematic shown in Figure 1. The performance of a CSTR is determined by the balance between the rate of input, output, and reaction rate within the reactor.

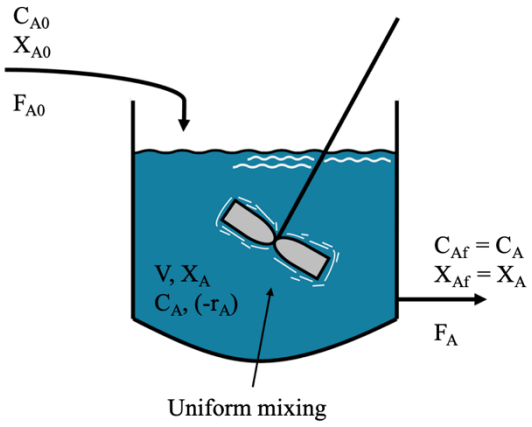


Figure 1: Schematic of a CSTR

For example, for a first-order irreversible reaction A→B, the mass balance for species A in a CSTR is given by equation 1.

$$F_{A0} = F_A + V(-r_A) \tag{1}$$

Where F_{A0} represents the molar flow rate of A entering the reactor, F_A represents the molar flow rate of A exiting the reactor, V the reactor volume, and $-r_A$ is A's rate of disappearance. Defining the residence time as the reactor volume divided by the flow rate, and knowing that the concentration equals the molar flow rate divided by the volumetric flow rate [3, 4, 22], the performance equation 2 can be derived from equation 1.

$$\tau = \frac{V}{Q} = \frac{C_{A0} - C_A}{-r_A} \tag{2}$$

Here, τ is the residence time, V the reactor volume, Q is the flowrate through the reactor, C_{A0} is the inlet concentration, and C_A is the outlet concentration of A. From equation 2, it is possible to evaluate the

performance of CSTRs. The objective is to minimize $\boldsymbol{\tau}$, resulting in a reactor that is just large enough to meet the conversion requirements of the same flowrate, thereby reducing the investment cost.

2.2 Plug Flow Reactor

A Plug Flow reactor (PFR) is an ideal reactor where reactants flow through a tubular reactor without axial mixing, creating perfect fronts where the composition changes along the length of the reactor. Each front of the volume acts as a differential element of the total volume, shown in Figure 2.

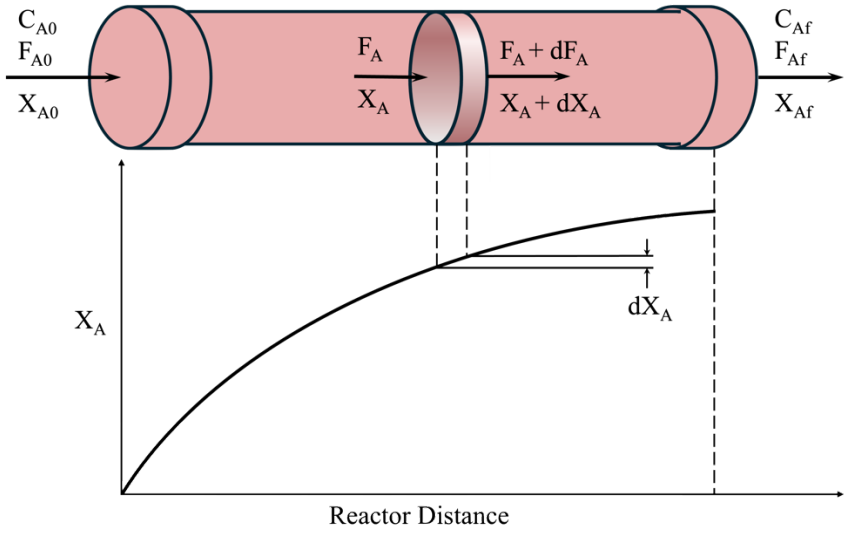


Figure 2: Schematic of a PFR

For the same first-order irreversible reaction A→B, the mass balance for species A in a PFR is represented by equation 3.

$$F_A = (F_A + dF_A) + dV(-r_A) \tag{3}$$

Where $\boldsymbol{F_A}$ represents the molar flow rate before the front, $\boldsymbol{(F_A + dF_A)}$ the molar flow rate after the front, and $\boldsymbol{dV}$ the differential volume of the front [3, 4]. After reconstructing the mass balance in terms of residence time $\boldsymbol{\tau}$ and concentration $\boldsymbol{C_A}$, integrating it across the reactor volume results in performance equation 4.

$$\tau = -\int_{C_{A0}}^{C_A} \frac{dC_A}{-r_A} \tag{4}$$

2.3 Levenspiel Plot

The Levenspiel plot can be utilized to compare and visualize reactor performance effectively. The plot graphically illustrates the correlation between the inverse of a reagent's reaction rate and its conversion. Aiding in the determination of the required reactor residence time needed to achieve the desired conversion [3], an example is shown in Figure 3.

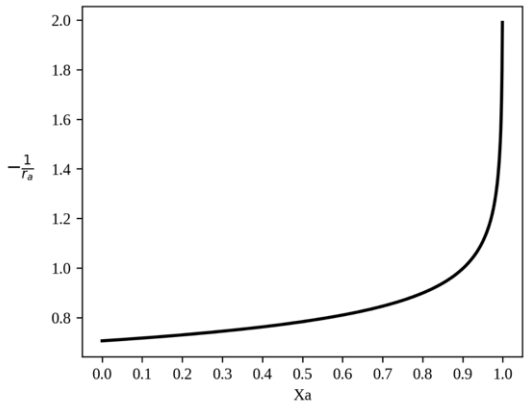


Figure 3: Example Levenspiel plot

Applying equation 4, on the Levenspiel plot reveals that a PFRs residence time is represented by the area under the curve from the initial to the final conversion level. By calculating this area and knowing the desired flow rate of the reactant, it is possible to determine the required reactor volume. This indicates that each segment along the length of the reactor contributes differently to the overall conversion, as illustrated by the area marked in red in Figure 4a. In contrast, applying equation 2 shows that the reaction rate remains constant throughout a CSTR due to complete mixing, which results in a uniform concentration throughout the reactor. This uniformity results in the reactor's performance being represented as a rectangle in the Levenspiel plot, as illustrated in Figure 4b. The rectangle's height corresponds to the inverse of the reaction rate at the outlet, while the width represents the total conversion achieved. This means that the reaction rate at the outlet determines the residence time in a CSTR.

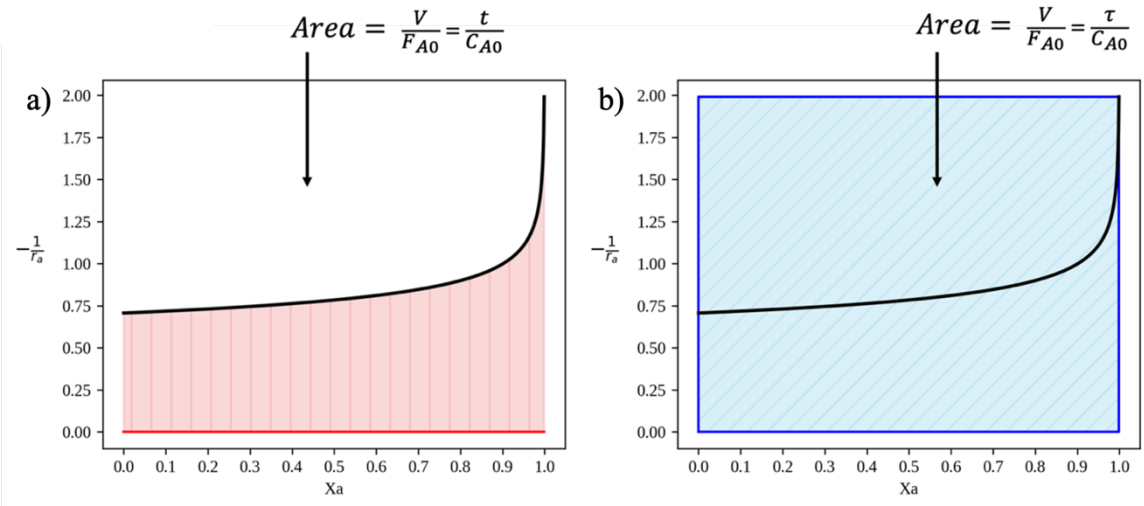


Figure 4: Graphical representation of the PFRs (a) and CSTRs (b) required residence time

2.4 Optimization with Levenspiel Plot

Working with reaction schemes that involve only one product, the setup optimization is best carried out using Levenspiel plots. The goal here is to achieve a setup with the shortest residence time for the desirable conversion. This means that the graphical surface area taken in by the reactors should be minimized. For example, Figure 5 shows a Levenspiel plot of a certain autocatalytic reaction [3, 4], where the goal is to achieve a conversion of 90%.

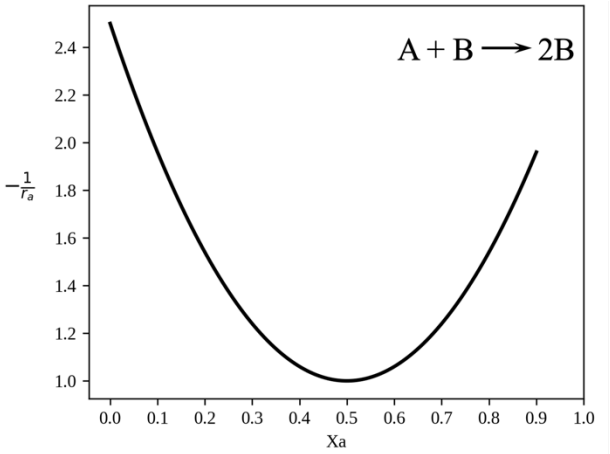


Figure 5: Example Levenspiel plot for an autocatalytic reaction

For instance, if the need is to attain a conversion rate of 90% for A, only a limited number of setups may be considered optimal. The first setup shown in Figure 6a consists of only a PFR. When using a single reactor, a PFR will have the shortest residence time in this case. However, this is not the best setup overall. Upon closer inspection, a shorter residence time can be achieved by using two reactors, with the first one being a CSTR followed by a PFR, shown in Figure 6b. The total residence time is the combined contribution from both reactors. The decrease in residence time is possible due to the outlet reaction rate of the CSTR, which determines the height of its rectangle. By adjusting the outlet conversion of the first reactor to align with the highest reaction rate, indicated at the minimum on the plot, it is possible to reduce the overall graphical area [3].

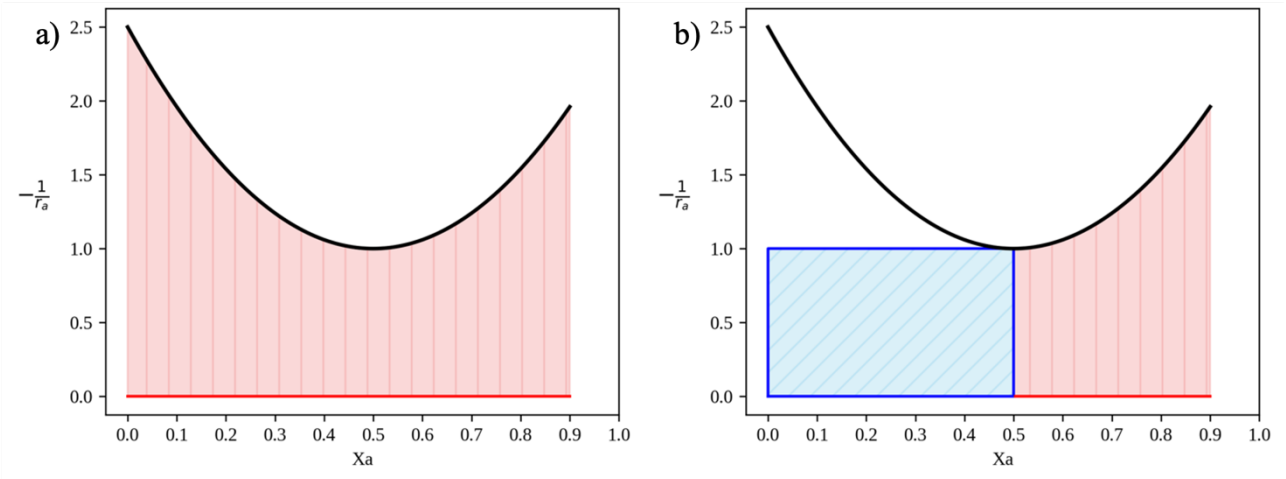
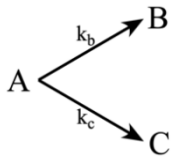


Figure 6: Levenspiel plot representation of the first (a) and second setup (b)

2.5 Parallel Reactions

Certain chemical reactions produce side products through parallel reaction pathways. A single reactant is transformed into different products simultaneously through distinct reaction pathways [3, 4, 23]. Following reaction is an example of this.



Parallel reactions are characterized by different reaction rate constants, in the case of the example, k_b, and k_c. The selectivity towards each product depends significantly on these reaction rate constants. The reaction rate constants, in turn, have a temperature dependency described by Arrhenius' law shown in equation 5.

$$k = k_0 \, e^{-\frac{E}{RT}} \tag{5}$$

Where T is the reaction temperature, R is the ideal gas constant, k_0 is the frequency, and E is the activation energy of the reaction. This indicates that the reaction rate constants change depending on the temperature and that an optimal temperature can exist for specific reaction mechanisms [3, 24].

2.6 Fraction Plot

Using a Fraction plot, as shown in Figure 7, is more effective than a Levenspiel plot for reaction schemes that involve parallel reactions [3]. The Fraction plot illustrates the relation between the overall fractional yield Φ and the reactant concentration.

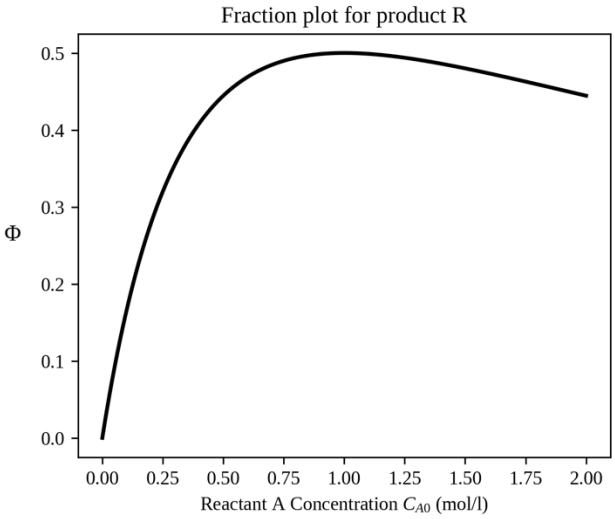


Figure 7: Example Fraction plot

The fraction of the product in the reactor is the ratio between the reaction rate of the desired product divided by the sum of its reactant's reaction. Known as the instantaneous fractional yield φ and is calculated according to equation 6 [3].

$$\varphi = \frac{dC_B}{-dC_A} \tag{6}$$

The instantaneous fractional yield is thus a function of Ca, the reactant concentration. In general, the concentration varies throughout the reactor, depending on the type of reactor used. To determine the amount of product formed, the overall fractional yield $\boldsymbol{\Phi}$ is calculated [3]. According to equation 7.

$$\Phi = \frac{C_{R_f}}{C_{A0} - C_{Af}} = \bar{\varphi}_{reactor} \tag{7}$$

Here, $\boldsymbol{C_{Rf}}$ is the product concentration at the outlet, $\boldsymbol{C_{a0}}$ the reactant's starting concentration, and $\boldsymbol{C_{Af}}$ the outlet concentration. The overall fractional yield is the mean of the instantaneous fractional yields at every point within the reactor [3]. To properly average this for a PFR where $\boldsymbol{C_a}$ changes throughout the reactor, equation 8 is used.

$$\Phi_P = \frac{1}{\Delta C_A} \int_{C_{A0}}^{C_{Af}} \varphi \, dC_a \tag{8}$$

For a CSTR where the reactant concentration equals $\boldsymbol{C_{Af}}$ throughout the whole reactor, provided there is proper mixing. The overall fractional yield will equal the instantaneous fractional yield evaluated at $\boldsymbol{C_{Af}}$ [3]. Once the overall fractional yield is calculated, the product outlet concentration can be calculated with equation 9.

$$C_{Rf} = \Phi\left(C_{A0} - C_{Af}\right) \tag{9}$$

It is possible to graphically represent the amount of product produced by a reactor using Fraction plots [3], as shown in Figure 8a. For a PFR, the total amount of product formed is determined by integrating the fraction curve from the initial to the final reactant concentration. The area under the curve reflects the amount of product produced, indicating that each segment along the reactor length contributes differently to overall production [3]. In contrast, for a CSTR shown in Figure 8b, the reactant concentration remains constant throughout the reactor due to complete mixing. This uniformity allows the production of the reactor to be represented as a rectangle on the fractional plot. The rectangle's height equals the instantaneous fractional yield evaluated at the outlet's concentration, and the width is the difference between the inlet and outlet concentration of the reactant [3]. This illustrates that the concentration of the product at the outlet determines the amount of product formed in a CSTR.

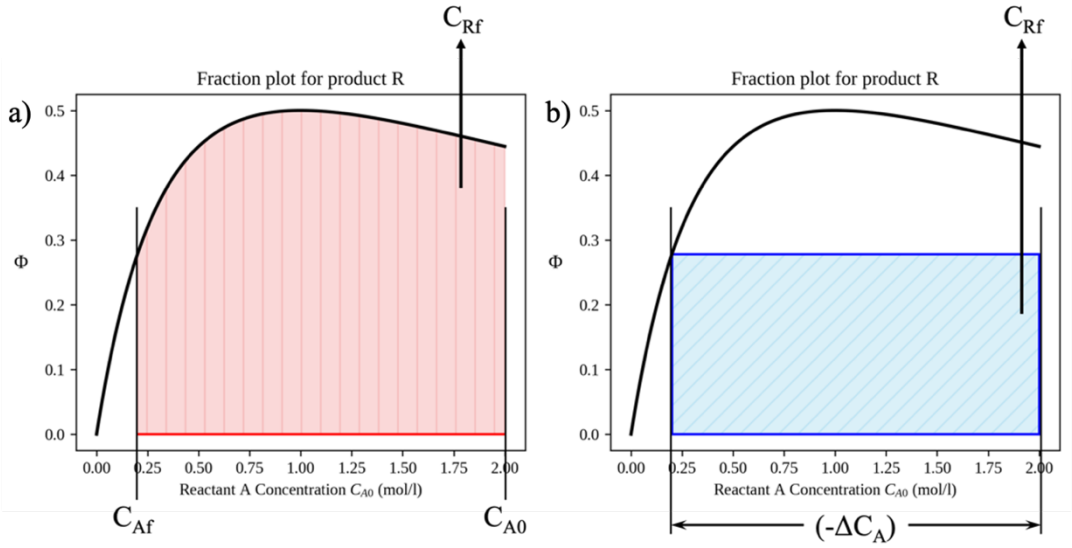


Figure 8: Graphical comparison between the produced product in a PFR (a) and a CSTR (b)

2.7 Optimization with Fraction Plot

Fraction plots optimize setups for parallel reaction schemes, maximizing a specific product's yield. Thus, maximizing the graphical surface area occupied by reactors is crucial [3]. Figure 9 illustrates a Fraction plot for a specific reaction scheme, intending to maximize the formation of product R.

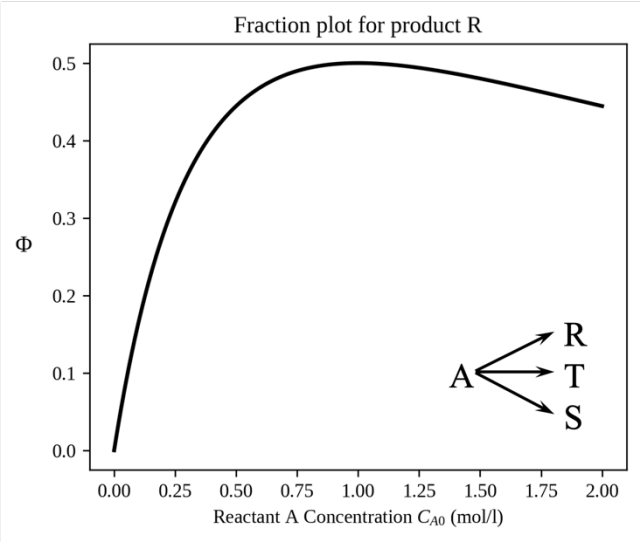


Figure 9: Example Fraction plot for certain reaction scheme

The initial configuration shown in Figure 10a illustrates the Fraction plot for a single PFR. While a single PFR can produce a considerable amount of the desired product, it may not be the most efficient approach. A closer examination of the Fraction plot in Figure 9 reveals a peak, suggesting that incorporating a second reactor, specifically a CSTR, could enhance the process [3]. The proposed strategy involves starting the reaction in a CSTR shown in Figure 10b, where the outlet concentration matches the peak concentration observed in the Fraction plot. The reaction is then completed in a PFR afterward, ultimately increasing the overall yield of the desired product. Once the necessary concentration levels across both reactors have been established, their respective residence times can be calculated [3].

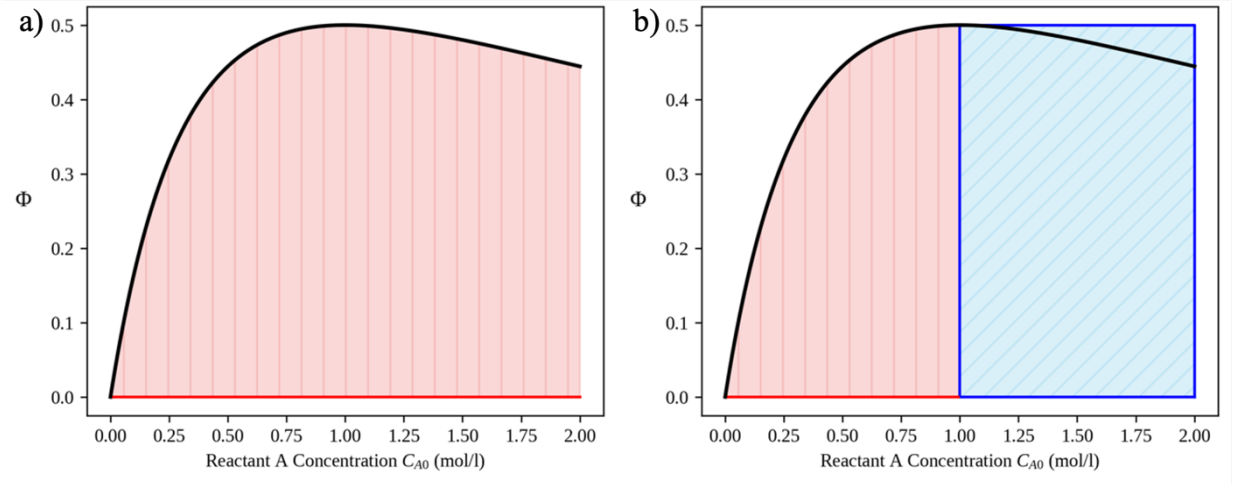


Figure 10: Fraction plot representation of the first (a) and second setup (b)

2.8 Genetic Algorithm

A GA is a stochastic search and optimization technique inspired by natural selection and genetics principles [8, 25, 26]. This involves a series of genetic operations such as selection, crossover, and mutation, enabling these algorithms to search large and complex solution spaces effectively [27, 28]. Figure 11 shows a general structure for a GA.

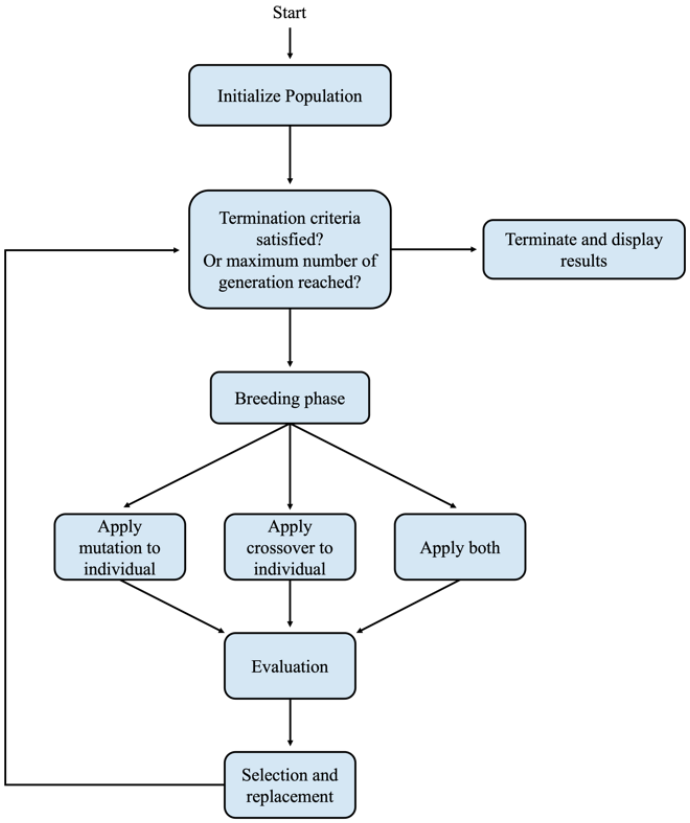


Figure 11: General structure for a genetic algorithm

The process begins with the initialization phase, where a population of individuals is generated randomly or through predefined heuristic methods to ensure diversity between the individual candidates [29]. Each individual in this population is a potential solution to the problem. For this thesis, individuals are encoded as a string of different reactors with or without their respective conversions, thus mimicking a setup.

Following the population creation, the breeding phase begins, here certain breeding operations like crossover and mutation are applied to selected pairs of individuals. Crossover is a genetic operator that imitates biological reproduction and recombination, allowing offspring to inherit beneficial traits from both parents, potentially resulting in a more fit individual, see Figure 12 [30]. In contrast, mutation introduces random genetic mutations to the offspring, maintaining genetic diversity. This randomness is essential to prevent premature convergence to local minima and ensures a thorough exploration of the solution space [31]. Further explanation regarding these operations is reported in section 3.1.5 of this thesis.

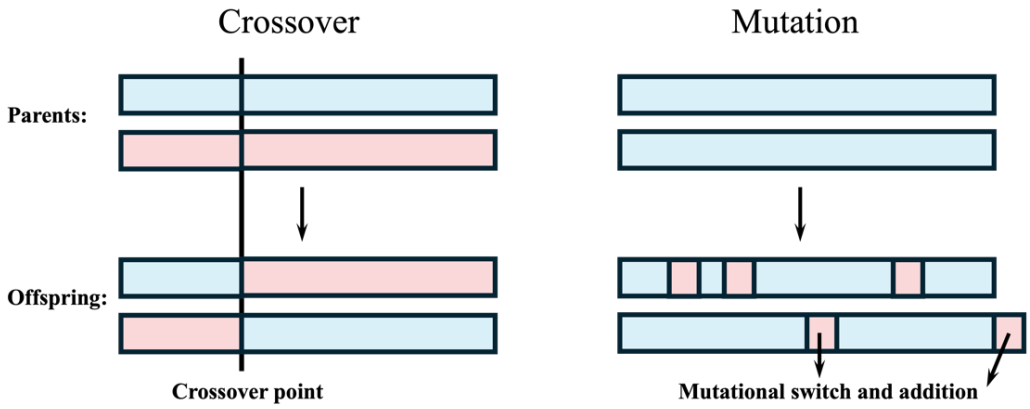


Figure 12: Schematic representation of the genetic breeding operations

After the breeding phase, a predefined fitness function evaluates each individual's fitness. This function assesses how well each individual solves the target problem by assigning a fitness score based on efficiency, determining which individuals are most likely to reproduce [29, 32].

Once the population is evaluated, the selection process begins. This phase simulates natural selection, where individuals with higher fitness are likelier to be chosen for reproduction [33]. The selection process is competitive, where individuals face off, and the fitter one is chosen through tournament selection. Alternatively, the process might involve some element of randomness, where the likelihood of an individual being selected depends on chance [34].

The GA continues through these steps until a specific stopping point is reached, such as reaching a maximum number of generations or reaching a point where no significant improvements are seen in fitness levels. When the algorithm converges, the best solution in the final population is usually considered the optimal solution to the problem [29, 35].

2.9 Differential Evolution

Differential Evolution (DE) is a population-based stochastic optimization GA widely used for solving complex, nonlinear, and multimodal optimization problems in continuous space. It includes steps such as mutation, crossover, and selection. Utilizing vectors to represent individuals, containing their direction and magnitude [36].

2.9.1 Initialization

A population can be initialized randomly or through alternative heuristic methods, such as Latin Hypercube Sampling (LHS). For random initialization, individuals are generated within the search space defined by the parameter bounds, according to equation 10.

$$x_{j,i,g} = Rand(0\,;1)\left(b_{j,max} - b_{j,min}\right) + b_{j,min} \tag{10}$$

Here, $\boldsymbol{g}$ represents the generational number, $\boldsymbol{j}$ is an input parameter within the vector $\boldsymbol{i}$, while $\boldsymbol{b_{j,max}}$ and $\boldsymbol{b_{j,min}}$ are the upper and lower bounds for $\boldsymbol{j}$. $\boldsymbol{Rand_j(0;1)}$ is a function that produces a random number uniformly distributed between zero and one for each input parameter $\boldsymbol{j}$. Once this process is completed for every individual, the entire parameter space should be fully covered, as seen in Figure 13 [37].

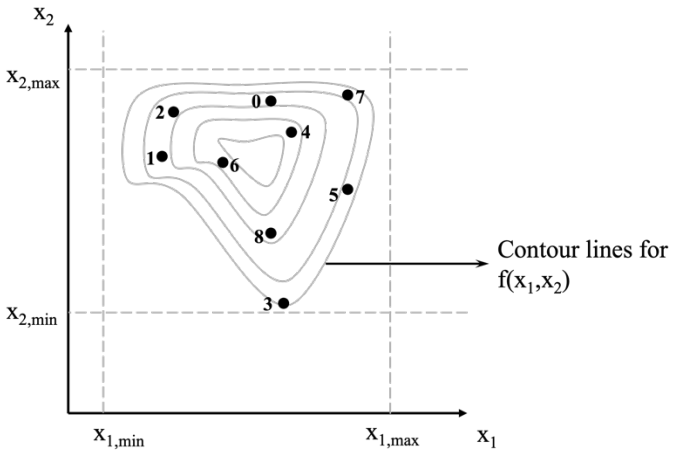


Figure 13: DE population, after initialization for random sampling

When using LHS, the resulting population will typically be more dispersed than a random initialized population. This approach divides the parameter space into slices, selecting one value from each slice. This process is described by equation 11 [38].

$$x_{j,k} = \frac{1}{F_k} \cdot \left(\frac{p_{j,k} - 1 + Rand_{jk}(0;1)}{N} \right) \tag{11}$$

In this equation, $x_{j,k}$ represents the value of the k^{th} parameter in the j^{th} vector. F_k is the distribution function of each parameter, describing its distribution. The variable $p_{j,k}$ is an integer that indicates which interval of the parameter the sample will occupy. $Rand_{j,k}(0;1)$ is a random number generator that provides an offset within the specified interval, and N is the total number of vectors to be generated. An example population generated through LHS is shown in Figure 14.

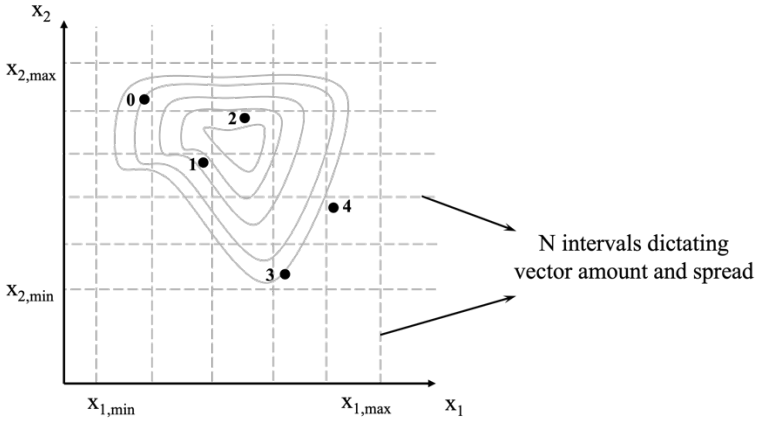


Figure 14: LHS population after initialization

2.9.2 Mutation

After initialization, DE mutates and recombines the population to form a new trial population [37]. The mutation involves adding a randomly sampled and scaled vector difference to a third vector, creating a mutant trial vector $\boldsymbol{v}_{i,g}$. This is done according to equation 12.

$$\boldsymbol{v}_{i,g} = x_{r0,g} + F\left(x_{r1,g} - x_{r2,g}\right) \tag{12}$$

Here, $\boldsymbol{F}$ is the scale factor, a positive real number that determines the rate of evolution, typically in the range of zero to one. The base vector $\boldsymbol{x_{r0,g}}$ is chosen randomly or driven by certain policy. However, it must differ from $\boldsymbol{x_{r1,g}}$ and $\boldsymbol{x_{r2,g}}$, which are also randomly selected from the individuals members [36].

2.9.3 Crossover

In addition to mutation, the DE algorithm also implements crossover strategies. Once the mutant trial vector is created, the crossover creates trial vectors out of parameter values from a randomly chosen target vector and the previously created mutant vector [37]. This is done according to the following probabilistic methodology:

if ($rand_j$ (0;1) ≤ CXPB):

$$\boldsymbol{u_{j,i,g}} = \boldsymbol{v_{j,i,g}}$$

else:

$$\boldsymbol{u_{j,i,g}} = \boldsymbol{x_{j,i,g}}$$

Here ***CXPB*** is the crossover probability, which is defined by the user. For a specific parameter $\boldsymbol{j}$, if a randomly generated number is less than or equal to ***CXPB***, the trial vector will inherit the parameter from the mutant vector. If the number exceeds ***CXPB***, the trial vector will retain the parameter from the target vector.

2.9.4 Selection

After applying mutation and crossover, all new trial vectors are formed. The selection process for the next generation follows, which involves evaluating the fitness of each vector using a predefined function. If the trial vector outperforms its original target vector, it replaces the target vector in the following generation. Otherwise, the target vector will keep its position in the following generation [36, 37].

2.10 Pareto Chart

The Pareto chart is an important tool in multi-objective optimization, representing the set of optimal solutions in which no objective can be improved without negatively impacting another. Widely used in engineering, to manage trade-offs between conflicting objectives [39, 40]. Each point on the Pareto front is a potential solution where improving one objective would cause at least one other objective to be degraded, an example is shown in Figure 15.

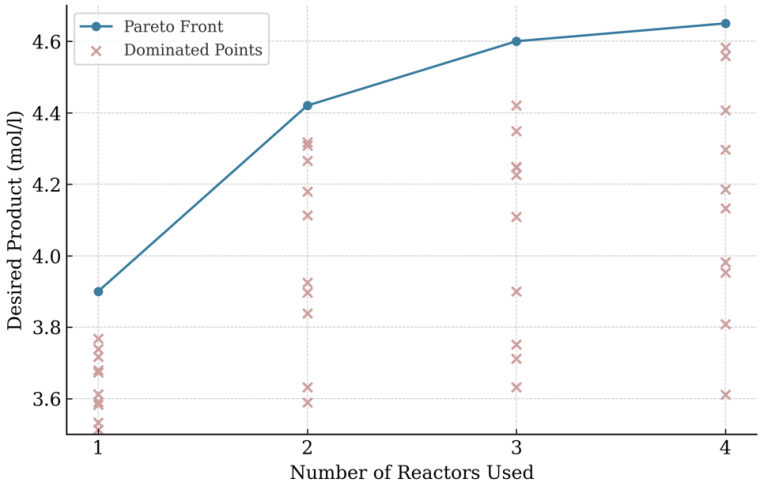


Figure 15: Example Pareto Chart

The axes in a Pareto front chart correspond to different objectives, and the connected points represent optimal solutions, known as the Pareto front. Solutions on the Pareto front are considered “non-dominated,” meaning no other solutions in the set outperform them across all objectives. For instance, in a bi-objective problem, a two-dimensional Pareto front is created, displaying the trade-off curve between the two objectives [41]. GAs, such as the Non-dominated Sorting Genetic Algorithm, often leverage Pareto Fronts to solve complex optimization problems by preserving solution diversity and guiding the search toward optimal trade-offs [42].

2.11 NSGA-II

The Nondominated Sorting Genetic Algorithm-II (NSGA-II) is known for its elitism and ability to maintain diversity in the population [42]. A key feature distinguishing NSGA-II from other genetic algorithms is its use of non-dominated sorting and crowding distance operators during the selection phase. This approach is important in effectively maintaining population diversity while highlighting superior "Elite" solutions. These mechanisms ensure that NSGA-II can manage a variety of complex problem spaces by promoting a diverse set of optimal solutions and allowing thorough exploration of the search space in multi-objective optimization scenarios [42].

2.11.1 Non-Dominated Sorting

Non-dominated sorting is a technique used to categorize the population into different layers or performance fronts based on the principle of Pareto dominance. A solution dominates another solution if it meets the following two criteria: 1) the solution performs no worse than another solution across all objectives, and 2) the solution is strictly better than the other solution in at least one objective [42].

Using this definition, the population is organized into a series of fronts. The first front is composed of solutions not dominated by any other member of the population, representing the set of optimal trade-offs among the objectives. The second front comprises solutions dominated solely by one or more solutions from the first. This process continues, with each subsequent front containing solutions dominated by those in the previous front. This structured approach allows for a clear understanding of solution quality regarding multi-objective optimization [42]. For an overview of the sorting algorithm, see Appendix A.

2.11.2 Crowding Distance

After sorting the solutions into fronts, the next step is to ensure diversity within each front through the crowding distance calculation. Crowding distance measures how dense the solutions are in the objective space around a particular solution, this is achieved through equation 13.

$$d_i = \sum_{j=1}^{M} \frac{f_j(i+1) - f_j(i-1)}{f_j^{max} - f_j^{min}} \tag{13}$$

Here M represents the number of objectives, $f_j(i+1)$ and $f_j(i-1)$ are the objective values of the neighboring solutions for objective j, f_j^{max} and f_j^{min} are the maximum and minimum values for objective j among all solutions in that front. Boundary solutions with no neighbors on one side are assigned a large crowding distance to ensure their selection during the selection procedure [42]. A visual representation of this is shown in Figure 16.

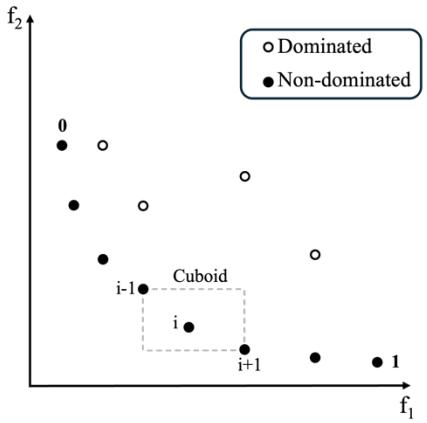


Figure 16: Example of a Crowding-distance calculation

2.11.3 Selection

After calculating the crowding distances, the breeding pool for the next generation is structured to enhance both quality and diversity within the population using the following strategy. The process begins by incorporating all members from the first front F1, which contains the optimal non-dominated solutions for the current population [42]. If the number of selected individuals from the first front is less than the desired population size, the algorithm proceeds to include members from the subsequent front. This process continues, incorporating individuals from progressively more dominated fronts until the accumulated selection pool size reaches or closely approximates the desired population size. The addition of the last front F3, will exceed the population limit, which means that selection will become selective within that front. Specifically, individuals with larger crowding distances are prioritized. This method ensures that the chosen individuals are well-distributed across the objective space. This selective procedure favors individuals who are spatially isolated from their peers, thereby maintaining a high level of diversity within the population [42]. This selection criteria ensures that the evolutionary process effectively explores the search space, which is crucial for uncovering optimal solutions across the multi-objective landscape. This methodology is made visual in Figure 17 [42].

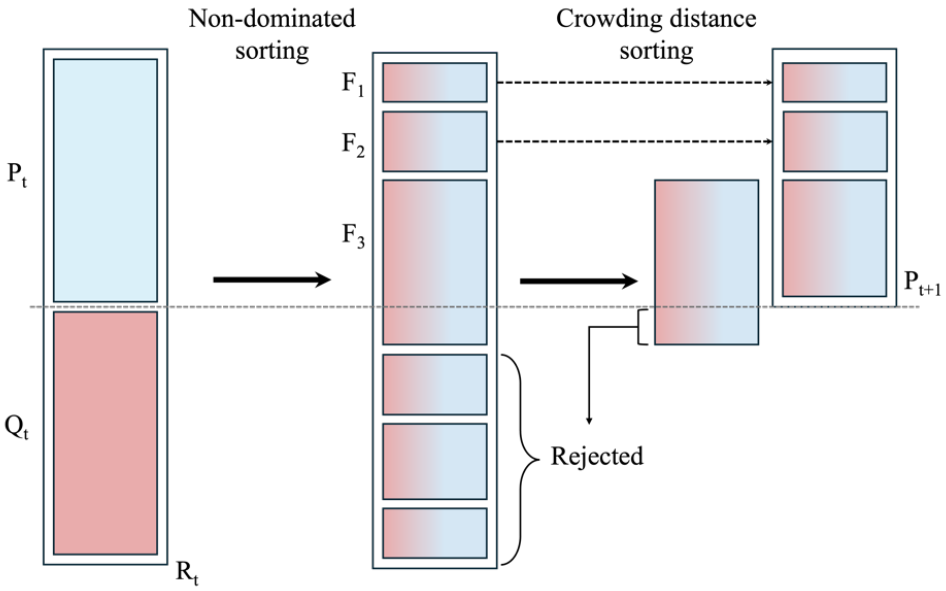


Figure 17: Schematic of the selection process in NSGA-II

2.12 SLSQP

Sequential Least Squares Programming (SLSQP) is an iterative method for solving non-linear optimization problems. Its objective is to find the optimal values for specific variables that minimize or maximize a target function while satisfying certain constraints [43]. The method approximates the original, complex problem with a simpler quadratic problem at each step.

Starting from an initial point, either determined or guessed, SLSQP creates a local model of the objective function using a second-order approximation, being easier to calculate and optimize than the original function. Additionally, it linearizes the constraints around the current point, transforming originally non-linear constraints into linear ones locally. These approximations lead to the formation of a quadratic programming subproblem, which is simpler to solve. This subproblem identifies the direction and magnitude for shifting the current real solution [44].

After determining the direction, SLSQP performs a line search to find the appropriate distance to move along that path and carry out the repositioning. After updating the current point, the algorithm also updates its Hessian approximation by using information gathered from the differences in gradients before and after the update. This adjustment ensures that the method gradually develops a better understanding of the underlying shape of the objective function and constraints. The process is repeated in each iteration, the improved Hessian approximation forms a new quadratic subproblem, which is then solved again. The current solution is adjusted until the conditions are met [45].

3 ALGORITHM DEVELOPMENT

3.1 Single Reaction Genetic Algorithm

The first algorithm developed in this work is a GA for single reactions (SRGA). This algorithm aims to optimize reactor setup configuration for reaction mechanisms where one reaction pathway is dominant, and the final reaction product is desired. A simple example of this is shown in the following reaction.

$$A + B \longrightarrow C$$

The aim of the GA is to minimize the number of reactors used in the setup and reduce the overall residence time required to achieve the desired conversion. A shorter residence time enables the use of smaller, more cost-effective reactors while reducing the number of reactors helps lower overall investment costs. SRGA follows the framework illustrated in Figure 18.

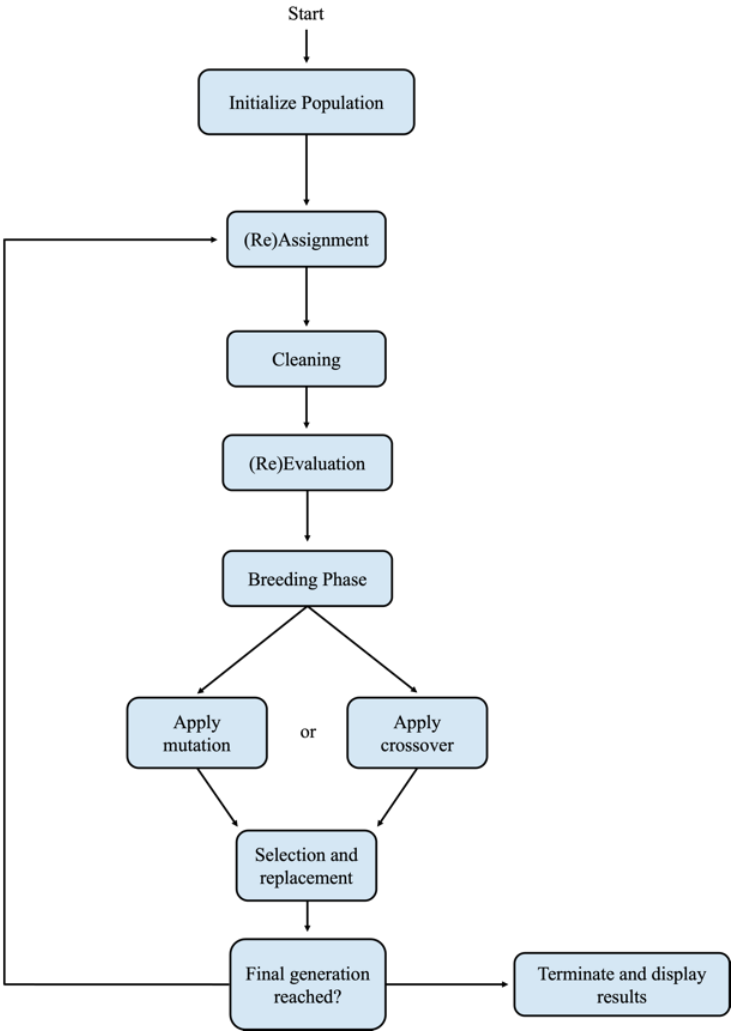


Figure 18: Schematic overview of SRGA's Framework

3.1.1 Initialization

The first step in developing SRGA is to define the shape and structure of each individual setup. In GAs, each individual is typically represented as a string of genes. For optimizing reactor configurations in this case, the genes will correspond to one of two types of reactors, namely CSTRs or PFRs, defined as strings that are recognizable by the other operations. Figure 19 shows an example gene sequence for a setup.

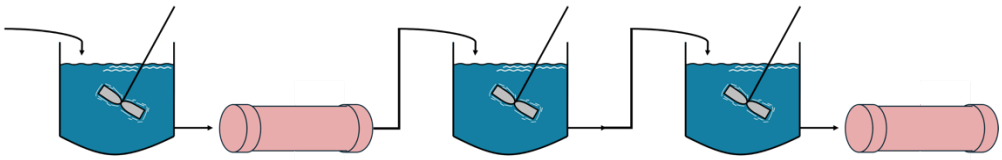


Figure 19: Example gene sequence

Each setup is also given the ability to contain its fitness values and corresponding fitness weights. The fitness value indicates a setup performance for the given problem and is assigned during evaluation. In case of SRGA, these are the total residence time and the reactor amount used. The fitness weights are used during selection to determine whether minimization or maximization occurs. By default, the GA is designed to maximize, therefore, multiplying the fitness by a negative number allows for minimization. Once this is done, a population of setups is initialized. The weight values used for SRGA are set to -0.5 since both fitness objectives complement each other when minimized.

3.1.2 Assignment

Once the population is initialized, each setup requires its reactors to be assigned a value for their conversion. DE is used here to determine the optimal conversions for the specified number of reactors, ensuring that each reactor is assigned the most efficient conversion and operates at maximum efficiency. This approach introduces a bi-layered optimization process, where DE fine-tunes reactor performance within each genetic algorithm iteration.

3.1.3 Cleaning

Before evaluating the setup, certain cleaning steps are performed. This is essential to avoid unnecessary computational tasks when evaluating the fitness of the setup. These actions are based on general principles of chemical reactor engineering [3, 4]. The cleaning process begins because a series of smaller PFRs in sequence will function identically to a single larger PFR. In this step, the extra PFRs are removed until only one remains. This concept is shown in Algorithm 1 and the result of this operation is visualized in Figure 20.

```
for reactor i, in setup:
    if reactor i, is not last and is a PFR:
        if next reactor j, is a PFR:
            remove reactor i
        end
    end
end
```

Algorithm 1: First step in the cleaning process

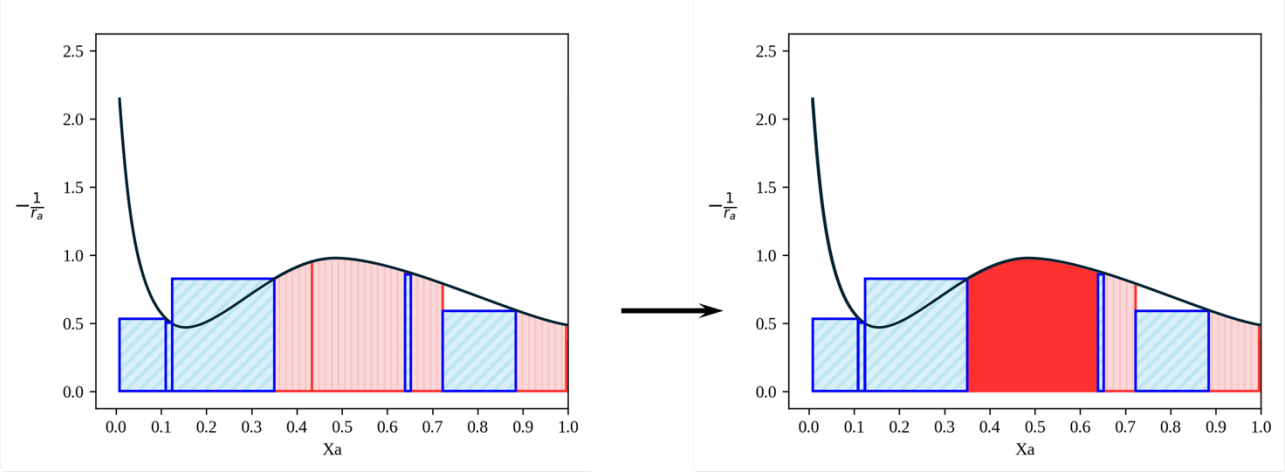


Figure 20: Removal operation of redundant serial PFRs

Once the multiple PFRs are reduced to one, the next step is identifying CSTRs that behave like a PFR. If the conversion range that a CSTR covers becomes increasingly narrow, it will approach the behavior of a similarly sized PFR. Having these types of small CSTRs in a system is unproductive since it does not differ from a PFR in its functioning. Therefore, the algorithm removes them from the setup and extends the conversion capabilities of the previous reactor to compensate for the one being removed. This is done according to Algorithm 2. The conversion threshold for SRGA is set at 0.03. Such a threshold was empirically determined, such that it does not negatively affect the search in the SRGA. Figure 21 shows the results of this operation. After filtering out the CSTRs that mimic PFRs, an additional check for consecutive PFRs is performed. The removal of certain CSTRs can result in previously separated PFRs being adjacent to each other, making this step necessary.

```
for reactor i, in setup:
    if reactor i, is a CSTR:
        if conversion difference between current and next reactor j, is smaller than 0.03:
            remove reactor i
        end
    end
end
```

Algorithm 2: Second step in the cleaning process

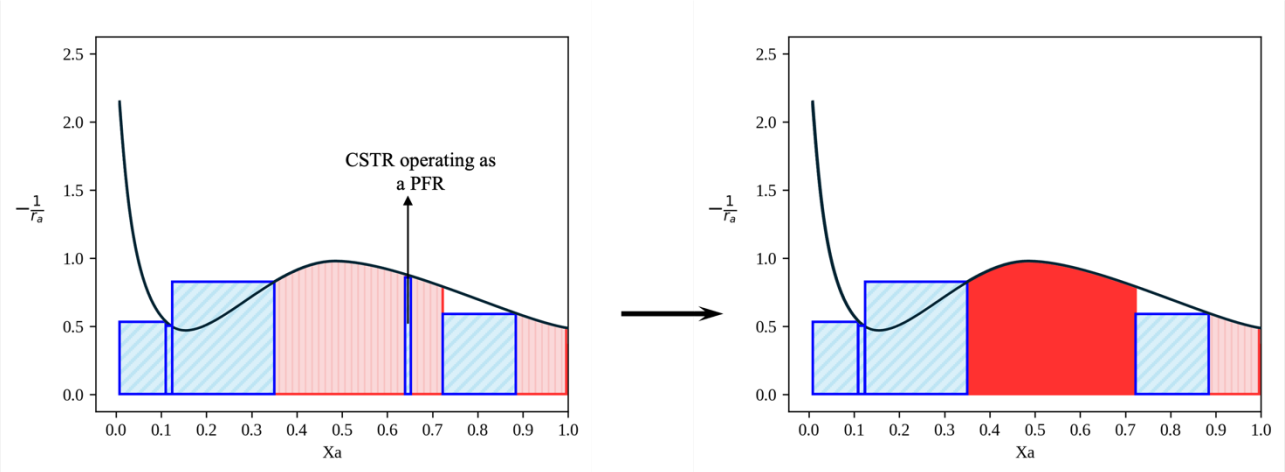


Figure 21: Recognition and removal of CSTR operating as a PFR

After filtering out the CSTRs that mimic PFRs, an additional check for consecutive PFRs is performed. The removal of certain CSTRs can result in previously separated PFRs being adjacent to each other, making this step necessary. Finally, a check for ultra-thin reactors is performed. During the reactor conversion assignment, two reactors may have conversion values that are close to each other, leading to the formation of one ultra-thin reactor. These ultra-thin reactors are generally not ideal for optimization and are, therefore, filtered out. The conversion threshold for this operation is set at 0.002, as the ultra-thin reactors were typically smaller than this value. This final procedure is carried out according to Algorithm 3. The results of this operation are shown in Figure 22.

```
for reactor i, in setup:
    if conversion difference between current and next reactor j, is smaller than 0.002:
        remove reactor i
    end
end
```

Algorithm 3: Final step in the cleaning process

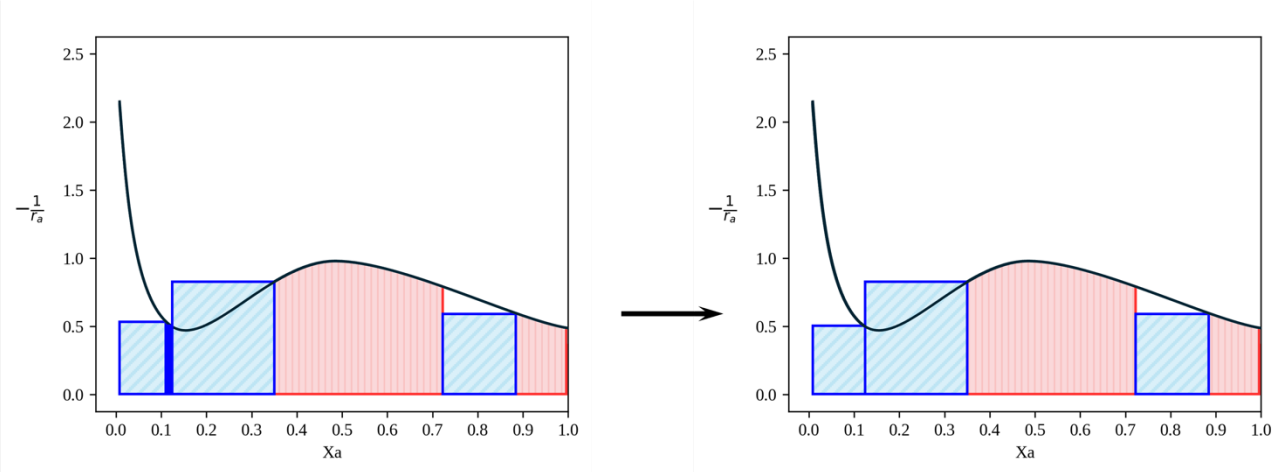


Figure 22: Removal of ultra-thin reactors

3.1.4 Evaluation

Once the setups are fully defined and cleaned, the next step is to define an evaluation function. This function will assess each setup's fitness by counting the number of reactors it includes and calculating the total residence time. Since a setup may contain different types of reactors, the evaluation function must account for these variations. It identifies each reactor type within the setup and applies the appropriate residence time calculations accordingly, as described in sections 2.1 and 2.2 of the literature study.

3.1.5 Breeding

During the breeding phase of SRGA, the parent population undergoes various genetic operations, like crossover and mutation to create the offspring population. Each of these operations is selected based on predefined probabilities. This approach maintains genetic diversity while improving solution space exploration.

3.1.5.1 Probabilities

The parent population generates a predetermined number of offspring. The operator used to create each offspring, whether it be mutation or crossover, is determined by user-defined probabilities: the crossover probability (CXPB) and the mutation probability (MUTP). The probabilities must sum to one, exceeding this causes the GA to malfunction. This process is outlined in Algorithm 4.

```
for parent in population:
    calculate chance between 0 and 1
    if chance is smaller than CXPB:
        apply Crossover
    else:
        apply Mutation
    end
end
```

Algorithm 4: Operation selection for the offspring

Here a random number generator produces a number between 0 and 1 that is uniformly distributed for each potential offspring. Depending on where this number falls, different genetic operations are executed. For SRGA to operate effectively.

3.1.5.2 Crossover

During this operation, two individual setups are selected from the population, and a crossover point is randomly chosen for each setup. The genetic material of these individuals is then combined to produce new offspring, potentially resulting in offspring that have better fitness than either parent alone. This step also facilitates the increase or decrease of reactors within a setup, making it crucial for exploring the solution space. An example of this operation is shown in Figure 23.

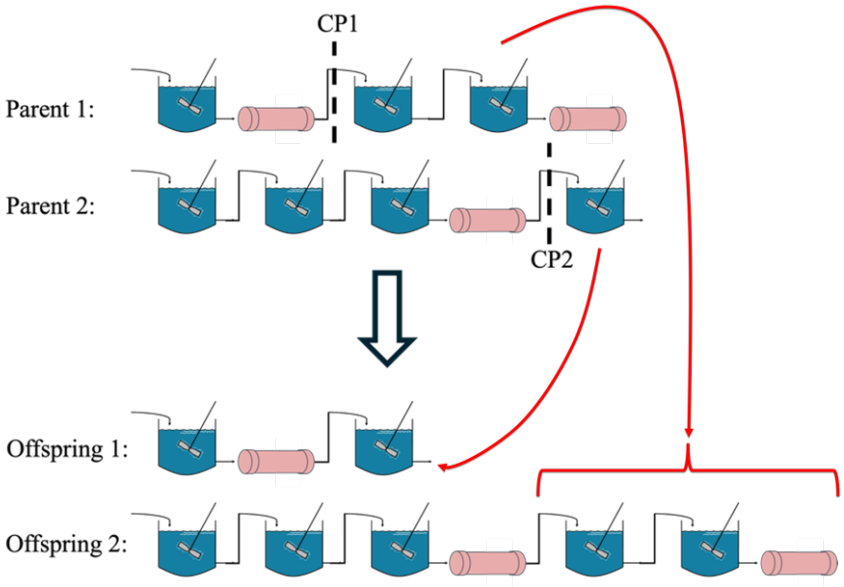


Figure 23: Visualisation of the crossover procedure

3.1.5.3 Mutation

The mutation operation in SRGA consists of two main actions, each determined by a specific probability: MUTPFLIP and MUTADD. MUTPFLIP involves switching one or more reactors within the existing sequence. If a reactor in the current sequence is a CSTR, there is a chance it will be switched to a PFR, and vice versa. MUTADD is based on human genome insertion mutation by adding a new reactor of random type at the end of the sequence [46]. This operation allows SRGA to explore new operational setup configurations and potentially more effective sequences. The implementation is done according to Algorithm 5.

```
for reactor in setup:
    calculate chance between 0 and 1
    if chance is smaller than MUTFLIP:
        if reactor type is "CSTR":
            set reactor type to "PFR"
        else:
            set reactor type to "CSTR"
        end
    end
end

calculate chance 2
if chance 2 is smaller than MUTADD:
    calculate chance 3
    if chance 3 is smaller than 0.5:
        add "PFR" to end of setup
    else:
        add "CSTR" to end of setup
    end
end
```

Algorithm 5: Mutation operation for SRGA

Both mutation strategies are important for diversifying the solution space and enhancing SRGAs evolving ability. The hypothesis here is that altering existing reactors while adding new reactors could help explore untested configurations, thereby increasing the chances of finding an optimal solution that could be missed by simple adjustments alone. Figure 24 illustrates this operation.

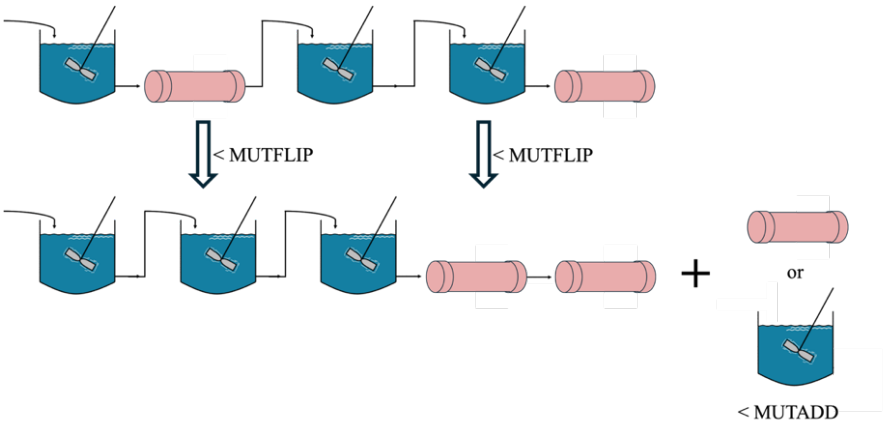


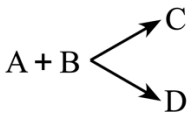
Figure 24: Visualization of the mutation procedure

3.1.6 Selection

After the breeding phase, the newly generated offspring must be re-evaluated for their fitness, as the original fitness values do not correspond to the new configurations of the offspring. Once the re-evaluation is complete, the selection process begins. In the case of the SRGA, the selection follows the NSGA-II procedure outlined in previous section 2.11. This selection occurs from a combined pool of both the parent and offspring populations.

3.2 Parallel Reaction Genetic Algorithm

The second algorithm developed in this work is a GA for parallel reactions (PRGA). This algorithm is designed to optimize reactor setups for parallel reaction systems where only one of the products is desired. The following reaction is a simple example of this.



PRGA aims to minimize the number of reactors in the setup while maximizing the yield of the desired product. Achieving higher product yields leads to more cost-effective reactors over time, while reducing the number of reactors lowers overall investment costs. Although PRGA shares some similarities with SRGA, key differences exist between the two, specifically, within the initialization, assignment, and breeding steps.

However, two major distinctions between PRGA and SRGA are that PRGA monitors the output concentration of each reactor, whereas SRGA works with a conversion ratio for its fitness evaluation. This approach is essential for creating fraction plots and for calculating product formation, as explained in section 2.5. Additionally, PRGA includes an extra refinement layer that optimizes reactor concentrations according to the SLSQP method.

In contrast, the cleaning, evaluation, and selection steps in PRGA are similar to those in SRGA, with a few minor distinctions. During the cleaning step, PRGA emphasizes the concentration of reactants, whereas SRGA focuses on the conversion. For the evaluation step, PRGA assesses the total formation of the desired product instead of calculating residence time, as SRGA does. The selection step remains unchanged in PRGA.

3.2.1 Initialization and Assignment

Each reactor is assigned an initial output concentration during initialization. The output concentration is considered a property of the reactor that evolves throughout the generations, rather than a simple assignment that is performed at each generation. The initial assignment for each reactor is done according to equation 14.

$$C_i = C_0 - \frac{D \cdot i}{T} \tag{14}$$

Where $\boldsymbol{C_i}$ is the output concentration of reactor $\boldsymbol{i}$, $\boldsymbol{C_0}$ is the initial concentration, $\boldsymbol{D}$ is the difference between the initial and desired final concentration, and $\boldsymbol{T}$ *is* the number of reactors in the setup.

3.2.2 Breeding

Unlike SRGA, reactor concentrations in PRGA are not predetermined at the beginning of each generation. As a result, breeding operations must account for the evolution of these reactor concentrations, in addition to factors already accounted for in SRGA such as the reactor and type. The operations are still based on the same probabilistic system employed in SRGA.

3.2.2.1 Crossover

The crossover procedure is mostly identical to the one used in SRGA. The key difference for PRGA is that the reactor output concentrations are internal properties of each reactor. Therefore, setups generated post-crossover may sometimes be unfeasible. The concentration at the end of each reactor should always be greater than or equal to that of the previous one. This issue is illustrated in Figure 25.

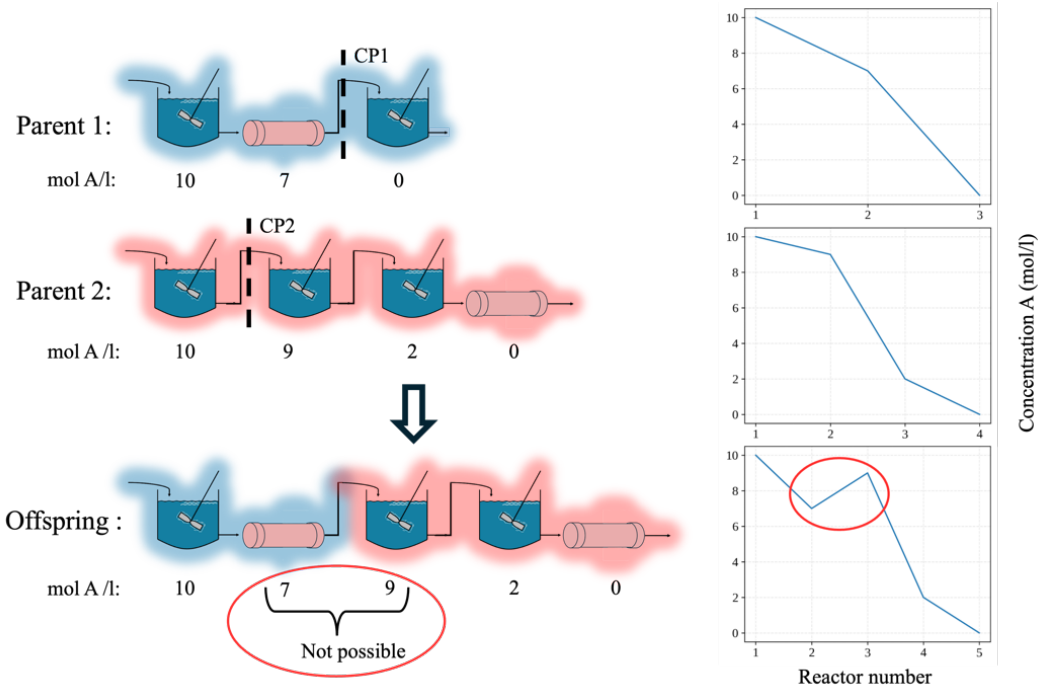


Figure 25: Illustration on impossible setup after crossover

The concentration profile observed is physically impossible. The concentration of the reactant cannot increase without altering the reaction balance or introducing an additional reactant input into a reactors. Neither of these scenarios applies in this thesis. To address this issue, a new term called "Energy" is introduced. In this context, the Energy of a reactor is defined as the absolute value of the difference between its output concentration and its input concentration. Algorithm 6 uses this term to normalize the concentrations of each reactor. This evaluates the feasibility of concentration profiles in reactor setups by comparing the total Energy increments to the concentration differences. It identifies infeasible setups and calculates the normalized output concentrations for those scenarios. This method is essential for steering the process towards optimal reactor concentrations.

```
for setup in offspring:
    calculate absolute Increments between each reactor
    Energy is sum of Increments
    Diff is difference between starting and target concentration
    if Energy is larger than Diff:
        add setup to impossible list
    end
end

for setup in impossible list:
    calculate absolute Increments between each reactor
    Energy is sum of Increments
    Fractions is division of Increments by Energy
    for reactor in setup:
        calculate cumulative Fraction
        reactant converted is multiplication between Diff and cumulative Fraction
        output concentration is subtraction of reactant converted from starting concentration
        assign output concentration
    end
end
```

Algorithm 6: Concentration normalization after crossover in PRGA

3.2.2.2 Mutation

In PRGA, mutation operates similarly to the process in SRGA. The first step, which involves switching reactors, is carried out in the same manner as in SRGA. However, the second step of adding reactors to the setup differs for PRGA. Instead of adding one single random reactor at the end of a setup, the mutation process in PRGA inserts a sequence of alternating reactors at a random position within the setup. The amount of added reactors is determined by a user-defined value called Dosage, an absolute value. Each reactor is assigned an output concentration alternating between the initial concentration and half of the initial concentration. After this, the concentrations are normalized according to previous Algorithm 6. This approach aims to enhance the likelihood of moving away from local minima and promoting the exploration of different setup configurations within the search space. It is executed according to Algorithm 7.

```
generate random number as chance 1
if chance 1 is smaller than MUTADD:
    choose random insertion position:
    generate random number as chance 2
    if chance 2 is smaller than 0.5:
        for number i in Dosage until i equals Dosage:
            if i is an uneven number:
                insert PFR with initial concentration
            else:
                insert CSTR with initial concentration divided by 2
            end
        end
    else:
        for number i in Dosage until i equals Dosage:
            if i is an uneven number:
                insert CSTR with initial concentration
            else:
                insert PFR with initial concentration divided by 2
            end
        end
    end
end
```

Algorithm 7: Insertion process for mutation in PRGA

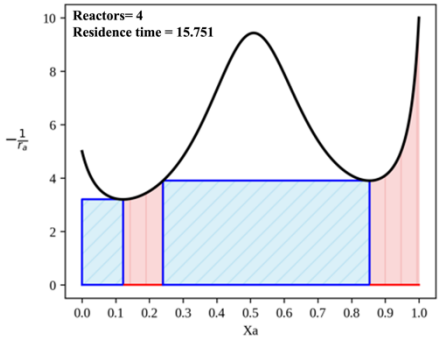
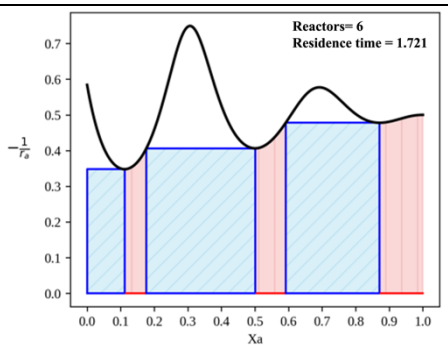
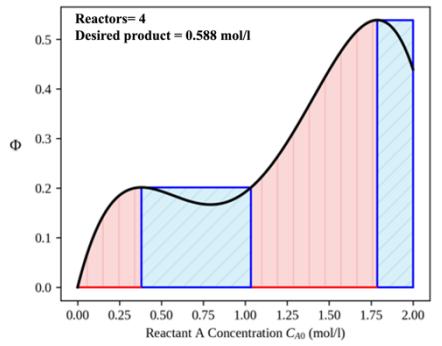
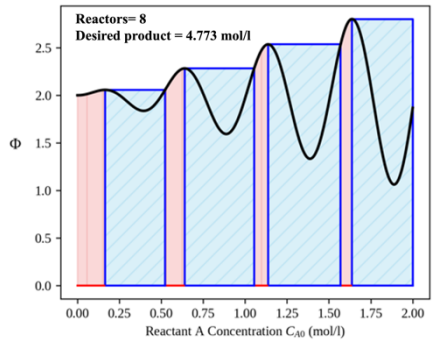
4 MATERIALS AND CASES

For the development of the GAs, Python 3.10 was utilized, employing DEAP 1.4 and SciPy 1.13.1 libraries. The use of Google Colab provided a robust platform for executing the Python code, ensuring efficient implementation and testing of the GAs [46, 47]. To assist in the grammar and spelling of this thesis, the following tools were utilized: Grammarly and ChatGPT [52, 53].

This thesis employed mathematical models of hypothetical chemical reactions inspired by the literature for validation and optimization purposes [49, 50]. The cases were first solved manually, to identify the optimal setup. Following, the GA's developed in section 3 were used to generate the optimal reactor configuration. A summary of the employed cases is reported in Table 1. In the graphs, the areas marked in red and blue represent PFRs and CSTRs, respectively.

Cases 1 and 2 were used for the optimization of SRGA. These cases represent hypothetical complex chemical reaction systems. The first case is a modification based on autocatalytic reactions [3]. The hypothesis here is that one of the reactants undergoes a reversible side reaction with itself to form a complex. As a result, the reaction rate is limited once the available reactant is used. The reactant is then recovered from the complex. The second case is a hypothetical representation of an oscillating biochemical reaction system, where one of the reactants undergoes a series of reactions in a closed circuit. Making its availability to react in the primary reaction periodical. For PRGA, cases 3 and 4 will be utilized, which are also hypothetical reaction schemes.

Table 1: Summary of the Cases used for the optimization

Case nr, mathematical equation and hypothetical reaction	Optimal Levenspiel or Fraction plot
Case 1: **equation:** $-ra = -9.9x^4 + 19.6x^3 - 11.9x^2 + 2.1x + 0.2$ **reaction:** $A + B \longrightarrow 2B + R$ $3B \rightleftharpoons [B_3]$	
Case 2: **equation:** $-ra = (1-x)^{1.1} \cdot \sin\big((1-x)\cdot 16\big) + 2$ **reaction:** $A + B \longrightarrow R$ $B \longrightarrow X$ $Z \longleftarrow Y$ $B \uparrow \quad X \downarrow$	
Case 3: **equation:** $\Phi = -9.97(1-x)^4 + 19.68(1-x)^3 - 11.96(1-x)^2 + 2.69(1-x)$ $x = \frac{C_{a0} - C_a}{C_{a0}}$ **reaction:** $A \rightleftharpoons C$ $A \longrightarrow R$ $A + B \rightleftharpoons 2B$	
Case 4: **equation:** $\Phi = (1-x)^{1.1} \cdot \sin\big((1-x)\cdot 25\big) + 2$ $x = \frac{C_{a0} - C_a}{C_{a0}}$ **reaction:** $A + B \longrightarrow 3R$ $B \longrightarrow X$ $Z \longleftarrow Y$ $B \uparrow \quad X \downarrow$ & $A + X \longrightarrow F$ $A + Y \longrightarrow G$ $A + Z \longrightarrow H$	

5 OPTIMIZATION

To assess the performance of the GAs during optimization, a metric referred to as the Total Accuracy is utilized. This metric is calculated as the average of two accuracy measurements: one evaluating the number of reactors involved and the other assessing the residence time or the formation of the desired product, depending on the specific GA being tested. Including both measurements in the Total Accuracy provides a more comprehensive evaluation of the GAs performance. The results would suggest negligible differences if only one accuracy metric were used. However, by incorporating reactor accuracy, the metric effectively scales the differences between the two measurements, offering a more balanced and meaningful performance comparison. The Total Accuracy is calculated according to equation 15.

$$Total\ Accuracy\ = \frac{\frac{r_c - |r_c - r_t|}{r_c} + \frac{p_c - |p_c - p_t|}{p_c}}{2} \cdot 100\% \tag{15}$$

Here, r_c and r_t are the correct and resulting reactor amounts from the test. For SRGA, p_c is the correct total residence time and p_t is the resulting total residence time from the test. On the other hand, for PRGA, p_c is the correct desired product amount, and p_t is the resulting desired product amount.

During the development of the GAs, it was observed that having more reactors in the initial setup than is required for the ideal configuration beneficial is for the performance of the GA. The reason for this lies in how the cleaning step operates in both algorithms, as it breaks down the initial configuration according to the predefined methodologies outlined in section 3.1.3. This results in an increased likelihood of quickly identifying the ideal setup. However, to optimize other parameters, the initial number of reactors was set to two to optimize other parameters as a worst-case scenario.

The following parameters are optimized in both cases: population size, breeding balance between mutation and crossover, and internal mutation probabilities. This is done in a sequential manner, to prevent combinatorial explosion of the test size. These parameters were chosen as these are the main hyperparameters of the GAs

5.1 SRGA

5.1.1 Population

The parameters in Table 2 were used for all the population tests. To accurately assess the performance, each population value is tested 100 times, and the results are averaged. The first run for Case 1 revealed that a maximum Total Accuracy occurred at a population size of five, followed by a steady decrease, as shown in Figure 26a. A second run focused solely on the first three population values to verify the reproducibility of the initial results. This run confirmed that the maximum is reproducible. Finally, a third run with a finer population resolution was carried out to identify the exact population value at which this maximum occurs. It was found that the true maximum occurs at a population value of four. The tests were repeated for Case 2, as shown in Figure 26b, no maximum was observed. Instead, a stable plateau was reached around a population value of five.

Table 2: SRGAs parameters for tests on the population size

Parameter	**Value**
Starting pop reactor amount	2
MUTP	0.3
CXPB	0.7
MutFlip	0.3
MutAdd	1
Generation amount	50
Optimization weights, w_n and w_t	-0.5 & -0.5
Starting concentration C_{a0}	4
Targeted Conversion	0.999

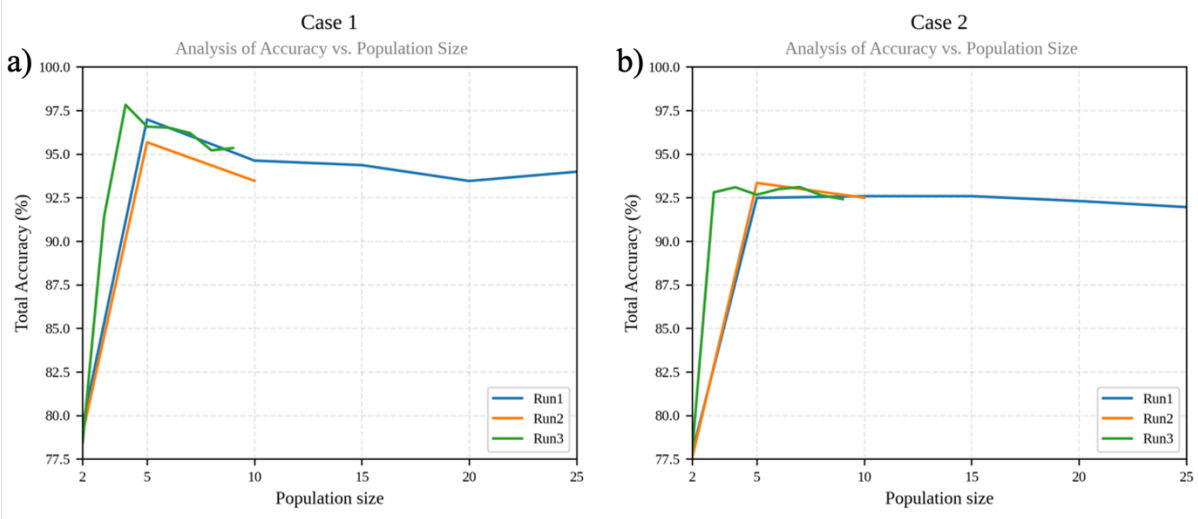


Figure 26: SRGAs population test results for Case 1(a) and 2(b), Total Accuracy vs Population size

The maximum observed in Case 1 is a non-common behavior in GA algorithms. Typically, an increase in population size would not result in a maximum followed by a decline in performance. However, the plateau-like progression seen in Case 2 does align with the expected outcome [51]. Indicating that this phenomenon occurs only with lower complexity problems. Attempts were made to determine the cause of this phenomenon. However, this was unsuccessful.

Beyond a certain threshold, increasing the population size yields diminishing returns in performance and keeps significantly increasing the required computational time, as shown in Figure 27. Based on the results, a population size of five was determined to be the most optimal for further optimization of the other parameters. This size was selected to ensure that the GA operates in a stable, plateau-like region while maintaining an acceptable computing time.

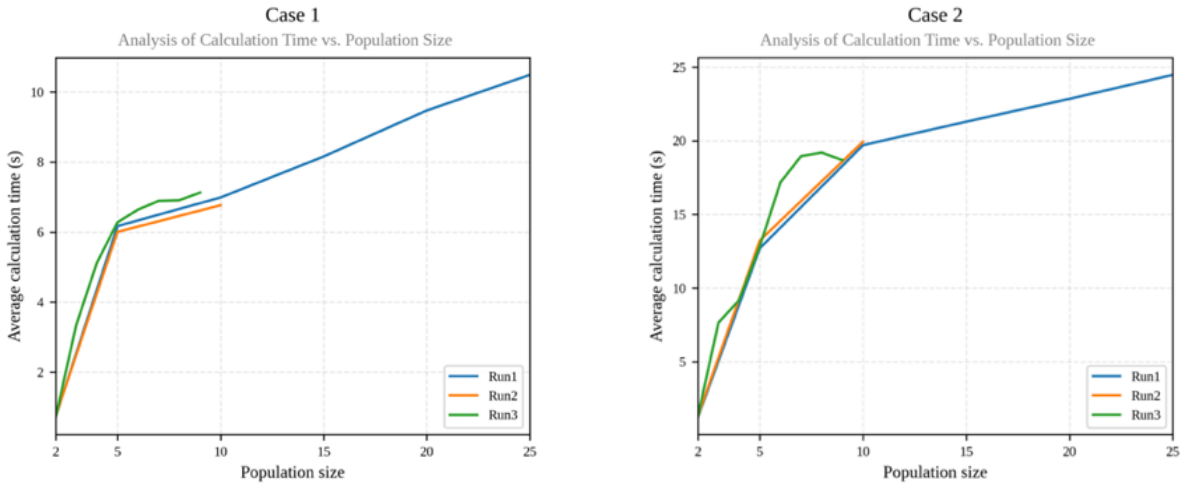


Figure 27: SRGAs population test results for both cases, Average calculation time vs Population size

Analyzing the optimization problem from a reactor engineering perspective, the overall performance of SRGA is compared across both cases presented in Figure 26. Case 1 reaches a Total Accuracy of 97.5%, while Case 2 stagnates at around 92.5%. SRGA is thus less successful for Case 2. This behavior can be explained by examining the Levenspiel plots of the reactions, shown in Figure 28. Case 2 exhibits more fluctuations in this plot, with minima closer to each other. Additionally, there is an overall steeper slope around the minima. Since the search resolution is the same for both cases, changes for a reactor will have a more significant impact on Case 2, leading to steeper local minima in the search space for the residence time. Thus lowering the chance of dislodgement, in the search for the true minimum.

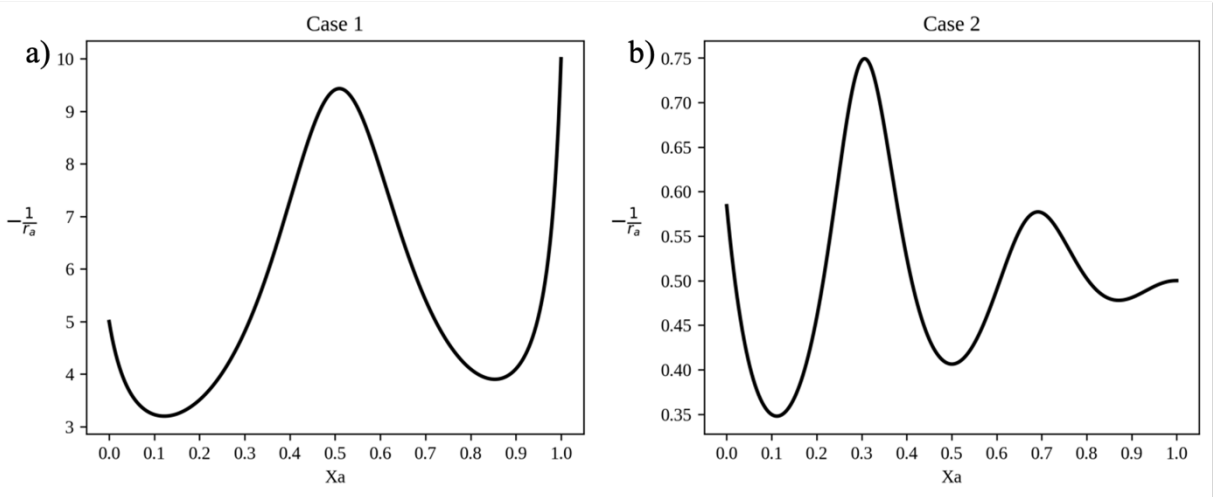


Figure 28: Levenspiel plot for Case 1(a) and Case 2(b)

5.1.2 Breeding balance

The parameters shown in Table 3 were used for all breeding balance tests. Each mutation and crossover combination was tested 100 times, and the results were averaged. For Case 1, a linear regression analysis of the Total Accuracy revealed that a lower mutation probability and a higher crossover probability are more advantageous for the performance of the SRGA. This becomes even more pronounced as the complexity of the problem presented to the GA increases, as shown for Case 2 in Figure 29.

Table 3: SRGAs parameters for tests on the breeding balance

Parameter	Value
Starting pop reactor amount	2
Population size	5
MutFlip	0.3
MutAdd	1
Generation amount	50
Optimization weights, w_n and w_t	-0.5 & -0.5
Starting concentration C_{a0}	4
Targeted Conversion	0.999

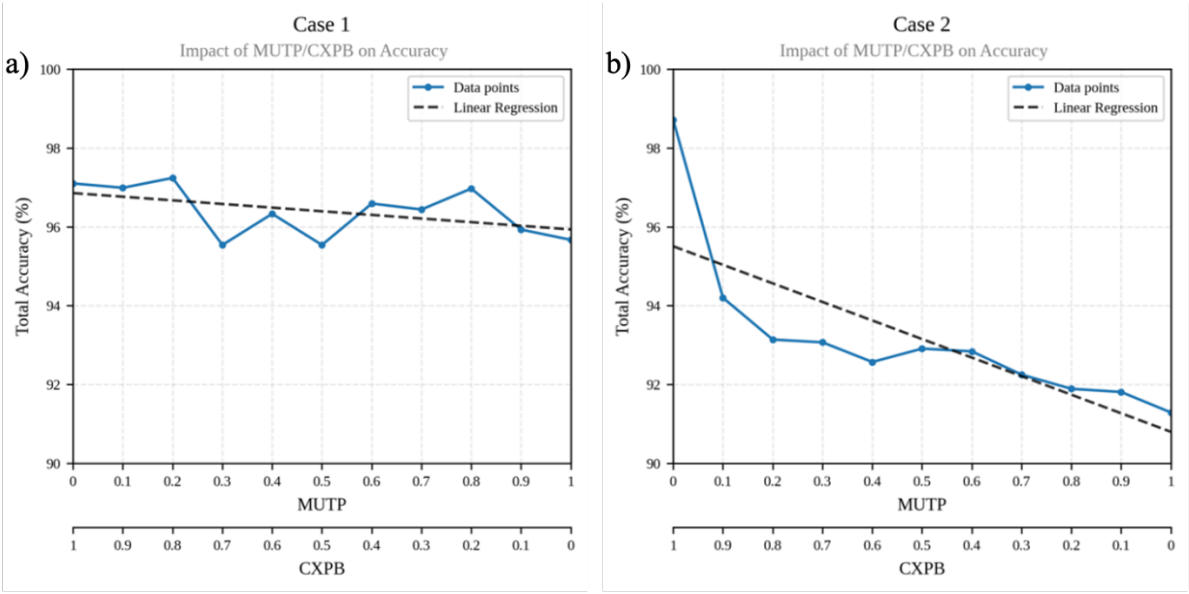


Figure 29: SRGAs breeding balance test results, Total Accuracy vs MUTP/CXPB for Case 1(a) and 2(b)

This behavior stems from the differences in how the two breeding operations function. The improved performance when the crossover is used more frequently can be attributed to its ability to incorporate multiple reactors into a system in one iteration. This expansion of the reactor chain length increases the likelihood of achieving the optimal configuration. In contrast, mutation adds only one reactor for each operation, resulting in a lower chain growth rate overall. This effect becomes more significant as the complexity of the problem increases and more reactors are needed to achieve the optimal solution.

The mutation and crossover probabilities will be set to MUTP=0.3 and CXPB=0.7, for the following test on the internal mutation probabilities. Although the results clearly indicate that the GA performs better when only crossover is used, especially in more complex scenarios. It may still be beneficial to explore the mutation operation in greater detail, as it could uncover configurations that improve the performance of the GA. This investigation is only possible if the mutation operation continues to be implemented.

5.1.3 Internal mutation probabilities

Since the internal mutation probabilities, MutFlip and MutAdd, are independent parameters, a LHS of the search space was performed. Each combination was tested 100 times. The parameters listed in Table 4 were used for all tests related to the internal mutation probabilities.

Table 4: SRGAs parameters for tests on the internal mutation probabilities

Parameter	Value
Starting pop reactor amount	2
Population size	5
MUTP	0.3
CXPB	0.7
Generation amount	50
Optimization weights, w_n and w_t	-0.5 & -0.5
Starting concentration C_{a0}	4
Targeted Conversion	0.999

The Latin hypercube scatterplot for Case 1 shows the Total Accuracy increasing with a higher MutFlip value. In contrast, MutAdd has a minimal impact on the Total Accuracy, as seen in Figure 30a. This becomes more evident as the problem's complexity increases, as seen in Case 2, illustrated in Figure 30b. For future use of SRGA, the Levenspiel plot should be evaluated in advance, and the MutAdd value should be set lower depending on the complexity.

A higher MutFlip value increases the number of reactors switched in a setup, effectively increasing the GAs ability to explore a wider range of potential solutions. The limited influence of MutAdd is likely due to it being overshadowed by the Crossover operation. Adding one reactor when the optimal solution requires many more is insufficient and thus shows little to no effect. To confirm these observations, another test on the breeding lance will be conducted with MutFlip= 0.9 and MutAdd= 0.0. The results will be compared with those from the previous tests.

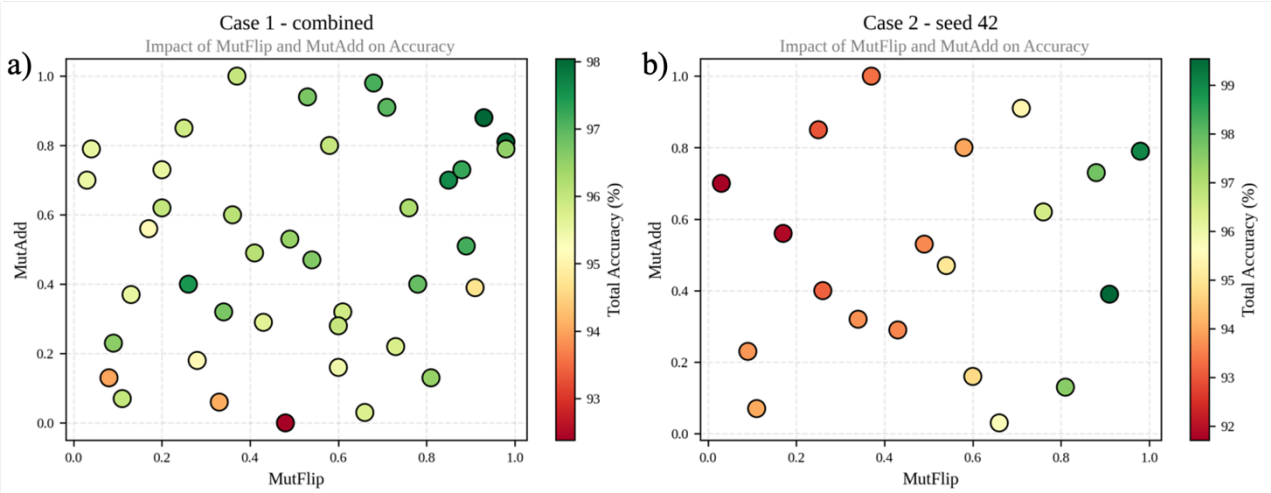


Figure 30: SRGAs internal mutation probabilities test results, MutAdd vs Mutflip for Case 1(a) and 2(b)

5.1.4 Breeding balance confirmation

The parameters shown in Table 5 were used for the second test on the breeding balance. Each combination is tested 100 times. The comparison between the first and second tests is shown in Figure 31. This result confirms that the effect of the mutation's addition during the breeding process is limited. It only becomes significant as the mutation probability approaches the upper end of its range. Another test with MutAdd set to 0 and MutFlip set to 0.3 confirmed the results from the previous test on internal mutational probabilities. For complex cases, a higher MutFlip is more beneficial.

Table 5: SRGAs parameters for tests on the breeding balance confirmation

Parameter	Value
Starting pop reactor amount	2
Population size	5
MutFlip	0.9
MutAdd	0
Generation amount	50
Optimization weights, w_n and w_t	-0.5 & -0.5
Starting concentration C_{a0}	4
Targeted Conversion	0.999

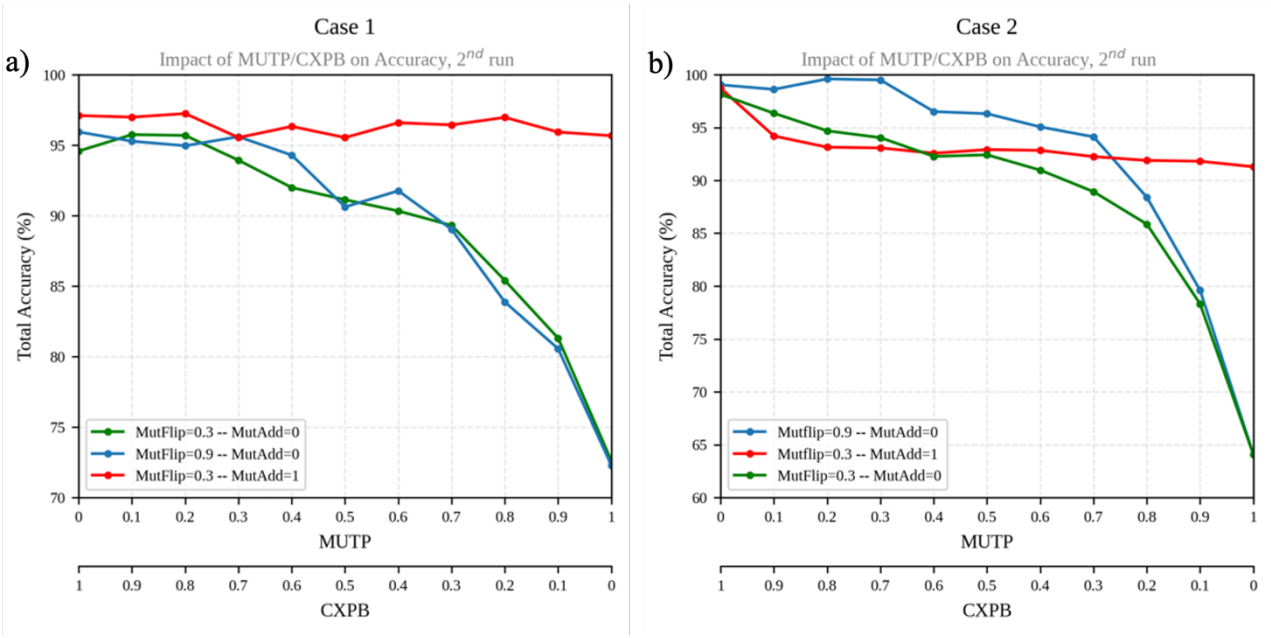


Figure 31: SRGAs breeding balance comparison, Total Accuracy vs MUTP/CXPB for Case 1(a) and 2(b)

When examining the optimization problem from a reactor engineering perspective, the Total Accuracy profile for the breeding balance of the SRGA can be compared in both cases, as illustrated in Figure 31. Case 2 shows a slight advantage due to the absence of additions caused by MutAdd. In contrast, Case 1 appears to experience a slight disadvantage. To further understand this, the reaction rate profiles shown in Figure 28 were analyzed again. In Case 2, as the reaction rate stabilizes towards the end, adding an additional reactor will not significantly affect the overall residence time. The change will result in only minor differences compared to the optimal state, which does not greatly impact the selection process of SRGA. In contrast, Case 1 differs due to the sudden deceleration of the reaction rate towards the end. Depending on the type of reactor used, there is a higher likelihood of selecting the correct final reactor.

5.2 PRGA

For PRGA, the generation limit has been set to 10 for all tests, which is significantly lower than the previous limit of 50 for SRGA. This adjustment was made because, during development, it was observed that higher generation limits resulted in only minimal differences in the outcomes. The Total Accuracy was consistently close to 100% for each parameter value. Thus, this reduced generation limit was implemented to better understand each tested parameter's impact.

5.2.1 Population

The parameters shown in Table 6 were used for the population tests. For PRGA, each population size was tested 1000 times, being significantly more compared to the 100 tests conducted for SRGA. This increase in testing was made possible by the improved computational efficiency of PRGA. Figure 32 compares the computational time per generation of SRGA and PRGA across different population sizes. The comparison clearly shows that PRGA is significantly more efficient. Even when dealing with more complex cases, PRGA outperforms SRGA, demonstrating faster performance than all cases used for SRGA, including the least complex case, Case 1. This efficiency is primarily due to PRGAs approach, which does not rely on DE to optimize reactor output concentrations at each generation. Instead, these concentrations are treated as an intrinsic property of the setup and are determined through the breeding operations, after which they are refined by SLSQP.

Table 6: PRGAs parameters for tests on the population size

Parameter	**Value**
Starting pop reactor amount	2
MUTP	0.3
CXPB	0.7
MutFlip	0.3
MutAdd	1
Generation limit	10
Optimization weights, w_n and w_p	-0.5 & 0.5
Starting concentration C_{a0}	4
Targeted Conversion	1
Power	C_{a0}
Dosage	2

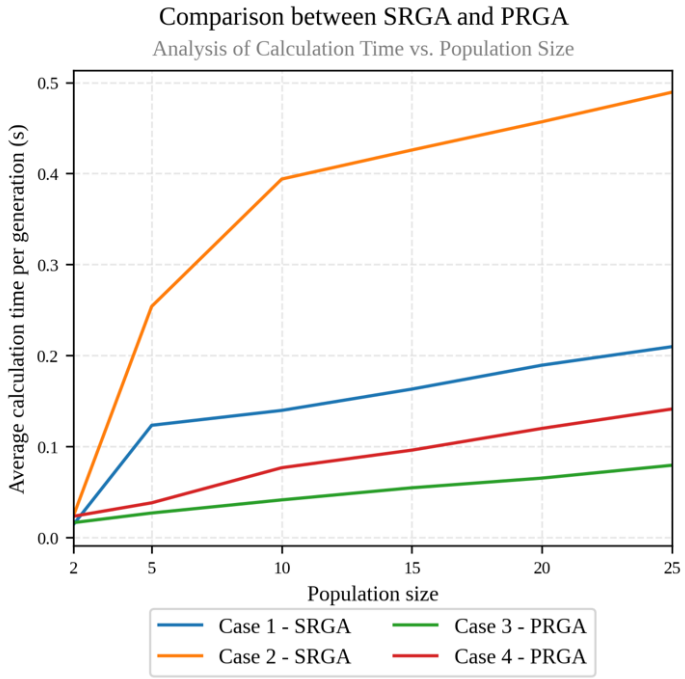


Figure 32: Comparison between SRGA and PRGA for computational efficiency

Figure 33 shows the effect of population size on Total Accuracy for Cases 3 and 4. Initially, Total Accuracy increases sharply and then stabilizes around 25. For Case 3, this configuration approaches 100%. However, this setup is insufficient for more complex problems, as observed in Case 4, which is expected due to the relatively small generation limit of 10. A population size of 5 was chosen for the following optimization tests, although a larger size would likely yield better results. This choice was made for the same reason as the reduced generation limit.

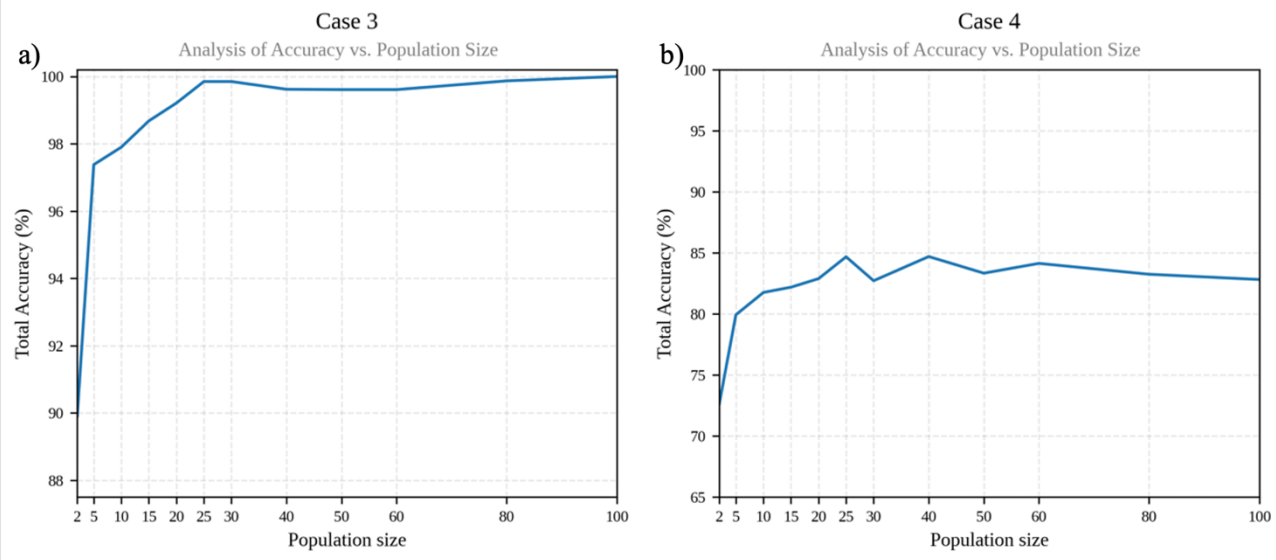


Figure 33: PRGAs population test results, Total Accuracy vs Population size for Case 1(a) and 2(b)

5.2.2 Breeding balance

The parameters in Table 7 were used for all breeding balance tests. Each mutation and crossover combination was tested 500 times. The results, shown in Figure 34, indicate that a higher MUTP positively impacts the performance of the GA. This trend holds true even in more complex scenarios, as seen in Case 4. The optimal range for MUTP seems to be from 0.6 to 0.9.

Table 7: PRGAs parameters for tests on the breeding balance

Parameter	Value
Starting pop reactor amount	2
Population size	5
MutFlip	0.3
MutAdd	1
Generation amount	10
Optimization weights, w_n and w_p	-0.5 & 0.5
Starting concentration C_{a0}	4
Targeted Conversion	1
Power	C_{a0}
Dosage	2

The mutation operation consists of two components: the random switching of reactors, determined by MutFlip, and the random insertion of reactors, determined by MutAdd. Either component can potentially increase the Total Accuracy. The switching component, promotes better exploration of the search space, thereby enhancing the chances of finding an optimal solution. While random insertion can speed up the process of reaching the optimal reactor amount. When comparing cases 3 and 4, it is notable that the increase in Total Accuracy for the initial MUTP values is greater in Case 3 than in Case 4. This can be seen on the left side of both plots in Figure 34. The difference is likely due to Case 3 requiring only four reactors for the optimal setup, allowing the GA to reach the optimal faster than Case 4, which requires eight reactors. Indicating that the latter component has a more pronounced impact. Further analysis of the internal mutation probabilities will provide additional insights into this behavior.

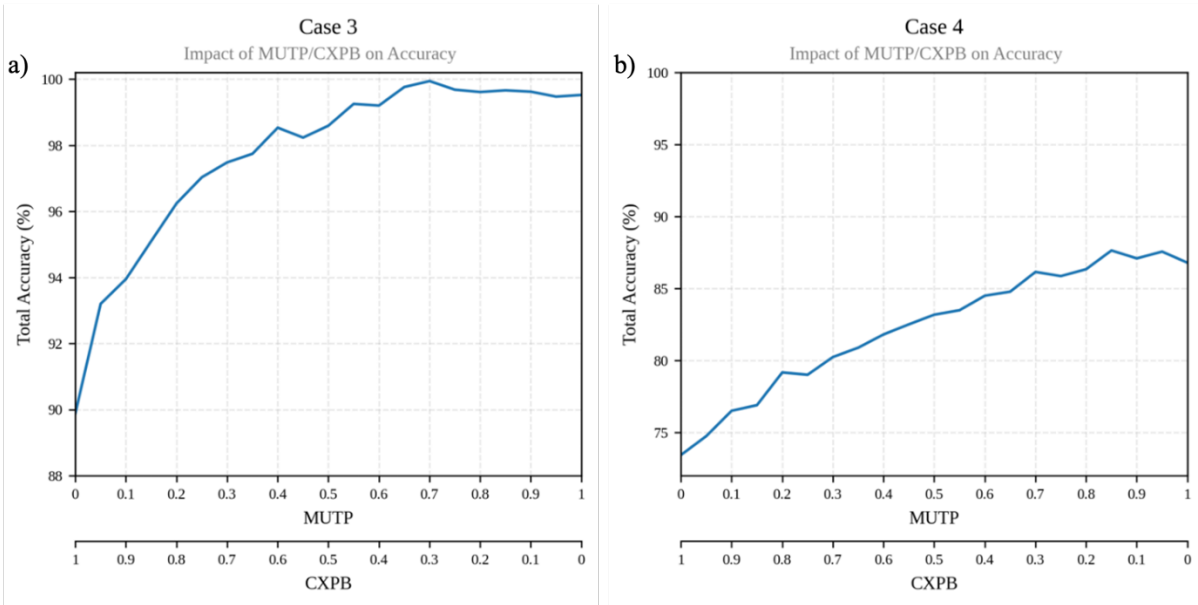


Figure 34: PRGAs breeding balance test results, Total Accuracy vs MUTP/CXPB for Case 1(a) and 2(b)

5.2.3 Internal mutation probabilities

To optimize the internal mutation probabilities of PRGA, a LHS analysis was performed, with each combination tested 500 times. Table 8 shows the parameter values used here, the results are shown in Figure 35. The Latin hypercube scatterplot for Cases 3 and 4 confirmed the previous hypothesis. The random insertion component of the mutation operation appears to have a more significant positive impact compared to the switching component. Interestingly, for the more complex problem presented in Case 4, the random switching component, influenced by MutFlip, seems to negatively impact the Total Accuracy as its probability increases. Thus, having a different behavior to that seen in SRGA.

Table 8: PRGAs parameters on tests for the internal mutation probabilities

Parameter	**Value**
Starting pop reactor amount	2
Population size	5
MUTP	0.75
CXPB	0.25
Generation amount	10
Optimization weights, w_n and w_p	-0.5 & 0.5
Starting concentration C_{a0}	4
Targeted Conversion	1
Power	Ca0
Dosage	2

The observed behavior is likely a result of the high mutation probability equal to 0.75, which increases the likelihood of the switching mechanism occurring. This effect is further amplified by its own probability MutFlip, increasing the switching frequency even more. As a result, there is an increased chance of forming configurations where two PFRs are arranged in series. Due to the cleaning phase, such setups are always reduced to a single PFR, effectively stagnating the optimization process. This issue is expected to primarily affect more complex problems where a higher number of reactors is needed. In simpler cases, fewer reactors are optimal and can be identified more quickly, reducing the impact of this effect.

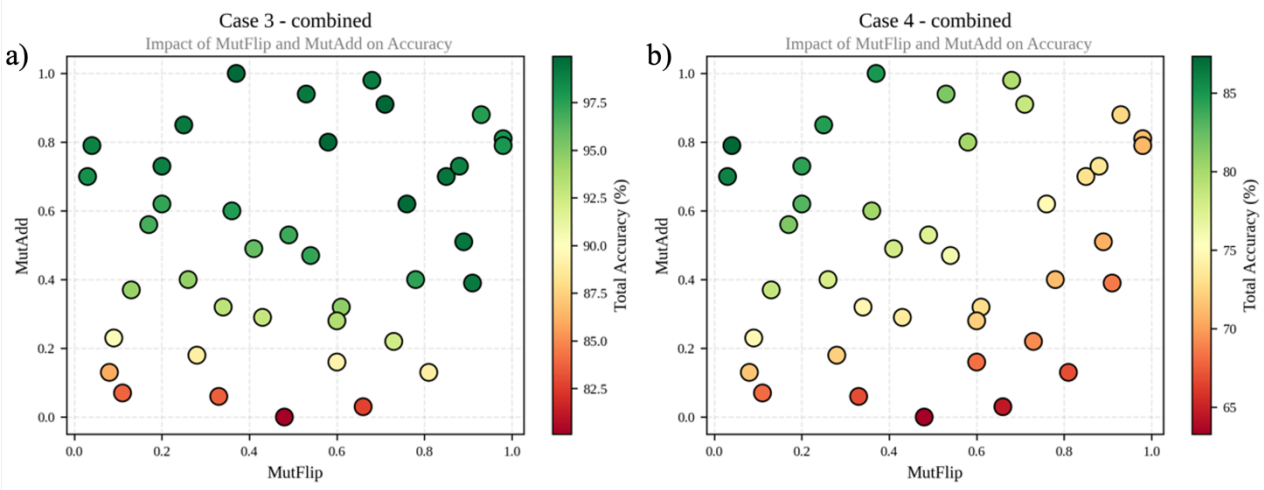


Figure 35: PRGAs internal mutation probabilities test results, MutAdd vs MutFlip for Case 1(a) and 2(b)

6 CONCLUSION

This thesis has explored the application of GAs for the optimization of reactor configurations in chemical engineering, with a particular emphasis on addressing complex reaction kinetics with ideal reactors. Successfully demonstrating the capability of GAs to identify optimal reactor setups that enhance efficiency by reducing residence time and maximizing desired product yields in complex reaction systems. By developing and testing two new tools, based on GAs, SRGA and PRGA, this study has introduced effective methodologies for optimizing both single-reaction and parallel-reaction scenarios. Thus having high potential in optimizing the design process, significantly accelerating workflow, and eliminating human bias in the design. The source code for both developed GAs is made available on the following official KU-Leuven GitLab repository: https://gitlab.kuleuven.be/cipt/master-theses/automatic-optimization-of-reaction-engineering-processes-iliyas-melnikov.git.

Key results demonstrated that SRGA is well-suited for optimizing setups in single-product reactions, effectively balancing the trade-offs between reactor count and operational efficiency. Similarly, PRGA proved effective for parallel reaction systems, emphasizing the maximization of specific product concentrations while accounting for competing pathways. Both algorithms were optimized using hypothetical cases, with the performance metrics' Total Accuracy and computational efficiency validating their robustness and potential for real-world application. PRGA however, demonstrated greater efficiency compared to SRGA, as it solved cases of similar complexity significantly faster. This advantage stems from PRGA not relying on optimization with DE, as seen in the SRGA framework. Instead, PRGA employs more customized genetic operations with refinement through SLSQP to achieve optimal intermediate concentrations as generations progress.

The study also highlighted the importance of parameter tuning, such as population size, breeding balance, and internal mutation probabilities in achieving optimal algorithm performance. Specifically, the findings indicated that a population size of five provided a balance between computational efficiency and solution quality. Mutation and crossover probabilities of 0.3 and 0.7, respectively, yielded the best performance for the SRGA. Additionally, higher MutFlip probabilities enhanced the exploration of solution spaces, whereas MutAdd probabilities had a less significant impact on SRGA.

For PRGA, which has different breeding operations, different results were observed concerning breeding balance and internal mutation probabilities. The optimal mutation and crossover probabilities were found to be 0.75 and 0.25, respectively. While a higher MutFlip negatively impacts performance in more complex scenarios, a higher MutAdd consistently yields positive results for PRGA.

This thesis contributes to the field of reaction engineering by providing a computational framework for optimizing reactor setups using GAs. The developed tools demonstrate significant potential for practical implementation and lay a foundation for future research to extend these methods to more complex, real-world scenarios. Possible future prospects include extending the application of these algorithms to non-ideal reactors, particularly for exothermic reactions where heat management plays a critical role. Another promising direction involves modeling real reactors based on residence time distributions derived from ideal reactor models, enabling more accurate performance predictions in industrial settings.

REFERENCES

[1] J. G. Segovia-Hernández, S. Hernández, E. Cossío-Vargas, and E. Sánchez-Ramírez, "Challenges and opportunities in process intensification to achieve the UN's 2030 agenda: Goals 6, 7, 9, 12 and 13," *Chemical Engineering and Processing - Process Intensification*, vol. 192, p. 109507, Oct. 2023, doi: 10.1016/j.cep.2023.109507.

[2] T. Rajabloo, W. De Ceuninck, L. Van Wortswinkel, M. Rezakazemi, and T. Aminabhavi, "Environmental management of industrial decarbonization with focus on chemical sectors: A review," *Journal of Environmental Management*, vol. 302, p. 114055, Jan. 2022, doi: 10.1016/j.jenvman.2021.114055.

[3] O. Levenspiel, *Chemical Reaction Engineering, 3rd Edition | Wiley*. 1998. [Online]. Available: https://www.wiley.com/en-us/Chemical+Reaction+Engineering%2C+3rd+Edition-p-9780471254249

[4] H. S. Fogler, *Elements of chemical reaction engineering, 5th ed*. Boston: Prentice Hall, 2016.

[5] C. He *et al.*, "A Review on Artificial Intelligence Enabled Design, Synthesis, and Process Optimization of Chemical Products for Industry 4.0," *Processes*, vol. 11, no. 2, Art. no. 2, Feb. 2023, doi: 10.3390/pr11020330.

[6] S. Dutta, *Optimization in Chemical Engineering*, 1st ed. Cambridge University Press, 2016. doi: 10.1017/CBO9781316134504.

[7] G. Joshi and M. B. Krishna, "Solving system of non-linear equations using Genetic Algorithm," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sep. 2014, pp. 1302–1308. doi: 10.1109/ICACCI.2014.6968423.

[8] D. E. Goldberg and J. H. Holland, "Genetic Algorithms and Machine Learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988, doi: 10.1023/A:1022602019183/METRICS.

[9] B. V. Babu and R. Angira, "Optimal design of an auto-thermal ammonia synthesis reactor," *Computers & Chemical Engineering*, vol. 29, no. 5, pp. 1041–1045, Apr. 2005, doi: 10.1016/j.compchemeng.2004.11.010.

[10] T. A.-N. Nguyen and T.-A. Nguyen, "Genetic Algorithms for Chemical Engineering Optimization Problems," in *Genetic Algorithms*, IntechOpen, 2022. doi: 10.5772/intechopen.104884.

[11] M.-V. Le, T.-A. Nguyen, and T.-A.-N. Nguyen, "Modeling and Optimization of the BSCF-Based Single-Chamber Solid Oxide Fuel Cell by Artificial Neural Network and Genetic Algorithm," *Journal of Chemistry*, vol. 2019, no. 1, p. 7828019, 2019, doi: 10.1155/2019/7828019.

[12] D. Mokeddem and A. Khellaf, "Optimal Solutions of Multiproduct Batch Chemical Process Using Multiobjective Genetic Algorithm with Expert Decision System," *Journal of Analytical Methods in Chemistry*, vol. 2009, no. 1, p. 927426, 2009, doi: 10.1155/2009/927426.

[13] M. C. A. F. Rezende, C. B. B. Costa, A. C. Costa, M. R. W. Maciel, and R. M. Filho, "Optimization of a large scale industrial reactor by genetic algorithms," *Chemical Engineering Science*, vol. 63, no. 2, pp. 330–341, Jan. 2008, doi: 10.1016/j.ces.2007.09.001.

[14] A. I. Hanopolskyi, V. A. Smaliak, A. I. Novichkov, S. N. Semenov, and C. Reviews, "Autocatalysis: Kinetics, Mechanisms and Design," vol. 2, pp. 2000026–2000027, 2020, doi: 10.1002/syst.202000026.

[15] N. Sbirrazzuoli, "Kinetic analysis of complex chemical reactions by coupling model-free and model-fitting analysis," *Thermochimica Acta*, vol. 719, p. 179416, Jan. 2023, doi: 10.1016/J.TCA.2022.179416.

[16] H. H. Kleizen, "Conceptual chemical process design in a sustainable technological world," *Journal of Cleaner Production*, vol. 14, no. 9–11, pp. 924–927, Jan. 2006, doi: 10.1016/J.JCLEPRO.2005.11.035.

[17] S. A. Vilekar, I. Fishtik, and R. Datta, "The steady-state kinetics of parallel reaction networks," *Chemical Engineering Science*, vol. 65, no. 10, pp. 2921–2933, May 2010, doi: 10.1016/J.CES.2010.01.022.

[18] T. Salmi *et al.*, "Application of semibatch technology on the investigation of homogeneously catalyzed consecutive and parallel-consecutive liquid-phase reactions: Kinetic measurements and modelling," *Chemical Engineering Science*, vol. 233, p. 116397, Apr. 2021, doi: 10.1016/J.CES.2020.116397.

[19] L. Meng *et al.*, "Exploring depolymerization mechanism and complex reaction networks of aromatic structures in chemical looping combustion via ReaxFF MD simulations," *Journal of the Energy Institute*, vol. 107, p. 101180, Apr. 2023, doi: 10.1016/J.JOEI.2023.101180.

[20] W. R. Penney, "Fast competitive/consecutive (C/C) chemical reactions," in *Computer-Aided Design of Fluid Mixing Equipment*, Elsevier, 2021, pp. 351–398. doi: 10.1016/B978-0-12-818975-7.00014-2.

[21] A. Bakhtiyorov, A. Elmanov, O. Maksudov, and A. Norkobilov, “Comparative Analysis of Esterification Reaction in Continuous Stirred Tank and Plug-Flow Reactors,” *Engineering Proceedings*, vol. 56, no. 1, Art. no. 1, 2023, doi: 10.3390/ASEC2023-15913.
[22] P. Jul-Rasmussen, X. Liang, X. Zhang, and J. K. Huusom, “Developing robust hybrid-models,” in *Computer Aided Chemical Engineering*, vol. 52, A. C. Kokossis, M. C. Georgiadis, and E. Pistikopoulos, Eds., in 33 European Symposium on Computer Aided Process Engineering, vol. 52. , Elsevier, 2023, pp. 361–366. doi: 10.1016/B978-0-443-15274-0.50058-5.
[23] A. V. Solokhin, S. L. Nazanskii, and T. V. Milyaeva, “Phase portrait of the dynamic system of parallel chemical reactions: An analysis of the effect of the rate constants,” *Theor Found Chem Eng*, vol. 44, no. 5, pp. 672–678, Oct. 2010, doi: 10.1134/S0040579510050064.
[24] I. W. M. Smith, “The temperature-dependence of elementary reaction rates: beyond Arrhenius,” *Chem. Soc. Rev.*, vol. 37, no. 4, pp. 812–826, 2008, doi: 10.1039/B704257B.
[25] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, “Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach,” *Information*, vol. 10, no. 12, Art. no. 12, Dec. 2019, doi: 10.3390/info10120390.
[26] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, “Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach,” *Information*, vol. 10, no. 12, Art. no. 12, Dec. 2019, doi: 10.3390/info10120390.
[27] J. Zhong, X. Hu, J. Zhang, and M. Gu, “Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms,” in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC’06)*, Nov. 2005, pp. 1115–1121. doi: 10.1109/CIMCA.2005.1631619.
[28] G. C. Pereira, M. M. F. de Oliveira, and N. F. F. Ebecken, “Genetic Optimization of Artificial Neural Networks to Forecast Virioplankton Abundance from Cytometric Data,” *JILSA*, vol. 05, no. 01, pp. 57–66, 2013, doi: 10.4236/jilsa.2013.51007.
[29] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. in Natural Computing Series. Berlin, Heidelberg: Springer, 2015. doi: 10.1007/978-3-662-44874-8.
[30] Y. Kaya, M. Uyar, and R. Tek\D{j}n, “A Novel Crossover Operator for Genetic Algorithms: Ring Crossover,” May 02, 2011, *arXiv*: arXiv:1105.0355. Accessed: Oct. 25, 2024. [Online]. Available: http://arxiv.org/abs/1105.0355
[31] I. Korejo, S. Yang, K. Brohi, and K. Z.U.A, “Multi-Population Methods with Adaptive Mutation for Multi-Modal Optimization Problems,” *IJSCAI*, vol. 2, no. 2, pp. 1–19, Apr. 2013, doi: 10.5121/ijscai.2013.2201.
[32] E. O. Alkafaween, “Novel Methods for Enhancing the Performance of Genetic Algorithms,” Jan. 25, 2018, *arXiv*: arXiv:1801.02827. doi: 10.48550/arXiv.1801.02827.
[33] A. Shukla, H. M. Pandey, and D. Mehrotra, “Comparative review of selection techniques in genetic algorithm,” in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Feb. 2015, pp. 515–519. doi: 10.1109/ABLAZE.2015.7154916.
[34] R. O.Oladele and J. S. Sadiku, “Genetic Algorithm Performance with Different Selection Methods in Solving Multi-Objective Network Design Problem,” *IJCA*, vol. 70, no. 12, pp. 5–9, May 2013, doi: 10.5120/12012-7848.
[35] M. Safe, J. Carballido, I. Ponzoni, and N. Brignole, “On Stopping Criteria for Genetic Algorithms,” in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, Eds., Berlin, Heidelberg: Springer, 2004, pp. 405–413. doi: 10.1007/978-3-540-28645-5_41.
[36] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997, doi: 10.1023/A:1008202821328.
[37] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution*. in Natural Computing Series. Berlin/Heidelberg: Springer-Verlag, 2005. doi: 10.1007/3-540-31306-0.
[38] M. Stein, “Large Sample Properties of Simulations Using Latin Hypercube Sampling,” *Technometrics*, vol. 29, no. 2, pp. 143–151, May 1987, doi: 10.1080/00401706.1987.10488205.

[39] A. Arias-Montano, C. A. C. Coello, and E. Mezura-Montes, “Multiobjective Evolutionary Algorithms in Aeronautical and Aerospace Engineering,” *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 5, pp. 662–694, Oct. 2012, doi: 10.1109/TEVC.2011.2169968.
[40] A. Asilian Bidgoli *et al.*, “Machine learning-based framework to cover optimal Pareto-front in many-objective optimization,” *Complex Intell. Syst.*, vol. 8, no. 6, pp. 5287–5308, Dec. 2022, doi: 10.1007/s40747-022-00759-w.
[41] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. in Genetic and Evolutionary Computation Series. Boston, MA: Springer US, 2007. doi: 10.1007/978-0-387-36797-2.
[42] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017.
[43] “minimize(method=’SLSQP’) — SciPy v1.14.1 Manual.” Accessed: Dec. 17, 2024. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html#optimize-minimize-slsqp
[44] *Numerical Optimization*. in Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. doi: 10.1007/978-0-387-40065-5.
[45] D. Kraft, “A Software Package for Sequential Quadratic Programming,” Institut für Dynamik der Flugsysteme Abteilung Regelung Oberpfaffenhofen, D-8031 Weßling, Jul. 28, 1988.
[46] “Insertion.” Accessed: Nov. 01, 2024. [Online]. Available: https://www.genome.gov/genetics-glossary/Insertion
[47] “DEAP documentation — DEAP 1.4.1 documentation.” Accessed: Oct. 30, 2024. [Online]. Available: https://deap.readthedocs.io/en/master/
[48] “SciPy -.” Accessed: Oct. 30, 2024. [Online]. Available: https://scipy.org/
[49] B. Novak and J. J. Tyson, “Design Principles of Biochemical Oscillators,” *Nat Rev Mol Cell Biol*, vol. 9, no. 12, pp. 981–991, Dec. 2008, doi: 10.1038/nrm2530.
[50] E. J. Crampin, S. Schnell, and P. E. McSharry, “Mathematical and computational techniques to deduce complex biochemical reaction mechanisms,” *Progress in Biophysics and Molecular Biology*, vol. 86, no. 1, pp. 77–112, Sep. 2004, doi: 10.1016/j.pbiomolbio.2004.04.002.
[51] O. Roeva, S. Fidanova, and M. Paprzycki, “Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling,” *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems*, pp. 371–376, 2013.
[52] OpenAI, "ChatGPT", GPT-4. [Online]. Available: https://chat.openai.com.
[53] Grammarly, "Grammarly", vers. 2025. [Online]. Available: https://www.grammarly.com.

LIST OF APPENDICES

Appendix A: NSGA-II sorting algorithm

```
initialize superiority tracking:
    for setup in setups:
        dominated_count = 0
        dominating_setups = Empty list
    end
end

initialize fronts:
    current front= empty list
    fronts = empty list
    next front = empty list
end

for setup A in setups:
    for setup B in setups except A:
        if A dominates B:
            dominated_count of B + 1
            add B to dominating_setups of A
        else:
            dominated_count of A + 1
            add A to dominating_setups of B
        end
    end
    if dominated_count of A is 0:
        add A to current_front
    end
end

pareto count = 0 + amount in current front
while pareto_count is smaller than amount to select:
    for setup A in current_front:
        for setup B in dominating_setups of A:
            dominated_count of B - 1
            if dominated_count of B equals 0:
                add B to next_front
                pareto_count + 1
            end
        end
    end
    add current_front to fronts
    set next_front to current_front
end
```

Appendix B: Individual seed results for cases regarding internal mutation probabilities

Note: Using a predefined function in SciPy, seeds 42 and 404 were utilized to generate a comprehensive set of measurement points that thoroughly cover the entire search space. In Case 1, both seeds were employed because relying on a single seed was insufficient. In contrast, for Case 2, seed 42 alone was sufficient.