Optimization of Dimensioning Problems in PlantLib Models Using FMU-Based Workflows

Master's thesis in Complex adaptive systems

Nils Hammar

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg 2025, 2026

www.chalmers.se

Master thesis 2026

Optimization of Dimensioning Problems in PlantLib Models Using FMU-based Workflows

Nils Hammar

CHALMERS

Department of Electrical engineering
Chalmers University of technology
Gothenburg 2026

Optimization of Dimensioning Problems in PlantLib Models Using FMU-Based Workflows

Nils Hammar

Supervisor: Andre Yamashita and Björn Söderlund, Optimation AB
Examinaer: Torsten Wik, Systems and Control, Electrical Engineering

Master's Thesis 2026
Department of Electrical engineering
Systems and Control
Chalmers University of technology
SE-412 96 Gothenburg
Telephone +46 76 948 77 99

.

Written in LaTeX
Gothenburg 2026

Optimization of Dimensioning Problems in PlantLib Models Using FMU-Based Work- flows

Nils Hammar
Department of Electrical engineering
Chalmers University of technology

Abstract

Optimation AB performs optimization of design parameters in industrial plants primarily through experience-based methods and manual trial-and-error testing in simulation models. The development of metaheuristic optimization algorithms has introduced new approaches for optimizing complex, nonlinear systems without requiring knowledge of the internal structure of the problem.
To assess the potential of these algorithms for industrial dimensioning problems, a pulp plant model developed in Dymola was exported as a Functional Mock-up Unit (FMU) and simulated externally using Python. An optimization problem was formulated in which storage tank volumes were defined as decision variables. The objective function was based on net present value (NPV), calculated by combining capital expenditure (CAPEX) and operational expenditure (OPEX) under fixed price assumptions.
The optimization framework was implemented in Python and applied to two baseline methods—Coordinate descent and Latin hypercube sampling—as well as two metaheuristic algorithms: Differential evolution and Simulated annealing. Before being applied to the real plant, the algorithms were tested and tuned on a simpler model designed to be nonsmooth and noisy. While all algorithms were able to identify tank sizes that improved economic performance compared to the baseline design, the metaheuristic methods demonstrated limited practical potential for the tested problem. This was primarily due to high computational cost and sensitivity to noise in the simulation results.

Keywords: Simulation-based optimization, Metaheuristic, Black-box, Industrial process modeling, Plant dimensioning, Differential Evolution, Simulated Annealing, Net Present Value (NPV).

Abbreviations

Below is the list of acronyms that have been used throughout the report, listed in alphabetical order:

CAPEX	Capital Expenditure
CEC	Congress on Evolutionary Computation
CLT	Central Limit Theorem
CRN	Common Random Numbers
DE	Differential Evolution
FMU	Functional Mock-up Unit
FMI	Functional Mock-up Interface
LHS	Latin Hypercube Sampling
MDT	Mean Down Time
MH	Metaheuristic
MTTF	Mean Time To Failure
NPV	Net Present Value
OPEX	Operational Expenditure
SA	Simulated Annealing

Contents

List of Tables

1
Introduction

1.1 Background

Optimation AB uses advanced simulation tools for the pulp and paper, mining, energy, and transport industries. One such tool is the Modelica library PlantLib OEE in Dymola, which can model entire industrial plants—such as pulp mills or cogeneration facilities—and simulate years of operation within minutes.
One important use of these models is production optimization. This includes adjusting plant settings, i.e. control engineering, but the models can also be used to redesign or replace entire plant components digitally before physical investment decisions are made. In this way, simulation models provide a powerful decision-support tool for evaluating long-term operational and economic consequences.
Currently, optimization of such models is typically performed through engineering experience and manual trial-and-error rather than by systematic algorithmic optimization. One reason for this is that the models are highly complex and cannot easily be expressed as differentiable mathematical functions. As a result, classical gradient-based optimization methods are generally not applicable.
However, there exist classes of optimization algorithms that do not require gradient information and that can operate on so-called black-box problems, where only input-output behavior is accessible. The plant model considered here belongs to this class. The targeted problem makes a challenging optimization problem characterized by:

- a black-box objective function,
- stochastic noise,
- limited evaluation budget due to long simulation times,
- and a potentially high-dimensional and nonconvex search space.

Such problems can be addressed using metaheuristic optimization algorithms. Metaheuristic algorithms have been widely applied to black-box engineering optimization problems [1]. In simulation-based settings with stochastic noise, performance depends strongly on evaluation budget and sampling strategies [2, 3]. However, their practical suitability for problems where a whole industrial plant is formulated as a black box simulation models remains less explored.

1.2 Aim

The aim is twofold: First, it seeks to provide an overview of the current state of metaheuristic optimization methods relevant to simulation-based industrial problems. Second, it evaluates the suitability of selected metaheuristic algorithms for

solving plant dimensioning problems in Optimation's Dymola-based models.
More specifically, the study investigates whether metaheuristic methods can efficiently improve the expected economic performance of a stochastic pulp plant simulation under a constrained evaluation budget, and which algorithmic strategies perform best under such conditions.

1.3 Approach

To address this aim, a full-scale pulp plant model was exported as a Functional Mock-up Unit (FMU) and integrated into a Python-based optimization framework. The optimization objective was formulated as a maximization of expected Net Present Value under stochastic reliability disturbances.
Three optimization methods were implemented and compared: Simulated Annealing, Differential Evolution, and Coordinate Search as a baseline reference. Since the objective function evaluations are noisy, a sequential paired-comparison framework using common random numbers was developed to manage statistical uncertainty under a fixed simulation budget.
To better understand algorithmic behavior under controlled conditions, a synthetic noisy test problem based on a modified Rastrigin function was also constructed and used for parameter tuning and methodological analysis.
The main results show that while all algorithms can improve upon a baseline plant configuration, none outperform simple Latin Hypercube sampling within the available budget. Furthermore, in the industrial model, simpler local search methods perform comparably to or better than the metaheuristic approaches. The findings suggest that high noise levels, long simulation times, and a relatively flat economic landscape near the optimum limit the practical advantage of sophisticated metaheuristic strategies in this setting.

1.4 Limitations

The focus is on a single industrial case study and a limited set of optimization algorithms. The evaluation budget is constrained by computational feasibility, which restricts the depth of exploration possible within each run. The economic model is simplified and does not include all real-world cost components.
Consequently, the results should be interpreted as an assessment of algorithm suitability under the specific conditions studied, rather than as a general statement about the effectiveness of metaheuristics in all industrial optimization contexts.

2

Theory

2.1 Mathematical optimization

Formally, a general optimization problem can be written as

$$\max_{x \in \mathcal{X}} f(x),$$

or equivalently as a minimization problem. The components of this formulation are central:

Decision variables. These are the controllable inputs of the system. Each component x_j corresponds to one design choice or parameter whose value influences the system performance. The dimension d of the decision vector defines the size of the search space.

Objective function. The function $f : \mathcal{X} \to \mathbb{R}$ is called the *objective function*. It quantifies the performance of a candidate solution. The goal of optimization is to find a decision vector x^* such that $f(x^*)$ is either maximal or minimal among all feasible solutions.

Constraints. The feasible set $\mathcal{X}$ is defined by constraints that restrict which solutions are admissible. These may include:

- **Bound constraints:** $l_j \leq x_j \leq u_j$,
- **Inequality constraints:** $g_i(x) \leq 0$,
- **Equality constraints:** $h_k(x) = 0$,

2.1.1 Multiple Objective Values

A general multi-objective optimization problem can be formulated as

$$\max_{x \in \mathcal{X}} \; F(x) = \max_{x \in \mathcal{X}} \; \Big(f_1(x), f_2(x), \ldots, f_m(x)\Big),$$

where each function $f_i : \mathcal{X} \to \mathbb{R}$ represents a distinct performance measure. Examples include maximizing economic return while minimizing environmental impact [4]. In contrast to single-objective optimization, there is generally no single solution that simultaneously optimizes all objectives. Instead, solutions are compared using the concept of *Pareto dominance*. A solution x^a is said to dominate x^b if it is no worse in all objectives and strictly better in at least one. The set of non-dominated solutions

forms the *Pareto front*, which represents the trade-off surface between competing objectives [4].

Local and global optima

A point x^* is called a *local optimum* if it is optimal within a small neighborhood of the point. It is a *global optimum* if it is optimal over the entire search space. Many real-world problems are nonconvex and may contain multiple local optima, making global optimization substantially more difficult [5]. A set is convex if any line segment between two points in the set lies entirely within the set.

Principles of conventional optimization

Classical optimization methods typically rely on structural properties of the objective function. If f is differentiable, gradient-based methods use first-order derivatives to determine descent or ascent directions. If second-order derivatives are available, curvature information can accelerate convergence through Newton-type methods [5]. The fundamental idea behind these methods is to exploit local information to iteratively move toward improved solutions. For example, gradient ascent updates a candidate solution according to

$$x_{k+1} = x_k + \alpha_k \nabla f(x_k),$$

where α_k is a step size. Under suitable smoothness and convexity assumptions, such methods can provide strong theoretical guarantees of convergence [5].

2.2 Metaheuristics

When gradients are unavailable, constraints are complex, or noise is present, conventional optimization techniques may become ineffective or impractical. For this reason, another class of optimization algorithms, useful without any knowledge of the objective function, has been developed, called metaheuristics.
Metaheuristics can be applied to black box problems, i.e problems with unknown internals, where only the inputs and outputs are known. The search principle is instead to evaluate the problem iteratively, where the output from the previous iteration is used to tweak the input slightly with the goal of getting closer to the optimum. Most metaheuristics implement an interplay of:

- **exploration:** discovering new regions of the search space,
- **exploitation:** refining known good regions.

They are widely used when objective functions are nonconvex, nonsmooth and when gradients are unavailable or unreliable.

2.2.1 Categorization of Metaheuristic Methods

There are several ways to categorize metaheuristic algorithms. Some surveys classify methods according to their source of inspiration (e.g., biological, physical, so-

cial), while others group them according to structural search mechanisms such as population-based versus single-solution methods [1].
In this work, the algorithms are categorized by mechanism. Most commonly used metaheuristics can be placed within one of the following broad classes: evolutionary algorithms, swarm-based methods, physics-based methods, trajectory-based methods, and hybrid approaches.

Evolutionary algorithms (EA). Evolutionary algorithms (EA) are population-based methods inspired by biological evolution. A set of candidate solutions evolves over generations through variation operators (mutation and recombination) and selection. Search progress is driven by differential survival of higher-quality individuals.
Representative examples include Genetic Algorithms (GA), Differential Evolution (DE), and Evolution Strategies (ES). Their population structure makes them well suited for multimodal (many local optima) problems and moderate-to-high dimensional search spaces [1].

Swarm-based methods. Swarm-based algorithms are inspired by collective behavior in natural systems, such as bird flocks or fish schools. Instead of explicit selection, individuals adapt their positions through information exchange with neighbors or global leaders.
Particle Swarm Optimization (PSO) is the most prominent example. Swarm methods often show rapid initial convergence due to strong information sharing, though they may require parameter tuning to avoid premature convergence [6].

Physics-based methods. Physics-based metaheuristics draw analogies to physical processes such as thermodynamics, gravity, or electromagnetism. Search behavior is modeled through artificial forces, energy transitions, or temperature-like control parameters.
Simulated Annealing (SA) is one of the earliest and most established examples. Although the physical metaphors differ, many such methods reduce to stochastic perturbation combined with structured acceptance criteria [7].

Trajectory-based methods. Trajectory-based metaheuristics operate on a single candidate solution and generate a path through the search space via iterative local modifications. Unlike population-based methods, they focus computational effort on one region at a time.
Examples include Hill Climbing, Simulated Annealing, and Tabu Search. These methods are attractive when objective evaluations are expensive, though they may require explicit mechanisms to escape local optima [1].

Hybrid methods. Hybrid metaheuristics combine components from multiple categories to exploit complementary strengths. A common pattern is to pair a population-based global search with a trajectory-based local refinement stage.

Hybridization has become increasingly common in recent literature, reflecting a shift toward mechanism-oriented improvements rather than the introduction of new metaphors [6].

2.2.2 Recent Developments in Metaheuristics

In recent years, the field of metaheuristics has experienced rapid expansion, with a substantial increase in the number of newly proposed algorithms across diverse optimization domains [8]. Surveys indicate continuous growth in bio-inspired and swarm-based methods, often introducing new hybridizations or adaptive control strategies to improve performance [9].
However, this rapid growth has also contributed to what is commonly referred to as the algorithm zoo phenomenon [7]. A considerable number of recently proposed algorithms are built around metaphorical analogies without clearly separating the narrative inspiration from the underlying search operators. In many cases, newly introduced methods exhibit only minor structural differences from established frameworks while being presented as new metaphors [7, 10].
Furthermore, contemporary metaheuristics often involve increasingly complex and niche-oriented designs. They may include layered hybrid mechanisms, dynamic parameter adaptation, and problem-specific heuristics that increase algorithmic sophistication [1]. Such enhancements can yield improvements in targeted applications, but also often increase difficulty of implementation.

2.3 Scalability and Dimensionality in Metaheuristic Optimization

In black-box optimization, the required number of objective function evaluations generally increases with problem dimensionality d, and benchmarking practice commonly defines evaluation budgets proportional to d. In frameworks such as BBOB/-COCO (Black-Box Optimization Benchmarking, COmparing Continuous Optimizers) [11] and the Congress on Evolutionary Computation (CEC) real-parameter optimization competitions [12], maximum budgets on the order of 10^4d to 10^5d evaluations are frequently adopted. In low-dimensional settings ($d \leq 10$), many classical metaheuristics are highly competitive and often reach near-optimal solutions within modest budgets. For very low dimensions ($d \leq 5$), many problems are often comparatively easy for both stochastic and deterministic methods; gradient-based and direct-search techniques may converge rapidly when applicable [5].
As dimensionality increases to moderate ranges ($10 \leq d \leq 50$), algorithmic differences become more pronounced and careful parameter control becomes important [13]. In higher dimensions ($d \geq 100$), performance degradation is commonly observed due to the curse of dimensionality, and evaluation requirements grow substantially. Certain methods, such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES), remain competitive [14]. Importantly, while evaluation budgets are often scaled linearly with d in benchmarking practice, the total computational cost does not necessarily scale linearly: population sizes, internal model updates, and

matrix operations may introduce superlinear complexity. Consequently, the scalability of metaheuristics depends not only on dimensionality but also on landscape structure, evaluation cost, and algorithm design choices.

2.4 Objective values and performance metrics for optimizers

Optimizer evaluation in deterministic settings can rely on best-found objective value after a fixed number of function evaluations. Under noise, this becomes more nuanced: reported performance depends on both search effectiveness and how uncertainty is reduced through replication.
Common performance metrics include:

- best-so-far mean estimate versus simulation cost,
- confidence interval width (estimation precision) at the best candidate,
- robustness across random seeds or independent runs,

If multiple objectives are considered, Pareto-based metrics such as hypervolume can be used, but for this work the primary focus is single-objective maximization; multi-objective metrics are therefore only briefly referenced and treated as out of scope for the main experimental design.

2.5 Noise

2.5.1 Noise handling strategies in metaheuristic optimization

A comprehensive survey of noisy evolutionary optimization outlines common noise-handling strategies. These include explicit resampling/averaging, adaptive sampling rules, robust selection schemes, fitness inheritance and filtering, and hybrid local validation of elite candidates [3]. The key message is that no single strategy dominates universally; effectiveness depends on noise magnitude, evaluation cost, and the optimizer's decision structure.

2.6 Common random numbers and paired stochastic comparison

2.6.1 Common random numbers and paired differences

A classical variance-reduction approach in simulation is to compare two designs using *common random numbers* (CRN), i.e., running both designs under the same random scenario/seed. If $Y(x, r)$ denotes one replication at design x under replication index r, the paired difference

$$D(r) = Y(x_{\text{new}}, r) - Y(x_{\text{old}}, r)$$

often has lower variance than unpaired differences, because shared stochastic variation cancels out.

2.6.2 Sequential comparison and indifference zones

Rather than deciding using a fixed number of replications, a sequential comparison procedure samples paired differences until a decision can be made with sufficient confidence or until a maximum effort is reached. An *indifference zone* can be introduced to avoid spending excessive simulation effort distinguishing very small performance differences that are practically irrelevant under noise [3].
In this work, the detailed decision rules and how they are integrated into the evaluation layer are described in the Methods/Implementation chapter, since they are part of the software infrastructure rather than general theory.

2.7 Statistical Foundations for Simulation Optimization

2.7.1 Stochastic objective functions and expectation

In simulation-based optimization, evaluating a decision vector x does not produce a deterministic value but a random outcome $Y(x)$ due to stochastic elements in the simulation model. The optimization problem is therefore formulated in terms of the expected objective value,

$$\max_{x \in \mathcal{X}} \mathbb{E}[Y(x)],$$

where $\mathbb{E}[Y(x)]$ denotes the expectation of the random objective [13].
Since the expectation is typically unavailable in closed form, it must be estimated using independent simulation replications. Let $Y(x, r)$ denote the simulation outcome at design x under replication index r. Given n independent replications, the sample mean estimator is

$$\bar{Y}_n(x) = \frac{1}{n} \sum_{i=1}^{n} Y(x, r_i).$$

Under independence and identical distribution of replications, the sample mean is an unbiased estimator,

$$\mathbb{E}[\bar{Y}_n(x)] = \mathbb{E}[Y(x)],$$

and its variance satisfies

$$\mathrm{Var}(\bar{Y}_n(x)) = \frac{\sigma^2(x)}{n},$$

where $\sigma^2(x)$ is the variance of a single replication [13].

2.7.2 Sampling distributions and confidence intervals

The Central Limit Theorem (CLT) implies that for sufficiently large n,

$$\sqrt{n}\Big(\bar{Y}_n(x) - \mathbb{E}[Y(x)]\Big) \xrightarrow{d} \mathcal{N}(0, \sigma^2(x)),$$

justifying approximate normal inference for sample means [13].
Because the variance $\sigma^2(x)$ is typically unknown, it is estimated using the sample variance

$$s^2(x) = \frac{1}{n-1}\sum_{i=1}^{n}\Big(Y(x,r_i) - \bar{Y}_n(x)\Big)^2.$$

A $(1-\alpha)$ confidence interval for $\mathbb{E}[Y(x)]$ is then constructed using the Student-t distribution,

$$\bar{Y}_n(x) \pm t_{1-\alpha/2,\,n-1}\frac{s(x)}{\sqrt{n}},$$

where $t_{1-\alpha/2,\,n-1}$ denotes the $(1-\alpha/2)$ quantile of the t-distribution with $n-1$ degrees of freedom [13]. The use of the t-distribution accounts for uncertainty in the variance estimate when n is finite.

2.7.3 Paired differences and common random numbers

When comparing two candidate solutions x_a and x_b, variance reduction can be achieved using common random numbers (CRN) [13]. Let

$$D(r) = Y(x_a, r) - Y(x_b, r)$$

denote the paired difference under replication index r, where both designs are evaluated using the same random seed.
If the covariance between $Y(x_a, r)$ and $Y(x_b, r)$ is positive, then

$$\mathrm{Var}(D) = \mathrm{Var}(Y(x_a)) + \mathrm{Var}(Y(x_b)) - 2\,\mathrm{Cov}(Y(x_a), Y(x_b)),$$

which is smaller than the variance of differences obtained from independent replications. Paired sampling therefore improves statistical efficiency in design comparisons [13].

2.7.4 Heteroskedastic noise

In many simulation models, the variance $\sigma^2(x)$ depends on the decision vector x. This phenomenon, known as *heteroskedasticity*, is common in stochastic simulation and simulation-based optimization [2]. Under heteroskedastic noise, fixed replication policies may be inefficient: low-variance regions may be oversampled, while high-variance regions require additional replications to obtain reliable comparisons.

3
Method

3.1 Problem definition

3.1.1 Dymola

Dymola is a commercial imlementation of the programming language Modelica, used for modeling complex dynamic systems. PlantLib is a Dymola library developed by Optimation for industrial process modeling. PlantLib provides predefined components representing common plant elements such as tanks, conveyors, pumps, and production units. By assembling these components, full-scale process plants can be modeled.
Given the many use cases of Dymola Plantlib models, it is not obvious what industrial model or what regarding the pulp plant model should be optimized. Determining an objective function to optimize was therefore formulated as one goal of the project. Ideally, the objective should be representative of what a client could want to optimize, but also a problem where the Dymola model shows strength, a problem on which meta heuristic optimization algorithms could be applied and a problem with the right level of difficulty for a thesis work.

3.1.2 Dymola pulp plant model

The industrial model used in this study is a full-scale, average-sized Swedish pulp plant model provided by Optimation . The plant has three types of variables. Firstly there are 148 parameters which are set before the run and kept constant through the run. These include everything from temperatures, pressures and production capacities of different plant parts to the sizes of the storage tanks. One important type of parameter are the so called reliability parameters, MDT (mean downtime) and MTTF (mean time to failure). These exist for many of the sub components and make them turn off randomly based on the mean time to failure and they stay down for a random time sequence based on mean downtime. Without reliability parameters, the plant production would be constant over time and deterministic. The idea with the reliability parameters is to implement realistic disturbance. The pulp plant model also includes 5 inputs which are very similar to parameters, with the difference that they don't need to be constant through the run, but could be set to differ over time. One instance in which this could be useful is if the input wood would have different moisture levels over time. In this particular model, the paper production is set as an input. It is however kept constant and the result is that the plant either produces at the set capacity or not at all. Thus, the production rate is

proportional to the percentage of time which the plant is producing.

Running the plant model also yields 215 outputs, which are all the momentary states of the plant parts, such as tank levels, flows and actual production rates.

Implementation of the Dymola Model in Python via FMU

The Dymola model was exported as a Functional Mock-up Unit (FMU) and integrated into Python using the Functional Mock-up Interface (FMI) standard. An FMU is a self-contained simulation package that contains compiled model code together with a model description (variable definitions, parameters, and metadata). The standard enables models created in one environment (e.g., Dymola) to be simulated and controlled from another (e.g., Python) without requiring the original modeling tool at runtime.

In this implementation, the FMU is treated as a black-box simulator. The Python infrastructure handles parameter injection, simulation control, and post-processing. Before each run, a parameter dictionary is constructed and mapped to the corresponding FMU variables.

The simulation is executed using FMI co-simulation, where the pulp plant model advances in fixed time steps via repeated `doStep` calls. The time step was set to 1200 s. This was increased from 600 s (used in the original application) since it nearly halved the computation time without any noticeable effect on the outcome for this model.

All simulations were executed on a workstation running Windows 10 (version 22H2), equipped with an Intel(R) Core(TM) i7-8700K CPU @ 3.70 GHz (6 physical cores, 12 logical threads) and 16 GB RAM. The optimization framework was implemented in Python 3.12, and up to 11 FMU simulations were executed in parallel using the multiprocessing module.

3.1.3 Plant traits

Before formulating an objective function based on the plant, the plant was run a few times through the FMU to determine reasonable simulation times for the experiment. Given that the goal of the experiment is to simulate steady state, it was important that the simulations were long enough to avoid "starting up effects" on production being dominant. For this, the pulp plant model was run for several years while the total production was plotted against time. In one trial, the experiment was run for 8 years, in which a subtle retardation in the production rate is shown after about three years. This is because pulp plant model is not entirely tuned for long time simulations, but according to the designer of the pulp plant, simulation times of around 0.5-3 years resembles steady state as well as a longer simulation, but without the errors. Some measured simulation times of the FMU are for 8 years, for 1 year and for 0.2 years. A reasonable time for a full run of an optimization algorithm is 12-24 hours

3.1.4 Optimization problem formulation

Decision variable

The chosen decision variables were the volume of the different storage tanks in the plant, for example log storage, raw water tank and lime silo. This is since it is realistic that a client want to change a few of these at once and want to determine optimal sizes. Given that metaheuristics usually have their greatest relative strength at higher dimensions (usually at least 5 decision variables, see Section 2.3), choosing tank sizes is reasonable given that there are 15 of them in the model. In the end, 8 of the tanks were chosen (see Table 3.2).

Objective values

Regarding what in the plant should be maximized there are in reality many features which are usually wished to be either maximized or minimized simultaneously. For example, the paper production is wished to be maximized while simultaneously the electricity purchases minimized. As discussed in theory, optimization algorithms could be implemented with several objective values, but limiting their number makes both implantation less complicated and results more interpretable.

Given that the typical end goal for a client is maximizing profits, it was desirable to translate the input or output values into either costs or revenues. There are in fact models which could translate most desires of a company into economical terms, including less concrete aspects such as emissions, but to limit the scope of the work, the economic aspects of tank investments were limited to investments costs, Capital expenditure, (CAPEX) and the prices of the raw materials and the product (operational expenditure, OPEX).

The model for determining the investment cost of a tank was based on an example presented from a client of Optimation. A typical investment has two costs: A planning cost and a building cost. It is assumed that the planning cost is the same no matter the size, and the capex calculations are therefore based solely on the building costs, which are scaling linearly at a price of 15 000 SEK/m^2 and the tank areas are assumed be cylinders with a height 3 times the radius.

The OPEX costs are also based on a simplified model, where the prices are constant and based on rounded standard industry prices for all the parameters in the plant model bearing economic relevance.

There are many costs with respect to the plants that the model does not consider. The idea of this model is not to be all encompassing, but that it should reflect how an investment would *change* the economical circumstances. An increase in production with x should represent a somewhat realistic increase in revenue, even if all costs are not taken into account. It is worth mentioning that the revenue from sold paper dominates the operational expenditures, (see Figure 3.1).

OPEX and CAPEX could be treated as separate quantities to be minimized simultaneously. However, to determine whether an increase in expected revenue justifies a one-time investment, economic evaluation methods are required to combine these quantities into a single measure. One such method is the net present value (NPV),

Table 3.1: Economic parameters used in the model. Prices are given in SEK.

Item	Price	Unit
Wood	700	SEK / tonne
Lime	2400	SEK / tonne
Bark fuel	500	SEK / tonne
Electricity (purchased)	250	SEK / MWh
Water	5	SEK / m^3
Wastewater	10	SEK / m^3
Paper (product)	-8000	SEK / tonne
Bark product (by-product)	-300	SEK / tonne

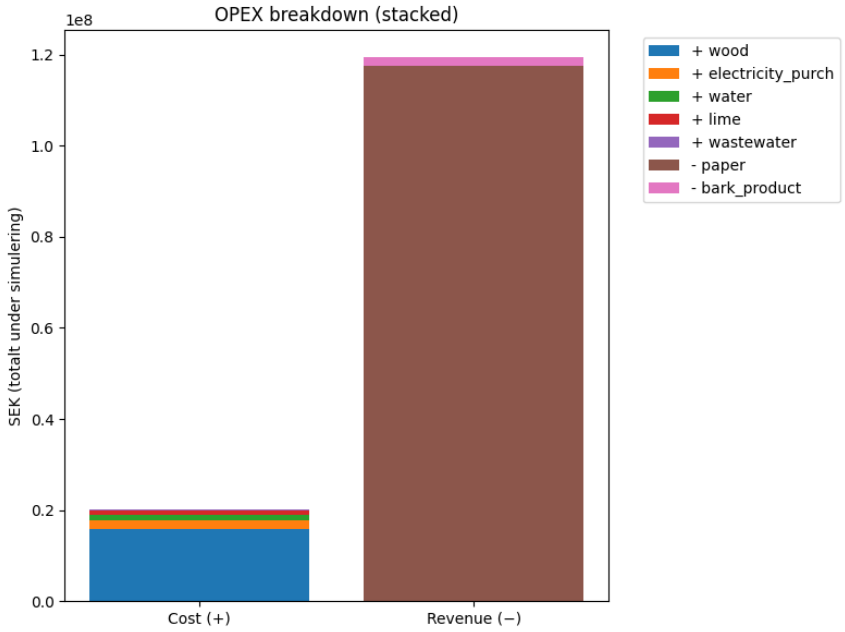

Figure 3.1: Breakdown of the operational expenditures for one year. The revenue from the amount of paper produced is dominating the operational expenditures.

or *nuvärde* in Swedish [15], defined as

$$\text{NPV} = \text{CAPEX} - \text{OPEX}_{\text{annual}} \sum_{t=1}^{T} \frac{1}{(1+r)^t}, \tag{3.1}$$

where r denotes the discount rate, representing the required rate of return, and T is the lifetime of the investment.

In this study, T is assumed to be 40 years and the discount rate is set to 5%. The analysis is conducted in real terms, excluding inflation, since all prices are assumed constant throughout the simulations. Revenue from paper sales is represented as negative operational expenditure.

3.1.5 Constraints

There is no theoretical upper or lower limit for the tank sizes, but for the cost scaling factors to be somewhat reasonable, an upper and lower limit to the tank sizes are introduced. Having a well defined search space is also necessary for many optimization algorithms. The upper and lower bounds are moderate scalings from the original values (base values) set in Dymola.

Table 3.2: Design parameter bounds and baseline values. (The unit is m^3

Parameter	Lower bound	Base value	Upper bound
Accumulator volume	100	200	1500
Blas tank volume	1000	5000	20000
Vedlager volume	1000	5000	20000
) Flislager volume	1000	5000	20000
Torn volume	1000	5000	20000
Stort torn volume	5000	15000	60000
Tunnluttank volume	200	1000	5000
Tjockluttank volume	50	200	2000

3.1.6 Seeds and parallelism

The objective is to maximize the expected value of the NPV. Due to the stochastic reliability parameters (MDT and MTTF), a single FMU run provides only one realization of the stochastic objective. Therefore, the objective value at a design point is estimated as the sample mean of multiple independent replications.
To reduce simulation time, replications are executed in parallel using Python's multiprocessing framework. In the present setup, up to 11 processes could be run in parallel with approximately the same wall-clock time as a single simulation.
One objective evaluation consists of n independent replications, each corresponding to one FMU simulation over a fixed time horizon. For example, 11 simulations of 0.5 years each are executed in parallel and their sample mean is treated as one aggregated evaluation corresponding to 5.5 effective simulation years. For paired comparisons, identical replication indices are used for both designs (common random numbers, CRN); the resulting paired-difference logic is described in Section 3.4.2

3.2 Problem characterization

3.2.1 Runs for diagnosing problem characteristics

Running the plant for 5.5 years using the parallelization takes roughly 260 seconds, which corresponds to 166 simulations in 12 hours. For this experiment, a simulation time of 12-24 hours is manageable. Given this relatively long simulation time of the pulp plant model, the majority of the experiment design and parameter tuning was conducted using a custom made, simplified model. To make the characteristics of the simplified model reasonably similar to the real model, the characteristics and

magnitude of the stochastic elements was determined. For this, two experiments were conducted: Latin Hypercube and neighborhood evaluation.

Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) [16] is a space-filling design method that generates representative samples within a bounded search space without requiring gradient information.
The method partitions each dimension into n equal-probability strata. In this study, $n = 80$, resulting in 80 design points. For each of the eight dimensions, one value is drawn at random from each stratum, and the strata are randomly permuted across dimensions. The design makes sure that each stratum is used exactly once. This guarantees uniform coverage of every marginal distribution while preserving randomness within each interval.
As a result, the full range of each variable is systematically explored, yielding a design with strong space-filling characteristics. Compared to a full factorial grid, LHS avoids the exponential growth in sample size as the dimensionality increases.
After the initial 80 evaluations, the five best-performing points were re-evaluated three additional times to reduce estimator variance. One evaluation of a single design point consists of 11 parallel simulation runs of 0.8571 years each, executed using independent random seeds across separate computational kernels. The estimated objective value at one point is computed as the sample mean across these independent replications.

Table 3.3: Main results from the Latin Hypercube Sampling (LHS) experiment, rounded to 4 significant numbers

Metric	Value
Best evaluation	1.919×10^{10}
Worst evaluation	1.598×10^{10}
Largest sample standard deviation	1.306×10^{9}
Smallest sample standard deviation	1.622×10^{8}
Median sample standard deviation	5.009×10^{8}

Neighbourhood Evaluation

To quantify the magnitude of local stochastic variability, a neighbourhood experiment was conducted. The purpose was to estimate the variability of paired differences between points that are extremely close in the search space, and to compare this to the variability between more distant points.
One design point was selected from the Latin hypercube sample that was close to the baseline tank configuration. From the selected point, a small perturbation was applied in all eight dimensions. Specifically, for each coordinate, a step corresponding to 10^{-4} of the total variable range was added or subtracted. The sign of the perturbation in each dimension was chosen randomly and then kept fixed for the subsequent steps.

From the initial point, three consecutive perturbations of equal length were applied in the same direction, resulting in four nearby points forming a short path in the search space. All evaluations were performed using identical replication settings. Pairwise differences between adjacent points were then computed, and the pooled standard deviation of these paired differences was used as an estimate of local noise.
For comparison, the same procedure was applied to 11 replications of four of the five best-performing points identified in the Latin hypercube experiment. The choice of 11 replications matches the level of parallelism used in later optimization experiments, where 11 independent half-year simulations are averaged to form one evaluation. This allows direct comparison between neighbourhood noise and non-neighbour noise under consistent replication structure.
The pooled standard deviation of paired differences for randomly selected Latin hypercube points was estimated at 354,100,663.3 SEK (simulation length 0.8571 years per replication). For neighbouring points, the corresponding value was 815,415,828 SEK (simulation length 0.54 years per replication).
Since the metaheuristic experiments use aggregated evaluations corresponding to approximately 5.5 simulation years (11 parallel runs of 0.5 years each), the noise levels were rescaled to reflect longer simulation time. Assuming independent increments, the variance scales linearly with simulation time, implying that the standard deviation scales according to

$$\sigma_{\text{large}} = \sigma_{\text{small}} \sqrt{\frac{t_{\text{small}}}{t_{\text{large}}}},$$

where t denotes simulation time. This scaling assumes independent increments over time and variance proportional to simulation horizon.
Applying this scaling yields estimated 5.5-year standard deviations of 140 million SEK for non-neighbor points and 257 million SEK for neighboring points.

3.3 Simplified Model: Noisy Kinked Rastrigin

To evaluate algorithmic behavior in a controlled setting, a synthetic test problem based on a modified Rastrigin function was constructed. The aim is not to replicate the pulp plant model in detail, but to reproduce its key structural and stochastic characteristics while remaining computationally inexpensive and fully reproducible.

Search domain and dimension. The problem is defined in $d = 8$ dimensions with box constraints

$$x \in [-5.12,\, 5.12]^8.$$

Deterministic component. The deterministic core is the classical Rastrigin function [17],

$$f_{\text{R}}(x) = 10d + \sum_{i=1}^{d} \left(x_i^2 - 10\cos(2\pi x_i) \right),$$

augmented with two nonsmooth modifications:

$$f_{\text{det}}(x) = f_{\text{R}}(x) + \alpha \sum_{i=1}^{d} |x_i| + \beta \sum_{i=1}^{d} \max(0,\ |x_i| - c)\,,$$

where the parameters are set to

$$\alpha = 0.15, \qquad \beta = 0.25, \qquad c = 1.5.$$

The ℓ_1-type term introduces nondifferentiability along coordinate hyperplanes, while the saturation term creates piecewise-linear growth outside a threshold region. These modifications break the symmetry and smoothness of the classical Rastrigin landscape.
The optimization problem is formulated as a maximization task by negating the deterministic function.

Stochastic structure. The stochastic objective value evaluated at decision vector x with replication seed r is defined as

$$Y(x, r) = -\left(f_{\text{det}}(x) + \varepsilon_{\text{seed}}(x, r) + \varepsilon_{\text{point}}(x, r)\right).$$

The noise model consists of two components:

1. **Seed-level noise:**

   $$\varepsilon_{\text{seed}}(x, r) = \sigma_{\text{seed}}(x)\, Z_r, \qquad Z_r \sim \mathcal{N}(0, 1),$$

   where the standard deviation depends on the normalized distance from the origin,

   $$\sigma_{\text{seed}}(x) = \sigma_{\min} + (\sigma_{\max} - \sigma_{\min})\, r(x).$$

   Here,

   $$\sigma_{\min} = 4.0, \qquad \sigma_{\max} = 32.0,$$

   and

   $$r(x) = \frac{\|x\|_2}{R\sqrt{d}}, \qquad R = 5.12.$$

   The quantity $r(x) \in [0, 1]$ ensures that noise magnitude increases linearly with distance from the center of the domain, producing heteroskedastic behavior.
2. **Pointwise noise under identical seed:**

   $$\varepsilon_{\text{point}}(x, r) = \sigma_{\text{point}}\, W_{x,r}, \qquad W_{x,r} \sim \mathcal{N}(0, 1),$$

   with

   $$\sigma_{\text{point}} = 12.5.$$

Both noise components are deterministic given the pair (x, r), ensuring full reproducibility.

Calibration rationale. The deterministic magnitude of the objective was estimated using a Monte Carlo experiment with 100 uniformly sampled design points. Noise parameters were selected so that the standard deviation of paired differences became proportional to the observed objective span, yielding a noise-to-signal ratio comparable to that measured in the pulp plant simulations. The scaling of the noise levels is made to be equal in size compared to the measured search space magnitude as in the pulp model. The search space is estimated to 150, which is not the absolute search space of the intercept, rather the difference between largest and smallest measured values from the Monte Carlo simulation. This is since the search space measured for the pulp plant is also just an estimation from a few rounds.

Resulting characteristics. The final model exhibits:

- Multiple local optima,
- Nonsmooth structure,
- Heteroskedastic stochastic noise,
- Partial correlation under common random numbers.

The pulp plant model therefore provides a controlled yet representative testbed for studying algorithmic robustness under noisy simulation conditions.

3.4 Experimental design and evaluation protocol

3.4.1 Simulation budget and reporting

To ensure fair comparison across optimization methods, all algorithms were run under a common simulation budget expressed as a maximum number of FMU replications (simulation calls). Each algorithm iteration may consume a variable number of replications because accept/reject decisions are based on sequential sampling of paired differences. Unless stated otherwise, each run logs the best-so-far objective value as a function of cumulative simulation calls.
After termination, the top-performing candidate solutions are re-evaluated with an increased replication count to obtain more accurate final estimates. This separates inexpensive search-time decisions from higher-confidence reporting.
comparison statistics

3.4.2 Stochastic paired differences

When objective function evaluations are subject to stochastic noise, direct comparison of sample means can lead to excessive simulation effort. This motivates decision rules based on paired comparisons under common random numbers [2].

Common Random Numbers and Pairing. Let $Y(\mathbf{x}, r)$ denote the outcome of the objective function evaluated at decision vector $\mathbf{x}$ using replication index r, where r uniquely determines the random seed. For any comparison between two solutions $\mathbf{x}_{\text{new}}$ and $\mathbf{x}_{\text{old}}$, the evaluator enforces *paired sampling* by using the same replication

indices for both points. That is, for each r,

$$D(r) = Y(\mathbf{x}_{\text{new}}, r) - Y(\mathbf{x}_{\text{old}}, r).$$

Using common random numbers (CRN) in this matched-pairs fashion can reduce the variance of difference estimators by inducing positive correlation between the two outputs, improving the efficiency of comparisons [13].

Sequential Estimation of Differences. Rather than fixing the number of replications in advance, differences are sampled sequentially, a common approach in simulation experimentation and ranking-and-selection style comparisons [13]. After each sampling stage, the evaluator computes the sample mean and standard error of the paired differences:

$$\bar{D} = \frac{1}{n}\sum_{i=1}^{n} D(r_i), \qquad \text{SE} = \frac{s_D}{\sqrt{n}},$$

where s_D is the sample standard deviation of the differences and n is the number of paired replications collected so far.
A one-sided Student-t critical value $t_{1-\alpha,\,n-1}$ is used to construct a symmetric confidence interval,

$$\left[\bar{D} - t_{1-\alpha,\,n-1}\,\text{SE},\ \bar{D} + t_{1-\alpha,\,n-1}\,\text{SE}\right],$$

which corresponds to a two-sided confidence level of approximately $1 - 2\alpha$ [13]. The significance level α used in the confidence interval is allowed to vary over the course of the optimization. Early in the search, a looser confidence level of $\alpha = 0.1$ is employed to encourage exploration, while later iterations use stricter confidence levels to promote reliable exploitation near promising solutions; this reflects the general exploration–exploitation tradeoff in simulation optimization [2]. The later confidense is the same as used everywhere else if not explicitly stated, $\alpha = 0.05$ or 95 percent confidence. Replications are assumed independent across seeds and normality is approximated.

Indifference Zone and Decision Rules. To avoid expending simulation effort on differences that are small relative to stochastic variability, an indifference-zone parameter $\delta > 0$ is introduced.
The value of δ is computed adaptively as

$$\delta = \max\left(\delta_{\text{rel}}|\hat{f}_{\text{best}}|, \delta_{\text{floor}}\right),$$

where $\hat{f}_{\text{best}}$ is the best estimated mean observed so far, δ_{rel} is a relative tolerance, and δ_{floor} is a problem-dependent absolute lower bound.
Given the confidence interval bounds (L, U) for $\bar{D}$, the comparison decision for a maximization problem is:

- **Accept** $\mathbf{x}_{\text{new}}$ if $L > \delta$
- **Reject** $\mathbf{x}_{\text{new}}$ if $U < -\delta$
- **Indifferent** if $L > -\delta$ and $U < \delta$

If none of these conditions are met, additional paired replications are sampled until a decision is reached or a maximum replication limit is exceeded, in which case the comparison is conservatively classified as indifferent.

3.5 Optimization algorithms

To evaluate the suitability of metaheuristic methods for noisy, simulation-based industrial optimization, two representative metaheuristic algorithms were selected: Simulated Annealing (SA) and Differential Evolution (DE). In addition, a coordinate search method was included as a deterministic baseline reference.
The metaheuristics were selected to represent two fundamentally different classes of search strategies. Simulated Annealing is a *trajectory-based* method that iteratively updates a single candidate solution using a temperature-controlled acceptance rule [18]. Differential Evolution is a *population-based* evolutionary algorithm that evolves a set of candidate solutions through mutation, crossover, and selection [19]. The distinction between trajectory-based and population-based metaheuristics is standard in the literature [1].
These two methods were chosen for several reasons. First, both have demonstrated strong empirical performance on nonconvex, derivative-free optimization problems where gradient information is unavailable [13]. Second, both are known to exhibit relatively rapid practical convergence in moderate dimensions compared to exhaustive sampling approaches [19]. Third, both methods can be naturally combined with stochastic objective evaluations and paired comparisons, making them suitable for simulation optimization under noise [2].
The two algorithms also differ structurally in how they balance exploration and exploitation. SA controls this tradeoff explicitly through its temperature schedule [18], whereas DE adapts search steps implicitly through scaled population differences [19].
As a baseline, coordinate search is included as a reference method. Coordinate search is a deterministic, derivative-free local search procedure that explores one coordinate direction at a time [13]. It provides a transparent and computationally lightweight benchmark. Comparing metaheuristics against this structured local method makes it possible to assess whether additional algorithmic complexity yields improved performance under noisy simulation conditions.
All algorithms interact with the objective function exclusively through the shared noisy evaluator described in subsection 3.4.2. The evaluator handles replication management, common random numbers, and statistical accept/reject decisions. The descriptions below therefore focus only on the search logic of each method.

3.5.1 Benchmark algorithm: Coordinate search

Coordinate search

Coordinate search (also referred to as coordinate ascent/descent) starts from a feasible point $x \in \mathbb{R}^d$, the algorithm perturbs a single component x_j while keeping all others fixed and accepts improving moves. The procedure is repeated in sweeps over all coordinates.

Step-size initialization. Each coordinate has an associated step size initialized as

$$\alpha_j^{(0)} = \texttt{step\_rel0} \cdot (u_j - \ell_j),$$

which scales the method with respect to the domain width [20]. A numerical lower bound on $(u_j - \ell_j)$ prevents degeneracy.

Move generation. For each coordinate j, two projected trial points are considered:

$$x^{(+)} = \Pi(x + \alpha_j e_j), \qquad x^{(-)} = \Pi(x - \alpha_j e_j),$$

where e_j denotes the j-th canonical basis vector and $\Pi(\cdot)$ denotes projection onto the feasible box. A local first-improvement rule is used: once an improving move along coordinate j is accepted, the algorithm proceeds to the next coordinate. The coordinate order may optionally be randomized each sweep [21].

Step-size adaptation and termination. If no improving move is found during a sweep, all step sizes are reduced:

$$\alpha_j \leftarrow \texttt{step\_shrink} \cdot \alpha_j \quad \forall j.$$

The algorithm terminates if:

- a maximum number of sweeps is reached,
- the global simulation budget is exhausted,
- no improvement occurs for a predefined number of sweeps,
- all α_j fall below $\texttt{step\_min\_rel}(u_j - \ell_j)$.

Coordinate search serves as a simple and computationally lightweight baseline representing structured local search [20].

3.5.2 Metaheuristic algorithms

Simulated Annealing

Simulated Annealing is implemented here as a single-solution stochastic search process driven by temperature-controlled acceptance decisions, originally proposed by Kirkpatrick et al. [18]. The method generates local perturbations of the current candidate and relies on probabilistic acceptance to regulate the balance between diversification and refinement throughout the run.

Proposal mechanism. At iteration t, a candidate solution $\mathbf{x}$ is perturbed by adding Gaussian noise scaled relative to the domain size,

$$\mathbf{x}_{\text{new}} = \Pi(\mathbf{x} + \gamma_t(\mathbf{u} - \mathbf{l}) \odot \boldsymbol{\varepsilon}),$$

where $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, I)$, $\gamma_t > 0$ is a step-size parameter, $(\mathbf{u} - \mathbf{l})$ denotes the coordinate-wise span of the search domain, $\Pi(\cdot)$ denotes projection onto the feasible box constraints and $\odot$ denotes element-wise multiplication.
Scaling the perturbation with the domain width ensures that the algorithm behaves consistently across variables with different magnitudes.

Acceptance rule. Let $Y(\mathbf{x}, r)$ denote one stochastic replication evaluated using common random numbers (CRN). For each proposal, a paired replication is performed and the difference

$$\Delta = Y(\mathbf{x}_{\text{new}}, r) - Y(\mathbf{x}, r)$$

is computed.
If $\Delta \geq 0$, the new solution improves the objective and is accepted. If $\Delta < 0$, the move is accepted with probability

$$p = \exp\left(\frac{\Delta}{T_t}\right),$$

which corresponds to the classical Metropolis acceptance rule [18].
The parameter $T_t > 0$ is called the temperature and controls the probability of accepting worsening moves. When T_t is large, even substantially negative values of d may be accepted, promoting global exploration. As T_t decreases, the probability of accepting worsening moves diminishes, and the algorithm behaves increasingly like a greedy local search. Because acceptance decisions are based on single paired replications, the method may occasionally accept moves due to stochastic fluctuations; however, the decreasing temperature reduces the long-run probability of noise-driven acceptance.

Cooling schedule. The temperature follows a geometric cooling schedule,

$$T_t = T_0 \alpha_T^t,$$

where $T_0 > 0$ is a chosen initial temperature and $0 < \alpha_T < 1$ is the cooling rate. This schedule gradually reduces exploration over time. A slow cooling rate (i.e., α_T close to 1) promotes extensive exploration, while faster cooling leads to more rapid convergence.

Termination criteria. The algorithm terminates when one of the following conditions is met: (i) a predefined maximum number of iterations is reached, (ii) the global simulation budget is exhausted, (iii) the proposal step size γ_t falls below a prescribed threshold, or (iv) no improvement in the best-so-far solution is observed over a specified number of reporting intervals. After termination, the best-performing candidate solutions are re-evaluated using higher replication counts to obtain accurate final estimates.

Differential Evolution

Differential Evolution (DE) is implemented in the classical *DE/rand/1/bin* variant [19]. The algorithm maintains a population of candidate decision vectors and iteratively improves them through mutation, crossover, and statistically gated selection. In the present application, each vector $\mathbf{x} \in \mathbb{R}^d$ represents a complete set of tank-level decisions subject to constraints $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$.

Initialization. Let N_p denote the population size. The initial population

$$\mathcal{P}^{(0)} = \{\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_{N_p}^{(0)}\}$$

is constructed by projecting the user-provided starting point $\mathbf{x}_0$ into the feasible domain and sampling the remaining $N_p - 1$ individuals uniformly within the box constraints.

Mutation (rand/1). For each target vector $\mathbf{x}_i^{(g)}$ in generation g, three distinct indices $r_1, r_2, r_3 \neq i$ are sampled from $\{1, \ldots, N_p\}$. A mutant vector is formed as

$$\mathbf{v}_i^{(g)} = \mathbf{x}_{r_1}^{(g)} + F\Big(\mathbf{x}_{r_2}^{(g)} - \mathbf{x}_{r_3}^{(g)}\Big),$$

where $F > 0$ is the mutation factor controlling the step magnitude. The difference term provides a population-driven search direction, yielding adaptive step sizes without gradient information.

Binomial crossover. A trial vector $\mathbf{u}_i^{(g)}$ is generated by combining the mutant $\mathbf{v}_i^{(g)}$ with the parent $\mathbf{x}_i^{(g)}$. For each coordinate $j \in \{1, \ldots, d\}$, the trial inherits the mutant component with probability $CR \in [0, 1]$, and otherwise retains the parent component. One randomly selected coordinate is always inherited from the mutant to ensure that $\mathbf{u}_i^{(g)} \neq \mathbf{x}_i^{(g)}$. The resulting vector is projected onto the feasible box.

Selection under noise. Each trial competes with its parent through the shared noisy evaluator. If the trial is accepted, it replaces the parent in the next generation; otherwise, the parent is retained. To avoid unnecessary replacements under nearties, an additional minimum improvement threshold $\Delta_{\text{accept}} \geq 0$ (default 0) may be imposed on the estimated paired difference.

Termination and reporting. The algorithm terminates when a predefined maximum number of generations is reached, the global simulation budget is exhausted, or no replacements are accepted for a specified number of consecutive generations. Upon termination, the evaluator performs a higher-precision re-evaluation of the best archived solutions and reports their sample means and confidence intervals.

Control parameters. Key parameters are the population size N_p, mutation factor F, crossover probability CR, maximum number of generations, and the acceptance threshold Δ_{accept}. The computational cost scales approximately linearly with N_p and the number of generations, with additional overhead from replication-based comparisons.

3.6 Implementation

3.6.1 Software structure

The optimization framework was implemented in Python using a modular design. The FMU is accessed through a model wrapper exposing a single evaluation interface. All optimization algorithms interact with the plant model only through a

shared evaluator component responsible for replication management, paired comparisons, and logging. This separation ensures that algorithm implementations remain independent of the FMU-specific details and that all methods are compared under identical noise-handling rules.

3.6.2 Caching and tolerance rounding

To avoid repeated FMU evaluations at numerically equivalent design points, a cache is maintained over previously evaluated solutions. Since the decision variables are continuous, each coordinate is mapped to a rounded key before cache lookup. For coordinate j, the rounding resolution is defined as

$$\varepsilon_j = \varepsilon_{\text{rel}}(u_j - \ell_j),$$

and the cached representation is obtained by rounding each coordinate to the nearest multiple of ε_j. This makes repeated evaluations at the same effective design point reusable across algorithms and across repeated comparisons, improving computational efficiency.

3.6.3 Tuning

Given the cheapness of the Rastrigin simulations, an effort of finding optimal values for a few key parameters was made before testing on the pulp plant.
To identify suitable parameter values for the real pulp plant simulations, both (i) a small set of evaluator parameters controlling noise-handling decisions and (ii) a small set of algorithm-specific hyperparameters controlling exploration. were tuned. The tuning objective was *average improvement* after a fixed number of objective evaluations, measured across many independent runs with different random seeds and random initial points.

Budgeted multi-run evaluation. Each candidate configuration (evaluator or algorithm parameters) was evaluated using 100 independent runs. For run k, (i) a distinct algorithm RNG seed, (ii) a distinct model initialization seed ensuring that each run starts from a different point in the search space, and (iii) a distinct evaluator base seed to decorrelate replication streams, were used. For each run, the best-so-far value at the target budget B objective calls was recorded. The performance metric was

$$\Delta(B) = f_{\text{best}}(B) - f_{\text{best}}(0),$$

and each configuration was summarized by the sample mean of $\Delta(B)$ across the 100 runs, that is the average improvement from the starting point across 100 runs. The evaluations are values including the noise, so the true improvements are unclear.

Two-stage tuning order. For each Algorithm, tuning was conducted in two stages:

1. **Evaluator stage.** A small grid search was performed over the evaluator parameters that most directly control the cost–quality trade-off in noisy comparisons: $(n_{\min}, n_{\max}, \delta_{\text{rel}})$, together with fixed choices for batch size and rounding tolerance. The configuration with the highest mean $\Delta(B)$ was selected.
2. **Algorithm stage.** With the evaluator fixed to the selected configuration, three algorithm-specific parameters were tuned using a one-at-a-time (coordinate-wise) search. Starting from the baseline configuration, a predefined range was swept for one parameter at a time while keeping the others fixed. The best-performing value was selected before proceeding to the next parameter.

This ordering was chosen because algorithm behavior depends strongly on how expensive and how conservative each accept/reject decision is; therefore the evaluator should be fixed first to provide a consistent notion of "progress per evaluation". It should be noted that in the cases when the best values were on the edge of the tested range, a test with new values outside of the old range should ideally have been used, but this was not done in this experiment due to limited time.

Table 3.4: Comparison of original and tuned parameter values. ΔB denotes average improvement after 200 simulations.

Alg.	Parameter	Original	Range Tested	Chosen	ΔB
DE	$n_{\max}$	4	$\{3, 4, 6\}$	3	60.3 → 76.5
	δ_{rel}	0	$\{0, 0.001, 0.005\}$	0	
	pop_size	5	$\{4, 5, 6, 8\}$	5	
	F	0.7	$\{0.5, 0.6, 0.7, 0.8\}$	0.5	
	CR	0.9	$\{0.6, 0.75, 0.9, 0.95\}$	0.95	
COORD	$n_{\max}$	3	$\{3, 4, 6\}$	6	47.7 → 73.0
	δ_{rel}	0	$\{0, 0.001, 0.005\}$	0.001	
	step_rel0	0.20	$\{0.05, 0.1, 0.2, 0.35, 0.5\}$	0.35	
	step_shrink	0.50	$\{0.3, 0.5, 0.7, 0.85\}$	0.3	
	rand._order	False	{False, True}	True	
SA	$n_{\max}$	4	$\{3, 4, 6\}$	3	30.5 → 62.2
	δ_{rel}	0	$\{0, 0.001, 0.005\}$	0	
	step_scale0	0.15	$\{0.05, 0.1, 0.2, 0.35, 0.5\}$	0.05	
	T_0	300	$\{10, 30, 100, 200, 400\}$	10	
	α_T	0.995	$\{0.97, 0.98, 0.99, 0.995\}$	0.97	

3.6.4 Final evaluation

The final evaluation quantifies how the three optimization methods perform on the full pulp plant NPV problem under a fixed simulation budget. To reduce sensitivity to initialization while keeping the experimental cost manageable, each method is executed from two starting points: (i) the baseline tank configuration provided with the Dymola model, and (ii) the best-performing design point identified in the Latin hypercube experiment (Table 3.3).

Budget and termination. Each run is conducted under a common budget expressed in FMU replications (`sim_calls`). The search phase is allowed to consume at

most $B = 200$ simulation calls, corresponding to roughly 12 hours. After the search phase terminates, the best candidate solution from the run is re-evaluated using an additional $n_{\text{final}} = 5$ independent replications (performed with fresh replication indices) to obtain a higher-confidence final estimate. The final reported performance for each run consists of: (i) the re-estimated mean NPV, and (ii) a confidence interval computed from the five independent post-run replications using a Student-t interval at the chosen confidence level.
Each replication corresponds to a fixed simulation horizon of 0.5 years, and independent replication indices are used to ensure statistical independence across samples.

4
Results

4.1 Pulpplant economic model

In the tables, all algorithm evaluations are written in the unit billion SEK. The in-run best estimate represents the average of all evaluations performed at that point. The number of evaluations it is based on is thus different depending on how many replications the evaluator used, usually 2-3 however. The high precision mean is based on 5 re-evaluations at the same point. Table 4.1 shows the final performance under a budget of 200 simulation calls, starting from the Dymola baseline configuration. Each solution is reported both with its in-run best estimated mean and its post-run high-precision re-evaluation. Table 4.3 shows the same results but starting in the best point from LHS. Below these tables, in Table 4.2 and 4.4 are the tank sizes corresponding to the results. Table 4.5 shows high precision evaluations of the two starting points.

Table 4.1: Final performance from baseline configuration

Algorithm	**Best in-run estimate**	**High-precision mean**	**Confidence interval**
Coord.	19.21	19.03	[18.93, 19.13]
SA	19.14	18.83	[18.68, 18.98]
DE	18.69	18.49	[18.26, 18.72]

Table 4.2: Final tank volumes (m^3) for the best solution found by each algorithm when starting from the Dymola baseline. The tank order follows Table 3.2.

Algorithm	**Tank sizes (m^3)**
Coordinate search	$\big(200,\ 11650,\ 13023,\ 5000,\ 11650,\ 53500,\ 1000,\ 200\big)$
Simulated Annealing	$\big(380,\ 4997,\ 2194,\ 1000,\ 11522,\ 43659,\ 533,\ 1048\big)$
Differential Evolution	$\big(932,\ 9737,\ 13023,\ 12868,\ 20000,\ 34621,\ 5000,\ 2000\big)$

Tank order: (Accumulator, Blas tank, Vedlager, Flislager, Torn, Stort torn, Tunnluttank, Tjockluttank).

Table 4.3: Final performance under a budget of 200 simulation calls, starting from the best Latin hypercube sample point.

Algorithm	Best in-run estimate	High-precision mean	Confidence interval
Coord.	19.69	19.26	[19.16, 19.35]
SA	19.48	19.12	[18.96, 19.28]
DE	19.48	19.27	[18.90, 19.39]

Table 4.4: Final tank volumes (m^3) for the best solution found by each algorithm when starting from the best Latin hypercube sample point. The tank order follows Table 3.2.

Algorithm	Tank sizes (m^3)
Coordinate search	$\big(688,\ 19753,\ 7042,\ 17378,\ 13369,\ 60000,\ 243,\ 1623\big)$
Simulated Annealing	$\big(198,\ 19753,\ 7042,\ 17378,\ 6719,\ 58625,\ 243,\ 1623\big)$
Differential Evolution	$\big(198,\ 19753,\ 7042,\ 17378,\ 6719,\ 58625,\ 243,\ 1623\big)$

Table 4.5: High-precision re-evaluation of the two starting points. Each value is computed using independent post-run replications with a Student-t confidence interval.

Starting point	High-precision mean	Confidence interval
Dymola baseline	17.72	[17.61, 17.90]
Latin Hypercube best	19.19	[19.08, 19.30]

5
Discussion

5.1 Tuning of toy model

Apart from serving as a method for identifying improved parameter settings for the real experiment, the best parameters for the Rastrigin model constitute a result in itself. Table 3.4 shows that all algorithms show a clear improvement after tuning, and that the performance ranking of the algorithms remains unchanged.
Most tuning parameters reflect better adaptation to the scale of the problem, but there are some interesting observations. To begin with, the δ_{rel} parameter appears to provide very limited benefit. This suggests that the risk of insufficient exploration may be greater than the risk of occasionally moving to a point that is actually worse. The preference for fewer replications per point points in the same direction, since fewer replications free up budget to evaluate more candidate solutions.
The strategy of measuring average improvement of estimated noisy values, rather than true improvement, has the advantage of making the results more comparable to those of the real model, where only noisy values are available. If more time had been available, it would have been appropriate to complement this with an evaluation of the improvement in noise-free objective values. It is entirely plausible that a strategy favoring exploration over careful evaluation does not truly find better solutions, but instead increases the probability of encountering noise realizations that temporarily inflate objective values — a form of sampling bias. More careful evaluation at each point would yield more accurate, but often lower, estimates, which the algorithm may not favor.
That simulated annealing clearly preferred shorter step sizes and lower initial temperatures shows that it is not only the number of evaluated points that matters. Larger step sizes and higher temperatures make the search behavior resemble a random walk.
The magnitude of the sampling bias appears largest for simulated annealing, suggesting that single-replication acceptance may be particularly susceptible to noise-induced overestimation.

5.2 Results from pulp plant model

In the pulp plant trials, there are indications of a similar bias toward points that appear large due to noise. The fact that all high-precision means are substantially lower than their corresponding in-run estimates points to this effect. While all algorithms were able to find better values than the original Dymola starting point,

none of them can be considered particularly powerful. None were able to find a better solution within 200 evaluations than what the Latin hypercube achieved in 80 evaluations.

When using the more realistic experimental design of starting from the best LHS point, the results are even less encouraging. Only coordinate ascent managed to improve the starting point, and only marginally.

Coordinate ascent, included as a non-metaheuristic baseline, was the strongest method also when starting from the original Dymola configuration. This suggests that the formulated problem may not be well suited for metaheuristic algorithms. One likely reason is that the high noise level, combined with the very limited budget, leaves insufficient room for the sophisticated balance between exploration and exploitation that metaheuristics rely on. Instead, the setting favors methods that make quick exploratory moves.

However, noise alone cannot fully explain the poor performance of the metaheuristics. In the Rastrigin problem, with noise levels calibrated to match those of the pulp plant model, the algorithms showed much more consistent improvement and larger average movements. This indicates that noise is not the primary issue, but rather other characteristics of the model.

The Rastrigin function was not constructed to represent the true landscape of the pulp plant, partly because assessing the real landscape would require extensive experimentation. However, there are some substantial differences between the two problems. The Rastrigin model is constructed such that large parts of the search space contain relatively low values, with fewer regions near the optimum. This can be seen clearly in a 2D representation and is even more pronounced in 8D. The pulp plant problem likely exhibits the opposite behavior. The lowest values in the search space correspond to simulations where parts of the system fail to operate entirely. Closer to the best-performing configurations, the landscape appears relatively flat, with expected improvements that are small compared to the noise level.

This flatness would explain the difficulty in improving upon the best LHS point. The algorithms might have produced larger movement from this starting point if more than the maximum of three replications per comparison had been allowed. However, even if this led to slightly more reliable decisions, it is arguably better to allow some movement under uncertainty than to remain indifferent due to statistically small differences.

Overall, the characteristics of the formulated problem appear to disfavor metaheuristics. While the problem is complex and hard to interpret, metaheuristics would require more simulations to be competitive. The primary reason for the poor performance is thus the long simulation time. Metaheuristics typically show their greatest strength when individual evaluations are inexpensive enough to allow many iterations, while the search space is large enough that systematic approaches such as grid search are infeasible.

The high noise levels also contribute to making the problem less suitable for metaheuristics. In one sense, high noise simply increases the effective cost of the problem, since sufficient replications can always reduce uncertainty. However, running the model for many years to approximate long-term production rates does not appear to be the most efficient approach. The Dymola model is designed to represent many

operational aspects of the plant, not merely the long-term average economic output. It is likely that a more specialized model could be constructed to estimate production and consumption rates directly, without simulating the full temporal dynamics over long horizons.

In summary, the results suggest that the limitation lies less in the algorithms themselves and more in the interaction between noise level, budget constraints, and landscape structure. Under a larger budget or a more evaluation-efficient model, metaheuristic methods may well show stronger relative performance.

6
Conclusion

This work has evaluated the performance of metaheuristic optimization methods for maximizing expected net present value in a noisy simulation model of an industrial pulp plant. While all algorithms improved through tuning and were able to outperform the original baseline configuration, none surpassed the best solution obtained by Latin Hypercube Sampling within the available budget. When initialized near promising regions, only coordinate search achieved marginal further improvement.
The results indicate that the main challenges arise from high noise levels and long simulation times and an apparently flat objective landscape near good solutions. Under these conditions, the advantages of metaheuristics are reduced, and simpler local search methods may perform equally well or better.
Overall, the study shows that the effectiveness of metaheuristic optimization depends strongly on problem structure.

Bibliography

[1] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Wiley, 2009.

[2] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury, "Simulation optimization: a review of algorithms and applications," *Annals of Operations Research*, vol. 240, no. 1, pp. 351–380, 2016.

[3] P. Rakshit, A. Konar, and S. Das, "Noisy evolutionary optimization algorithms—a comprehensive survey," *Swarm and Evolutionary Computation*, vol. 33, pp. 18–45, 2017.

[4] K. Miettinen, *Nonlinear Multiobjective Optimization*. Boston, MA: Kluwer Academic Publishers, 1999.

[5] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[6] M. Abdel-Basset, G. Manogaran, S. Mirjalili, and M. Irfan, "A comprehensive review of nature-inspired metaheuristic algorithms for engineering applications: Classification, applications, and future trends," *Expert Systems with Applications*, vol. 92, 2018.

[7] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[8] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[9] I. J. Fister, X.-S. Yang, I. Fister, and J. Brest, "A brief review of nature-inspired algorithms for optimization," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 3, pp. 267–281, 2013.

[10] C. Aranha, C. L. C. Villalón, F. Campelo, M. Dorigo, R. Ruiz, M. Sevaux, K. Sörensen, and T. Stützle, "Metaphor-based metaheuristics, a call for action: the elephant in the room," *Swarm Intelligence*, vol. 16, pp. 1–6, 2022.

[11] N. Hansen, A. Auger, D. Brockhoff, T. Tušar, and D. Tušar, "Coco: A platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, vol. 36, no. 1, pp. 114–144, 2016.

[12] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Díaz, "Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization," Nanyang Technological University, Tech. Rep., 2013.

[13] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. Prentice Hall, 2010.

[14] N. Hansen, "The cma evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*, J. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, Eds. Springer, 2006, pp. 75–102.

[15] A. Isaksson, B. Lantz, and H. Löfsten, *Industriell ekonomi – Grundläggande ekonomisk analys*, 2nd ed. Studentlitteratur AB, 2018.

[16] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[17] L. A. Rastrigin, "Systems of extremal control," in *Theoretical Foundations of Engineering Cybernetics Series*. Moscow: Nauka, 1974.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[19] 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, 1997.

[20] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.

[21] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS