AI Enhanced Facility Layout Optimization

Supervisor: Dr. F.A. Bukhsh
Critical Observer: MSc. N. Bouali
Author: Yigit Bezek

University of Twente
Creative Technology

Abstract

Throughout the last 20 years, the increased demand of customised products and services has affected many industries. Particularly manufacturing industry has gone through a transformation driven by Industry 4.0 with the demand of more efficient and adaptable production systems and workflows. Despite the technological advancements, many of the manufacturing companies still rely on manual, experience based approaches for facility layout planning. This results in suboptimal designs that impacts the productivity, energy consumption and operational costs. This challenge affected the competitive corrugated cardboard industry specifically due to complex material flows, diverse product mixes and multiple processing stages creating a complex optimisation problem that exceeds human cognitive capabilities.

This experimental research addresses the facility layout optimization problem at a corrugated cardboard manufacturer where the current manual trial and error based approach requires weeks and months of iterative design cycles leading to inefficient layouts with unpredictable performance. To address these challenges this paper investigates the following research question: how AI enhanced generative design can be leveraged to optimize throughput in custom cardboard facility layouts while integrating with existing design workflows and simulation tools. With this research question a hybrid genetic algorithm based optimization system that combines NSGA-II with two tier evaluation pipeline was developed. This system employs process level gene encoding to reduce the search space, integrating A* pathfinding for spatial validation and using both surrogate model and discrete event simulation for fitness assessment. This approach aims to generate multiple Pareto optimal layouts balancing infrastructure costs (total conveyor length) and operational efficiency (throughput and path ratio).

Experimental results and evaluation demonstrates that this developed system serves primarily as a proof-of-concept for AI-enhanced layout optimisation rather than a deployable industrial solution. The results show a reduction in the conveyor length and improvements in throughput optimisation. However the evaluation remains constrained by several reasons: the system was tested against a "golden layout" only reference layout provided by industrial partner for comparison purposes and due to lack of complexity of the system the real factory layout's could not be used in comparison. Component variety is limited to seven transport systems and extensive parameter tuning is required. The future development should focus on increasing the complexity by integrating all the transport system component that it can be validated against a real full scaled factory layout, as scaling with the NSGA-III so that it can be customised with more objectives and not only two and integrating with real time operational data.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Faiza Bukhsh and my company supervisor Can Olmezoglu for her guidance and support throughout this research project. I am also thankful to the team at Fraunhofer Innovation Platform for Advanced Manufacturing at the University of Twente (FIP-AM@UT) for providing me with the opportunity to work on this project, and to Van den Bos CM and Van den Bos Robotics for their collaboration.

Contents

List of Figures

1 Introduction

Manufacturing industries are undergoing a radical transformation which is also known as Industry 4.0. This is a revolutionary age for manufacturing industry. It envisions a future in which manufacturing systems are modular and intelligent, enabling products to direct their own production sequence while maintaining cost effectiveness [4]. Industry 4.0 is a promise for manufacturing sector aiming to enhance their efficiency, productivity and quality. In reality the manufacturers are still holding on to expensive trial and error based approach, resulting in repetitive extended design cycles and suboptimal performance and making design stages labor intensive and highly dependent on expert driven insights [5]. Companies need to advance their systems and workflows with digital optimizations, data driven applications and technological automations in order to keep up with the competitive industry [1].

A key milestone within this revolutionary transformation is the push to optimize in material handling. Material handling is defined as the movement, storage, control, and protection of materials/goods/products throughout their entire product life cycles [2].

Efficient material handling is key to increase the throughput and lower the operational costs aligning closely with the goals of Industry 4.0 to enhance productivity and quality. When it is done poorly this can lead to great problems such as high operational costs, extensive design times and throughput bottlenecks in system designs.

This problem is especially relevant to custom transport systems, which also serve specialized workflows in industries such as corrugated cardboard manufacturing. Inefficient layout in these environments can cause higher operational costs, significant throughput bottlenecks, and difficulty predicting the true cost of design changes or expansions.

1.1 Problem statement

This subsection introduces the facility layout problem at Van den Bos, formulates the main and sub research questions and outlines how this paper follows CRISP-DM method to answer them.

This project addresses such issues for Van den Bos CM and Van den Bos Robotics. Van den Bos helps companies in the cardboard industry by providing them engineering processes and solutions through the state-of-art transport system and software for corrugated cardboard industry. Currently Van den Bos relies on expert driven, manual trial and error based approach. The status quo approach for configuring system layouts suffers from excessive time-consuming cycles of trial and error (the process can take up to one month until the initial layout plan is drafted), repetitive proposals, excessive energy consumption, extended production queues and

inefficient layout design leading to reduced throughput and wasted resources and money.

In colloboration with the University of Twente, Fraunhofer Innovation Platform (FIP-AM@UT) is developing a simulation environment for Advanced Manufacturing; this project aims to introduce AI enhanced generative design to automate the layout optimization process.

1.2 Research questions

Therefore following research question is defined:

Main-RQ: How can AI enhanced generative design be leveraged to optimize throughput in custom cardboard facility layouts, specifically integrating the design workflows and simulation tools?

To operationalize this question, the study will address three sub research questions. These sub questions explore the state-of-the-art in facility layout optimization, data and parameter requirements (e.g., machine speeds, buffer sizes), and how best to validate improvements in throughput.

Sub-RQ 1: To what extent can computational or algorithmic approaches (e.g., genetic algorithms, reinforcement learning, other metaheuristics) optimize the throughput in facility layout problems?

Sub-RQ 2: Which layout constraints, machine parameters, and throughput metrics are most critical for modeling in an AI enhanced generative design?

Sub-RQ 3: How will optimized layouts be validated and assessed in terms of throughput compared to the status quo?

1.3 Research Approach & Thesis Road-map

This research adopts the Cross Industry Standard Process for Data Mining (CRISP-DM). CRISP-DM is a well established framework guiding systematic and predictive analysis for data driven projects. The agile approach of CRISP-DM will ensure the flexibility and adaptability of implementation for new challenges and findings in the process of this research instead of a rigid waterfall method. The methodology consists of six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment [12]. The agile approach of the CRISP-DM is depicted in Figure 1.

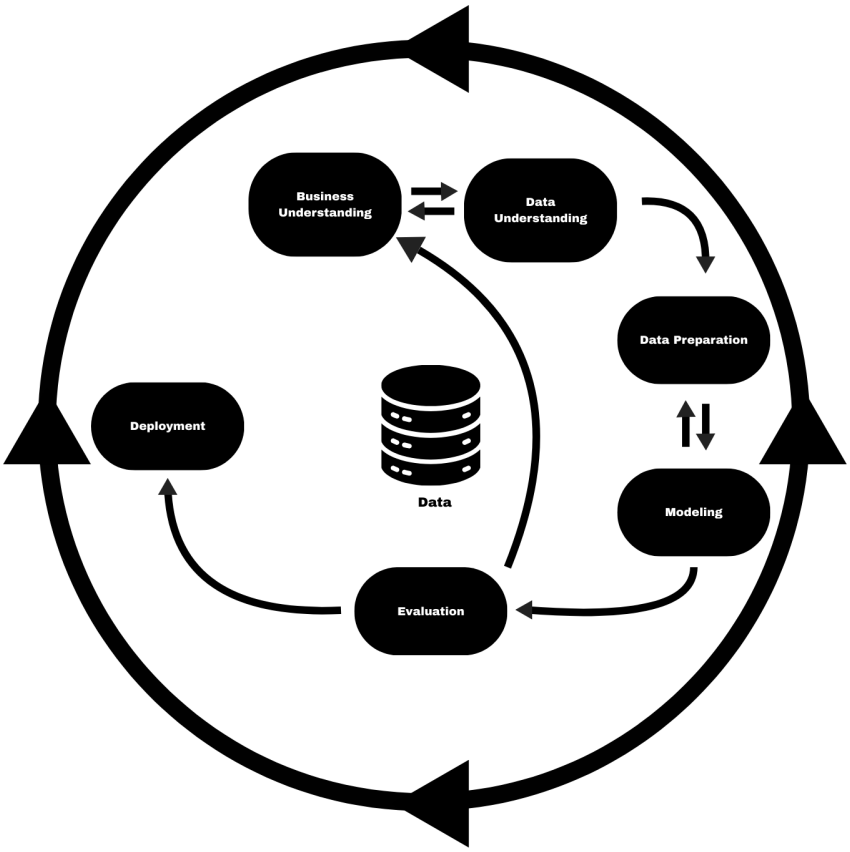

Figure 1: The agile CRISP-DM framework

Below, each step is discussed in terms of clarifying their relevance and implementation.

Business Understanding:

The first phase is devoted to understand the project objectives and requirements from the perspective of Van den Bos's facility layout optimization challenge. Here the facility layout problems, business objectives and success criteria are defined primarily in chapters one and two as a foundational understanding of this research.

Data Understanding & Preparation:

In this phase exploration and assessment of available facility layouts, machine performance metrics, constraints and operational flows at Van den Bos will be performed. The simulation data from Fraunhofer Innovation Platform will be analyzed for integration of proposed generative design. Detailing the models and preparation strategies chapter four is going to include JSON schemas for layouts and excel data structures for synthetic data imitation of production orders.

Modeling:

The selection of appropriate algorithm and the implementation is going to be discussed in chapter four and five. Where the chapter four is describing and detailing the system design choices in theory and the next chapter realization, detailing the actual practical implementation

from the theory level to the practical level including the system realizations with iterative agile approach in the process of implementation.

Evaluation:

The developed model in the previous phase will be evaluated and assessed to determine the effectiveness in addressing the facility layout problem of Van den Bos. Metrics such as the throughput and utilization will be used for the comparison of experimental results of the GA optimized solutions against the baseline in chapter six.

Deployment:

Even though the traditional CRISP-DM framework involves deployment, this paper will focus on developing and validating the concept and method rather than a fully working market ready operational system. The output of this thesis won't be deployable but chapter eight will provide a road map and steps for future deployment and integration of proposed solution to Van den Bos's workflow.

This chapter introduced the facility layout problem of Van den Bos articulating the initial research question of this research and setting a clear plan for agile and iterative project plan adopting the CRISP-DM framework. By mapping the phases of this framework with each thesis chapter it is ensured that business objectives, data assets and development choices will remain traceable throughout this paper. The next chapter is going to deepen the understanding of the business and data understanding phases by analyzing the Van den Bos's current manual approach and going through available state-of-the-art algorithms that can automate and optimize layout generation. This analysis will provide evidence base for selecting the most suitable optimization approach and refining the research questions accordingly.

2 Background Research

Building on the industrial problem identified and CRISP-DM roadmap presented in chapter one, business and data understanding foundations are going to be developed. This will be analyzed through two complementary scopes; business and science, then will be combined to form an overall understanding. First, a dissection and analysis of the status quo will be performed to understand the specific challenges of Van den Bos. Subsequently, this chapter will systematically explore literature that will map the state-of-the-art optimization techniques and algorithms for the facility layout problem. Finally combining and assessing the strengths and weaknesses of both scopes identification of research gap and justification of selected approach will be presented before moving on with latter chapters.

2.1 Business Lens: Current Manual Approach at Van den Bos

In order to understand the specific challenges that Van den Bos faces with, their existing workflow will be studied in detail. A critical part of Van den Bos's service involves planning, configuring and creating facility layouts into existing (brownfield) or new (greenfield) production sites. The configuration and planning of these facility layouts currently involves many iterative cycles between and within internal and external stakeholders as it is depicted below in Figure 2.

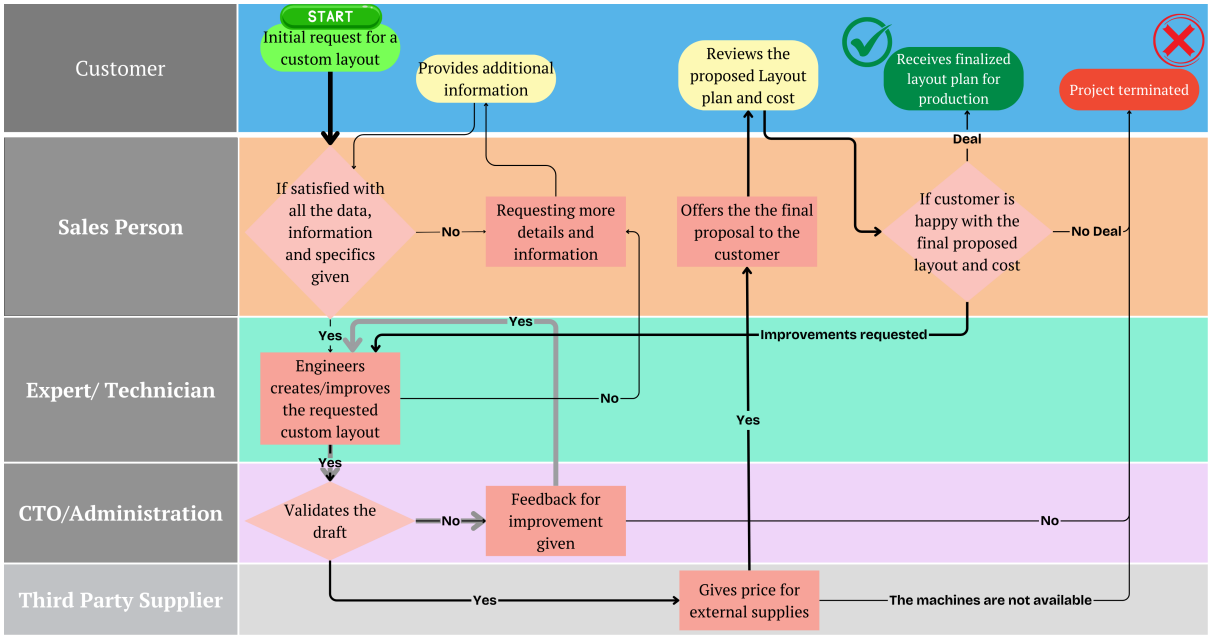

Figure 2: Status-Quo Van den Bos

Workflow synopsis: This flow portrays the planning and configuration of a layout, including internal (Engineering Department, Sales Department and Administrative Department) and external stakeholders (Third Party Suppliers and Customer). The cycle begins with a Customer requesting a planning solution. A Salesperson then receives this inquiry and assesses it based on its complexity and whether sufficient information is given. The inquiry is then routed to the Engineering Department or back to the Customer for more information. Once the information is transferred to the Engineering Department, here, a similar process of requesting specific information may begin here which will result in the inquiry being sent back to the Salesperson to Customer, and so on, until no information is needed. Afterwards, the Engineering Department creates an initial draft and transfers this draft to Administration, usually the CTO. The CTO reviews the layout and then either (1) approves it, (2) sends it back to the Engineering Department for improvement, or (3) cancels the project. When the CTO approves the layout,

the layout is forwarded to their Third Party Supplier for a cost estimate. After receiving a cost estimate, the Salesperson offers the final layout to the Customer. After reviewing the layout plan and pricing, the Customer will either (1) sign the contract (approval is quite rare in the first draft), (2) cancel the project, or (3) request more changes to the layout. The sales person routes the requested revisions and changes back to Engineering Department and the cycle begins again. This cycle iterates until the Customer approves and signs the contract or the project is terminated.

While this multi stakeholder workflow leverages valuable expert knowledge, it is associated with several inefficiencies and challenges. These challenges directly impact the effectiveness and efficiency of the layout planning.

Observed bottlenecks: Below in Figure 3 shows a visualization summary of the observed bottlenecks of the Van den Bos's status quo. The excessive time elapsed between the first inquiry

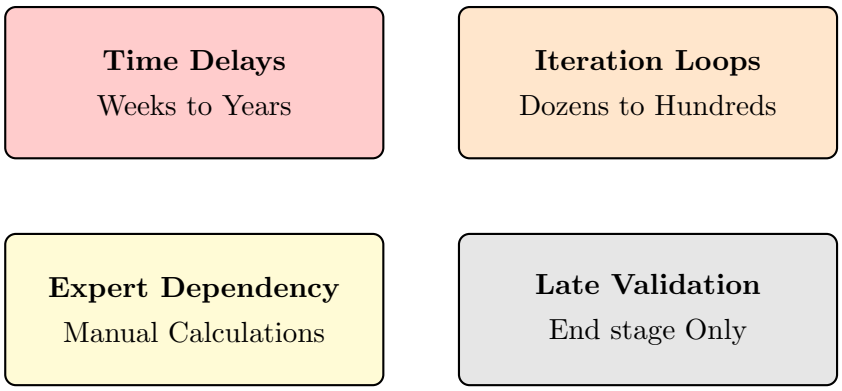

Figure 3: Key bottlenecks in Van den Bos's current facility layout planning process

and the approved final layout design can range from a month to multiple years. Another inefficiency leading to higher quotation times is the loops within the workflow. Given the dynamic nature of the industry and the number of internal and external stakeholders, there exists an excessive amount of review loops in the workflow. Some of which can range from a dozen to hundreds of iterations. The fundamental reason for these iterative loops in the status quo is the computational complexity of a layout design creation. This process in the status quo heavily depends on expert-driven insight. This complexity makes it challenging for experts to perform and verify crucial calculations. This leads to errors, miscalculations, and oversight of important factors that then require review cycles for correction. Another bottleneck is the lack of early performance validation. Fraunhofer Innovation platform has developed a simulation tool that aids the visualization; however, it is used in late in the process and primarily used to validate a design that is already created manually by an expert. It cannot generate or suggest

optimization on top of the created layouts, hence many sub-optimal drafts reach the customer unchanged.

These challenges have a direct impact on the business, resulting in higher proposal costs, slow response to customer requests, and unpredictable cost and throughput of delivered proposals.

Implications for the optimization project Diving in depth to the specific business case provides critical insights which are critical for the development of a effective optimizing solution. Rather than only optimizing their current system the proposed system should address each bottleneck identified in previous subsection while preserving the human tacit expert knowledge. In order to achieve that three primary requirements have been identified:

Data requirements:

The optimizer must accept heterogeneous inputs (product mix, machine specs, hall geometry). For this heterogeneous input there must be a dedicated pipeline for data preparation (CRISP-DM).

User interaction:

Sales and Engineering Departments must stay in the loop. The tool must show layouts that can balance efficiency for multiple objectives within one layout. The choices of these Pareto optimal (efficiency within the multi-objective context) layout must be explained to the expert, enabling expert judgment [16].

Evaluation metrics: Success will be evaluated through multiple indicators that evaluates both the operational efficiency and the business value. These includes throughput improvements, infrastructure cost reduction (through minimizing the conveyor length), optimizing the routing efficiency, improved resource utilization and consistent generation of feasible layouts(satisfying the spatial and operational requirements and constraints). These metrics are going to be detailed in the latter chapter three with how much improvement the system is targeting or how are these evaluation metrics are satisfied.

This business portrait highlights the need to shift from a fragmented, time consuming process to a data driven, semi automated approach that can generate and validate optimized layouts production. The next section therefore adopts a scientific lens reviewing the state-of-the-art optimization algorithms and identifying the technique best suited to closing the gaps identified here.

2.2 Scientific Lens

Building on the detailed analysis of Van den Bos's business case and its challenges, this section will adopt a scientific lens to explore computational and algorithmic approaches for facility

layout optimization. The objective is to systematically review state-of-the-art methods, identify their strengths and weaknesses and justify the selection of the most suitable approach for a proposed AI enhanced generative design solution.

To identify relevant state-of-the-art optimization techniques for facility layout optimization, a systematic literature search was conducted across multiple databases including ScienceDirect, IEEE Xplore, Springer, Web of Science, Google Scholar and ACM Digital Library. Keywords used in various combinations included: "facility layout optimization", "facility layout design", "genetic algorithm", "manufacturing systems", "flexible manufacturing systems", "simulated annealing", "tabu search", "hybrid algorithms", "hybrid optimization", "meta-heuristics", "dynamic facility layout", "unequal area facility layout", "multi-objective"/"multi-population", "digital twin", "reinforcement learning", "material handling", "corrugated industry" and "Industry 4.0". The search primarily focused on peer-reviewed journal articles, Conference proceedings, Books (foundational concepts) and online resources (for industry standards and definitions) published from the last 20 to 25 years, with specific attention paid to studies that address multi objective optimization, complex constraints and integration with simulation manufacturing contexts.

Exclusion Criteria:

- Purely theoretical papers
- Single-objective optimization studies
- Static layout-only papers
- Publications without quantifiable performance metrics
- Papers lacking comparative analysis with established methods or benchmarks
- Publications without consideration of material flow patterns
- Papers lacking integration with other systems

2.2.1 Facility layout Optimization Methods: A Review

Optimization methodologies for the Facility Layout Problem (FLP) are broadly categorized in current literature. These different search approaches can be grouped into traditional heuristics, hybrid algorithms and modern data driven approaches. Understanding their characteristics and trade-offs is crucial for selecting a suitable approach for the complex and specific needs of Van den Bos.

Traditional Heuristics Traditional Heuristics are commonly used methodology where a partial search algorithm is used within an optimization problem [8]. The most widely used traditional heuristics are Genetic Algorithms (GA), Simulated Annealing (SA), and Tabu Search (TS).

GAs are based on Charles Darwin's evolutionary theory which adopts the concept of survival of the fittest approach in the context of optimization [9]. In the paper by Rajasekharan et al. (1998) [10] the authors propose a genetic algorithm for a flexible manufacturing systems. Multiple metrics were successfully integrated within the algorithm despite increasing the problem complexity. The results show that GAs are effective in computational time reduction and generating optimal layouts for complex systems. GAs versatility in order to explore large solution sets makes it a foundational approach in FLP.

Simulated Annealing (SA) and Tabu Search (TS) are also well established examples of traditional metaheuristics. While they also adopt a search algorithm, they differ from the GAs evolutionary strategy. SA is an optimization method trying to find a global minimum of a cost function, mimicking the annealing of molten down crystalline lattice that is cooling down [11]. This resembles the cold and hot game but it settles when cold(the algorithm converging to local solutions while the temperature decreases and exhibiting more exploratory behavior on higher temperatures), and wanders away when it's hot. McKendall et al.(2006) has proposed two SAs in Dynamic Floor Layout Problems (DFLP). The first variant utilized pair wise exchange to explore neighbor nodes. The latter approach implemented a "look ahead/look back" strategy. This strategy allowed two facilities to work in a cycle so that less computational cost would be spent. As a result, both approaches improved the layouts that were generated compared to other computational algorithms [13].

Tabu Search (TS) is another classical meta-heuristic. Glover and Laguna (1998) describe it as a framework that equips a basic local search with memory rules so it can move past locally optimal traps and probe wider regions of the solution space [14]. Building on this idea, Liang and Chao (2008) introduced a parallel, multi tabu strategy for facility-layout optimization that balances diversification (pushing the search into unexplored areas), and intensification (examining promising regions in depth). Their algorithm also uses tailored neighborhood moves such as backward/forward shifts, left/right shifts, and adjacent department swaps. Experiments suggest this method surpasses competing approaches (including GA, SA and neural network variants) by roughly four to five percent in cost objectives [41].

Traditional heuristics provide robust frameworks in FLP but each has limitations. GAs face scalability issues with very large and complex layouts, SAs performance are sensitive to

parameter tuning and TSs can be trapped in a complex local optima if memory structures are not designed carefully.

Hybrid Approaches In this section hybrid algorithms are reviewed and analyzed with comparison to traditional heuristics. To mitigate their individual setbacks researchers have combined two or more traditional heuristic's, making a Hybrid approach.

Mir and Imam (2001) developed a hybrid approach from a standard Simulated Annealing method. This was achieved by fine tuning parameters such as: temperature schedule, acceptance rules, and local improvement steps which is a good fit for layout specific constraints [15]. With these parameter tweaks, these traditional heuristics became a hybrid variant. The experiments showed lower costs and higher throughput compared to an unmodified approach of SA.

Besbes et al. (2020) developed a unique approach by combining GA and A* path finding algorithm. This unique combination proposed is specifically relevant with the facility layout problems that involve transport cars within the network design (VDB, which manufactures transport cars and conveyors and are a related fit to this problem set). This hybrid approach iteratively refines a layout by shortest path and obstacle calculations, proving the effectiveness of this approach in constrained layouts [23].

Pourvaziri and Naderi (2014) proposed a hybrid approach integrating multi population GA with SA achieving better solution set quality while improving computational efficiency. Where multi population GA increased the global exploration and the use of SA enhanced the local refinement [18].

Overall hybrid approaches have demonstrated enhanced capability in handling complex constraints and multi objectives facility layout problems.

Modern Approaches The shift towards Industry 4.0 with the increased computational power and data availability manufacturing industry began to adopt more adaptive and data driven approaches for FLP. Where methods such as Machine Learning (ML) and Reinforcement Learning (RL) are becoming more and more popular. These methods aim for layouts that can adapt to dynamic conditions and changing environments.

One recent example has been explored by Chalak Qazani and Parvaz (2022). Researchers have looked into three ML architectures: Long Short Term Memory networks(LSTM) which is good at capturing sequential patterns and dependencies for layout evaluation, Multilayer Perceptrons (MLPs) which are neural networks that are capable in learning non linear, complex relationships between the layout parameters and performance metrics and Adaptive Neuro Fuzzy Inference Systems (ANFIS) this approach combines the learning capabilities of a neural

network system with interpretability of non binary fuzzy logic systems. The key differences of these three mL architecture is their unique approach: LSTMs handling time series data more effectively, MLPs providing pattern recognition for static relationships while ANFISs offering human interpretable heuristics next to the learning capabilities. These three approaches were combined with traditional heuristics, evaluating the optimization of component placement in manufacturing industry. The simulations of finite elements supplied the training data and the learners were combined with the search heuristics, demonstrating improvements in accuracy and optimization efficiency [20].

Klar et al. (2021) demonstrated the potential of RL by implementing Double Deep Q Learning agent for automated layout planning. This agent used to iteratively place equipment on a grid, receive transport time feedback meanwhile updating its policy system, it gradually reached layouts that had shorter travel distances in between equipment. Their study indicates that RL can handle the combinatorial complexity of layout problems and yield high performing optimized solutions with the use of continuous learning policy [21].

Choi et al. (2025) proposed an integrated, simulation based framework that combines linear programming (it is a mathematical optimization technique to achieve the best outcome in linear problems) for line balancing with discrete event simulation for cell and buffer sizing. This was done in a digital-twin environment (an identical digital replica of the physical manufacturing facility enabling real time monitoring, testing and validation) for the final layout validation. The combinatorial approach showed reduction in overall design time and boosted manufacturing efficiency. Albeit substantial computational resources were required [22].

Modern approaches offer significant potential for dynamic adaptation and potentially better levels of automation given flexible adaptability. However they require substantial amount of good quality training data, computational power and costly development processes. This may cause feasibility challenges in certain industry contexts.

2.2.2 Research gap and Approach Selection Justification

The reviewed literature provides a wide landscape of algorithmic approaches for FLP. Each of these approaches vary within their suitability relevant to the challenges that identified in Business Lens of the status quo for VDB.

Handling Complex Constrains: Methods that optimizing layouts for UA-FLP [28, 37] and hybrid approaches that integrates path finding algorithms [23] are highly relevant for status quo. Van den Bos's needs to place machines of different dimensions within complex building geometries that may also include constraints (aisle, pillar or any type of construction obstacle).

Multi Objective Optimization: Studies demonstrating multi objective GAs [34, **?**] and simulation based evaluation [27] align with the balancing of the multi objective optimization need in the context of manufacturing industry. Within the status quo, Van den Bos need to optimize multiple objectives ; throughput, cost, energy usage, and especially ergonomic/safety factors, since each piece of industrial machinery has a safety distance that can be dangerous if breached. Moreover, they must maintain a good balance between these objectives.

Dynamic Demands: Approaches for DFLP [13, 18, 30] are crucial for Van den Bos. The dynamic nature of the industry brings the need to adapt layouts for continuously changing customer demands and product mixes over time.

Integration with Simulation: The approaches that integrates GAs with discrete event simulations [27] and methods that uses visualization tools [29] are highly relevant for the status quo. VDB needs a way of integrating the existing simulation tool of Fraunhofer. This is crucial for accurate layout performance evaluation which is going to aid in the evaluation of a layout for the proposed solution.

Justification of GA based hybrid variant approach selection: Despite the wide variety of existing methods found, a key gap remains, which is how to provide a practical, feasible and integrative AI enhanced generative design solution tailored specifically to the context of custom corrugated cardboard facility layouts. Even though the modern ML/RL approaches may hold the highest promise for future dynamic adaption and best optimization possible, their implementation may be the bottleneck. Implementation of modern approaches require resources, data availability, high computational power, and long term investment. Traditional heuristics, while foundational and proficient at low complexity problem optimization, may struggle with problems of a complex or multi objective nature, without enhancements. Hybrid approaches offer a balance, combining the robustness of the traditional heuristics with strategies for context specific challenges.

Given the project constraints, the need for an initial, feasible solution and the specific complexities of VDB workflow analysis (diverse machine sizes, complex flow types, multi-objective goals); Genetic Algorithms with a hybrid framework, leveraging simulations and fine tuning parameters emerge as the most suitable and practical approach.

1. Versatility,

2. The proven ability of handling multi objective problems and constraints,

3. The relative tractability and feasibility compared to data and computational power hungry ML/RL approaches for an initial implementation
4. Ease of compatibility with simulation
5. Ease of early validation which is crucial for status quo

The technical arguments from current literature demonstrate the sound basis of GA in the context of FLP and the discussions with the client and supervisor have confirmed this direction. This selection is aligning with the Sub-RQ 1 which posits; to what extent can computational or algorithmic approaches optimise throughput in FLP? The research shows that GA's are well equipped to handle the complex nature and relevant objectives. Providing a strong foundation to tackle the status quo of VDB.

2.2.3 Conclusion of Scientific Lens

In conclusion, the literature offered a rich spectrum of optimization techniques for facility layout problems. Traditional heuristics represented the foundational and robust approaches as the modern methods represented the cutting edge applications. The analysis of the literature and existing work confirmed the selection of GA's. Especially when integrated with simulation tools for performance evaluation it may address the complex custom layout challenges faced by VDB. Based on the scientific review and feasibility considerations, a GA based approach was selected as the foundation for the proposed AI enhanced generative design solution. The following chapters will detail the development and the implementation of this selected approach. It will focus on how it leverages and handles data, constraints and parameters (addressing Sub-RQ 2). Later chapters will then explore how the resulting layouts will be evaluated against the status quo in terms of throughput (addressing Sub-RQ 3).

3 Requirements

The purpose of this chapter is to translate the overarching goal of this research into a concrete set of engineering requirements. These requirements are crucial for the design, implementation and evaluation of this thesis. Requirements will state what the prototype must do (functional), how well it must do it (non functional) and how success will be measured (KPIs). Each design choice that will be presented later in this chapter is grounded in these requirements which are categorized using the MoSCoW prioritization framework. The formulation of these requirements are compiled by stakeholder meetings with the Van den Bos, gap analysis of state of the art

facility layout research (section 2.4) and iterations made during the development of the MVP, which will be further discussed in the realization chapter.

3.1 Functional Requirements

Functional requirements specify the essential features and behaviors the system must perform. In the table below, each requirement is tagged with its corresponding system component, along with an explanation of its relevance and its assigned priority level.

Table 1: Functional Requirements

Field	**Requirement**	**Rationale**	**Priority**
Layout Generation	Given a seed layout (initial inquiry of the facility that has the specification of the facility with corrugator, converter and End of Line) the system must generate a complete factory layout.	Replaces the long quotation time of manual trial and error based approach which is currently done by engineers.	**Must**
Connectivity	Every machine's output port must reach the next processing step, that starts with corrugator and ending at the EOL, via a directed path of conveyors, turntables or transfer cars.	This ensures that all the stacks are completing the full life cycle and the layouts are ready for full simulation evaluation.	**Must**
Spatial Validation	Components can not overlap obstacles, pillars or with other components with ensuring safety precautions of machine clearance $\geq$ 500 mm around every path.	Mandatory for real world resembling realistic layouts.	**Must**

Continued on next page

Table 1 – continued from previous page

Field	Requirement	Rationale	Priority
GA Interface	The GA must encode high priority process decisions (the machine order that a stack should take, routing preferences for the flow of stacks and buffer strategy) rather than coordinates.	Reduces the combinatorial search space and allows spatial validation modules to guarantee geometric feasibility.	**Must**
Evaluation	Every candidate layout should pass a fast surrogate check and a minority of an elite subset of them must run through a full discrete event simulation for throughput and utilization metrics.	Balances speed and fidelity. Where surrogate check covers the low fidelity in short amount of time dealing with the geometrical problems and discrete event simulator covers the high fidelity by tracking the full cycle of production of each material that is going to be produced and processed.	**Should**
Data Export	The best layouts shall be written to JSON plus a PNG schematic.	Enables review by engineers and management.	**Must**
Simulation Integration	The system must be able to interface with the simulation through the JSON/Excel formats.	Ensures the integration of the whole system with simulation.	**Must**
Multi-objective Support	The system must be able handle and optimize at least two objectives simultaneously.	Ensures the multi-objective optimization capability of the system.	**Must**
Order Flexibility	The system should be able handle complex and big sizes of orders.	Increases the system's relevance to real life simulations where the complexity and numbers of orders can be mixed (different dimensions and flute types of cardboard).	**Should**

3.2 Non-functional Requirements

Non functional requirements define the quality of the system behavior and the quality of functions that support the system. This ensures that the prototype is not just functional but also relevant and practical for deployment.

Table 2: Non-functional Requirements

Field	**Requirement**	**Priority**
System Runtime	The system must deliver at least one feasible layout within 15 minutes for the mvp (a pilot that can operate with min of 4 machines) and fully developed layout within and hour	**Must**
Extensibility	Adding a new transport module (e.g. AGV, overhead crane, robot arm) may require no longer than 2 developer days and no GA redesign.	**May**
Reproducibility	With fixed random seed the GA must reproduce objective values within ±1% variance. error = $\frac{\lvert \text{Objective}_{\text{run1}} - \text{Objective}_{\text{run2}} \rvert}{\text{Objective}_{\text{run1}}}$	**Must**
Transparency	Every optimization run must generate diagnostic logs (component utilization, stack journeys) and progress plots.	**Must**

3.3 Success Validation metrics (KPIs)

This subsections identifies and quantifies the systems success in optimizing the facility layouts compared to VDB's status quo. The selected metrics are measuring the improvements within specific challenges identified in previous chapter.

Table 3: Key Performance Indicators

Metric	**Goal**	**Rationale**
Total conveyor length (mm)	Minimize	Crucial for material-handling cost.
Path-ratio (actual / Manhattan)	Minimize	Important for routing efficiency (operational cost).
Machine utilization (%)	Maximize	Crucial for identifying the bottlenecks and inefficient layouts.Utilization $= \frac{\text{Active Time}_{\text{MachineA}}}{\text{Active Time}_{\text{MachineA}} + \text{Idle Time}_{\text{MachineA}}}$
Throughput (stacks / hour)	Maintain or improve by $\geq 5\%$ the baseline throughput.	Crucial for evaluating whether the system remains the throughput of the companies design or improves it with its own layout.
Feasibility rate per generation	Feasibility is reached $> 90\%$ by tenth generation.	Crucial for keeping the search of GA focused on feasible layouts. Saves simulation time.
Hyper-volume	Starting with a rise and plateau	Hyper Volume provides a single score of Pareto front which is crucial in evaluating how well each layout is balancing given multi objectives.

This chapter serves the role as a foundation for upcoming design choices and ensuring these choices will be relevant to the specific challenges of VDB. The next chapter will detail the selected design choices into technical architectures, data models and algorithmic approaches in order to fulfill the identified requirements and success metrics.

4 Methodology

This chapter presents the methodology for the development and implementation of the proposed solution: Genetic Algorithm Based generative design system addressing the optimization of facility layouts.

Building on the problem context and literature review presented in earlier chapters, this section outlines:

- Section 4.1 The modular system architecture.
- Section 4.2 Data models and preparation strategies (addressing Sub-RQ 2: Identifying the

critical layout constraints, machine parameters, and throughput metrics).

- Section 4.3 The discrete event simulation environment.
- Section 4.4 The GA configuration (4.4.1), path finding and spatial validation with A* search algorithm,(4.4.2) two tier evaluation(4.4.3), NSGA-II parameters(4.4.4), Fitness Function(4.4.5), Genetic Operators(4.4.6) and Constraint handling & validation(4.4.7).
- Section 4.5 The integrated workflow of the overall optimization process.

Chapters one and two establish the business understanding, including project objectives, scope, and success criteria. This chapter covers the Data understanding and preparation phase of CRISP-DM framework by elaborating different types of important data for the GA system, the UI facility layout configurator of FIP, order (synthetic data) generator tool (which is used to be able to run the simulation tool) and the simulation tool. It details their structure, formats and the dependencies on each other (see section 4.2).

This section also describes the gathering cleaning, arranging, transforming and integrating the raw data into formats that is suitable for both the simulation and genetic algorithm. In order to achieve this defining the JSON schema for layouts and parsing process of the Excel files that has the order data is a crucial steps which is discussed in section 4.2.

Also the evaluation phase of CRISP-DM is partially covered by with an early stage evaluation mechanism that is adopted by the proposed solution. This will be talked in fitness function of GA serving as an internal evaluation loop and the two tier simulation integration that is assessing the candidates layouts based on chosen and defined metrics (see Section 4.4.3 and 4.4.5).

4.1 System Architecture Overview

The proposed AI enhanced generative design system is built upon a modular architecture to facilitate iterative development, testing, integration and future enhancements. The iterative cycle is orchestrating design cycles including design, simulation and optimization allowing the discovery and enhancements of the suboptimal layout designs. The system is comprised into four interconnected components:

1. Layout Configurator (LC)
2. Order generator (OG)
3. Discrete event simulator (DES)

4. Genetic Algorithm (GA)

Each component plays a crucial role in the overall system workflow, which can be seen below in Figure 4:

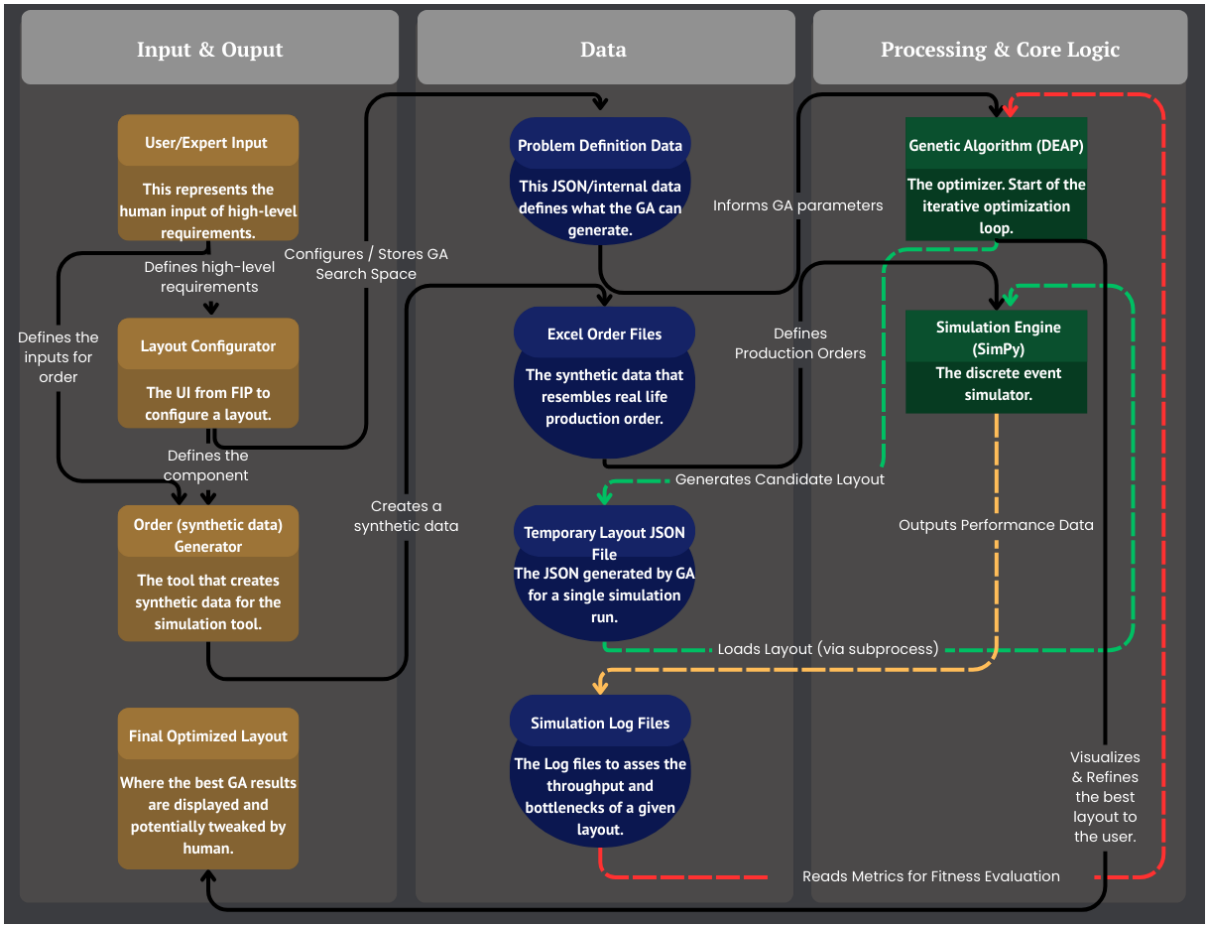

Figure 4: System Architecture Overview showing data flow between components

Nodes:

- Left - Input & Output Layer:

  Layout Configurator (PyQt5 UI), Order Generator

- Middle - Data Layer:

  Problem Definition Data, Problem Definition Data, Excel Order Files, Temporary Layout JSON File and Simulation Log Files

- Right- Processing & Core Logic:

  Simulation Engine (SimPy), Genetic Algorithm (DEAP)

Arrows & Data Flows:

1. User/Expert Input → Layout Configurator
2. Layout Configurator → Order (synthetic data) Generator
3. User/Expert Input → Order (synthetic data) Generator
4. Layout Configurator → Problem Definition Data
5. Order (synthetic data) Generator → Excel Order Files
6. Problem Definition Data → Genetic Algorithm
7. Genetic Algorithm → Temporary Layout JSON File
8. Temporary Layout JSON File → Simulation Engine
9. Excel Order Files → Simulation Engine
10. Simulation Engine → Simulation Log Files
11. Genetic Algorithm → Temporary Layout JSON File

    Over here the system adopts an optimization feedback loop from GA to Temporary Layout JSON File step until the number of generations of the GA is completed this cycle iterates. Once the final layout selected then iterative cycle ends with a final layout design

12. Genetic Algorithm → Final Optimized Layout

Layout Configurator (LC) (PyQt5 UI): This is a tool that provides a user friendly graphical interface using the PyQt5. It has two purposes; one it enables experts/users to configure and visualize a layout design manually which can be saved in JSON format. Secondly, it is used by expert/user to define high level parameters and constraints for GA's search space. This includes the available component types (Corrugator, Conveyor, WIP, Converter and EOL), their properties (dimensions, operational ranges, speed and their connections.

Order Generator (OG) (Synthetic Data Generator): This tool is dedicated for generating synthetic and realistic production order data. The generated data is in .xlsx format defining the simulated factory layout's order demand. This generator allows flexible and custom order configurations (e.g. sheet numbers, sheet dimensions, flute types and machine schedules) in order to generate realistic, challenging and diverse scenarios for the simulation. This tool is a

crucial input for the DES and critical for the AI enhanced generative design by addressing Sub-RQ (Evaluation of the proposed solution). The adoption of realistic resemblance of the synthetic data is a helpful approach enabling consistent and repeatable performance evaluations.

Discrete Event Simulation (DES) (SimPy): DES is the dynamic evaluation core of the generative design workflow. This module evaluates the given factory layouts dynamically. It receives a JSON defined layout from the GA or LC and an Excel defined production order set from the OG as input. With the use of SimPy framework this module models the time based progression of the stacks (stacks of cardboards with given dimensions) through various components (Corrugator, Conveyor, WIP, Converter and EOL). This time based progression calculation also accounts the speeds, capacities, processing times and inter component interaction of each component available. The DES outputs detailed activity log for each component, stack and performance metrics which is crucial for evaluating the layout efficiency.

Genetic Algorithm (GA) (DEAP): This is the module that acts as AI enhanced optimizer. It leverages the DEAP framework for automation and optimization factory layouts. This module operates on a population of candidate layouts (represented as a chromosome) and applies evolutionary operators (selection, crossover and mutation) that is defined by the problem definition data. Then for each generated candidate the GA performs a DES run and collects performance metrics from the output of DES. From this output it calculates a fitness score. The fitness score is used for evolution which enables the algorithm to prioritize the selection of layouts that optimize the defined objectives such as throughput/ material handling efficiency. The GA is the core module that drives the discovery of novel and efficient layouts.

4.2 Data Understanding & Preparation

This section describes the crucial data sources, preparation of those data sources and the formatting within the system workflow. It describes how the data is prepared for both DES and GA. This approach of data understanding and preparation also addresses to Sub-RQ 2 which is crucial for identifying of critical layout constraints, machine parameters and throughput metrics for the AI enhanced generative design.

4.2.1 Facility Layout Data Model (JSON Structure)

The facility layout configuration is represented as JSON (JavaScript Object Notation) array of objects (an example can be seen below in Listing 1). Each object in this array describes a specific component on the factory with their unique properties. These components are corrugator (a

machine that is responsible for the production of corrugated cardboard from rolls of sheets) [7], conveyor (a stationary transport system that moves materials from one location to another location and the movement is limited to single axis with a fixed input and output), WIP (Work-In-Progress Area) (a short term buffer or storage area formed by multiple conveyors. Primary purpose is to manage the variations in the production flow rates between machines. Preventing overflow and ensures continuous material availability minimizing the idle time and contributing to the machine utilization.), Transfer Car (A movable conveyor system that is designed to efficiently transport stacks of cardboard between conveyors and machines. Optimizes layout flexibility), Converter/ Machine/ Processor (a processing machine that performs specialized operations dependent on the machine such as cutting, printing, folding...) and End of Line (EOL) (The final stage of the production process where all the stacks are collected and palletized to be sent to the client).

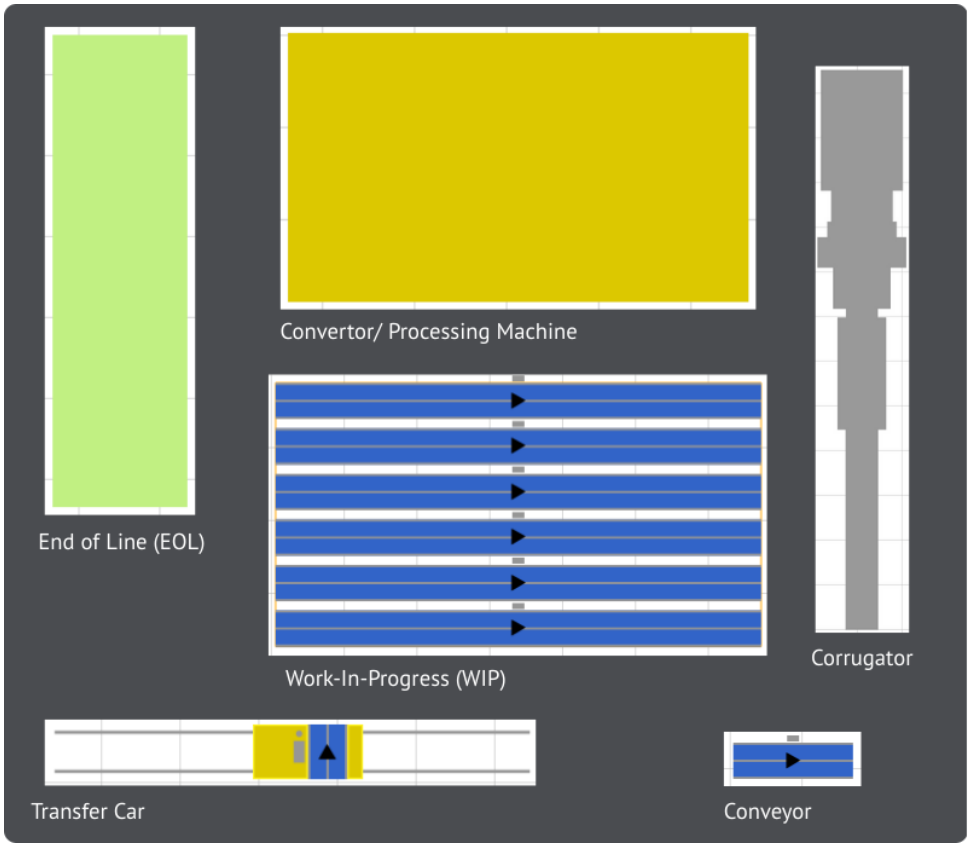

Figure 5: The available components in the system

The current available components in the system are shown in Figure 5. These components are visual representations of the building blocks of the GUI and the Generative AI enhanced design.

Listing 1: Representative snippet of facility layout in JSON structure

```
{
    "type": "corrugator",
    "name": "Corrugator",
    "length": 2000,
    "outputs": ["Conveyor_1", "Conveyor_2"]
},
{
    "type": "conveyor",
    "name": "Conveyor_1",
    "length": 2000,
    "width": 1000,
    "speed": 18,
    "inputs": ["Corrugator"],
    "outputs": ["Conveyor_3"]
},
{
    "type": "machine",
    "name": "Cut",
    "length": 10000,
    "width": 5833,
    "speed": 100,
    "width_change_factor": 10,
    "length_change_factor": 0,
    "inputs": ["Conveyor_Cut"],
    "outputs": ["Conveyor_7"]
},
{
    "type": "wiparea",
    "name": "WIP",
    "length": 30000,
    "width": 60000,
    "capacity": 3600000000,
    "nr_of_conveyors": 6,
    "speed": 15,
    "inputs": ["TransferCar_1"],
    "outputs": ["TransferCar_2"]
},
{
    "type": "transfer car",
    "name": "TransferCar_2",
    "length": 1400,
    "width": 1000,
    "speed": 10,
    "nr_of_conveyors": 1,
    "inputs": ["WIP"],
    "outputs": ["Print", "Cut"]
},
{
```

```
    "type": "eol",
    "name": "EOL",
    "inputs": ["Conveyor_6", "Conveyor_8"]
}
```

Attributes: Each component in the JSON structure represents a distinct factory element with specific attributes that define its operational characteristics and spatial requirements.

- Type: (String) Specifies the type of the Component category
- Name: (String) A unique identifier name for the component
- Length, Width: (Numeric) Physical dimensions of the components (in mm)
- Speed: (Numeric) Operational speed of the component (m/min)
- Max_stack_height: (Numeric) Maximum allowable stack height chosen from the min height of convertors
- width_change_factor, length_change_factor: (Numeric) Percentage of change in the dimensions of sheets after processing.
- Capacity (WIP): (Numeric) Storage capacity of the WIP
- nr_of_conveyors (Transfer_Car, WIP): (Numeric) Number of internal conveyors
- Inputs, outputs: the inter component connection relationship

Justification: JSON was selected as a data model due to: Ease of inspection due to human readable format, ease of explaining complex structure due to hierarchical nature of the format and the support with Python environments, supporting to streamline the parsing and generation process. The schemeless structure offers flexibility for addition/change of component attributes without rigid schema migrations. This is highly effective and beneficial for iterative design cycles.

Generation: The layout is generated by GA and visualized by PyQt5 based LC from a JSON input. The same library is used in LC that also allows for the creation of JSON format (see above the Listing table 1) of a layout design from human/expert specification input (an example of visualization can be seen in Figure 6).

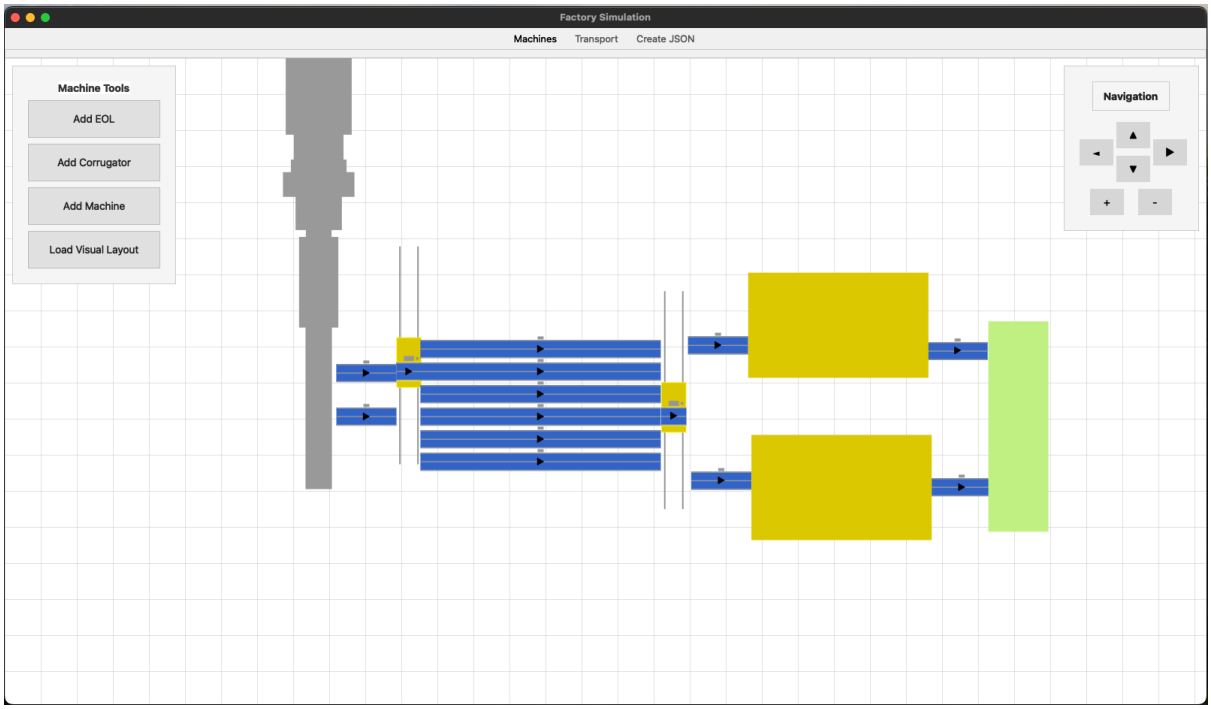

Figure 6: Sample Layout Design Visualization

4.2.2 Production Order Data Model (Excel Structure)

The operational synthetic data is produced from the OG. The OG expects a JSON file describing the specifications of the factory layout. Then the production details is outputted in Excel format which specifies the sequence and characteristics of orders to be processed. Table 4 represents sample excel format of the synthetic data generated.

Table 4: Representative sample of production orders in excel

Batch ID	Flute Type	Order ID	Length (mm)	Width (mm)	Sheets # number of	Time (min)	Machine
1	D	1	1100	1300	1700	9.71	Cut
2	BC	9	900	1200	2300	10.82	Print
3	A	11	850	1300	2300	9.93	Cut
4	E	19	850	1200	1500	6.53	Print
5	D	25	750	1300	1700	6.82	Print

This Excel data is parsed to create Order objects, encapsulating individual order details. All the attributes of the objects in the sample Excel file is explained in the Table 5 below. This includes a schedule attribute derived from the machine columns. The schedule is the sequence of machines that each stack from the order must visit. The stack height is derived from the minimum machine height. This is a simplification approach that handles the complexity of the restack machines that are separate machines used to re-stack before a processing machine to

adjust the height for their inputs.

Table 5: Production order parameter definitions and ranges

Parameter	**Description**	**Unit**	**Example Range**
Batch ID	Groups related orders for scheduling	#	1–100
Flute Type	Corrugated board fluting specification	A to F	A, B, BC, C, D, E, F
Order ID	Unique identifier for individual orders	ID number	1–102
Sheet Length	Product length dimension	mm	500–2000
Sheet Width	Product width dimension	mm	500–2000
Number of Sheets	Production quantity per order	sheets	10–100000
Producing Time	Processing duration estimate	minutes	0–10
Machine Assignment	Primary processing equipment	-	Cut, Print

This production order data serves as the dynamic input for the factory simulation. By providing realistic and varied demand scenarios, this data plays a crucial role in assessing the efficiency, identifying bottlenecks, and evaluating the overall behavior of a given layout design.

4.2.3 Performance Metrics for Fitness Evaluation

The quality of a single layout design is assessed by logged key performance indicators (KPIs) from the simulation results. These metrics are logged in a text file and they are crucial for the formulation of the fitness function. The crucial performance metrics that is going to be systematically extracted from the simulation are:

Total Simulation time: Total execution time of simulation from the first stack to the production of the last stack.

Total Number of Produced Stacks: The aggregate output quantity of finished products from the factory.

Total Number of Produced Cardboard(in meters): Total meters of cardboard produced by the corrugator.

Total Number of Square Meters Produced: Cumulative area of all finished stacks that have reached the EOL.

Machine Utilization: The ratio of total time a machine is engaged and working, to both active processing and any idle time spent waiting for input or output to be delivered. It is calculated as $(machine.total\_working\_time/simulation\_end\_time) \times 100$.

Component Throughput (Stacks): The utilization ratio in component level. The total number of stacks that have successfully passed through a particular component

WIP Maximum Number of Stacks (`max_nr_stacks`) and Time of Max Stacks (`time_max_stacks`): Metrics showing the utilization level of WIP's in the facility layout.

These diagnostics in low and high level metrics are primary source for the GAs fitness function. These are generated after a simulation run in a text file where a core part of the GA implementation is going to extract and aggregate these KPIs to its comprehensive fitness score for each candidate layout.

4.3 Simulation Environment Implementation

The Discrete Event Simulation developed by FIP provides a foundation for performance evaluation for each candidate layout instance. This provides a realistic and dynamic assessment of the proposed configurations. This section discusses the framework of simulation, components of the factory and their interactions.

The simulation is constructed on SimPy framework which is a Python based framework designed for discrete event simulations. This facilitates the modeling of the active elements like the components and their time-based interactions. The core concept of the SimPy are SimPy.Environment (the clock of the simulation), SimPy.Process (representing the components like machines or conveyors) and SimPy.Container (for modeling the queues and capacities) are replicating the behavior of a factory.

Models of the Components The logic have been analysed and implemented from the programmable logic controller documentation VDB has provided. Each physical component listed in the Figure 5 is modeled with their own operational logic and attributes in the SimPy environment.

The corrugator is responsible for generating cardboard stacks from the data of OG. It models the production time based on the stack dimensions and the configured speed of the corrugator. It handles the initial dispatch of stacks to output conveyors. The conveyor is representing the transport belts modeling the stack movement based on given the length and speed. It manages the container space and queues for the stacks. The logic in core is resembling the PLC code form the VDB environment the interruption for new stacks and external signals for stack removal.

Convertor / Machine is responsible for the processing stations (currently only cut/ print). Each machine simulates the processing time and can modify the dimensions of the stacks.

WIP reprints Work In Progress buffer areas. It models a finite capacity for temporary storage for the stacks. Also processes stacks for the waited time and manages the queue for the stacks to be picked up.

EOL component serves as a final destination of all stacks. It gathers the completed stacks and helps with the outputting the overall simulation performance metrics.

The communication between the components are handled with the SimPy.Store and SimPy.Event object synchronizations. Which resembles a factory flow with a downstream of components asking frmm upstream sources. In order to run the simulation first these components are created dynamically from the JSON configuration file. All the instances are set with the given specifications and connections. Then the SimPy process starts for all the active components and the simulation proceeds until the last component which is the EOL signaling the completion of all orders. The metrics are then outputted on log files from this simulation.

4.4 Genetic Algorithm Methodology

The GA is implemented as the core optimization metaheuristic to explore and refine factory layouts. This approach was chosen for its versatility to handle complex and multi objective problems (addressing the Sub-RQ 1). This section details the design and operational principles of GA and its implementation.

Representing a complex factory layout within the GA chromosome presents a significant challenge. A simple numerical list commonly reviewed in GA literature is insufficient for this optimization problem. It is insufficient for heterogenous component types, the diverse parameters and complex logic of a facility layout and workflow.

4.4.1 Chromosome Design (process level genes)

Chromosome design is one of the most crucial part of GA based solution. GA takes this chromosome and evolves it over generations while each gene value get to be translated to real world factory components of a final layout. The design proposed for this research is a 12 integer array. The choice of integers for each gene so that the GA operators (crossover and mutation) can stay simple. Each gene is encoding all the key design decisions on the process level rather than raw coordinates. The main reason for this is the vastness of the search space coming with raw geometry encoding in genes. This choice satisfies the spatial validation FR(see Table 1) If an average layout has 30 components with hold raw coordinates(x, y, orientation) encoded genes and in a 100m sample hall with 1m of each grid and 10 allowed values per variable creating a search space of $10^{3x30} = 10^{90}$ possibilities compared to process level design with 12 discrete genes assuming each has at most 5 choices(where the current proposed approach has at most 3 choice but for the more developed version) with a search space of 5^{12} possibilities.

Table 6: Process-level gene structure in the enhanced GA chromosome

Gene ID	Gene type	Values	Rationale
0–2	`machine_order`	6 permutations	routing sequence
3	`wip_type`	0: single 1: twin 2: bypass	insert buffer style(bottleneck calculations)
4	`corridor_slice`	0: top 1: mid 2: bot	assistance for A* search window
5	`turntable_policy`	0: never 1: auto 2: always	angle change detection in conveyor path for turntable insertion
6–8	`speed_tier`	0–2	set conveyor speed tier
9–11	not used	to be implemented	future transport modules

Above Table 6 shows what each gene encodes: First three are for the machine sequence where a flow can only go through one processing machine or chained process (example: first to be cut then to be colored(print) and then to be folded) is required depending on the orders, the second gene type is for buffer strategy (allows the use of different buffer type integration in a layout), then the third gene type is reserved for routing preference (helping for the search algorithm to find the route faster in a smaller search space, the next gene is for the use of turntables, the three genes are for the three different conveyor types where any can have a different speed type(due to the different type of conveyors). The last two genes are reserved for future models to be implemented.

4.4.2 A^* Path Finding & Spatial Validation

After the process of encoding the seed layout into a list of anchor points (The components in facility that has fixed locations: Corrugator, Obstacles, Convertors, EOL(End of Line) and facility dimensions) they will be decoded in order to initialize the path building. These anchor point are crucial for the algorithm. When the genes are decoded the high level genes specify the corridor/order preference between these anchors instead raw coordinates. Before any routing with the A^* the factory hall is discretized. This is done according to the smallest unit which will be a simplified constraint(relates to real life example of a pillar) of 3m radius with 0.5m safety distance resulting in the choice of square grid with 5m cells (flexible choice of grid depends on the smallest component that is going to be in the layout for example layout case it was the pillar). Each center of the cell is a node in a graph where the vertical and the horizontal

neighbors are joined by edges. The traversal cost from these neighbors are the euclidean distance between their centers with 5m. The octile heuritics is chosen because it can efficiently estimate distances in 8 grid(horizontal, vertical and diagonal movements) environment [17]. To find the cheapest collision free route in between two anchors A^* search with heuristic of a octile distance is calculated:

$$\text{Straight move cost} : D = 5m \tag{1}$$

$$\text{Diagonal move cost} : D_2 = \sqrt{2}\, D = 7.07 \tag{2}$$

$$h(dx, dy) = D(dx + dy) + (D_2 - 2D)\min(dx, dy) \tag{3}$$

$$h = 5(dx + dy) - 2.93\min(dx, dy) \tag{4}$$

This is admissible on eight connected grid ensuring A^* to return the true shortest path. The process level genes such as corridor slice benefits the A^* search by restricting the range of y nodes the algorithm should visit and the turntable policy decides whether a sharp angle is resolved by a turntable or letting the conveyor path to continue around the curvature. The spatial validation also checks for overlapping by expanding each component contour by 500 mm. The infeasible layout will be penalized steering the GA towards feasible layouts.

4.4.3 Two tier evaluation pipeline Fitness function

The research shows efficiency in runtime of surrogate models in computationally costly optimization contexts [35]. To balance the fidelity and speed of our system, the evaluation is split in two tiers.

Overview of two tier architecture Two tier evaluation consists of a fast surrogate model that screens all the candidates and a high fidelity DES evaluation that is a through check which is also more costly. The reason of this architecture choices is to ensure broad design exploration while managing and reserving the expensive simulation resources. The pipeline is operating on four stages sequentially: starting with evaluation of all the candidates in the population with the surrogate model, then elite selection based on the surrogate results, proceeding with DES evaluation of the elite selected candidates and combining the fitness of both evaluation results as seen below in Figure 7.

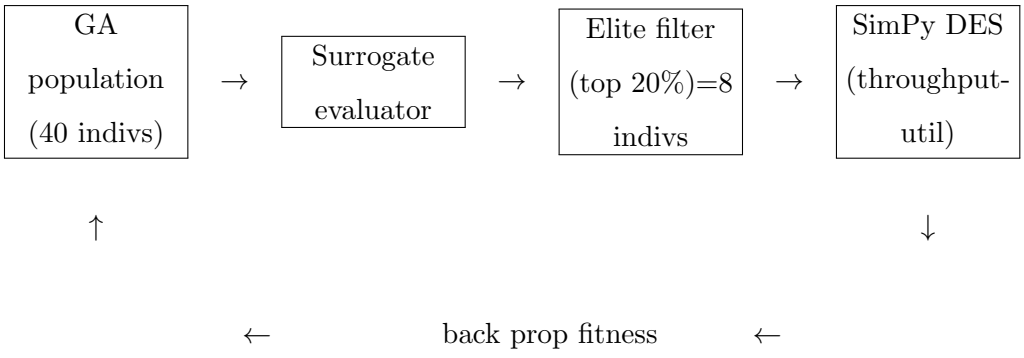

Figure 7: Two-tier evaluation pipeline with fitness back propagation

The Figure 7 shows the whole example population of forty candidates which goes through the fast surrogate check. In order to balance the fidelity and feasibility only the top 20th percentile undergoes through full simulation evaluation and with the example scenario given in the table the 40 pop x 20 gen x 0.2(top 20th percent) = 160 simulation runs which satisfied the system runtime NFR(non functional requirement) ensuring that the system will run under 60 mins.

Tier 1 Surrogate evaluation The surrogate model provides a fast assessment of the layout quality within miliseconds range. The evaluation is done on four metrics that aid the full simulation performance.

Objectives of Tier 1 The surrogate models will have two objectives and two constraints. One of the objective is that minimizing the total conveyor length. This is s simple summation of all conveyor lengths in a layout, serving a proxy for MHC and overall system complexity. Other objective will be the path ratio efficiency. The system will measure the efficiency of the routing by considering the number of hops in between components and the number of intermediate conveyors. the path ratio that is getting closer to one will indicate an optimal routing.

Constraints of Tier 1 Overlapping and the connectivity is crucial for DES evaluation. For the initial phase when constructing a layout from the anchor seed layout Axis-Aligned Bounding Box(AABB) approach will test component by component identifying if the next component to be added is going to overlap [43]. Since the genetic operators can break this validity of layouts during the fitness evaluation the overlapping can still be the case. Given full layout more efficient sweep line approach of the Bentley Ottmann algorithm will be used during fitness evaluation to identify if the layout has overlaps [42].

Another constraint the system should be penalizing is the connectivity which will also affect

the ability of running a full simulation. The system will utilize graph traversal algorithm to verify that material flow(starting from Corrugator to MachineA/B/C to EOL) exists between all the required components pairs. This will ensure the production viability of a produced layout.

Tier 2 DES evaluation The DES evaluator provides a high fidelity evaluation through full discrete event simulation. The selection mechanism ensures that only the high performing candidates are selected; as given example case in Figure 7 only the most promising %20 percentile is being evaluated by DES. The selection strategy will prioritize the feasible solutions based on Tier 1 constraint satisfaction and performance rankings. This will help to ensure the limited budget for the DES evaluation focuses on operational requirements as layouts showing promise for optimizing the targeted objectives.

Back Propagation & Fitness Integration The final fitness will be employed by merging both evaluation tiers preserving both of their benefits. Candidates that are evaluated by DES receive a high fidelity score while the remaining candidates retain their surrogate results. This combination creates a mixture of high fidelity population where the selection is guided by the most accurate information available for that individual.

4.4.4 NSGA-II parameters

The Non dominated Sorting Genetic Algorithm has been chosen because of the ease of handling several objectives (the current proposals objectives in the pareto front are total belt length and the path ratio(actual/manhattan)), being able to keep a diverse solution set instead of single solution and that it never looses the best layout so that the progress never falls back. The multi-objective trade of is really crucial for the future extensibility of the project with other crucial objectives in the industry energy consumption, component price, throughput and more dimensions could be added in the Pareto front.

Table 7: NSGA-II parameters and values

Parameter	**Value**	**Rationale**
Population	40	In each generation 40 candidate layout are kept alive at the same time which is big enough of variety and satisfies the system runtime NFR by keeping the system runtime under one hour.
Generations	20	20 generations with 40 candidates in each generation resulting in 800 total candidate is big enough of search while remaining maintainable budget for the event simulation
Crossover	uniform, %50	50 percent is standard coin flip rate where each gene has 0.5 chance of getting copied from parent A or B
Mutation	bit flip, %8	adds small percent of different variety of search so that GA does not get stuck.
Stop rule	if system reached 20 generations or HV has not improved by more than 1 ten percent in five gens(HV plateau< 0.001)	Ensures the convergence as avoiding wasted time if the search has already reached a plateau on exploring better layouts.
Seed	42	Just a random number generator which has this value as being the answer to the universe.

Above the Table 7 presents the parameter. The parameter values with the two tier evaluation pipeline satisfy the NFRs system runtime and reproducibility by resulting towards similar Pareto front.

4.5 Integration Workflow

This section explains the complete automated integration workflow for optimizing the factory layout generation. This workflow includes integration of the DA and DES. It is a iterative process where GA proposes the layouts and DES dynamically evaluates their performance back to GA refining its search based on the feedback to increase the optimization of throughput in

the layout configurations.

5 Realization

In this chapter, implementation and evolution of the proposed methodologies and system specifications are going to be discussed. In chapter two, where the Genetic Algorithm based simulation integration was chosen the single objective GA evaluated every layout directly within the simulation. Since %99 of the layouts were infeasible the simulation runs were unrealistic. This was because the spatial awareness of each component is directly related to the connectivity of the components. Where in the first proposed approach the created layouts were just randomly generated around the facility without realistic connectivity(the locations were discovered but the input and the output connections were remained) this was a good base as a proof of concept but was not close enough for realistic scenarios. For the spatial awareness to develop A^* path finder and fast surrogate evaluator found to be effective with an increase of %83 and runtime below 90 minutes. The switch to NSGA-II with the addition of two tier fitness loop produced stable Pareto front within 60 minutes as reaching our KPI and maintaining the throughput. Following parts of this chapter is going to describe the final architecture(5.2), gene to layout pipeline (5.3), path finding pipeline(5.4) and the validation(5.6).

5.1 Final Architecture Overview:

The final architecture has evolved upon the proposed architecture from the chapter 4. The modularity of the system remained but the data flow has changed. In the old architecture the data would be in loop from the GA module to DES into fitness function where as now builder module(for the geometry and spatial awareness) and surrogate evaluation is added and the new flow starts from the GA to Builder module then to a fast evaluation of surrogate check and then to DES as seen below in Figure:8. These refactored loops with the sub-module additions helped the system to create feasible layouts and complete DES runs of those layouts.

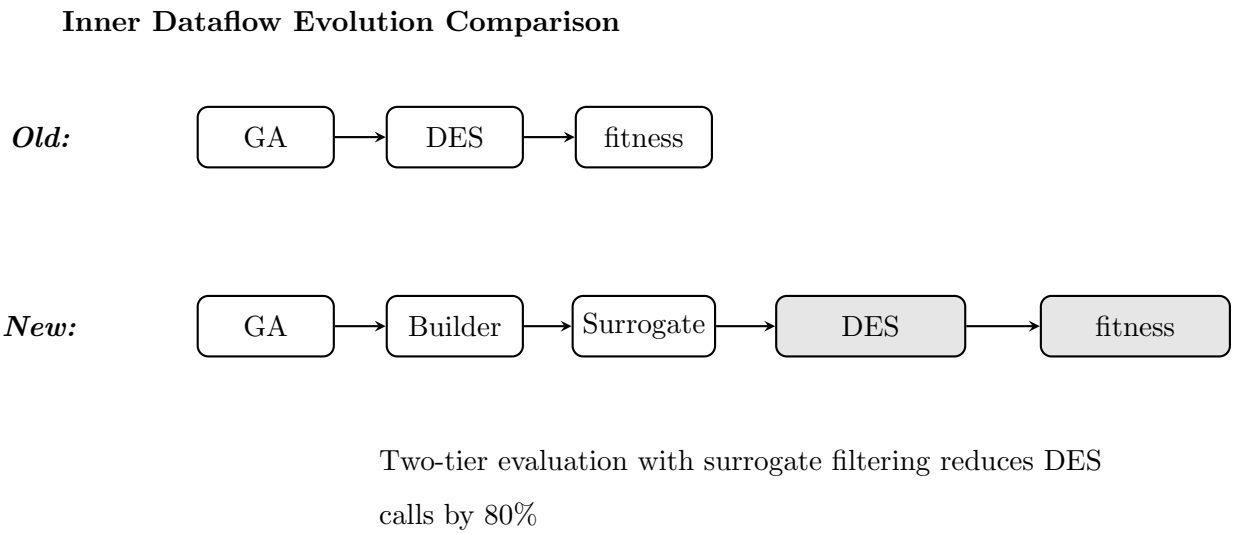

Figure 8: Evolution of the GA workflow: Direct evaluation (old) vs. two-tier evaluation pipeline (new)

Not only the data flow but also new archtitecture has added a parallel executions for the simulation runs by Python's multiprocessing.Pool.

```
#use of parallel run of the DES with the top %20 percentile of the layouts
elite_f = pool.map(full_evaluator.evaluate, elite_layouts)

#create an identifier for layout and see ife we have evaluated it then reuse
    the cache result if not computer teh evaluationa nd store the result for
    future use
layout_hash = hash_layout(lay)
if layout_hash in cache:
    f = cache[layout_hash]
else:
    f = full_evaluator.evaluate(lay)
    cache[layout_hash] = f
```

Listing 2: Implementation of multiprocessing.Pool + cache code snippet

As the detailed csv logs the final architecture is reusing the DES cache reducing the average evaluation time and eliminating the duplicate runs. The bGA module now has NSGA-II and two tier evaluation and the rest of the configurators and utils remained same the visualization of the final architecture can be seen in Figure 4. Each Layer contributes to a different module of our system. The layers consist of Interface, Builders, GA and utilities.

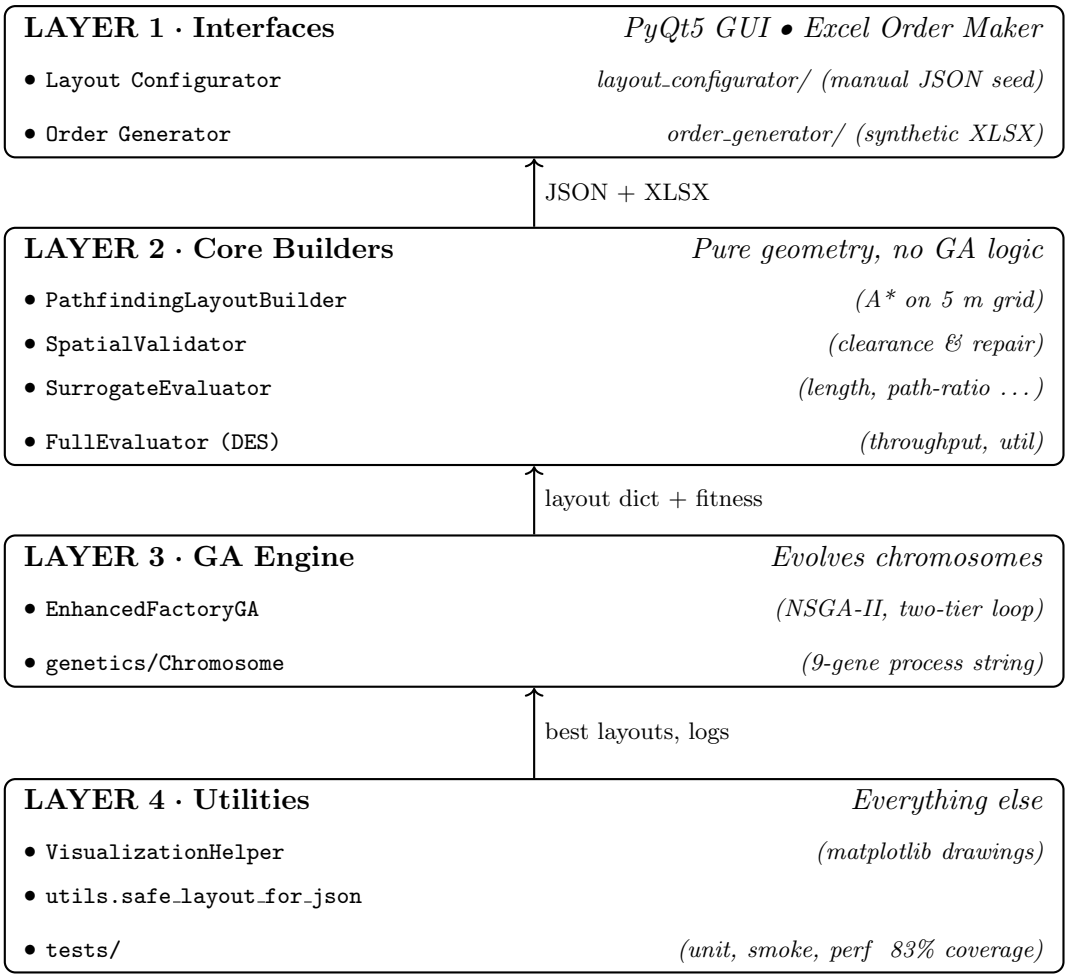

Figure 9: Four-layer architecture of the AI-enhanced facility layout optimization system

The above figure depicts the moduar architecture of the system starting with the first layer. It has the user interfaces that the user is expected to create or customize, the seed layout and order file that is meant to be fed to the builders. These interfaces did not change form the proposed approach in chapter 4.

5.2 Genotype to Phenotype

This chapter explains how the encoded chromosomes are decoded to layouts happening in between the GA module and the Builder module. The same gene structure has been implemented as proposed in the chapter 4 with the high level genes repacing the raw coordinates. Below there is a flow diagram of what is happening in between those modules once a seed layout is initialized and order file was initialized in the first module the interfaces. See the figure10 below to observe what happens after that system creates encodes a chromosome and decodes it to a layout.

The random created chromosomes by the NSGA-II call and for each chromosome created a

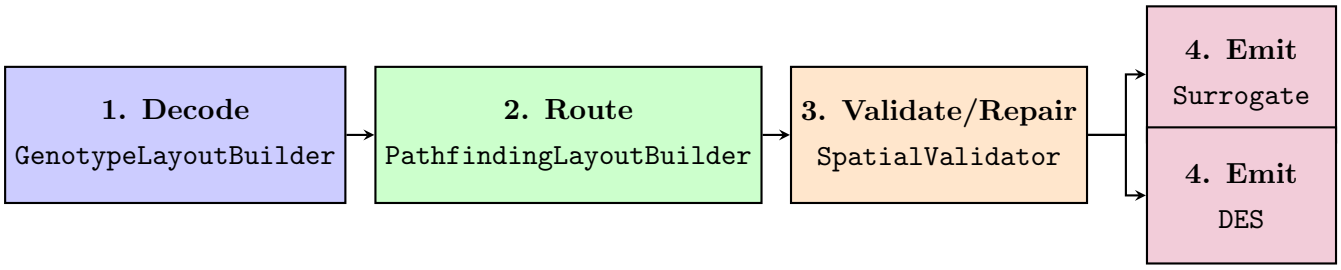

Figure 10: Four-step pipeline from chromosome to layout evaluation

genotype layout builder is called with the given chromosome and the anchors that are set and fed by the user. The genotype layout builder decodes and reads the process level genes and builds an ordered way point list and passes it to pathfindingLayoutBuilder class. Example way point list: L=[Corrugator, M_1, M_2, M_3, EOL]. Then for each consecutive pair $(L_(i_1), L_(i_2))$ in the list routing is the computed with A^* search following the corridor preference with 5m grids and each cell being a node. The calculated paths are converted to straight segments from the grid cells and forming a conveyor with length, $x_location$, $y_location$ and angle. Within the same class the turntable and wip insertion is decided based on the decoded genom dict. The created layout is given to spatial validator to check if a layout is feasible, overlapping, out of factory boundaries. After the checks if any of the rules are vilated, the layout is penalized. Then the Genotypebuilder assembles a dictionary, gives it to DES and surrogate evaluator for evaluation.

5.3 Two–tier evaluation

The evaluation is split in two as stated in chapter 4. The evaluators are designed in a conditional merge strategy. For each generation the process start with surrogate evaluator finds a proxy score for all individuals. Given these proxy scores a small subset %20 is of promising layouts are selected. The chosen elite layouts are evaluated by the high-fidelity but computationally expensive DES composing the final fitness vector for one individual. If a layout was evaluated by DES then its proxy score was overwritten with the results of DES. This ensures that GA selection pressure is guided by the more accurate information available for more promising layouts. The provided snippet shows the flow state above in the code starting from the decoding to surrogate all to Filtering the elites to DES evaluation and finishing by merging them. Another crucial implementation is how the assumption of surrogate model is creating an accurate proxy is validated. For each elite that is evaluated by both tiers evaluators, the root mean square(RMSE) is calculated. This RMSE metric provides that surrogate is effectively guiding the GA and and

```python
#use of parallel run of the DES with the top %20 percentile of the layouts
elite_f = pool.map(full_evaluator.evaluate, elite_layouts)

# 1. Decodes all individuals in population 'X' into layouts
layouts = [builder.build(chrom) for chrom in X]

# 2. Tier-1: Get fast surrogate scores for the entire population
surrogate_scores = surrogate.batch_score(layouts)
f_surrogate = surrogate_scores[:, [0, 3]] # Objectives: (length and path-ratio
g_surrogate = surrogate_scores[:, [1, 2]] # Constraints (overlap, connectivity)

# 3. Select elite individuals based on surrogate performance
#    (prioritizes feasible solutions first)
elite_indices = select_elites(f_surrogate, g_surrogate, k=int(pop_size * 0.2))

# 4. Tier-2: Evaluate elites with high-fidelity DES
f_des = np.full_like(f_surrogate, np.nan) # Initialize with NaN
g_des = np.full_like(g_surrogate, np.nan)

for idx in elite_indices:
    layout_hash = hash_layout(layouts[idx])
    if layout_hash in cache:
        # Use cached DES result
        des_result = cache[layout_hash]
    else:
        # Run expensive DES evaluation (can be parallelized)
        des_result = full_evaluator.evaluate(layouts[idx])
        cache[layout_hash] = des_result # Store in cache

    # Populate DES results for this elite individual
    if des_result is not None:
        f_des[idx] = des_result[[0, 3]]
        g_des[idx] = des_result[[1, 2]]

# 5. Final Merge: Use DES scores where available, otherwise use surrogate
    scores
has_des_score = ~np.isnan(f_des[:, 0])
out["F"] = np.where(has_des_score[:, None], f_des, f_surrogate)
out["G"] = np.where(has_des_score[:, None], g_des, g_surrogate)
```

Listing 3: Implementation of two tier fitness

allows to observe if the surrogate starts to drift or becomes a poor predictor for the GA. Below the code snippet shows the implementation of the RMSE calculation.

```python
# Calculates surrogate RMSE for monitoring
if has_des.any():
    # ...
    rmse = np.sqrt(np.mean((surrogate_vals[valid_mask] - des_vals[valid_mask])
        **2))
    self.surrogate_rmse = rmse
```

Listing 4: RMSE monitoring

5.4 Testing & validation

Through out the development process multi layered testing strategy was implemented to ensure correctness and performance. This system evolved as the project evolved moving form simple component checks to full system evaluation and validations. Final testing process can be categorized into 4 levels respectively; Unit testing, system testing, regression testings and performance testing.

Unit Testing From the lowest level of individual function and classes to higher level of modular integrations 77 test were developed. By these test the core logic of critical modules and collaborative working tested.

System Testing (Smoke) Before running a full, time consuming optimizing run a smoke test has performed. This is a rapid test of the system with small population and minimal number of generations. The goal is not to find the optimal solution but to confirm the whole pipeline of the system form genotype creation to surrogate evaluation to DES simulation executes without any errors or crashes.

Regression Testing The baseline layout file was used to be compared for each new run after any significant change in the system.

Performance Benchmarking The system was evaluated against the non-functional requirements verifying that surrogate evaluator could process full population within milliseconds and the entire system run would be within one hour budget.

Table 8: Validation matrix and results

Test level	of cases	Purpose & examples
Unit	77	verifying small functions and classes • `SpatialValidator.has_overlap()` returns `False` for non-intersecting AABBs. • `Chromosome.from_list()` raises on wrong gene count.
Smoke	12	quick full system runs with small params (pop = 6, gen = 2). Confirms that the pipeline produces ≥1 feasible layout and no issues.
Regression	5	Compare current best-layout JSON against company baseline layout
Performance	4	Surrogate run is within ms for 40 layouts, A* path 2 ms per branch andfFull optimisation < 60 min

Coverage. Running `pytest {q {cov` reports 7 failed 71 passed. Critical modules (`builder`, `validator`, `evaluator`) exceed 90 %. Log-parsing glue and GUI code constitute most of the errors and failing

Key libraries

`SimPy` 4.1.0	discrete-event engine
`pymoo` 0.6.1	NSGA-II, sampling, indicators
`numpy` 1.26.4	vector maths
`networkx` 3.3	connectivity checks
`matplotlib` 3.8.4	visualisation
`pytest` 8.0.2	testing framework
`pytest-benchmark` 4.0	perf gates

Threading / parallelism All SimPy evaluations run in a `multiprocessing.Pool`(size = CPU cores = 16). Surrogate calculations stay single-threaded NumPy.

5.5 Lessons learned from pivots

The evolution of the GA was stating with a version that could not create any feasible layouts. Then integrating a builder and A* has increased the feasibility but the runtime issue didn't met the targeted KPI's. Surrogate/DES two tier evaluation with NSGA-II model gave the

scalability and balancing the multi objective approach as reducing the whole system time in the proposed solution. The final evolved architecture after the realization phase can be seen below in Figure 11.

These insights frame the discussion in Chapter 6 and chart a roadmap for future industrial deployment.

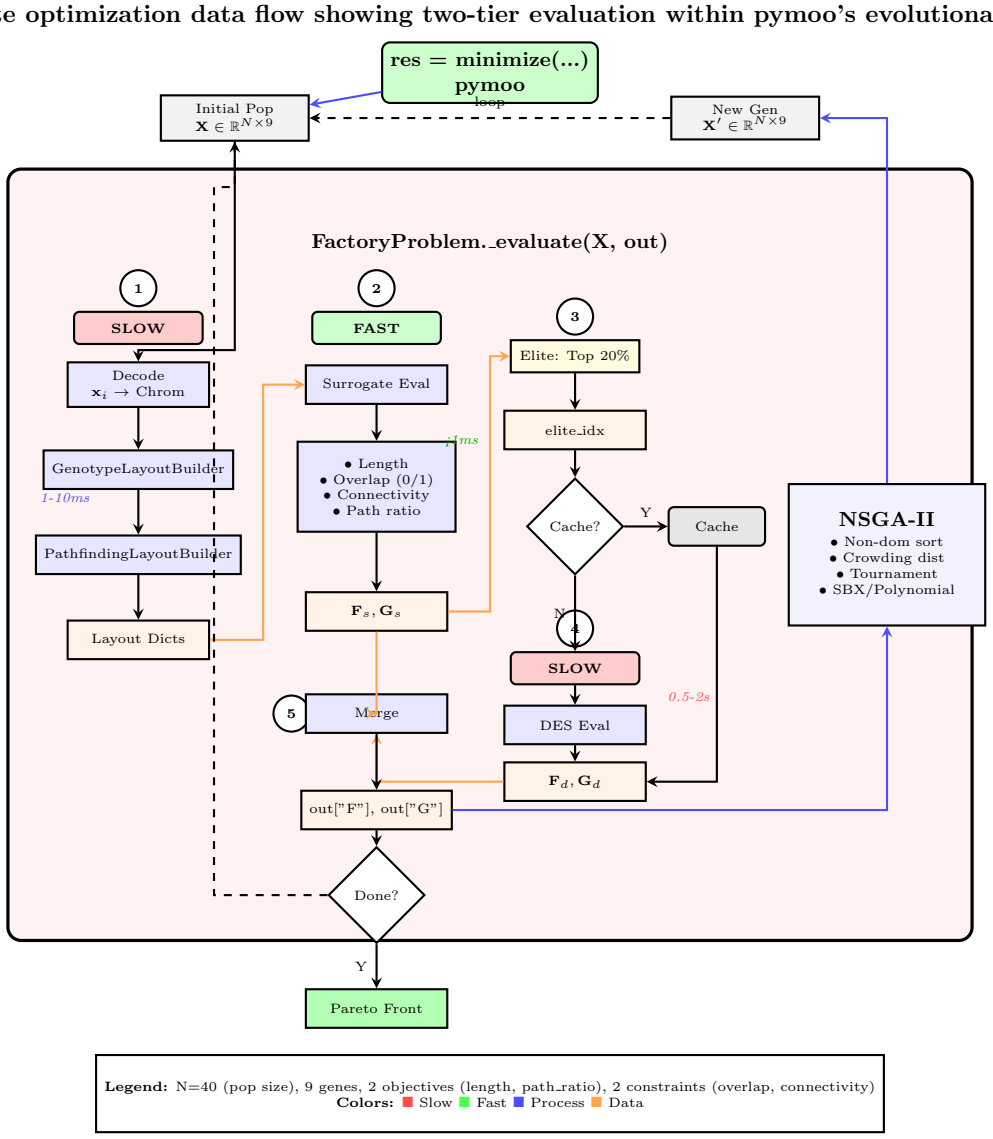

Figure 11: Complete optimization data flow showing two-tier evaluation with NSGA-II

6 Results & Analysis

The previous chapter detailed how the proposed solution in chapter 4 was practically implemented and built on chapter 5. This chapter evaluates how good does the implemented system performs. All the runs were executed in the same computer and python environment: CPython

3.11.4 ×64. All the experiments used same population of 40, generation of 20, seed 42 and the two tier fitness budget 160 SimPy evaluations. Baseline layout from VDB is used as control design and as a result five key performance indicators are reported for each layout that is generated is total conveyor length(mm) path ratio(actual/manhattan), avg throughput(stacks per hour), mean machine utilization and feasibility(overlaps, connectivity).

The chapter proceeds with global convergence curves to KPI comparisons, bottleneck inspection, an ablation study and sensitivity analysis.

6.1 Experimental set-up

Hardware:Apple M3, 24 GB RAM. Software: CPython 3.11.4, SimPy 4.1.0, pymoo 0.6.1. Baseline Layout:

Listing 5: Seed Layout


```
{
  "bounds": {
    "x_min": 0,
    "x_max": 400000,
    "y_min": 0,
    "y_max": 250000
  },
  "obstacles": [
    {"type": "pillar", "x": 100000, "y": 80000, "radius": 5000},
    {"type": "pillar", "x": 200000, "y": 80000, "radius": 5000},
    {"type": "pillar", "x": 300000, "y": 80000, "radius": 5000},
    {"type": "pillar", "x": 100000, "y": 170000, "radius": 5000},
    {"type": "pillar", "x": 200000, "y": 170000, "radius": 5000},
    {"type": "pillar", "x": 300000, "y": 170000, "radius": 5000}
  ],
  "components": [
    {
      "id": "Corrugator",
      "type": "corrugator",
      "x": 50000,
      "y": 125000,
      "length": 2000,
      "max_width": 3500,
      "speed": 200,
      "sheet_thickness_mm": 4,
      "max_stack_height": 1000
    },
    {
      "id": "Conveyor_1",
      "type": "conveyor",
      "x": 80000,
```

```
    "y": 110000,
    "length": 40000,
    "width": 1000,
    "speed": 18,
    "angle": 0
  },
  {
    "id": "Conveyor_2",
    "type": "conveyor",
    "x": 80000,
    "y": 140000,
    "length": 40000,
    "width": 1000,
    "speed": 18,
    "angle": 0
  },
  {
    "id": "TransferCar_1",
    "type": "transfer_car",
    "x": 130000,
    "y": 125000,
    "length": 1400,
    "width": 1000,
    "speed": 10,
    "nr_of_conveyors": 2
  },
  {
    "id": "WIP",
    "type": "wip",
    "x": 150000,
    "y": 125000,
    "length": 30000,
    "width": 60000,
    "capacity": 3600000,
    "nr_of_conveyors": 6,
    "speed": 15
  },
  {
    "id": "TransferCar_2",
    "type": "transfer_car",
    "x": 185000,
    "y": 125000,
    "length": 1400,
    "width": 1000,
    "speed": 10,
    "nr_of_conveyors": 3
  },
  {
    "id": "Conveyor_Print",
```

```
    "type": "conveyor",
    "x": 193000,
    "y": 85000,
    "length": 35000,
    "width": 1000,
    "speed": 18,
    "angle": 270
  },
  {
    "id": "Print",
    "type": "machine",
    "x": 200000,
    "y": 50000,
    "length": 10000,
    "width": 5833,
    "speed": 100,
    "batch_time": 120,
    "width_change_factor": 20,
    "length_change_factor": 10
  },
  {
    "id": "Conveyor_Cut",
    "type": "conveyor",
    "x": 193000,
    "y": 165000,
    "length": 35000,
    "width": 1000,
    "speed": 18,
    "angle": 90
  },
  {
    "id": "CutCrease",
    "type": "machine",
    "x": 200000,
    "y": 200000,
    "length": 10000,
    "width": 5833,
    "speed": 100,
    "batch_time": 180,
    "width_change_factor": 10,
    "length_change_factor": 0
  },
  {
    "id": "Conveyor_Glue",
    "type": "conveyor",
    "x": 195000,
    "y": 125000,
    "length": 95000,
    "width": 1000,
```

```
    "speed": 18,
    "angle": 0
  },
  {
    "id": "Glue",
    "type": "machine",
    "x": 300000,
    "y": 125000,
    "length": 10000,
    "width": 5833,
    "speed": 100,
    "batch_time": 90,
    "width_change_factor": 0,
    "length_change_factor": 0
  },
  {
    "id": "Conveyor_3",
    "type": "conveyor",
    "x": 210000,
    "y": 55000,
    "length": 140000,
    "width": 1000,
    "speed": 18,
    "angle": 0
  },
  {
    "id": "Conveyor_4",
    "type": "conveyor",
    "x": 210000,
    "y": 195000,
    "length": 140000,
    "width": 1000,
    "speed": 18,
    "angle": 0
  },
  {
    "id": "Conveyor_5",
    "type": "conveyor",
    "x": 310000,
    "y": 125000,
    "length": 35000,
    "width": 1000,
    "speed": 18,
    "angle": 0
  },
  {
    "id": "EOL",
    "type": "eol",
    "x": 350000,
```

```
      "y": 125000
    }
  ],
  "connections": {
    "Corrugator": ["Conveyor_1", "Conveyor_2"],
    "Conveyor_1": ["TransferCar_1"],
    "Conveyor_2": ["TransferCar_1"],
    "TransferCar_1": ["WIP"],
    "WIP": ["TransferCar_2"],
    "TransferCar_2": ["Conveyor_Print", "Conveyor_Cut", "Conveyor_Glue"],
    "Conveyor_Print": ["Print"],
    "Print": ["Conveyor_3"],
    "Conveyor_Cut": ["CutCrease"],
    "CutCrease": ["Conveyor_4"],
    "Conveyor_Glue": ["Glue"],
    "Glue": ["Conveyor_5"],
    "Conveyor_3": ["EOL"],
    "Conveyor_4": ["EOL"],
    "Conveyor_5": ["EOL"]
  },
  "anchors": {
    "Corrugator": {
      "x": 50000,
      "y": 125000,
      "fixed": true
    },
    "Print": {
      "x": 200000,
      "y": 50000,
      "fixed": true
    },
    "CutCrease": {
      "x": 200000,
      "y": 200000,
      "fixed": true
    },
    "Glue": {
      "x": 300000,
      "y": 125000,
      "fixed": true
    },
    "EOL": {
      "x": 350000,
      "y": 125000,
      "fixed": true
    }
  }
}
```

Seed layout converted to JSON;

Listing 6: Seed Layout

```
bounds = {
        'x_min': 0, 'x_max': 400000,  # 400m x 250m factory
        'y_min': 0, 'y_max': 250000
    }

    obstacles = [
        {'type': 'pillar', 'x': 100000, 'y': 80000, 'radius': 5000},
        {'type': 'pillar', 'x': 200000, 'y': 80000, 'radius': 5000},
        {'type': 'pillar', 'x': 300000, 'y': 80000, 'radius': 5000},
        {'type': 'pillar', 'x': 100000, 'y': 170000, 'radius': 5000},
        {'type': 'pillar', 'x': 200000, 'y': 170000, 'radius': 5000},
        {'type': 'pillar', 'x': 300000, 'y': 170000, 'radius': 5000},
    ]

    anchors = {
        'Corrugator': {
            'x': 50000, 'y': 125000,
            'speed': 200,
            'max_width': 3500,
            'sheet_thickness_mm': 4,
            'max_stack_height': 1000
        },
        'Print': {
            'x': 200000, 'y': 50000,
            'batch_time': 120
        },
        'CutCrease': {
            'x': 200000, 'y': 200000,
            'batch_time': 180
        },
        'Glue': {
            'x': 300000, 'y': 125000,
            'batch_time': 90
        },
        'EOL': {
            'x': 350000, 'y': 125000
        }
    }
```

mixed-order file `orders_mix120.xlsx` (34 batches). The baseline was simulated once with the same order file to yield reference KPIs.

6.2 Convergence behavior

Figure 12 shows (a) hyper-volume (HV), (b) feasible layouts, (c) total length and (d) path ratio over time 10 generations.

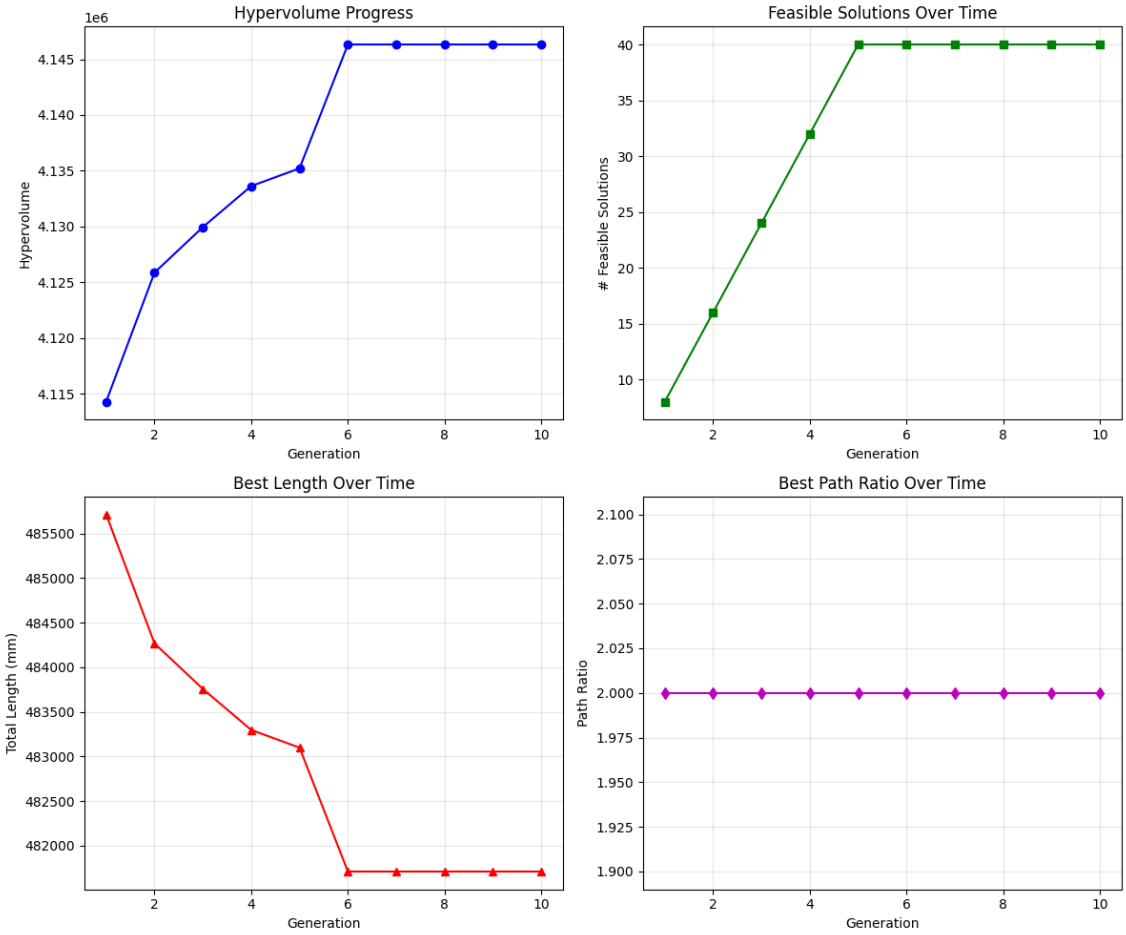

Figure 12: Convergence curves: (a) HV; (b) feasible individuals, (c) shortest total path length and (d) path ratio.

The hyper volume evolution shows fast initial progress from 4.114×10^6 at generation 1 to 4.147×10^6 by generation 6, representing %0.8 improvement. The curve shows steep inclining in early generations followed by stabilization. This indicates a fast start and efficient exploration of the objective space in the early phase and reaching saturation with identifying the most optimal layout. The plateau after generation 6 means that the algorithm has converged to a stable Pareto front. The feasibility progression figure shows the number of feasible solutions throughout the evolution of the populations. Starting from 8 feasible solutions (%20 of the population) in the first generation it increases to 32 feasible layouts by the fourth generation and reaching full feasibility by the 5th generation. The rapid improvement in the feasibility within the early

generations validates the effectiveness of the spatial validator and the smart builder modules guiding the search towards valid layouts while remaining the diversity. The last two tables shows the optimization of total length in the best layout. The total length decreases from 485,500 mm in generation 1 to 481,101 mm by generation 6 with achieving a 4,399 mm (%0.9) reduction in total length. The steepest improvement occurs in generation 6 which after stabilizes. The path ratio achieves the optimal value of 2 by the first generation and maintains the throughput indicating the progressive layout builder is successful in constructing efficient routes. Overall the convergence analysis revealing three important facts: fast convergence within 6 generations with demonstrating computational efficiency, the algorithm reaches %100 feasible layouts by the fifth generation confirming effective constraint handling and lastly stable optimization with consistent path efficiency while minimizing the length. This smooth convergence validates the two-tier evaluation strategy is effective despite the fact only %20 of candidates are simulated.

6.3 Pareto front

Figure 13 presents final Pareto front after 20 generations. The plot has total conveyor length(x axis) against path ratio(x axis)for all 38 feasible solutions the GA found. The color gradient represents the throughput ranging form 100 to 125 stacks/hour.

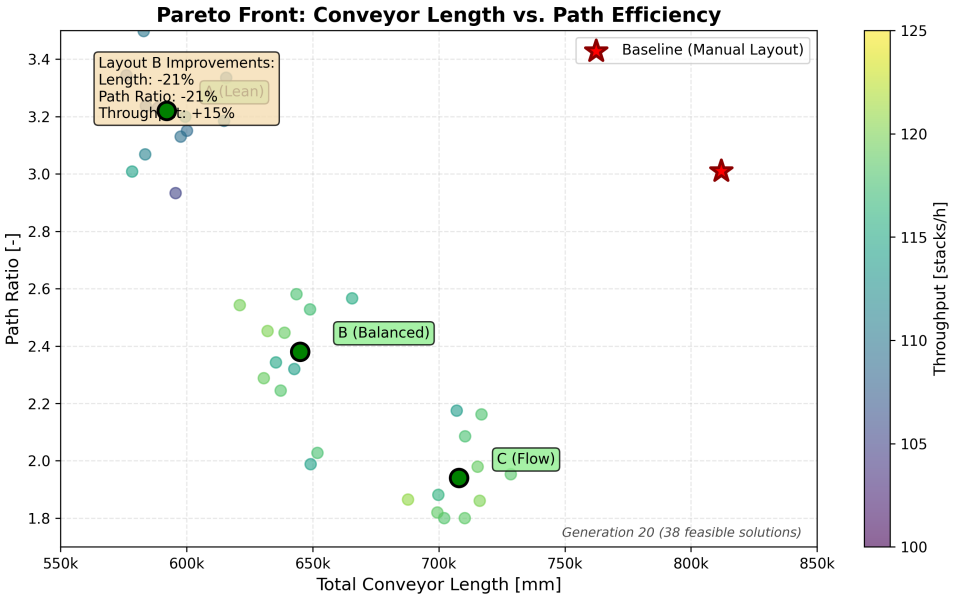

Figure 13: Pareto front after 20 generations (colour = throughput [stacks h^{-1}]).

The generated solutions forms a trade off curve between two objectives, starting from 580,000 to 740,000mm for total length and ranging from 1.8 to 3.4 path ratio. This shows the ability of the algorithm to explore the diverse solutions. Each configuration of layout in the Pareto front

represents a balance between the compactness and the efficiency of the routing in a layout. The GA optimized solutions are performing better compared to baseline layout(represented as a red star being positioned at 810,000mm length and 3.0 path ratio) which confirms the effectiveness of the algorithm. The Layout A on the left top corners achieves minimization at the expense of routing efficiency with a length of 580,00mm to 3.2 path ratio. Resulting with %29 reduction in total length to baseline layout. Layout B adopting a more balanced approach with 645,00mm and 2.38 path ratio this solutions performs well within balancing the both objectives. Layout B achieves %21 reduction in conveyor length(167,00mm saving form the length) with %15 throughput increase to 117 stacks per hour. Layout C located at the bottom right with length of 7120,00mm and the lowest pat ratio with 1.94 this configuration prioritizes the flow efficiency. The path ratio being the closes to one meaning that it was the closest to theoretical minimum possible length. Chge color gradient shows strong a correlation between the routing efficiency and production capacity. Solutions in the lower region of Pareto front observed in lighter colors also representing higher throughput layouts(being greater than 115 stacks per hour). The relation of reduced material travel distance increasing the production rates also show that efficient routing can be potentially dealing better with bottlenecks in the system. The distribution of 38 feasible solution across the objective space demonstrates the effectiveness of GA maintaining diversity throughout the evolution as showing better performance compared to baseline layout.

6.4 KPI comparison with baseline

Table 9: Baseline vs. GA layouts (KPIs)

Layout	**Length [mm]**	**Path-ratio**	**Throughput**	**Util. [%]**
Baseline	812 000	3.01	102	54.3
A (Lean)	592 000	3.22	110	61.5
B (Bal.)	645 000	2.38	117	66.9
C (Flow)	708 000	1.94	119	68.1

Table 9 provides a quantitative comparison between the baseline layout and three GA generated configurations from the Pareto front with four key parameters being the infrastructure cost(total length), routing efficiency(path ratio), production capacity (throughput) and machine utilization. All GA generated solutions show reduction in the infrastructure cost from %27.1 on configuration A, %20.6 on configuration B and %12.8 n configuration C. These reduction of total length directly relate to capital saving of a facility infrastructure. The third column

of the table presents path ratio performance showing the trade off between compactness and route efficiency of a layout. The lean configuration prioritizing the minimum infrastructure does not show much improvement on the path efficiency. Reaching a path ratio of 3.22 shows inefficient routing resulting in less throughput with 110 stacks per hour which is also directly related to lack of bottleneck management where the machine utilization also staying low %54.3. The balanced configuration B shows good improvement on both the reduction in total length with 167,00mm saving and balancing this objective with routing efficiency of 3.22. Also results in a moderately better utilization rate and throughput compared to lean configuration. The flow configuration has the highest length with 708.00mm but the most efficient path building in the layout with 1.94. This results with the highest utilization rate %68.1 and throughput 119 stacks per hour. This is a expected result since configuration c prioritizes the operational efficiency over infrastructure costs(total length). Overall all the configuration seems to perform better than the baseline layout.

6.5 Turn-table ablation study

To evaluate the impact of turntable insertion in layouts an ablation study completed. Three cases were tested: never insert, conditional insert and insert at direction change. The table below show no effect of turntable in the total length of a stack to take.

Table 10: Effect of turn-table policy on layout composition

Policy	**Length [mm]**	**Turntables**	**Components**
Never	481,101	0	65
Auto	481,101	1	66
Always	481,101	3	68

6.6 Sensitivity to order mix

In optimization context, it is common practice to test the solution with different complexities of data. Repeating the GA on orderslowmix.xlsx (had 24 batches) and ordershighmix.xlsx (had 100 batches) does not affect the Pareto front the GA finds the most optimized solution for both order types. This means that GA is working well and that it found a robust layout that can handle the complex orders efficiently.

7 Discussion limitations

This chapter presents an analysis of the AI enhanced generative design system which is developed for VDB. In this analysis examining the performance, implications and alignment with the paper's research objectives are found. Current chapter is going to follow the structure as: what the results showed, what they meant for the industry, why they occurred and future insights for implementation on business case.

7.1 Interpretation of Results

The GA achieved measurable improvements over the baseline layout across the key metrics. The balanced layout option B demonstrated %20 reduction in both conveyor length and the path ratio. The machine utilization and the throughput also showed increase. These improvements directly address VDB's core challenges of inefficient layouts and unpredictable performance. The business case can save great amount of money with 21 percent of reduction in conveyor length. This capital saving in the industry is carrying a high potential to increase the chance of survival of VDB in the competitive sector.

7.2 Algorithm Behaviour Analysis

The rapid convergence with reaching optimal solutions within 6 generations can be attributed to three key design choices; Process level encodings rather than raw coordinates (reducing the search space), two-tier evaluation (balancing the cost of time and total number of evaluated layouts) and intelligent building blocks (guaranteeing the feasible layouts and eliminating the infeasible ones which plagued the initial prototypes).

7.3 Addressing main and sub RQ's

7.3.1 Main-RQ:

The research demonstrates high promise that AI-enhanced generative design through NSGA-II based hybrid approach can be effective in optimizing facility layouts by: automating the layout generation process (which may took weeks), evaluating thousands of configurations rather than solely relying on human intuition, balancing multiple objectives and integrating with simulation tools.

7.3.2 Sub-RQ 1:

Given the time and money budget for the project the extent of computational optimization is substantial achieving throughput improvements while reducing the infrastructure costs. Even with the current simplistic approach the GA's ability is far beyond a human's capability of exploring 800 layouts under 60 minutes.

7.3.3 Sub-RQ 2:

Critical parameters that are identified are: machine processing times (affecting bottleneck identification), buffer capacities (affecting the flow of material), conveyor speeds and spatial constraints. The gene encodings were successful in capturing these parameters in computational format.

7.3.4 Sub-RQ 3:

Validation through the discrete event simulation is shown to be essential revealing in early prototypes the geometric optimization alone was not sufficient. The two-tier approach balanced validation thoroughness with computational feasibility.

7.4 Technical Limitations

The current implementation does not include a holistic pipeline which is ready for the business case for VDB where different visualizing tools are utilized in different stakeholders. The results from the comparison with the optimized layout does not reflect results in a full scoped factory scale it lacks the complexity of a real factory layout. This reference layout was designed as a proof of concept for testing the simulation environment rather than representing a real world factory configuration which is given by the company. The discrete event simulation is not resembling the real life to full extent such as it is not accounting the acceleration/deceleration dynamics or detailed mechanical constraints. The Pareto front has two objectives but in order to get better prediction NSGA-III need to be integrated for a 3-d analysis of the three objectives. These can all affect the actual performance. The current system does not support the whole library of documents of VDB but only 6 components. The rest of the components and modules need to be developed and integrated. Even when those components are integrated the system might not be keeping up with all the metrics and KPIs that have been reached and more computational hungry approach can be required.

7.5 Practical Constraints

While meeting the system runtime requirement the larger facilities might require longer runtimes or different and stronger models. Even though the order generation is realistic enough to create vast amount of orders the match in machine utilization could not be evaluated. In corrugated cardboard industry a third of a day the corrugator is kept running and the rest two thirds of the day converters are meant to be fully active. This scenario could not be tested due to lack of data of the clients of our project company.

8 Conclusion

This paper has addressed the challenge of facility layout optimization in the corrugated cardboard industry where traditional adopted manual approaches result in weeks to months long design cycles with unpredictable performance. The main objective was to develop and validate an AI enhanced generative design system that could automate layout generation while maintaining integration with existing workflows and simulation tools.

The research successfully developed a hybrid genetic algorithm based optimization system that combines NSGA-II with a two tier evaluation pipeline. The system has demonstrated the ability to : Automate layout generation(from weeks to under 60 minutes), generate multiple Pareto optimal solutions (while balancing the operational efficiency and infrastructural costs), achieving 100 percent layout feasibility within 5 generations (with the use of intelligent spatial validation) and extended simulation capabilities. However, these achievements must be studied within the evaluation matrices. The golden layout (the comparison reference layout) that is provided by the company was a proof of concept test layout not including the optimized layout quality or including the complexity of real factory layout. While the system has shown improvements, these metrics reflect improvement over a non optimized reference rather than industry used working layouts. This lack of multiple realistic baseline layouts limits the ability of this paper to claim the performance improvements. Although, it can be stated that the system successfully demonstrated technical feasibility of AI enhanced layout optimization, and successful framework integration for multi objective optimization in facility designs, as well as successfully combining two tier(high and low fidelity) evaluation to optimization validation.

Reflection on Methodology and Limitations The CRISP-DM framework provided a valuable structure for this paper, especially in maintaining alignment between the business understanding of the company and the technical implementation of the proposed solution. The

evaluation constraints faced were: single baseline comparison limits the validity, the lack of access to real production optimized layout for benchmarking means that the results cannot determine if improvements represent achieving genuine optimization beyond human capabilities. The technical limitations were: simplified simulation assumptions of the real world dynamics, lack of component integration(there are many more transport system components that form a realistic factory layout but only 7 were implemented) and the parameter tuning based on a single test case may not generalized.

Contributions Despite these limitations this paper makes several contributions. Starting with technical contributions, the process level encoding schemes reduced the search space from 10^{90} to5^{12} possibilities, it extended a single layout simulator into a parallel working flexible evaluation tool. The two tier evaluation pipeline reduced simulation calls %80 while maintaining the solution quality. Even though it would be false to claim definitive performance superiority, the research proves automated layout generation is feasible. Multi objective optimisation is able to balance multiple industry goals and ai enhanced approaches can explore solution spaces beyond human capability.

8.1 Recommendations for project company VDB

In order to develop a deployable system, the company must take steps in many facets of their production. The company should begin with introducing a small number of transport system and continue to increase the number of systems while validating performance in a single facility. The following changes should also be implemented for a deployable system: standardize the order file formats, establish representative production scenarios. Develop educative workshops for sales engineers interpreting the Pareto fronts, connect and map the points between the existing workflow and the optimizer, add more complex technologies (the stack sucking robotic arm, AGVs), create an easily understandable web based UI, implement metrics to compare the optimized layouts with actual operational performance, deploy real-time optimization with real-time sensing and actuating by supporting the system to be dynamic re-optimization based on real-time data, and finally optimize in multiple facilities and integrate machine learning and train on the real-time data.

8.2 Future work

The multi-objective expansion is one of the most crucial next steps for the GA-based system. The system can be beneficial in multiple aspects, such as, energy consumption and component

cost. Another crucial future extension can be the dynamic layouts for DFLP by developing reconfigurable layouts that also adapt to changing product demands without any physical reconstruction. Machine learning integration using the historical performance data to train on and improve the simulation accuracy and optimization efficiency will boost the capabilities of the system to a next level.

This multi-fidelity surrogate and discrete simulation based hybrid GA model represents an approach of balancing fidelity and computational cost. This system serves primarily as a proof-of-concept for AI-enhanced layout optimization rather than a deployable industrial solution. Despite the limitations and constraints of this paper this research contributes as a step towards the Industry 4.0 vision of intelligent and adaptive manufacturing. By automating layout optimization and maintaining human oversight, the system offers a path for traditional manufacturers to benefit from AI models and methods, leading to a better chance to compete in the industry.

Declaration on Use of AI Tools

During the development of this thesis, I used OpenAI's ChatGPT to support the research process in the following ways: to brainstorm and refine ideas for the methodology and to review the papers grammar and academic tone. The content and analysis presented in this work reflect my own understanding, interpretation and original contributions. AI tool used only to enhance the quality and readability of the final text

9 References

References

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014. doi: 10.1007/s12599-014-0334-4.

[2] G. B. B. Vieira, G. S. Pasa, M. B. N. O. Borsa, G. S. Milan, and A. Pandolfo, "Materials handling management: A case study," *Journal of Operations and Supply Chain Management*, vol. 4, no. 2, pp. 19–30, Jul.–Dec. 2011.

[3] A. Kusiak, "Smart manufacturing," *International Journal of Production Research*, vol. 56, no. 1–2, pp. 508–517, 2018.

[4] P. Zheng, S. Lin, C. Chen, and X. Xu, "Smart, connected open architecture product: An IT-driven co-creation paradigm with lifecycle personalization concerns," *International Journal of Production Research*, vol. 56, no. 8, pp. 2661–2682, 2018.

[5] B. A. Peters, "Genetic algorithms for layout optimization in manufacturing systems," *Computers & Industrial Engineering*, vol. 35, no. 3–4, pp. 535–538, 1998.

[6] "Transport Systems," *VandenBos CM.* [Online]. Available: https://vandenbos-cm.nl/en/products/transport-systems/ [Accessed: 01-Apr-2025].

[7] T. Pereira, A. S. L. Neves, F. J. G. Silva, R. Godina, L. Morgado, and G. F. L. Pinto, "Production Process Analysis and Improvement of Corrugated Cardboard Industry," *Procedia Manufacturing*, vol. 51, pp. 1395–1402, 2020. doi: 10.1016/j.promfg.2020.10.194.

[8] T. Weise, "Metaheuristics," in *Global Optimization Algorithms – Theory and Application*, 2009. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/metaheuristics. [Accessed: 08-Apr-2025].

[9] V. Sharma and A. Kumar, "Genetic Algorithms," in *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems*, New Delhi: Springer India, 2015, pp. 15-37.

[10] M. Rajasekharan, B. A. Peters, and T. Yang, "A Genetic Algorithm For Facility Layout Design In Flexible Manufacturing Systems," *International Journal of Production Research*, vol. 36, no. 7, pp. 1–22, Jul. 1998, doi: 10.1080/002075498193958.

[11] D. Bertsimas and J. Tsitsiklis, "Simulated Annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, Feb. 1993, doi: 10.1214/ss/1177011077.

[12] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, "CRISP-DM 1.0: Step-by-step data mining guide," SPSS Inc., 2000. [Online]. Available: https://web.archive.org/web/20220401041957/https://www.the-modeling-agency.com/crisp-dm.pdf

[13] A. R. McKendall Jr., J. Shang, and S. Kuppusamy, "Simulated annealing heuristics for the dynamic facility layout problem," *Computers & Operations Research*, vol. 33, no. 8, pp. 2431–2444, Aug. 2006, doi: 10.1016/j.cor.2005.02.021.

[14] F. Glover and M. Laguna, "Tabu Search," in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer, 1998, pp. 2093–2229.

[15] M. Mir and M. H. Imam, "A hybrid optimization approach for layout design of unequal-area facilities," *Computers & Industrial Engineering*, vol. 39, no. 1-2, pp. 49–63, 2001, doi: 10.1016/S0360-8352(00)00065-6.

[16] "Pareto Optimality," *ScienceDirect*, Elsevier. [Online]. Available: https://www.sciencedirect.com/topics/engineering/pareto-optimality. [Accessed: 17-May-2025].

[17] A. Patel, "Introduction to A*: Heuristics," *Stanford Theory Group*, Stanford University. [Online]. Available: https://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html. [Accessed: 01-Jan-2025].

[18] H. Pourvaziri and B. Naderi, "A hybrid multi-population genetic algorithm for the dynamic facility layout problem," *Applied Soft Computing*, vol. 24, pp. 457–469, Nov. 2014, doi: 10.1016/j.asoc.2014.06.051.

[19] G. Schuh, P. Burggräf, M. Lenders, and F. Breitenbach, "Towards the next level of facility layout planning: Combining digital twins and AI-driven optimization,"

[20] M. R. Chalak Qazani, H. Parvaz, and S. Pedrammehr, "Optimization of fixture locating layout design using comprehensive optimized machine learning," The International Journal of Advanced Manufacturing Technology, vol. 122, pp. 2701–2717, 2022.

[21] M. Klar, M. Glatt, and J. C. Aurich, "An implementation of a reinforcement learning based algorithm for factory layout planning," Manufacturing Letters, vol. 30, pp. 1–4, 2021.

[22] S. H. Choi and B. S. Kim, “Intelligent factory layout design framework through collaboration between optimization, simulation, and digital twin,” Journal of Intelligent Manufacturing, vol. 36, pp. 1547–1561, 2025.

[23] M. Besbes, M. Zolghadri, R. C. Affonso, et al., “A Methodology for Solving Facility Layout Problem Considering Barriers: Genetic Algorithm Coupled with A* Search,” Journal of Intelligent Manufacturing, vol. 31, no. 3, pp. 615-640, 2020.

[24] A. Mader and W. Eggink, “A Design Process forfor Creative Technology,” in *International Conference on Engineering and Product Design Education*, University of Twente, The Netherlands, Sep. 4-5, 2014, pp. 1-6.

[25] Priya Darshini A. and Arun Kumar S., ”Augmented Reality for Seamless Integration with STEM Education,” International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), vol. 8, no. 5, pp. 195-204, 2021. doi: 10.32628/IJSRSET218126.

[26] M. Rajasekharan, B. A. Peters, and T. Yang, ”A Genetic Algorithm for Facility Layout Design in Flexible Manufacturing Systems,” International Journal of Production Research, vol. 36, no. 1, pp. 95-110, 1998.

[27] F. Azadivar and J. Wang, ”Facility Layout Optimization Using Simulation and Genetic Algorithms,” International Journal of Production Research, vol. 38, no. 17, pp. 4369-4383, 2000.

[28] F. G. Paes, A. A. Pessoa, and T. Vidal, ”A Hybrid Genetic Algorithm with Decomposition Phases for the Unequal Area Facility Layout Problem,” European Journal of Operational Research, vol. 256, no. 3, pp. 742-756, 2017.

[29] N. Beri and R. Manickam, ”Genetic Algorithm-Assisted Visualization in Industrial Metaverse: Factory Layout Planning,” in Industry 4.0 and Advanced Manufacturing (I-4AM 2024), LNME, 2025, pp. 165-177.

[30] T. Dunker, G. Radons, and E. Westkämper, ”Combining Evolutionary Computation and Dynamic Programming for Solving a Dynamic Facility Layout Problem,” European Journal of Operational Research, vol. 165, no. 3, pp. 55-69, 2005.

[31] J. Chae and B. A. Peters, ”A Simulated Annealing Algorithm Based on a Closed Loop Layout for Facility Layout Design in Flexible Manufacturing Systems,” International Journal of Production Research, vol. 44, no. 22, pp. 4567-4587, 2006.

[32] P. C. Kulkarni and K. Shankar, "Genetic Algorithm for Layout Problems in Cellular Manufacturing Systems," in Proc. IEEE Int'l Conference on Industrial Engineering and Engineering Management, 2007.

[33] V. M. Pillai, I. B. Hunagund, and K. K. Krishnan, "Design of Robust Layout for Dynamic Plant Layout Problems," Computers and Industrial Engineering, vol. 61, no. 3, pp. 813-823, 2011.

[34] K. S. N. Ripon, K. Glette, M. Hovin, and J. Torresen, "A Multi-Objective Evolutionary Algorithm for Solving Integrated Scheduling and Layout Planning Problems in Manufacturing Systems," in Proc. IEEE Conference on Evolving and Adaptive Intelligent Systems, 2012.

[35] H. Tong, C. Huang, L. L. Minku, and X. Yao, "Surrogate models in evolutionary single-objective optimization: A new taxonomy and experimental study," Information Sciences, vol. 567, pp. 146–168, 2021. doi: 10.1016/j.ins.2021.02.039.

[36] H. Pourvaziri and B. Naderi, "A Hybrid Multi-Population Genetic Algorithm for the Dynamic Facility Layout Problem," Applied Soft Computing, vol. 24, pp. 457-469, 2014.

[37] J. M. Palomo-Romero, L. Salas-Morera, and L. García-Hernández, "An Island Model Genetic Algorithm for Unequal Area Facility Layout Problems," Expert Systems with Applications, vol. 68, pp. 151-162, 2017.

[38] Y. Peng, T. Zeng, L. Fan, Y. Han, and B. Xia, "An Improved Genetic Algorithm-Based Robust Approach for Stochastic Dynamic Facility Layout Problems," Discrete Dynamics in Nature and Society, vol. 2018, Article ID 1529058, 2018.

[39] S. Tejashwin, R. Ghadai, K. Patil, and S. K. Pramanick, "A novel genetic algorithm approach for optimizing facility layout of flexible manufacturing systems with looped-layout material flow pattern," *Applied Soft Computing*, vol. 124, p. 109035, 2022. doi: 10.1016/j.asoc.2022.109035.

[40] A. Palange and P. Dhatrak, "Lean Manufacturing: A Vital Tool to Enhance Productivity in Manufacturing," *Materials Today: Proceedings*, vol. 46, pp. 729–736, 2021. doi: 10.1016/j.matpr.2020.12.193.

[41] L. Liang and W. Chao, "The strategies of tabu search technique for facility layout optimization," *Automation in Construction*, vol. 17, pp. 657–669, Aug. 2008. doi: 10.1016/j.autcon.2008.01.001.

[42] Bentley, J. L., & Ottmann, T. A. (1979). Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers, 28(9), 643-647.

[43] de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). Computational geometry: Algorithms and applications (3rd ed.). Springer-Verlag.

Appendices

A Technical Implementation Details

A.1 Complete Chromosome Structure

Table 11: Complete gene mapping for enhanced GA chromosome

Gene	Type	Value Range	Description
0	machine_order[0]	0-2	First machine in sequence (0: Print, 1: Cut, 2: Glue)
1	machine_order[1]	0-2	Second machine in sequence
2	machine_order[2]	0-2	Third machine in sequence
3	wip_type	0-2	Buffer configuration (0: single, 1: twin, 2: bypass)
4	corridor_slice	0-2	Y-axis routing preference (0: top, 1: mid, 2: bottom)
5	turntable_policy	0-2	Turntable insertion (0: never, 1: auto, 2: always)
6	speed_tier[0]	0-2	Conveyor speed tier 1 (0: slow, 1: medium, 2: fast)
7	speed_tier[1]	0-2	Conveyor speed tier 2
8	speed_tier[2]	0-2	Conveyor speed tier 3
9-11	reserved	0	Reserved for future transport modes

A.2 A* Pathfinding Algorithm Implementation

Listing 7: Core A* pathfinding implementation with octile heuristic

```
def astar_path(start, goal, grid, obstacles):
    """
    A* pathfinding with octile distance heuristic

    Args:
        start: (x, y) starting position in mm
        goal: (x, y) goal position in mm
        grid: Grid object with cell_size
        obstacles: List of obstacle dictionaries

    Returns:
        List of (x, y) waypoints or None if no path exists
    """
    def octile_distance(dx, dy):
```

```
        D = grid.cell_size  # 5000mm
        D2 = D * math.sqrt(2)  # diagonal cost
        return D * (dx + dy) + (D2 - 2*D) * min(dx, dy)

    def get_neighbors(node):
        x, y = node
        neighbors = []
        # 8-directional movement
        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1),
                       (-1,-1), (1,-1), (-1,1), (1,1)]:
            nx, ny = x + dx, y + dy
            if (0 <= nx < grid.width and 0 <= ny < grid.height
                and not is_obstacle(nx, ny, obstacles)):
                neighbors.append((nx, ny))
        return neighbors

    # Convert mm to grid coordinates
    start_grid = grid.mm_to_grid(start)
    goal_grid = grid.mm_to_grid(goal)

    # Priority queue: (f_score, g_score, node, path)
    open_set = [(0, 0, start_grid, [start_grid])]
    closed_set = set()
    g_scores = {start_grid: 0}

    while open_set:
        f, g, current, path = heapq.heappop(open_set)

        if current == goal_grid:
            # Convert path back to mm coordinates
            return [grid.grid_to_mm(p) for p in path]

        if current in closed_set:
            continue
        closed_set.add(current)

        for neighbor in get_neighbors(current):
            if neighbor in closed_set:
                continue

            # Calculate movement cost
            dx = abs(neighbor[0] - current[0])
```

```
            dy = abs(neighbor[1] - current[1])
            move_cost = grid.cell_size * math.sqrt(dx*dx + dy*dy)

            tentative_g = g + move_cost

            if neighbor not in g_scores or tentative_g < g_scores[neighbor]:
                g_scores[neighbor] = tentative_g
                h = octile_distance(
                    abs(neighbor[0] - goal_grid[0]),
                    abs(neighbor[1] - goal_grid[1])
                )
                f_score = tentative_g + h
                heapq.heappush(open_set,
                    (f_score, tentative_g, neighbor, path + [neighbor]))

    return None  # No path found
```

B Data Structures and Formats

B.1 Complete JSON Schema

Listing 8: Complete JSON schema for facility layouts

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Facility Layout Schema",
  "type": "object",
  "required": ["bounds", "components", "connections"],
  "properties": {
    "bounds": {
      "type": "object",
      "properties": {
        "x_min": {"type": "number"},
        "x_max": {"type": "number"},
        "y_min": {"type": "number"},
        "y_max": {"type": "number"}
      },
      "required": ["x_min", "x_max", "y_min", "y_max"]
    },
    "obstacles": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "type": {"enum": ["pillar", "wall", "restricted_area"]},
```

```
        "x": {"type": "number"},
        "y": {"type": "number"},
        "radius": {"type": "number"},
        "width": {"type": "number"},
        "height": {"type": "number"}
      }
    }
  },
  "components": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "id": {"type": "string"},
        "type": {"enum": ["corrugator", "conveyor", "machine",
                          "wip", "transfer_car", "eol"]},
        "x": {"type": "number"},
        "y": {"type": "number"},
        "length": {"type": "number"},
        "width": {"type": "number"},
        "angle": {"type": "number"},
        "speed": {"type": "number"},
        "capacity": {"type": "number"},
        "nr_of_conveyors": {"type": "integer"},
        "batch_time": {"type": "number"},
        "width_change_factor": {"type": "number"},
        "length_change_factor": {"type": "number"}
      },
      "required": ["id", "type", "x", "y"]
    }
  },
  "connections": {
    "type": "object",
    "patternProperties": {
      ".*": {
        "type": "array",
        "items": {"type": "string"}
      }
    }
  },
  "anchors": {
    "type": "object",
    "patternProperties": {
      ".*": {
        "type": "object",
        "properties": {
          "x": {"type": "number"},
          "y": {"type": "number"},
          "fixed": {"type": "boolean"}
```

```
          }
        }
      }
    }
  }
}
```

C Additional Algorithms and Methods

C.1 Surrogate Evaluation Function

Listing 9: Surrogate evaluation implementation

```
class SurrogateEvaluator:
    """Fast geometric evaluation without simulation"""

    def evaluate(self, layout):
        """
        Returns: [total_length, overlap_penalty,
                  connectivity_penalty, path_ratio]
        """
        metrics = np.zeros(4)

        # 1. Total conveyor length
        total_length = 0
        conveyors = [c for c in layout['components']
                     if c['type'] == 'conveyor']
        for conv in conveyors:
            total_length += conv.get('length', 0)
        metrics[0] = total_length

        # 2. Overlap penalty
        overlap_count = 0
        components = layout['components']
        for i in range(len(components)):
            for j in range(i+1, len(components)):
                if self._components_overlap(components[i],
                                            components[j]):
                    overlap_count += 1
        metrics[1] = overlap_count * 10000  # Heavy penalty

        # 3. Connectivity check
        disconnected = 0
```

```
        connections = layout.get('connections', {})
        for comp_id, outputs in connections.items():
            for output_id in outputs:
                if not self._connection_valid(comp_id,
                                              output_id,
                                              components):
                    disconnected += 1
        metrics[2] = disconnected * 12000

        # 4. Path ratio (actual / Manhattan)
        path_ratio = self._calculate_path_ratio(layout)
        metrics[3] = path_ratio

        return metrics

    def _components_overlap(self, c1, c2):
        """AABB collision with 500mm clearance"""
        clearance = 500

        # Get bounding boxes
        bb1 = self._get_bounding_box(c1)
        bb2 = self._get_bounding_box(c2)

        # Expand by clearance
        bb1 = self._expand_box(bb1, clearance)
        bb2 = self._expand_box(bb2, clearance)

        # Check overlap
        return not (bb1['x_max'] < bb2['x_min'] or
                    bb2['x_max'] < bb1['x_min'] or
                    bb1['y_max'] < bb2['y_min'] or
                    bb2['y_max'] < bb1['y_min'])

    def _calculate_path_ratio(self, layout):
        """Ratio of actual path to Manhattan distance"""
        # Find key path: Corrugator -> Machine -> EOL
        corrugator = next(c for c in layout['components']
                          if c['type'] == 'corrugator')
        eol = next(c for c in layout['components']
                   if c['type'] == 'eol')

        # Manhattan distance
```

```
        manhattan = (abs(eol['x'] - corrugator['x']) +
                     abs(eol['y'] - corrugator['y']))

        # Actual path length (sum of conveyors in main flow)
        actual = self._trace_path_length(layout,
                                         corrugator['id'],
                                         eol['id'])

        return actual / manhattan if manhattan > 0 else 999
```

C.2 Two-Tier Fitness Merge Strategy

Listing 10: Fitness merge implementation

```
def merge_fitness_scores(surrogate_scores, des_scores, elite_mask):
    """
    Merge surrogate and DES evaluations

    Args:
        surrogate_scores: (pop_size, n_obj) from fast eval
        des_scores: (pop_size, n_obj) with NaN for non-elites
        elite_mask: (pop_size,) boolean array

    Returns:
        merged_scores: (pop_size, n_obj) final fitness
    """
    merged = np.copy(surrogate_scores)

    # Override with DES scores where available
    valid_des = ~np.isnan(des_scores[:, 0])
    merged[valid_des] = des_scores[valid_des]

    # Log statistics
    n_des = np.sum(valid_des)
    print(f"Evaluated␣{n_des}/{len(elite_mask)}␣with␣DES")

    # Calculate RMSE for validation
    if n_des > 0:
        common_mask = elite_mask & valid_des
        if np.any(common_mask):
            surr_vals = surrogate_scores[common_mask]
            des_vals = des_scores[common_mask]
```

```
29
30              rmse = np.sqrt(np.mean((surr_vals - des_vals)**2))
31              print(f"Surrogate RMSE: {rmse:.2f}")
32
33              # Warn if surrogate is diverging
34              if rmse > 50000:  # 50m threshold
35                  warnings.warn("Surrogate accuracy degrading")
36
37      return merged
```

D Experimental Data

D.1 Sample Order Mix Specifications

Table 12: Characteristics of test order sets

Order Set	Batches	Orders	Sheets	Mix	Complexity
orders_low.xlsx	24	72	125,840	Uniform	Low
orders_mix120.xlsx	34	102	203,576	Mixed	Medium
orders_high.xlsx	100	300	612,450	Diverse	High
orders_stress.xlsx	150	450	1,024,800	Extreme	Very High

E Glossary of Terms

AABB Axis-Aligned Bounding Box - simplified collision detection method

Chromosome Encoded representation of a layout solution in the GA

Corrugator Machine producing corrugated cardboard from paper rolls

CRISP-DM Cross Industry Standard Process for Data Mining

DES Discrete Event Simulation - time-based simulation method

DFLP Dynamic Facility Layout Problem

EOL End of Line - final collection point in production

FLP Facility Layout Problem

GA Genetic Algorithm - evolutionary optimization method

HV Hypervolume - multi-objective performance metric

MHC Material Handling Cost

NSGA-II Non-dominated Sorting Genetic Algorithm II

Octile Distance
Heuristic for 8-directional grid pathfinding

Pareto Front Set of non-dominated solutions in multi-objective optimization

Path Ratio Actual path length divided by Manhattan distance

RMSE Root Mean Square Error - accuracy metric

SimPy Python framework for discrete event simulation

Transfer Car Mobile platform for transporting stacks between conveyors

UA-FLP Unequal Area Facility Layout Problem

WIP Work In Progress - buffer storage area

F List of Abbreviations

Abbreviation	**Full Form**
AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
DEAP	Distributed Evolutionary Algorithms in Python
FIP-AM@UT	Fraunhofer Innovation Platform for Advanced Manufacturing
FR	Functional Requirement
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LC	Layout Configurator
ML	Machine Learning
NFR	Non-Functional Requirement
OG	Order Generator
PyQt5	Python Qt5 GUI Framework
RL	Reinforcement Learning
SA	Simulated Annealing
TS	Tabu Search
UI	User Interface
VDB	Van den Bos
XML	eXtensible Markup Language