

Resource reconfiguration and optimization
in Two-sided Robotic Assembly Line
Balancing Problem

BY

Bayan Ahmed Alzain

A Thesis Presented to the

COLLEGE OF GRADUATE AND INTERDISCIPLINARY STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

INDUSTRIAL AND SYSTEMS ENGINEERING

December 2024

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **BAYAN AHMED ALZAIN** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN INDUSTRIAL AND SYSTEMS ENGINEERING**.

Thesis Committee

Dr. Haitham Saleh (Adviser)

Prof. Salih Duffuaa (Member)

Dr. Anas Alghazi (Member)

Dr. Anas Alghazi
Department Chairman

Prof. Suliman Al-Homidan
Dean of Graduate Studies

Date



©Bayan Ahmed Alzain
2024

This thesis is dedicated to my beloved family, whose constant support and encouragement have been the cornerstone of my journey.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Haitham Saleh, for his invaluable guidance, support, and encouragement throughout this thesis journey.

I am also grateful to my committee members, Dr. Anas Alghazi, and Prof. Salih Duffuaa, for their insightful feedback and suggestions, which significantly improved the quality of this work.

Special thanks to my family and friends for their constant support and motivation during challenging times.

Lastly, I would like to thank King Fahad University of Petroleum and Minerals, whose funding and resources made this research possible.

Contents

ACKNOWLEDGMENTS	v
LIST OF TABLES	ix
LIST OF FIGURES	xi
LIST OF SYMBOLS	xiii
LIST OF ABBREVIATIONS	xiv
ABSTRACT (ENGLISH)	xv
ABSTRACT (ARABIC)	xviii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement: The Rebalancing Challenge	2
1.2 Research Objectives	3
1.2.1 Objective 1: Extend the Reconfiguration Model to Include Two-Sided Assembly Lines	3
1.2.2 Objective 2: Address Multi-objective Optimization	4
1.2.3 Objective 3: Solve the Reconfiguration Model for Two- Sided Robotic Assembly Line	4
1.3 Research Questions	5
CHAPTER 2 LITERATURE REVIEW	8
2.1 Reconfigurable Manufacturing Systems	8
2.2 Two-Sided Assembly Lines	11

2.2.1	Two-Sided Robotic Assembly Line Balancing	13
2.3	TALBP Solution Methodologies	15
2.3.1	Exact and heuristic methods	16
2.3.2	Solution Methodologies for RALBP	18
2.3.3	Solution Methodologies for TRALBP	20
2.3.4	Multi-Objective Mathematical Programming	21
2.4	Current Research Gaps	23
CHAPTER 3	MATHEMATICAL FORMULATION	25
3.1	Problem formulation	25
3.1.1	Assembly line rebalancing approach	26
3.1.2	Model assumptions	27
3.1.3	Notations	28
3.1.4	Cost model	31
3.1.5	Mathematical formulation	33
3.1.6	Linearization	37
3.2	An illustrated example	42
CHAPTER 4	SOLUTION METHODOLOGIES	45
4.1	Exact Methods	45
4.2	Meta Heuristic Approaches	46
4.2.1	Encoding And Decoding	47
4.2.2	Neighbor Structure	51
4.2.3	Non-dominated Sorting Genetic Algorithm II (NSGA-II)	52
4.2.4	Restarted Simulated Annealing Algorithm (RSA)	57
4.2.5	Restart Mechanism	61
4.2.6	Cooling schedule	62
CHAPTER 5	EVALUATION EXPERIMENTS	64
5.1	Benchmark of instances	65
5.2	Parameter calibration	67

5.3	Experiment Design	70
5.4	Experiment Results	71
5.5	Comparison matrices	76
CHAPTER 6 RESULTS AND CONCLUSION		86
6.1	Evaluation metrics	86
6.2	Solver Capability	87
6.3	Exact Model Results	88
6.3.1	Results Discussion	94
6.4	RSA results	97
6.4.1	Results Discussion	104
6.5	RSA vs solver	106
6.6	Conclusion	107
APPENDIX A SUPPLEMENTARY MATERIAL		111
REFERENCES		127
VITAE		139

List of Tables

Tab. 3.1	All possible products of binary variables $z_{ijkp} := x_{ijk} \cdot x_{pjk}$	41
Tab. 3.2	All possible products of binary variables $z_{irjk} := x_{ijk} \cdot y_{rjk}$	41
Tab. 3.3	All possible scenarios of linear constraints	42
Tab. 3.4	Numerical example	42
Tab. 3.5	Cost & Energy Data	43
Tab. 3.6	Capability Matrix	43
Tab. 3.7	Gurobi Solver Outcomes	44
Tab. 4.1	Decoding procedure for TRALBP-TT with capability aspect	50
Tab. 4.2	Pseudocode for NSGA-II	55
Tab. 4.3	Non-dominated sorting algorithm	57
Tab. 5.1	The parameters and their levels	71
Tab. 5.2	Orthogonal arrays for RSA parameters	72
Tab. 5.3	Orthogonal arrays for NSGA-II parameters	72
Tab. 5.4	Selected levels for all parameters of both algorithms. .	75
Tab. 5.5	Average results from the three evaluation metrics for different problems.	78
Tab. 5.6	Summary of Results for CP Metric	80
Tab. 5.7	Summary of Results for SP Metric	81
Tab. 5.8	Minimum objective values achieved by both NSGA-II and RSA.	85
Tab. 6.1	Evaluation metrics	87

Tab. 6.2	Performance of Gurobi solver in solving problems across Greenfield and Brownfield scenarios	88
Tab. 6.3	Test results of P65	106
Tab. 6.4	Random trial RSA runs for problems P12 and P16, under Brownfield mode settings with two mated stations given $nr = 20$	107
Tab. A.1	P9: Task Dependencies: Predecessors and Successors .	111
Tab. A.2	P9: Task Types Assignment	112
Tab. A.3	P9: Robot Capabilities for Tasks 1 to 9	112
Tab. A.4	P9: Robot Processing Times for Tasks 1 to 9	112
Tab. A.5	Robot Costs	112
Tab. A.6	Energy Consumption Per Robot (PCr)	112
Tab. A.7	P24: Robot Processing Times for Tasks 1 to 24	113
Tab. A.8	P24: Capability Matrix for Robots (Tasks 1 to 24) . .	113

List of Figures

Fig. 2.1 Reconfigurable Flow Lines [3].	11
Fig. 2.2 Two-sided Robotic Assembly Line	15
Fig. 3.1 Flowchart of Greenfield and Brownfield Approaches	26
Fig. 4.1 Task permutation and robot vector	48
Fig. 4.2 Insert and swap operators	52
Fig. 4.3 Multi-point crossover	53
Fig. 5.1 Non-dominated solutions in the TP	69
Fig. 5.2 Main Effects Plots for Convergence and S/N Ratio (RSA) . .	73
Fig. 5.3 Main Effects Plots for Convergence and S/N Ratio (NSGA-II)	74
Fig. 5.4 NSGA-II and RSA Pareto fronts at $r = 60$	83
Fig. 5.5 NSGA-II and RSA Pareto fronts at $r = 80$	84
Fig. 6.1 Interaction Plots of run times across different scenarios	89
Fig. 6.2 Interaction Plots of mean efficiency across different scenarios .	90
Fig. 6.3 Interaction Plots of mean total cost across different scenarios	91
Fig. 6.4 Interaction Plots of mean cycle time across different scenarios	92
Fig. 6.5 Interaction Plots of mean total energy consumption across dif- ferent scenarios	93
Fig. 6.6 3D Plots showing the interaction between cycle time, total en- ergy, and total cost for different problem setups (P9, P12, P16, P24) in Greenfield and Brownfield scenarios	94
Fig. 6.7 Comparative Analysis of Average Cycle Time (CT) for Large Cases Between Greenfield and Brownfield Modes Across Vari- ous Robot Alternatives ($nr = 20$ to 80).	98

Fig. 6.8 Comparative Analysis of Total Energy (TE) for Large Cases Between Greenfield and Brownfield Modes Across Various Robot Alternatives ($nr = 20$ to 80).	99
Fig. 6.9 Comparative Analysis of Total Cost (TC) for Large Cases Be- tween Greenfield and Brownfield Modes Across Various Robot Alternatives ($nr = 20$ to 80).	101
Fig. 6.10 Comparative Analysis of Average Robot Numbers for Large Cases Between Greenfield and Brownfield Modes Across Vari- ous Alternatives ($nr = 20$ to 80).	102
Fig. 6.11 Comparative Analysis of Efficiency (E) for Large Cases Be- tween Greenfield and Brownfield Modes Across Various Robot Alternatives ($r = 20$ to 80).	104

LIST OF SYMBOLS

C_{inv}	Investment cost, in \$
C_{prc}	Processing cost, in \$
C_{sav}	Savings cost, in \$
d_{rjk}	Binary variable: 1, if robot r is initially assigned to mated station j at side k ; 0, otherwise
$\underline{EC}$	Vector of investment costs for robots, in \$
f_{rec}	Labor cost of reconfiguration expressed in dollars per second (\$/s)
$\underline{PC}$	Vector of processing costs per second for robots, in \$/s
PC_r	Energy consumption of robot r per time unit when operational
pc_r	Processing cost per second for robot r , in \$/s
$\underline{SC}$	Vector of savings costs for robots, in \$
sc_r	Savings cost for robot r , in \$
SPC_r	Energy consumption of robot r per time unit in standby mode
$t_{a,r}$	Assembly time for robot r , in seconds
$t_{d,r}$	Disassembly time for robot r , in seconds
t_{ir}	Time taken by robot r to process task i
$t_{m,r}$	Moving time for robot r , in seconds

LIST OF ABBREVIATIONS

ALBP	Assembly Line Balancing Problem
DMLs	Dedicated Manufacturing Lines
FMSs	Flexible Manufacturing Systems
MBO	migrating bird optimization algorithm
MOMP	Multi-Objective Mathematical Programming
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PSO	particle swarm algorithm
RALBP	Robotic ssembly line balancing problem
RAMs	Reconfigurable Assembly Machines
RMS	Reconfigurable Manufacturing System
RM _s	Reconfigurable Machines
RMT	Reconfigurable Machine Tools
RSA	Restarted Simulated Annealing
TALBP	Two-sided assembly line balancing problem
TRALBP	Two-sided Robotic Assembly Line Balancing problem
TRALBP-TT	Two-Sided Robotic Assembly Line Balancing Problem with Task Types

THESIS ABSTRACT

NAME:

TITLE OF STUDY:

MAJOR FIELD: DATE OF DEGREE:

April 24, 2025

Efficiently managing an assembly line involves allocating tasks across equipment and workstations to minimize production costs. In the manufacturing environment, companies encounter obstacles in achieving optimal productivity and cost-effective assembly processes. Additionally, adapting to workload variations or quality standards often calls for the adoption of flexible production systems. To manage these challenges, manufacturing companies recognize the importance of regularly adjusting assembly lines. However, most of the literature in this field focuses on new production facilities that start with no pre-existing resources, commonly referred to as a greenfield approach. In contrast, balancing the line considering existing infrastructure and initial conditions is known as a brownfield approach. Among the various assembly line layouts, a two-sided assembly line has proven to be beneficial compared to a one-sided line in several ways. The core issue in

modern manufacturing is optimizing two-sided robotic assembly lines, particularly under the constraints of brownfield production. Current approaches largely ignore existing resources, leading to a gap in practical and efficient solutions. To address this research problem, this study presents a new mixed-integer programming model for two-sided assembly lines to simultaneously minimize cycle time, total reconfiguration cost, and energy consumption. To evaluate the model, a composite dataset integrating benchmark problems has been utilized. Based on the results from small-sized benchmark problems, the solver demonstrates higher efficiency in Brownfield scenarios, where it benefits from existing infrastructure and initial conditions. This efficiency is evident in faster solving times, lower total costs, and better management of cycle times and energy. In contrast, in Greenfield scenarios, where no existing infrastructure is leveraged, the solver faces greater challenges, particularly as the number of robots and the complexity of problems increase. To manage large-sized cases, two meta-heuristics are implemented: Restarted Simulated Annealing Algorithm (RSA) and Non-dominated Sorting Genetic Algorithm II (NSGA-II). Consequently, new encoding and decoding procedures have been developed to handle the extended model, which includes task types and incorporates robots' capabilities. Based on evaluation metrics, the RSA has provided a more diverse set of high-quality solutions compared to NSGA-II. Furthermore, RSA achieved shorter cycle times not only in small-sized cases but also in large cases with a considerable number of robot alternatives. Across all considered scenarios, the total cost in the Brownfield mode is consistently lower than in the Greenfield mode. However, while RSA provides consistent performance regardless of the

number of robots or tasks, there are limitations in the solver's ability to optimize at larger scales, particularly in Greenfield scenarios.

ملخص الرسالة

الاسم: عنوان الدراسة:

التخصص: تاريخ الدرجة العلمية:

إدارة خطوط التجميع بكفاءة تتطلب تخصيص المهام عبر المعدات ومحطات العمل لتقليل تكاليف الإنتاج. تواجه الشركات في بيئات التصنيع تحديات في تحقيق الإنتاجية المثلى وعمليات التجميع ذات التكلفة الفعالة. بالإضافة إلى ذلك، فإن التكيف مع تقلبات حجم العمل أو معايير الجودة يستدعي في الغالب تبني أنظمة إنتاج مرنة. لمواجهة هذه التحديات، تدرك الشركات أهمية إجراء تعديلات دورية على خطوط التجميع. ومع ذلك، تركز معظم الدراسات في هذا المجال على المرافق الإنتاجية الجديدة التي تبدأ بدون موارد مسبقة، وهو ما يعرف باسم النهج الأخضر (Greenfield). في المقابل، فإن تحقيق التوازن في الخط مع الأخذ بعين الاعتبار البنية التحتية والظروف الأولية القائمة يُعرف بالنهج البني (Brownfield). من بين الترتيبات المختلفة لخطوط التجميع، أثبت خط التجميع ثنائي الجوانب أنه أكثر فائدة مقارنة بالخط أحادي الجانب في عدة جوانب. تتمثل المشكلة الأساسية في التصنيع الحديث في تحسين خطوط التجميع الروبوتية ثنائية الجوانب، خاصة تحت قيود الإنتاج بنهج بني. تتجاهل معظم الأساليب الحالية الموارد المتوفرة، مما يؤدي إلى فجوة في الحلول العملية. لمعالجة هذه المشكلة البحثية، يقدم هذا البحث نموذجًا برمجيًا جديدًا للأعداد الصحيحة المختلطة لخطوط التجميع ثنائية الجوانب بهدف تقليل وقت دورة الإنتاج، إجمالي تكلفة إعادة تهيئة خط الإنتاج، واستهلاك الطاقة في وقت واحد. تم تقييم النموذج باستخدام مجموعة بيانات مدمجة تشمل بيانات لخطوط إنتاج حقيقية. بناءً على النتائج المستخلصة من المشاكل الصغيرة الحجم، يظهر النموذج كفاءة أعلى في سيناريوهات النهج البني، حيث يستفيد من البنية التحتية والظروف الأولية القائمة. تتضح هذه الكفاءة من خلال أوقات حل أسرع، وانخفاض إجمالي التكاليف، وإدارة أفضل لأوقات دورة الإنتاج والطاقة. بالمقابل، يواجه النموذج تحديات أكبر في سيناريوهات النهج الأخضر، حيث لا يتم استغلال البنية التحتية القائمة، خاصة مع زيادة عدد الروبوتات. لإدارة الحالات الكبيرة الحجم، تم تطبيق اثنتين من الخوارزميات: خوارزمية التلدين المحاكي

(RSA) وخوارزمية الفرز الجينية (NSGA-II) ونتيجة لذلك، تم تطوير إجراءات جديدة للترميز وفك الترميز للتعامل مع النموذج الموسع الذي يشمل أنواع المهام وقدرات الروبوتات. بناءً على معايير التقييم، أظهرت خوارزمية RSA مجموعة أكثر تنوعًا من الحلول عالية الجودة مقارنة بـ NSGA-II بالإضافة إلى ذلك، حققت RSA أوقات دورات أقصر ليس فقط في الحالات الصغيرة الحجم ولكن أيضًا في الحالات الكبيرة مع عدد كبير من خيارات الروبوتات. عبر جميع السيناريوهات المدروسة، كانت التكلفة الإجمالية في وضع النهج البني أقل باستمرار مقارنة بوضع النهج الأخضر. ومع ذلك، ورغم الأداء المتسق لخوارزمية RSA بغض النظر عن عدد الروبوتات أو المهام، هناك قيود في قدرة النموذج على تحسين الأداء في الحالات الكبيرة الحجم، خاصة في سيناريوهات النهج الأخضر.

CHAPTER 1

INTRODUCTION

Efficiently managing an assembly line need allocating tasks, across equipment and workstations to minimize production costs. This is commonly referred to as the Assembly Line Balancing Problem (ALBP). Within production environment, companies encounter obstacles when working for optimal productivity and cost-effective assembly processes. These challenges are due to shorter product lifetimes, increased competition, and increasing customer requirements. Furthermore, adapting to variations in workloads or quality standards often requires the adoption of flexible production systems.

In order to deal with these challenges, a lot of manufacturing companies recognize the importance of regularly adjusting assembly lines. Reconfiguring or making modifications to an existing assembly line, including the allocation of assembly tasks to the line workstations, is known as assembly line rebalancing. However, most of the literature in this field focuses on new production facilities that start with no pre-existing resources, commonly referred to as a greenfield approach. Nevertheless, the ever-changing demands of a modern market require

continual adjustments to assembly lines and production capabilities, introducing additional constraints to the ALBP.

Many variations exist in assembly line layouts, among which two-sided assembly lines are highly used to manufacture products that are both large in size and produced in significant quantities, such as cars and trucks. This particular layout of the assembly line utilizes both the right and left sides simultaneously. Based on real cases, the two-sided assembly line has proven to be beneficial compared to a one-sided line in several ways, including shorter line lengths, decreased throughput time, reduced costs for tools and fixtures, and minimal material handling requirements [1].

1.1 Problem Statement: The Rebalancing Challenge

The core issue in modern manufacturing is optimizing two-sided robotic assembly lines, particularly under the constraints of reconfigurable production. Current approaches largely ignore the existing resources, leading to a gap in practical and efficient solutions. To address this research problem, this study aims to address the complex nature of rebalancing two-sided assembly lines by developing a multi-objective model. The model is designed with dual primary objectives: first, to manage the inherent complexity associated with the rebalancing of two-sided assembly lines, and second, to simultaneously minimize cycle time, energy consumption and reconfiguration costs. This approach is essential for achieving

sustainable and cost-effective manufacturing in response to dynamic market demands.

1.2 Research Objectives

This research is structured around three primary objectives. Each objective focuses on a distinct aspect of the assembly line’s reconfiguration, ranging from extending existing models to accommodate two-sided operations to incorporating advanced multi-objective optimizations. These objectives are crafted to tackle the technical challenges specific to robotic assembly systems and to enhance overall operational efficiency and sustainability. The detailed objectives are as follows:

1.2.1 Objective 1: Extend the Reconfiguration Model to Include Two-Sided Assembly Lines

The purpose of this objective is to extend existing reconfiguration models from single-sided to two-sided assembly lines to address the unique requirements and challenges of two-sided configurations. The approach to achieve this involves updating current models to accurately represent the complexities associated with two-sided operations. The expected outcome of this objective is a comprehensive reconfiguration model for two-sided assembly lines, which effectively optimizes both layout efficiency and resources usage, making it adaptable for a variety of industrial settings.

1.2.2 Objective 2: Address Multi-objective Optimization

The second objective of this research is to address multi-objective optimization within the Reconfiguration Model, focusing on integrating additional key performance indicators such as cycle time and energy consumption alongside the existing objective of total reconfiguration cost. This objective aims to enhance the model's capability to simultaneously optimize multiple critical factors, thereby aligning the production process with broader sustainability and efficiency goals. The method to achieve this involves incorporating advanced optimization techniques that can effectively balance these competing objectives, providing a solution that optimizes overall operational efficiency. The anticipated result of this objective is a robust, multi-objective reconfiguration model that can deliver optimal solutions under varying operational constraints, thereby significantly improving the performance and sustainability of two-sided robotic assembly lines.

1.2.3 Objective 3: Solve the Reconfiguration Model for Two-Sided Robotic Assembly Line

The third objective is to solve the newly developed reconfiguration model for two-sided assembly lines using both exact and meta-heuristic methods. The goal is to demonstrate the model's applicability and effectiveness in optimizing re-balancing processes across different solution methodologies. This objective will be achieved by employing a dual-method approach. Exact methods like linear programming or branch-and-bound algorithms will be used to establish optimal solution bench-

marks where feasible. In parallel, various meta-heuristic approaches, including optimization algorithms and heuristic strategies, will be explored to address instances where exact methods are impractical due to the complexity and nature of the model. Successfully applying and solving the model using both exact and meta-heuristic methods will validate its performance. This dual approach will provide comprehensive insights into optimal re-balancing strategies for two-sided assembly line contexts.

1.3 Research Questions

Within the above framework, the study will address three critical research questions:

1. How can existing reconfiguration models for single-sided assembly lines be effectively adapted for two-sided assembly lines, with a focus on the triple objectives of minimizing total reconfiguration cost, reducing energy consumption, and decreasing cycle time?

This question aims to explore the required modifications and innovative approaches needed to address the unique operational dynamics of two-sided assembly lines. It emphasizes the need for a comprehensive model that balances cost-efficiency in reconfiguration processes with a strong focus on energy conservation and cycle time reduction, aligning with the changing demands of modern, sustainable manufacturing practices.

2. What methodologies and techniques can be employed to solve the extended

reconfiguration model for two-sided robotic assembly lines?

This question explores suitable methodologies, such as optimization algorithms or exact methods, that can effectively solve the newly developed model. It is essential for ensuring that the theoretical model developed can be practically applied to real-world scenarios, thus bridging the gap between theory and practice.

3. How can the theoretical model be adapted to be effectively solved using metaheuristics?

This question investigates how the theoretical reconfiguration model can be transformed to facilitate resolution through metaheuristic approaches. This includes determining the specific adaptations required to encode the problem suitably for metaheuristics and identifying which types of metaheuristic algorithms are most effective for this type of problem. This approach aims to enhance the practical applicability and efficiency of the model in real-world manufacturing settings.

By addressing these questions, the research aims to contribute significantly to the field of assembly line balancing, especially in the context of two-sided assembly lines, which have been less explored within the field of re-balancing. This approach not only furthers the theoretical understanding but also aims to provide practical solutions that enhance efficiency and sustainability in production systems. The structure of this report is as follows: Chapter 2 presents a comprehensive literature review, followed by Chapter 3, which introduces the mathematical model.

Chapter 4 details the solution methodologies, and chapter 5 presents evaluation experiments. Finally, Chapter 6 discusses the results and concludes the report with a summary and recommendations for future research.

CHAPTER 2

LITERATURE REVIEW

2.1 Reconfigurable Manufacturing Systems

Due to the current competitive market requirements, manufacturing companies must design cost and time efficient systems as well as being environmentally sustainable. This trend is expected to continue according to the "Manufacturing Challenges 2020" report from the USA. As noted by the National Research Council in 1998, a significant challenge identified is the capability to rapidly reconfigure manufacturing systems to adapt to dynamic demands nature [2]. This requires a Reconfigurable Manufacturing System (RMS) which represents one of the most recent concepts in manufacturing. Within this framework, parts of machines, machine systems, and material handling tools can be integrated, removed, added, and exchanged to meet production requirements. This adaptability is driven by the need to adjust quickly and efficiently to changing demands. For diverse production types, RMS is considered a suitable manufacturing paradigm since it acts as a flexible facilitator for this diversity. Therefore, it represents a mixed ver-

sion from the two manufacturing systems commonly utilized in industry: Flexible Manufacturing Systems (FMSs), and Dedicated Manufacturing Lines (DMLs).

As stated by Koren, who introduced RMS, while DMLs are cost-effective, they often underperform in various scenarios, particularly amid global competitive pressures, leading to losses. Meanwhile, FMSs adapt to product changes but are not configured for modifications in the system structure. As a result, neither system is adequately prepared to manage sudden changes in the market, like fluctuations in demand or changes in standards. RMS provides the strengths of both systems by integrating the flexibility of FMSs with the extensive manufacturing capacity of DMLs. This targeted design allows RMS to effectively adapt to situations like machine breakdowns or market fluctuations while maintaining performance. It offers a solution for scenarios where both productivity and the ability to respond quickly to uncertainties are crucial. The RMS concept has increasingly captured the scientific community's attention, where over 60% of the research papers on this topic were published between 2010 and 2017. Furthermore, it is found that the majority of studies related to RMS have been formulated as optimization problems [2], [3].

Reconfigurable Machines (RMs) are key elements in RMS. Their primary role is to give these systems the flexibility needed to adapt quickly to changes in market conditions. It used to provide the system with the ability to manage a designated family of products given their inherent flexibility where these machines are characterized by a modular design. They consist of various interchangeable physical parts that could be quickly exchanged to provide a wider range of functional capa-

bilities. Therefore, the manufacture of diverse product types to meet new product requirements or emerging market demands is facilitated by the modular design of these machines. There are four types of RMs and each of these machine types contribute to the efficiency of modern manufacturing systems by giving their distinct functions and applications. However, the primary focus of the research will be on Reconfigurable Assembly Machines (RAMs), commonly employed in assembly lines. Given their modular design, they could be adapted for assembling products that share specific characteristics [2], [3].

The layout in which reconfigurable machines are arranged is a critical factor that defines the type of the manufacturing system. Furthermore, choosing the appropriate layout for these machines is a key decision during the RMS design phase. It has an effect on the manufacturing system's productivity and the quality of the products produced. Various layout types exist within RMS, which vary based on the process type and the machinery involved. This research specifically concentrates on Reconfigurable Flow Lines. Such lines are made up of multiple workstations, with each station containing one or more RMs designed to carry out specific tasks. The fundamental layout of these lines is presented in Figure 2.1. Typically, Reconfigurable Flow Lines are employed in settings like reconfigurable assembly or machining lines. RAMs contain various workstations with parallel, identical centers, each equipped with adaptable resources for task execution. This design enhances the system's responsiveness to product modifications, as it allows for easy reconfiguration of centers or resource reallocation among workstations [3].

Due to the fact that RMS consist of reconfigurable machines, numerous re-

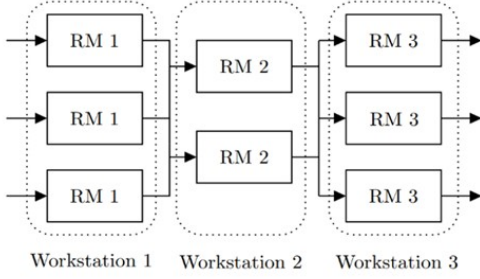


Figure 2.1: Reconfigurable Flow Lines [3].

searchers have linked the performance metrics to the overall performance of the RMS. There have been various studies focusing on Reconfigurable Machine Tools (RMT), exploring a range of objective functions and performance metrics. Typically, these objective functions are evaluated at both the machine and system levels. However, in both cases, cost has been the most frequently considered objective. These expenses are generally categorized into three primary classifications: RMS reconfiguration/rebalancing costs, investment/capital expenses, and RMS operational cost. This research will specifically concentrate on Reconfiguration cost, which arises from the addition, relocation, or removal of machines in the RMS. Reference [3] in the literature provides a summary of the relevant RMS objective functions that have been explored. Among these, cost-related objectives have received the most attention, while the sustainability considerations of RMS have only been addressed in a limited number of studies [3].

2.2 Two-Sided Assembly Lines

On an assembly line, various tasks are assigned and executed at different workstations. Every workstation is responsible for completing its assigned tasks within a

set, constant timeframe, known as the cycle time. The objective of the ALBP is to efficiently distribute tasks to the workstations while considering one or more factors for optimization [4]. Regarding the real word applications, the simple straight single-sided assembly line balancing problem might be too theoretical to represent the real situation requirements. Therefore, the other versions of different layouts are extensively studied to handle the real production [5].

In terms of straight assembly line layouts, assembly line balancing problems are categorized as single-sided, featuring single-sided workstations, and as the Two-sided assembly line balancing problem (TALBP). It was first introduced by Bartholdi in 1993 [6]. In a two-sided layout, there are mated-stations, a pair of two opposite workstations where tasks are divided and processed on both sides in parallel. Moreover, the two-sided layout offers the advantage of assembling large-sized products. Furthermore, in practical applications, it can offer many benefits compared to a one-sided layout, including shorter line lengths, less material handling, decreased throughput times, and reduced expenses for tools [1].

Another distinguishing aspect of Two-Sided Assembly Line Balancing (Two-ALB) as compared to one-sided ALB lies in the restrictions imposed on the directions in which tasks can be performed. In this assembly process, there are tasks that must be performed on the left (type L) or right (type R) side, and the rest of tasks might be assigned on any side (E-type). Taking into account the direction of operations modifies the approach to dealing with precedence relations and cycle time constraint. Due to the presence of two sides at each station, idle time can occur in some cases, especially among tasks allocated to the same paired station.

For instance, there are situations where a task cannot be finished if its immediate predecessor is the task on the opposite side. Consequently, a robot must wait until the task on the opposite side is completed before it can begin its work. As a result, line balancing must take into consideration the sequence-dependent completion time of tasks. This dependency on the sequence affects the cycle time [7].

In two-sided assembly lines, three extra constraints are essential: zoning constraints, positional constraints, and synchronism constraints. The Positive Zoning Constraint requires that a specific set of tasks be allocated to the same mated station. Conversely, the Negative Zoning Constraint prohibits assigning a set of tasks to the same mated station. The positional constraint ensures that certain tasks must be assigned to a specific workstation. The synchronism constraint ensures that paired tasks must be executed simultaneously on facing sides of a mated-station. These added constraints significantly increase the complexity of the TALBP [1], [6].

2.2.1 Two-Sided Robotic Assembly Line Balancing

Nowadays, many industries are leveraging robots in their assembly lines due to increased manpower costs and customized demands for a wide range of products. An assembly line using robots or comparable automated machinery for tasks like welding, handling, and part assembly is classified as a Robotic Assembly Line (RAL). In the last few years, Robotic ssembly line balancing problem (RALBP) has been attracting more interest in the academic community. RALBP refers

to the operation of efficiently allocating tasks to workstations by strategically assigning the most suitable robot to each task [8].

Currently, industries that assemble products that are both large in size and produced in significant quantities are utilizing two-sided robotic assembly lines. The schematic diagram of a two-sided robotic assembly line is shown in Figure 2.2. In this case, Two-sided Robotic Assembly Line Balancing problem (TRALBP) combine the significant characteristics of two-sided assembly line balancing and RALBP. Hence, there are two concerns that must be handled: task allocation and robot assignment. In existing research, TRALB problems are commonly classified into two categories. Type I TRALB focuses on the minimization of the number of stations within a constant cycle time, while Type II TRALB focuses on the reduction of the cycle time for a group of workstations. The execution of automated equipment in assembly lines has provided production flexibility and contributed to improving product quality [5]. However, as robotic assembly lines maintain their operations, the expense of energy consumption continues to be a major cost factor. Furthermore, with the increasing prices of energy, this financial aspect becomes increasingly critical and cannot be ignored. Also, decreasing energy consumption can maintain the competitiveness of industries and lower pollution levels [9].

The process of allocating robots to workstations is tackled in different RALBP models using a variety of methods: exact, heuristic, meta-heuristic, and simulation approaches. The majority of the exact solution strategies used for Simple ALBP can also be adapted for RALBP, regardless of the variances in model formulation. These exact methods include branch and bound techniques, (mixed)

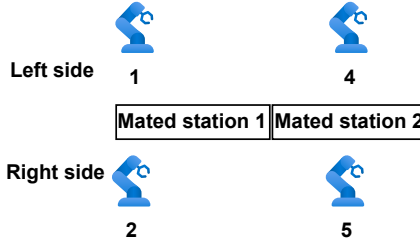


Figure 2.2: Two-sided Robotic Assembly Line

integer programming (IP), and others. More information can be found in [10]. Assembly line balancing problems considering single-side assumptions are recognized as NP-hard. Hence, TRALBP problems are also classified in the NP-hard category because of their increased complexities [9]. Much of the existing research considering straight robotic assembly lines, typically involving the assembly of a single product. However, recent studies increasingly report on various assembly line configurations and the assembly of multiple products. Yet, research on TRALBP remains limited.

2.3 TALBP Solution Methodologies

The methods applicable to TALBP are broadly categorized into exact methods, heuristic methods, and meta-heuristic methods. Although the application of exact and heuristic methods to TALBP has been relatively limited, they offer an invaluable benchmark and insight into the problem’s nature. Meanwhile, meta-heuristic methods constitute the highest percentage of the research efforts in TALBP, due to their ability to provide satisfactory results within an acceptable computational timeframe [4].

Meanwhile, the complexity and the extensive search space inherent in TRALBP, particularly when compared to one-sided RALBP, along with their classification as NP-hard problems, demand a comprehensive methodological approach. Hence metaheuristic methods have emerged as a promising approach for addressing RALBP, research suggests that exact methods have demonstrated significant effectiveness in complex single-side assembly line extensions and robotic assembly line problem, even within a reasonable timeframe for computation [11]. This insight is crucial, considering the new production system requires significant investments by the company, thus making the optimization of these investments essential. In the following sections, the studies of solution methodologies will be summarized as follows: first, exact methods used for two-sided assembly lines will be discussed, followed by studies on single robotic assembly lines, and concluding with two-sided robotic assembly lines.

2.3.1 Exact and heuristic methods

Many exact algorithms and heuristics have been developed to optimize the balance of traditional one-sided assembly lines. However, their application to balancing two-sided lines is limited. Concerning exact methods, only three studies have been documented for solving TALBP. A station-oriented enumerative algorithm was developed to address small-sized problems, while subsequent advancements expanded the use of branch-and-bound methods to solve larger problem instances [12], [13], [14].

The Branch-and-Bound Algorithm represents a classic optimization technique

that systematically divides the problem into smaller parts (branching) and evaluates the bounds of the solution space (bounding) to efficiently find the optimal solution [15]. Among the exact methods, further studies have explored the application of the branch-and-bound method to other specific challenges in SALBP, such as assembly line balancing with station paralleling, two-sided assembly line balancing, and multi-manned assembly line balancing. These studies demonstrate the versatility and robustness of this approach in handling different operational constraints [10].

Although numerous heuristic methods have been utilized for simple ALBP, their applications to TALBP remain relatively limited. [6] initially developed a first-fit heuristic aimed at minimizing the number of workstations, marking the first application of such methods to TALBP [6]. Subsequently, [16] introduced a group assignment procedure that assigns tasks in groups rather than individually [16]. This method proved to be more effective than the first-fit heuristic initially developed by Bartholdi in 1993. [17] employed a heuristic approach that leveraged the computer method of sequencing operations for assembly lines. This method successfully identified optimal solutions. Another significant study involved applying the first-fit rule from [6] to tackle TALBP-II. However, the performance of this heuristic method was surpassed by a genetic algorithm presented in [1]. Heuristic approaches can quickly generate feasible solutions; however, their performance is often unsatisfactory as the problem size increases. The presented studies are applied in TALBP. More studies for heuristic approaches applied to single sided assembly lines are presented in [18].

2.3.2 Solution Methodologies for RALBP

[19] were the first to investigate the RALBP, for which they developed a mathematical formulation and proposed a heuristic. [8] addressed the type-I RALBP using two distinct methods: one based on a genetic algorithm and the other on a branch and bound method. [20] tackled the type-II RALBP with a genetic algorithm, specifically focusing on optimizing cycle time. With an increasing emphasis on energy conservation, [21] emerged as pioneers by highlighting the importance of reducing energy consumption in single-sided robotic assembly lines, utilizing PSO.

[22] conducted the first study to model a type-II parallel RALBP, analyzing it through three heuristic approaches based on beam search. These approaches demonstrated better performance compared to other evolution algorithms. [23] incorporated three key features into the RALBP: workstation paralleling, robot assignment, and the use of various robotic arms per station. They developed a mixed-integer linear model and utilized CPLEX solver. [23]. Li, Janardhanan et al. (2018) explored the RALBP with model sequencing, developing both a restarted simulated annealing algorithm and a co-evolutionary algorithm to tackle the problem. These algorithms were then applied on a real applications of robotic welding assembly line [24]. [25] applied the migrating bird optimization algorithm (MBO) to address the RALBP, comparing it with seven traditional heuristic methods. They found that the MBO algorithm outperformed these methods in large-sized problem instances with a U-shaped layout [25].

Furthermore, [5] modeled task sequencing impact on the setup times and used MBO metaheuristic on this problem. Their experiments showed that the MBO algorithm performed better than the other meta-heuristics [5]. [26] developed a algorithm distribution based to minimize energy consumption in RALBP, finding it more effective than both the (NSGA-II) and the particle swarm algorithm [26]. Others created a multi-objective algorithm (MOEA/D) with a decomposition mechanism, addressing RALBP by considering changeover times dependent on task sequencing [27]. [28] improved the classic dragonfly algorithm to effectively address a U-shaped RALBP with integrated sequencing decisions. Recently, authors in [29] introduced a firefly algorithm that utilizes probability attraction to solve parallel RALBP. their aim is to simultaneously minimize cycle time and total power consumption. [30] modeled the algorithm of memory-based cuckoo search to solve an RALBP designed to produce multiple models using a single configuration.

[11] introduced an integer-programming model to manage dynamic resource reconfiguration for robotic assembly lines. They focused on a RALBP under brownfield constraints, adapting available assembly lines to fluctuating market demands [11]. in [31], the authors implemented a two-phase approach for dynamic capacity RALBP with focus on the cost: the first phase minimized capital costs under initial demand conditions, and the second phase aimed to reduce the combined capital and reconfiguration costs across various demand scenarios. [32] developed a split-based approach within a metaheuristic framework to tackle two versions of the sequence-dependent tasks in RALBP, differing in the assignment

capacity for robot types [32]. [33] modeled a mixed-integer programming model to optimize a bi-objective scenario, focusing on minimizing both production time and carbon emissions. They enhanced a particle swarm algorithm (PSO) with specific modifications rules to manage the problem [33].

2.3.3 Solution Methodologies for TRALBP

[1] were the first in presenting a mathematical formulation for two-sided assembly line problems, serving as the foundational basis for subsequent research in robotic two-sided assembly lines [1]. The most significant literature on two-sided robotic assembly lines includes the following contributions: [34] developed RTALBP-II to minimize cycle time and employed a co-evolutionary particle swarm optimization algorithm as its solution [34]. Subsequently, [35] introduced two algorithms of discrete cuckoo search and were found to be more effective than the co-evolutionary PSO algorithm. [36] tackled the two-sided mixed-model RALBP, which includes times for setups within their mathematical formulation and solved the model using SA algorithm. Furthermore, [37] modeled TRALB problem by a new mixed integer linear programming model aimed at optimizing cycle time and proposed a set of metaheuristics for the problem. Their research implemented thirteen different metaheuristic methodologies, such as PSO, genetic algorithm (GA), bees algorithm (BA), discrete cuckoo search (DCS), two variations of the artificial bee colony (ABC) algorithm, and the MBO algorithm [37]. More recently, [38] developed search method based on the variable neighborhood to solve a mixed-model, two-sided RALBP with the objective of minimizing production time while includ-

ing setup times.

However, research targeting the power consumption in assembly line balancing remains less explored and is primarily centered around single-sided and U-shape lines. [9] are the first to publish a paper addressing a two-sided robotic assembly line, aiming to simultaneously minimize the consumption of energy and cycle time. Their study investigated power consumption in RALBP-II two-sided layout and implemented a multi-objective. Then, they solve it using simulated annealing algorithm to reduce both power consumption and production time [9].

Lastly, in the context of rebalancing two-sided robotic assembly lines, only two studies have specifically addressed this issue with a multi-objective focus. Both considered minimizing cycle time and reconfiguration costs—one utilizing a heuristic algorithm [39], while the other proposed a modified genetic algorithm [40].

2.3.4 Multi-Objective Mathematical Programming

Solving Mathematical Programming (MP) problems with a single objective function is generally straightforward. However, Multi-Objective Mathematical Programming (MOMP) involves multiple objective functions, where typically, no single solution optimally satisfies all objectives simultaneously. In such cases, decision makers aim to identify solutions that provide suitable trade-offs rather than a singular optimal solution. In MOMP, the traditional concept of optimality is supplanted by Pareto optimality. Pareto optimal solutions are those that are non-dominated; a non-dominated solution cannot be improved in one objective

without compromising its performance in at least one other objective function [41].

Generally, the most commonly used methods for managing multiple objectives in conjunction with exact models are the weighting method and the ϵ -constraint method. These approaches are capable of generating a diverse Pareto set of solutions, even in cases of relatively large MOMP. Nevertheless, the ϵ -constraint method offers several benefits compared to the weighting method [41]. In the weighting method, multiple objectives are combined into a single objective through a linear combination, with each objective assigned a specific weight. This method aims to explore the feasible region of the problem to identify solutions that maximize or minimize the combined objective. However, as it relies on a weighted sum, the solutions it produces are typically 'corner solutions' or 'extreme solutions' within the feasible region. These solutions, while efficient, represent extreme cases where one or more objectives are prioritized at the expense of others. A significant drawback of this method is redundancy; different sets of weights may lead to the same corner solution, resulting in repetitive and inefficient optimization efforts.

Conversely, the ϵ -constraint method involves optimizing one of the objective functions while treating the remaining objectives as constraints. This modification of the original feasible region allows the method to explore a broader range of solutions, including those that are not at the extremes. As a result, the ϵ -constraint method is capable of generating non-extreme efficient solutions, thereby providing a richer and more diverse set of efficient options. Each iteration is likely to produce a unique solution, effectively mapping out a more comprehensive representation

of the efficient solution set.

Another important consideration in the weighting method is how the objective functions are scaled, which significantly impacts the results. This method combines multiple objectives into a single objective function by applying weights to each. However, if these objectives are not on a similar scale, the weights might not accurately reflect the intended importance or influence of each objective. Conversely, the ϵ -constraint method does not require this kind of scaling. Since it focuses on optimizing one objective at a time, with the others merely bounded (but not combined), there is no need to align their scales. Each objective's influence is controlled individually through the constraints, rather than through a combined effect, as in the weighting method. This distinction simplifies the process and helps avoid potential biases introduced by scaling in multi-objective optimization scenarios. However, if obtaining a broad and diverse set of solutions is critical, the ϵ -constraint method might be preferable due to its ability to explore beyond the extremes and generate a richer set of efficient solutions. On the other hand, the weighting method might be suitable for scenarios where finding extreme solutions is more desirable or acceptable [42].

2.4 Current Research Gaps

From the literature, it can be seen that there is a strong emphasis on cost objectives in RMS, with most studies examining single-sided line layouts and giving less attention to sustainability aspects. In the field of robotic assembly lines, limited

studies exploring energy efficiency, primarily centering around one-sided or U-type assembly lines. Hence, this research proposes extending the model from a single-sided to a two-sided robotic reconfiguration format. This significant contribution not only broadens the scope of RMS but also integrates the important objectives of minimizing cycle time, reconfiguration costs and energy consumption, therefore contributing to the promotion of sustainable manufacturing practices in RMS.

In this study, we develop a formulation that integrates two distinct models. This formulation expands the single-sided robotic reconfiguration model by [11] by incorporating elements from the two-sided model presented by [9]. Notably, [11] highlights that existing models, such as SALBP or RALBP, do not account for different types of tasks. To address this, they introduced the Robotic Assembly Line Balancing Problem with Task Types (RALBP-TT) [11], which includes various task types such as handling, joining, and separation, to better simulate real-world production scenarios. Additionally, the model by [11] was the first to consider the savings from selling unused robots, alongside the investment and processing costs. In this paper, we further adapt the reconfiguration model to accommodate the two-sided aspects with a multi-objective optimization approach that aligns with the minimization of cycle time and energy consumption objectives introduced by [9]. As a result, we refer to our extended model as the Two-Sided Robotic Assembly Line Balancing Problem with Task Types (TRALBP-TT). To the best of our knowledge, this is the first model to incorporate task types into a two-sided robotic assembly line framework.

CHAPTER 3

MATHEMATICAL FORMULATION

3.1 Problem formulation

The formulation of TRALBP-TT is detailed in this section, which includes the assignment of tasks to robots and the assignment of robots to workstations. An introduction to the overall approach is provided first, followed by the statement of assumptions, the definition of variables, and the cost model. Subsequently, the Mixed Integer Programming (MIP) formulation of the problem will be presented. While the initial model is non-linear due to certain interactions between tasks, robots, and workstations, we will discuss the linearization of these non-linear elements in a later section.

3.1.1 Assembly line rebalancing approach

In contrast with the greenfield approach which focuses on creating new assembly lines, the brownfield approach allows for a flexible and adaptable assembly line configuration that can seamlessly transition based on changing production demands. It operates as a cycle that can be executed iteratively, continually optimizing the assembly line to respond to new requirements while utilizing existing Greenfield assignments. Figure 3.1 illustrates the decision-making process and the relationship between the Greenfield and Brownfield approaches for production requirements. The process begins by determining whether an existing assembly line is available. If not, the Greenfield approach is chosen, leading to new investments. If an assembly line exists, the Brownfield approach is selected, followed by a decision on whether to reuse existing resources. Some resources may be reused to meet the new production requirements. If resources cannot be reused, they are sold, and the process may involve new investments to address different production requirements. The new solution then serves as input for the Brownfield approach for future rebalancing.

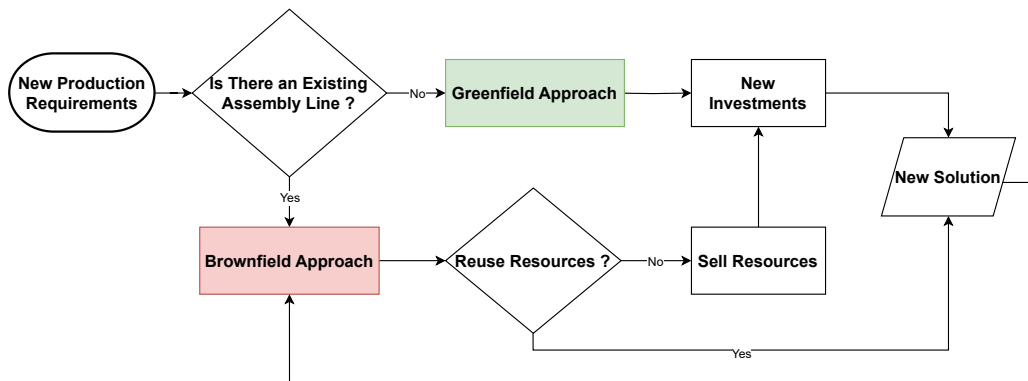


Figure 3.1: Flowchart of Greenfield and Brownfield Approaches

3.1.2 Model assumptions

In the development of a robust model for two-sided robotic assembly line configurations, certain foundational assumptions are made to simplify the process and focus the research. The following assumptions are considered in the reconfiguration of the robotic assembly line for both Greenfield and Brownfield approaches. These assumptions apply to both the Greenfield and Brownfield approaches. However, for the Greenfield approach, there is no initial solution available.

1. A single product is considered.
2. Task times are dependent on the assigned robot that will perform the task, and are deterministic.
3. Robots setup times are ignored.
4. The solution space includes at least one robot that is feasible per task, with associated costs. These costs include saving, investment, and processing costs.
5. The existing line assignment solution is available.
6. In the solution space, not every robot is required to be assigned.
7. The total time at a workstation must be within a specified cycle time.
8. A workstation can perform any task, given the robot assigned can process it.

9. Material handling and transportation times are neglected or accounted into the task times.
10. Setup times between tasks are neglected.
11. Each robot's power consumption is given, and the energy consumption is estimated accordingly.
12. Each side of a mated station is assigned exactly one robot.
13. There are three task types: 1) separation, (2) handling, and (3) joining.

3.1.3 Notations

Indices

i, h, p : Tasks Indices.

r : Robot type.

j, g : Mated-stations.

(j, k) : Workstation of mated-station j at k side.

k : A side of the line; $k = \begin{cases} 1, & \text{left side} \\ 2, & \text{right side} \end{cases}$

Sets

I : Set of tasks, $I = \{1, 2, \dots, i, \dots, nt\}$.

J : Set of mated-stations, $J = \{1, 2, \dots, j, \dots, nm\}$.

R : Set of robot types, $R = \{1, 2, \dots, r, \dots, nr\}$.

$S(i)$: Set of immediate tasks that are successors to task i .

$P(i)$: Set of immediate tasks that are predecessors to task i .

$S_a(i)$: Set of all tasks that are successors to task i .

$P_a(i)$: Set of all tasks that are predecessors to task i .

P_0 : Set of tasks that do not have an immediate predecessor.

AL : Set of tasks that must be assigned to left direction, $AL \subseteq I$.

AR : Set of tasks that must be assigned to right direction, $AR \subseteq I$.

AE : Set of tasks that can be assigned to either direction, $AE \subseteq I$.

$C(i)$: Set of tasks whose operation directions are opposed to those of task i ;

$$C(i) = \begin{cases} AL, & \text{if } i \in AR \\ AR, & \text{if } i \in AL \\ \emptyset, & \text{if } i \in AE \end{cases}$$

$K(i)$: A set of integers indicating the preferred directions of the task i ; $K(i) =$

$$\begin{cases} \{1\}, & \text{if } i \in AL \\ \{2\}, & \text{if } i \in AR \\ \{1, 2\}, & \text{if } i \in AE \end{cases}$$

PC : Set of task-station pairs that indicate positional constraints (tasks that must be assigned to a particular mated station).

PZ : Set of tasks for positive zoning constraints (a pair of tasks that must be assigned to the same side of a single mated station).

NZ : Set of task pairs for negative zoning constraints (a pair of tasks that must be assigned to different mated stations).

SC : Set of task pairs for synchronism constraints (a pair of tasks that must be assigned to opposite sides of a single mated station).

Parameters

t_{ir} : Time taken by robot r to process task i .

M_1 : Large positive number.

PC_r : Energy consumption of robot r per time unit when operational.

SPC_r : Energy consumption of robot r per time unit in standby mode.

$\underline{EC}$: Vector of investment costs for robots, in \$.

ec_r : Investment cost for robot r , in \$.

$\underline{PC}$: Vector of processing costs per second for robots, in \$/s.

pc_r : Processing cost per second for robot r , in \$/s.

$\underline{SC}$: Vector of savings costs for robots, in \$.

sc_r : Savings cost for robot r , in \$.

$t_{a,r}$: Assembly time for robot r , in s.

$t_{d,r}$: Disassembly time for robot r , in s.

$t_{m,r}$: Moving time for robot r , in s.

d_{rjk} : 1, if robot r is initially assigned to mated station j at side k ; 0, otherwise.

f_{rec} : Labor cost of reconfiguration expressed in dollars per second (\$/s).

Decision Variables

y_{rjk} : 1, if robot r is assigned to mated-station j at side k ; 0, otherwise.

x_{ijk} : 1, if task i is assigned to mated-station j at side k ; 0, otherwise.

CT: Cycle time.

t_i^f : Finishing time of task i .

TE: Total energy consumption.

EC_{jk} : Energy consumption on side k of mated-station j .

Auxiliary Variables

z_{ip} : 1, if task i is assigned earlier than task p in the same mated station; 0, otherwise.

3.1.4 Cost model

The formulation of the cost model is primarily based on the single-sided robotic reconfiguration model proposed by [11]. Rebalancing the line requires a decision regarding the available robots. In such a scenario, a company might need to decide whether to sell a robot or reuse it after rebalancing the line.

The cost model consists of three main components: savings cost (C_{sav}), investment cost (C_{inv}), and processing cost (C_{prc}). The savings cost vector $\underline{SC} \in \mathbb{R}^{nr}$, shown in Equation (3.1), captures the net savings associated with trading in or selling each piece of equipment or robot r . Each element of SC is adjusted by subtracting the costs related to reconfiguration, which include the times for as-

sembly ($t_{a,r}$), disassembly ($t_{d,r}$), and moving ($t_{m,r}$), all multiplied by a labor cost factor (f_{rec}). The savings cost C_{sav} , as shown in Equation (3.2), is computed by combining these adjusted savings values with the binary indicator d_{rjk} , which represents whether a robot was initially assigned to a station-side combination. The term $(1 - y_{rjk})$ ensures that savings are only considered when a robot is removed in the new solution.

The second element is the investment cost C_{inv} , which represents the new robot investments, as shown in Equation (3.3). It uses the investment cost per robot (ec_r), which is an element of the vector $\underline{EC} \in \mathbb{R}^{nr}$, and incorporates the decision variable y_{rjk} to indicate new assignments. The term $(1 - d_{rjk})$ ensures that investment costs are only counted when a robot is assigned to a station where it was not previously assigned. Moving robots between stations is not considered. If an already available robot is suitable for a different task at another station, a new robot investment is considered more cost-effective. This is because, in combination with the processing cost over the long-term time horizon, investing in a new robot provides a better overall cost outcome compared to reusing an existing robot at a new station and incurring the disassembly and assembly costs [11].

The third element is the processing cost, C_{prc} , which accounts for the operational expenses associated with using robots at their assigned stations. This aspect is represented by a vector $\underline{PC} \in \mathbb{R}^{nr}$, and it may include a range of costs such as energy consumption, inspection machine expenses, environmental impact fees, maintenance charges, and additional variable costs defined by the company.

As shown in Equation (3.4), it is calculated using the processing cost rate (pc_r) for each robot and the decision variable y_{rjk} , which indicates active assignments in the new solution. This term reflects the ongoing costs of operating the robots in their allocated stations.

$$\underline{SC} = \begin{bmatrix} sc_1 - (t_{a,1} + t_{d,1} + t_{m,1}) \cdot f_{\text{rec}} \\ \vdots \\ sc_{nr} - (t_{a,nr} + t_{d,nr} + t_{m,nr}) \cdot f_{\text{rec}} \end{bmatrix} \quad (3.1)$$

$$C_{\text{sav}} = \sum_{r=1}^{nr} \sum_{j=1}^{nm} \sum_{k=1}^2 (sc_r - (t_{a,r} + t_{d,r} + t_{m,r}) \cdot f_{\text{rec}}) \cdot d_{rjk} \cdot (1 - y_{rjk}) \quad (3.2)$$

$$C_{\text{inv}} = \sum_{r=1}^{nr} \sum_{j=1}^{nm} \sum_{k=1}^2 ec_r \cdot y_{rjk} \cdot (1 - d_{rjk}) \quad (3.3)$$

$$C_{\text{prc}} = \sum_{r=1}^{nr} \sum_{j=1}^{nm} \sum_{k=1}^2 pc_r \cdot y_{rjk} \quad (3.4)$$

3.1.5 Mathematical formulation

$$\mathbf{Min} \ CT \quad (3.5)$$

$$\mathbf{Min} \ C = -C_{\text{sav}} + C_{\text{inv}} + C_{\text{prc}} \quad (3.6)$$

$$\mathbf{Min} \ TE = \sum_{j=1}^{nm} \sum_{k=1}^2 EC_{jk} \quad (3.7)$$

$$EC_{jk} = \sum_{r=1}^{nr} \sum_{i=1}^{nt} PC_r t_{ir} x_{ijk} y_{rjk} + \left(\sum_{r=1}^{nr} SPC_r \cdot y_{rjk} \right) \cdot \left(CT - \sum_{r=1}^{nr} \sum_{i=1}^{nt} t_{ir} x_{ijk} y_{rjk} \right) \quad \forall j \in J, k \in \{1, 2\} \quad (3.8)$$

$$\sum_{j \in J} \sum_{k \in K(i)} x_{ijk} = 1 \quad \forall i \in I \quad (3.9)$$

$$\sum_{g \in J} \sum_{k \in K(h)} g \cdot x_{h g k} \leq \sum_{j \in J} \sum_{k \in K(i)} j \cdot x_{ijk} \quad \forall i \in I - P_0, h \in P(i) \quad (3.10)$$

$$t_i^f \leq CT \quad \forall i \in I \quad (3.11)$$

$$t_i^f - t_h^f + M_1(1 - \sum_{k \in K(i)} x_{ijk}) + M_1(1 - \sum_{k \in K(h)} x_{hjk}) \geq \sum_{k \in K(i)} \sum_{r=1}^{nr} t_{ir} y_{rjk} \quad (3.12)$$

$$\forall i \in I - P_0, h \in P(i), j \in J$$

$$t_p^f - t_i^f + M_1(1 - x_{ijk}) + M_1(1 - x_{pjk}) + M_1(1 - z_{ip}) \geq \sum_{r=1}^{nr} t_{pr} y_{rjk}$$

$$\forall i \in I, p \in \{r \mid r \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < r\}, j \in J, k \in K(i) \cap K(p) \quad (3.13)$$

$$t_i^f - t_p^f + M_1(1 - x_{ijk}) + M_1(1 - x_{pjk}) + M_1(z_{ip}) \geq \sum_{r=1}^{nr} t_{ir} y_{rjk}$$

$$\forall i \in I, p \in \{r \mid r \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < r\}, j \in J, k \in K(i) \cap K(p) \quad (3.14)$$

$$t_i^f + M_1(1 - x_{ijk}) \geq \sum_{r=1}^{nr} t_{ir} y_{rjk} \quad \forall i \in I, j \in J, k \in K(i) \quad (3.15)$$

$$\sum_{r=1}^{nr} y_{rjk} = 1 \quad \forall j \in J, k \in \{1, 2\} \quad (3.16)$$

$$\sum_{j=1}^{nm} \sum_{k=1}^2 y_{rjk} \leq 1 \quad \forall r \in R \quad (3.17)$$

$$x_{ijk} = 1 \quad \forall (i, (j, k)) \in PC \quad (3.18)$$

$$x_{ijk} - x_{hjk} = 0 \quad \forall (i, h) \in PZ, j \in J, k \in K(i) \cap K(h) \quad (3.19)$$

$$\sum_{k \in K(i)} x_{ijk} + \sum_{k \in K(h)} x_{hjk} \leq 1 \quad \forall (i, h) \in NZ, j \in J \quad (3.20)$$

$$x_{ijf} - x_{hjk} = 0 \quad \forall (i, h) \in SC, j \in J, k \in K(h), f \in K(i), k \neq f \quad (3.21)$$

$$t_i^f - t_i = t_h^f - t_h \quad \forall (i, h) \in SC \quad (3.22)$$

$$z_{ip} = \sum_{j \in J} \sum_{k \in K(i) \cap K(p)} x_{ijk} \cdot x_{pjk} \quad \forall i \in I, p \in \{r \mid r \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < r\} \quad (3.23)$$

$$y_{rjk} \in \{0, 1\}, \quad \forall r \in R, j \in J, k \in \{1, 2\} \quad (3.24)$$

$$x_{ijk} \in \{0, 1\}, \quad \forall i \in I, j \in J, k \in K(i) \quad (3.25)$$

$$t_i^f \geq 0, \quad \forall i \in I \quad (3.26)$$

$$CT \geq 0 \quad (3.27)$$

$$TE \geq 0 \quad (3.28)$$

$$EC_{jk} \geq 0, \quad \forall j \in J, k \in \{1, 2\} \quad (3.29)$$

$$z_{ip} \in \{0, 1\}, \quad \forall i \in I, p \in \{r \mid r \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < r\} \quad (3.30)$$

Objective function (3.5) minimizes the cycle time. Objective function (3.6) minimizes the total reconfiguration cost. Objective function (3.7) minimizes the total power consumptions on all workstations. Constraints (3.8) measure the energy consumption on side k of mated-station j . The first term of constraints (3.8) calculates the operational energy consumption at station (j, k) , and the subsequent term determines the standby power consumption. Constraints (3.9) ensure that each task is assigned to only one side of a workstation within the mated-station. Constraints (3.10) ensure the precedence relations between the tasks. Constraints (3.11) are the cycle time constraints, where the finishing time represents the cumulative time at the workstation. It is determined by constraints (3.12) to (3.15), which ensure that tasks only start after their predecessors are completed, accounting for both processing times and task dependencies. Constraints (3.12) to (3.14) are designed to manage the sequence-dependent finishing times. When task h is the immediate predecessor of task i at the same mated-station, constraints (3.12) are activated and reduce to $t_i^f - t_h^f \geq \sum_{r=1}^{nr} t_r y_{rjk}$. If tasks i and h are allocated to the same station but there is no precedence relationship, then constraints (3.13) and (3.14) are activated. Specifically, if task i is assigned before the task p , constraints (3.13) are activated and reduce to $t_p^f - t_i^f \geq \sum_{r=1}^{nr} t_{pr} y_{rjk}$. In all other instances, constraints (3.14) are activated, reducing to $t_i^f - t_p^f \geq \sum_{r=1}^{nr} t_{ir} y_{rjk}$.

Constraints (3.15) check that the finishing time of task i is greater than or equal to its operational time. Constraints (3.16) ensure that each workstation side has exactly one robot assigned to it. Constraints (3.17) indicate that not all available robots need to be assigned, and that each robot can be assigned to at most one workstation.. Constraints (3.18) manage the positional constraints. Constraints (3.19) are the Positive Zoning constraints, whereas constraints (3.20) are the Negative Zoning constraints. Constraints (3.21) and (3.22) ensure that every pair of synchronized tasks is assigned to opposite sides of the same workstation, and they must start at the same time. Constraints (3.23) define the auxiliary variable z_{ip} , which helps sequence tasks according to their precedence relationships, if any, which are enforced by constraints (3.13) and (3.14). Constraints (3.24) to (3.30) define the domain restrictions for all variables.

3.1.6 Linearization

To accurately represent a real-world optimization problem, it is often essential to incorporate non-linear terms in the mathematical formulation to effectively model the unique operational characteristics of the decision problem. In the above model, non-linearity arises from two types of variable interactions: the multiplication of binary variables and the multiplication of binary and continuous variables. The multiplication of binary variables occurs when expressing task precedence and assignment relationships, particularly in the case of the indicator variable z_{ip} , as described in Equation (3.23), where the product of two binary variables is used to determine whether a task is assigned earlier than another within the same station.

Nonlinearity is also introduced in the expression of total energy consumption, namely Equation (3.8), where the product of the binary variable y_{rjk} and the continuous variable CT is used to calculate the energy consumed based on robot assignment and cycle time. Additionally, the binary variables in this expression also interact with each other, leading to both binary-binary and binary-continuous terms.

The inclusion of non-linear terms typically escalates the computational complexity of the optimization model, thereby increasing the time needed to solve it. One effective approach to managing non-linear terms in operations research is the use of exact transformations. This method replaces non-linear equations or functions with an exact equivalent linear programming (LP) formulation, thereby creating valid inequalities [43].

To transform our model into a *linear* mixed-integer programming problem, these non-linear terms are linearized by introducing auxiliary variables and necessary linearization constraints that ensure they accurately represent the original non-linear relationships. Specifically, a term like $CT \cdot y_{rjk}$, which involves the multiplication of a binary variable and a continuous variable, can be replaced by an auxiliary variable w_{rjk} .

To linearize the non-linear constraints (3.23) containing a product of two binary variables, we introduce an auxiliary binary variable z_{ijkp} , and introduce the following linearization constraints for all $i \in I$, and $p \in \{r \mid r \in I - (P_a(i) \cup$

$S_a(i) \cup C(i)$ and $i < r$ }:

$$z_{ip} = \sum_{j \in J} \sum_{k \in K(i) \cap K(p)} z_{ijkp} \quad (3.31)$$

$$z_{ijkp} \leq x_{ijk} \quad \forall j \in J, k \in K(i) \cap K(p) \quad (3.32)$$

$$z_{ijkp} \leq x_{pjk} \quad \forall j \in J, k \in K(i) \cap K(p) \quad (3.33)$$

$$z_{ijkp} \geq x_{ijk} + x_{pjk} - 1 \quad \forall j \in J, k \in K(i) \cap K(p) \quad (3.34)$$

$$z_{ijkp} \in \{0, 1\} \quad (3.35)$$

To linearize the non-linear binary-binary products, $x_{ijk} \cdot y_{rjk}$, in constraints (3.8), we introduce auxiliary binary variables z_{irjk} , and equations (3.36) to (3.39) for every $i \in I, r \in R, j \in J$, and $k \in \{1, 2\}$:

$$z_{irjk} \leq x_{ijk} \quad (3.36)$$

$$z_{irjk} \leq y_{rjk} \quad (3.37)$$

$$z_{irjk} \geq x_{ijk} + y_{rjk} - 1 \quad (3.38)$$

$$z_{irjk} \in \{0, 1\} \quad (3.39)$$

Then, for the binary-continuous products in the same constraint, $CT \cdot y_{rjk}$, we introduce auxiliary variables w_{rjk} , and equations (3.40) to (3.43) for every

$r \in R, j \in J$, and $k \in \{1, 2\}$:

$$w_{rjk} \leq M_2 \cdot y_{rjk} \quad (3.40)$$

$$w_{rjk} \leq CT \quad (3.41)$$

$$w_{rjk} \geq CT + M_2 \cdot (y_{rjk} - 1) \quad (3.42)$$

$$w_{rjk} \geq 0 \quad (3.43)$$

Where M_2 is a large positive number. Then, for $j \in J$, and $k \in \{1, 2\}$, constraints (3.8) are re-written as follows:

$$EC_{jk} = \sum_{r=1}^{nr} \sum_{i=1}^{nt} PC_r \cdot t_{ir} \cdot z_{irjk} + \sum_{r=1}^{nr} SPC_r \cdot w_{rjk} - \sum_{r=1}^{nr} \sum_{i=1}^{nt} SPC_r \cdot t_{ir} \cdot z_{irjk} \quad (3.44)$$

All possible scenarios for the linearization constraints are presented in Tables 3.1, 3.2, and 3.3. Further studies that applied this method are presented in [43].

Table 3.1: All possible products of binary variables $z_{ijkp} := x_{ijk} \cdot x_{pjk}$

x_{ijk}	x_{pjk}	$x_{ijk} \cdot x_{pjk}$	Constraints	Imply
0	0	0	$z_{ijkp} \leq 0, z_{ijkp} \leq 0, z_{ijkp} \geq -1, z_{ijkp} \in \{0, 1\}$	$z_{ijkp} = 0$
0	1	0	$z_{ijkp} \leq 0, z_{ijkp} \leq 1, z_{ijkp} \geq 0, z_{ijkp} \in \{0, 1\}$	$z_{ijkp} = 0$
1	0	0	$z_{ijkp} \leq 1, z_{ijkp} \leq 0, z_{ijkp} \geq 0, z_{ijkp} \in \{0, 1\}$	$z_{ijkp} = 0$
1	1	1	$z_{ijkp} \leq 1, z_{ijkp} \leq 1, z_{ijkp} \geq 1, z_{ijkp} \in \{0, 1\}$	$z_{ijkp} = 1$

Table 3.2: All possible products of binary variables $z_{irjk} := x_{ijk} \cdot y_{rjk}$

x_{ijk}	y_{rjk}	$x_{ijk} \cdot y_{rjk}$	Constraints	Imply
0	0	0	$z_{irjk} \leq 0, z_{irjk} \leq 0, z_{irjk} \geq -1, z_{irjk} \in \{0, 1\}$	$z_{irjk} = 0$
0	1	0	$z_{irjk} \leq 0, z_{irjk} \leq 1, z_{irjk} \geq 0, z_{irjk} \in \{0, 1\}$	$z_{irjk} = 0$
1	0	0	$z_{irjk} \leq 1, z_{irjk} \leq 0, z_{irjk} \geq 0, z_{irjk} \in \{0, 1\}$	$z_{irjk} = 0$
1	1	1	$z_{irjk} \leq 1, z_{irjk} \leq 1, z_{irjk} \geq 1, z_{irjk} \in \{0, 1\}$	$z_{irjk} = 1$

Table 3.3: All possible scenarios of linear constraints

y_{rjk}	CT	Constraints	Imply
0	$0 \leq CT \leq M$	$w_{rjk} \leq 0, w_{rjk} \leq CT, w_{rjk} \geq CT - M, w_{rjk} \geq 0$	$w_{rjk} = 0$
1	$0 \leq CT \leq M$	$w_{rjk} \leq M, w_{rjk} \leq CT, w_{rjk} \geq CT, w_{rjk} \geq 0$	$w_{rjk} = CT$

3.2 An illustrated example

To present the main characteristics of the considered model, this section provides a simplified example problem involving 9 tasks and 2 mated-stations, solved considering a line depot of 5 alternative robots r . Assembly tasks are indivisible basic operations. Precedence relations between tasks are provided a priori in the form of a precedence graph and fixed. Table 3.4, contains the tasks, as well as their successors, directions, and the types. There are three task types considered in this study: (1) separation, (2) handling, and (3) joining.

Table 3.4: Numerical example

Task	Successors	Preferred direction	Task type	r1	r2	r3	r4	r5
1	4	L	1	2	2	3	2	2
2	5,6	R	3	3	3	4	3	3
3	6	E	2	2	2	3	2	2
4	7	L	3	3	3	4	3	3
5	7,8	R	3	1	1	2	1	1
6	9	E	3	1	1	2	1	1
7	-	E	2	2	2	3	2	2
8	-	L	2	2	2	3	2	2
9	-	E	2	1	1	2	1	1

For each robot, there are three types of costs: investment cost, processing cost,

and saving cost. The investment cost is incurred if the robot was not available in the current solution of the brownfield case. The saving cost is obtained if the robot was used in the current solution of the brownfield case but is not considered in the new solution. The processing cost applies to each robot used in the new solution. Table 3.5 shows these costs as well as the energy consumption for each robot per unit of time. Robot capabilities are determined based on their investment costs,

Table 3.5: Cost & Energy Data

Robot	Investment Cost	Processing Cost	Saving Cost	Energy Consumption
1	18283	7200	828	2.4
2	20567	21600	2056	1.6
3	8028	7200	802	1.5
4	22319	7200	2231	2.4
5	28019	14400	2801	1.5

suggesting that more expensive robots possess a broader range of task capabilities. This relationship is depicted in a capability matrix as shown in Table 3.6, where a ‘1’ indicates that a robot can perform a specific task, and a ‘0’ signifies its inability. For example, since robot 3 has the lowest investment cost among the robots, it consequently has limited capabilities, as it is unable to perform tasks of type 2, which involve handling processes.

Table 3.6: Capability Matrix

Robot	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
3	1	1	0	1	1	1	0	0	0
4	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1

This example was solved using an exact method with the Gurobi solver, and

the solutions are presented in Table 3.7. In this example, the initial allocation solution considered is shown in Figure 3.1. Therefore, the problem is solved as a brownfield case. The solver found two solutions for this example, as presented in Table 3.7. The first solution involved reusing the robots from the initial solution, which resulted only in processing costs. In the second solution, the allocation involved reusing robots 1, 4, and 5, while selling robot 2 and investing in robot 3. Consequently, with the savings from selling robot 2, the total cost was 41,972, which is lower compared to the first solution’s cost of 50,400. The energy calculations are the same as those presented in [9].

Table 3.7: Gurobi Solver Outcomes

Solver	CT	Total cost	Total energy
Gurobi	5	50400	34.64
	6	41972	38.43

CHAPTER 4

SOLUTION METHODOLOGIES

This research will initially apply exact methods to establish a benchmark for optimal solutions, which is essential for understanding the upper limits of solution quality and for validating the results obtained from other methods. Following this, the study will explore meta-heuristic methods, taking advantage of their demonstrated effectiveness in managing the multi-objective and complex nature of TRALBPs. This dual approach, combining the precision of exact methods with the efficiency of meta-heuristics, aims to offer a comprehensive and practical solution framework, thus making a contribution to both theoretical and practical advancements in TRALBP optimization.

4.1 Exact Methods

Exact methods are well-known for yielding optimal solutions; however, their application in solving two-sided models is limited, as evidenced in the literature. Moreover, their use in addressing multi-objective problems becomes more chal-

lenging due to their limitations in handling multiple dimensions, especially as the problem size increases. Despite these challenges, many current programming languages and solvers can effectively handle small-sized problems with multiple objectives, providing a diverse set of solutions. Therefore, in this study, small-sized problems with a reasonable number of robotic alternatives are solved and analyzed using the MOA package in Julia, which supports various multi-objective optimization algorithms, such as the weighted sum and ϵ -constraint. Additionally, the Gurobi 9.5.2 solver is employed to enhance the solution process. Initial experimental results indicate that using the weighted sum with different sets of weights often leads to identical solutions, resulting in repetitive and inefficient optimization runs, a finding corroborated by the literature. Meanwhile, the ϵ -constraint method provided a richer and more diverse set of efficient options.

4.2 Meta Heuristic Approaches

Meta-heuristic algorithms are advanced computational methods designed to solve complex optimization problems that are often beyond the reach of traditional, deterministic algorithms. These approaches are particularly valuable in scenarios where the solution space is vast, and exact methods fail to find a practical solution within a reasonable time frame. Meta-heuristics are not limited to finding a single optimal solution; instead, they are capable of exploring a diverse set of potential solutions, making them well-suited for multi-objective optimization problems. Such problems require multi dimensional optimization of multiple ob-

jectives, necessitating a balance that meta-heuristic techniques can efficiently find multiple trade-offs.

Among many meta-heuristic algorithms, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and Restarted Simulated Annealing (RSA) stand out for their robustness and effectiveness. NSGA-II, an evolution of the earlier Genetic Algorithm (GA), excels in handling multi-objective problems by classifying solutions into different fronts based on dominance criteria [44]. This facilitates the identification of a set of optimal trade-off solutions known as the Pareto front, thus ensuring a diverse solution set that does not compromise one objective for another. On the other hand, RSA, an adaptation of the classical Simulated Annealing algorithm, introduces a mechanism to escape local optima by periodically resetting the algorithm's state, thus broadening the exploration of the solution space [9].

This study employs both NSGA-II and RSA to address a complex multi-objective scenario, exploring their effectiveness in navigating and optimizing the trade-offs inherent in the problem. After applying both algorithms, specific evaluation metrics are utilized to compare their performance in terms of obtaining Pareto front solutions. Based on this comparative analysis, the better-performing method will be selected for further detailed analysis.

4.2.1 Encoding And Decoding

In metaheuristics, solution encoding refers to the representation of a problem's solution in a structured format suitable for the algorithm to process. Common

Task permutation	3	2	6	1	9	4	5	7	8
------------------	---	---	---	---	---	---	---	---	---

Robot allocation	14	19	15	10
------------------	----	----	----	----

Figure 4.1: Task permutation and robot vector

encoding formats include binary strings, real-valued vectors, permutations, or custom representations, depending on the problem type. The encoding determines how solutions are manipulated during optimization operations like mutation, crossover, or neighborhood search [44]. To encode solutions in this study, two vectors are employed: the task permutation vector and the robot allocation vector [34], [37]. The task permutation vector specifies the task assignments, while the robot allocation vector determines the allocation of robots. To illustrate the concept, consider an example depicted in Figure 4.1, which involves a problem with 9 tasks and 4 robots, solved considering 20 alternative robots. In the task permutation vector, tasks that are positioned earlier have higher priority and are assigned first, provided they meet the precedence relations. In the robot allocation vector, each robot's position indicates the station and side to which it is assigned. For example, robot 14 is assigned to the first station on the left side, and robot 19 to the first station on the right side.

Both a decoding process and an initial cycle time are necessary to transform the encoding into a feasible solution. The task permutation vector and the initial cycle time serve as inputs for the decoding scheme. The detailed assignment

of tasks to each mated-station is outlined based on this decoding scheme. It was originally presented by [37], which addresses the TRALB problem, including setup times. However, in this study, setup times are ignored, and the procedure is extended to accommodate varying robot capabilities and to select the robot with the lowest investment cost, as explained in Table 4.1 below.

Table 4.1: Decoding procedure for TRALBP-TT with capability aspect

Input: Task permutation, and initial cycle time
Step 1: Open a new mated-station. Step 2: Identify the assignable tasks for both sides of the current mated-station, ensuring adherence to precedence, direction, and cycle time constraints. Step 3: Assign the robot with the lowest cost capable of performing the tasks in the assignable set. Verify the robot’s capability each time the assignable task set is updated; if the current robot cannot perform the updated set, reassign a capable robot. Step 4: If no assignable tasks remain and there are unassigned tasks, return to Step 1. Else Execute Step 5. Step 5: Determine the allocation of tasks to a side. Select the side with greater available capacity, or default to the left side if capacities are equal. Step 6: Select the task with the least sequence-dependent idle time from the assignable task set that is in an earlier position and assign it to the selected side. Step 7: Adjust the capacities of both sides of the current mated-station. Step 8: If all tasks have been assigned: Calculate total energy consumption and total reconfiguration cost. Terminate the decoding procedure. Else Go to Step 2.
Output: Detailed task assignment, robot allocation vector, cycle time, total energy consumption, and total cost.

In Step 2, a task is considered assignable if it satisfies precedence, direction,

and cycle time constraints. However, if the current mated-station is the last one, the cycle time constraint may be relaxed. In such cases, the maximum value of the finishing times of tasks across all mated-stations is considered as the cycle time for the feasible solution. The initial cycle time is estimated using the following formula:

$$CT = 2 \cdot \frac{\sum_{i \in I} \sum_{r \in R} t_{ir}}{nr \cdot 2 \cdot nm}$$

Where nm represents the number of mated stations and nr the number of utilized robots. This formula and an iterative mechanism to update the cycle time are integrated into the evolution process of the RSA algorithm, as detailed in [37].

4.2.2 Neighbor Structure

In literature, the insert, swap and mutations operators are widely applied to different types of assembly line balancing problems in meta heuristics algorithms as they help enhance exploration capacity and expand the search space. These neighbor structures are commonly used to generate new solutions or create new generations. In this study, to generate a new neighbor solution from the current task permutation vector in RSA , both the insert operator and the swap operator are employed. Initially, a task permutation vector is randomly selected. Subsequently, either the insert operator or the swap operator is applied at random to derive a new vector. Examples of both the insert and swap operators applied to a task permutation vector are illustrated in Figure 4.2. Unlike previous studies

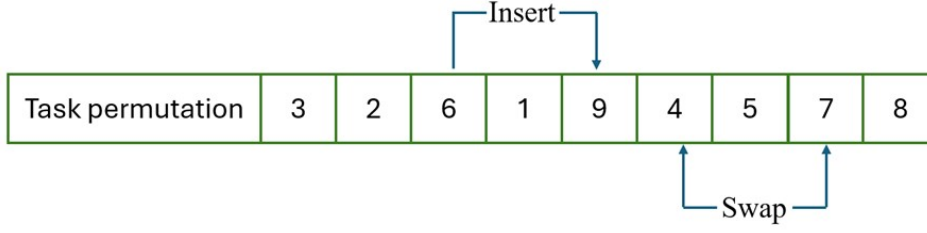


Figure 4.2: Insert and swap operators

[34], [37], the robot vector in this study is derived as part of the overall solution, rather than being independently generated, due to the integration of the robot's capabilities.

In NSGA-II, a multi-point crossover technique is used to generate an offspring from two parent solutions. Figure 4.3 illustrates a simple example. In this example, the crossover points selected are positions 3, 5, and 7. Using these points, segments are defined that will be swapped from Parent 2 into the offspring, which initially starts as a copy of Parent 1. The segments [7, 6, 5] and [5, 4, 3] from Parent 2 are used to replace the segments [3, 4, 5] and [6, 7, 8] in Parent 1, resulting in the presented offspring. This crossover method ensures diversity in the population and combines traits from both parents to potentially create a more optimal solution.

4.2.3 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II is a multi-objective evolutionary algorithm based on non-dominated sorting, designed to reduce computational complexity. It addresses limitations such

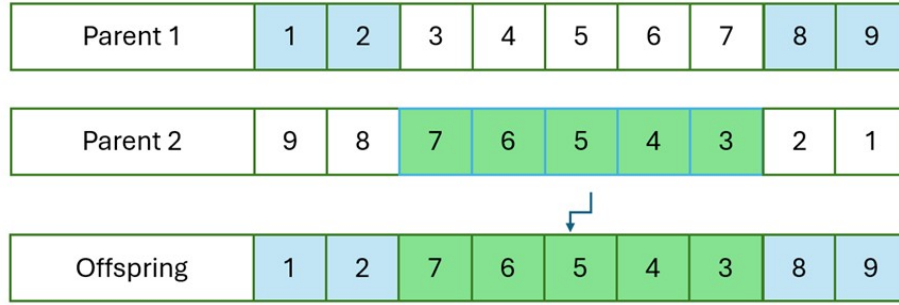


Figure 4.3: Multi-point crossover

as non-elitism and the necessity of specifying a sharing parameter [44]. Due to its robustness and efficiency, NSGA-II has been widely adopted by researchers to solve diverse multi-objective optimization problems across various fields.

The process begins with the generation of a random population of size $nPop$, a parameter of the algorithm. Throughout the iterative cycles, the objective values of each individual in the population are evaluated using a designated function. Following this, individuals are ordered through a non-dominated sorting process. Each individual is assigned a rank corresponding to their level of non-domination, where individuals in the first front have the lowest rank, and those in successive fronts have incrementally higher ranks. Subsequently, the Crowding Distance is calculated for members within each front to maintain diversity in the solution space.

A Boolean tournament selection operator based on crowded-comparison is employed, necessitating the evaluation of both the rank and the crowding distance for each individual in the population. Initially, this operator selects two members from the population. If these members have the same rank, the one with the larger

Crowding Distance is chosen. If their ranks differ, the individual with the lower rank is selected. Subsequently, a new offspring population of size n is generated through selection, followed by crossover and mutation processes. This results in a combined population of original and new members, totaling $nPop + n$ in size. The pseudocode for NSGA-II is depicted in Table 4.2 [45].

Table 4.2: Pseudocode for NSGA-II

Procedure: Typical NSGA-II

Begin

Step 1. Determine population size ($nPop$), crossover rate (β), mutation rate (r), and iteration numbers (Itr).

Step 2. Randomly generate the first population.

Step 3. Sort individuals using Non-dominated Sorting (NS) Algorithm and calculate Crowding Distance for each of them.

3.1. Randomly choose a few individuals from the population.

3.2. If the individuals belong to different fronts, choose the one with the lower front number.,

ELSE select one with higher Crowding Distance.

Step 4. Apply crossover and mutation operators to the parents to produce offspring.

Step 5. Sort parents and offspring using Non-dominated Sorting (NS) Algorithm and determine the Crowding Distance for each individual.

Step 6. Choose the top individuals based on rank and crowding distance from the current population and offspring to form the next generation.

Step 7. Check the termination condition and return to Step 3 if it has not been met.

END

Furthermore, a population of exactly $nPop$ size is achieved through a sorting process. This process involves two distinct steps: sorting the solutions first by

their Crowding Distances in descending order, and then by their ranks in ascending order. The newly formed population is then used to generate the subsequent generation by systematically iterating through these stages. This procedure continues until a predefined termination condition is met. The pseudocode for the Non-dominated Sorting (NS) algorithm is illustrated in Table 4.3 [45].

Table 4.3: Non-dominated sorting algorithm

Procedure: Typical Non-dominated Sorting (NS) Algorithm

Begin

Step 1. Put $l = 1$ and also for all the population's individuals $Sp = \{\}$, $Np = 0$.

Where:

l : is the counter of front.

Sp : donates set of population dominated by individual number p (p^{th} member).

Np : the number of the p^{th} member is dominated by the others.

Step 2. For each two members of the population (like p and q):

IF q is dominated by p , THEN add q to the set of Sp and $Nq = Nq + 1$.

IF p is dominated by q , THEN add p to the set of Sq and $Np = Np + 1$.

Step 3. Put members valued with $Np = 0$ in first front (F_1).

Step 4. Put $Q = \{\}$ as a default for the front of F_1 .

Step 5. For each member of F_l (like p), if q is a member of Sp , subtract a unit from Nq , and if $Nq = 0$, add q to the Q (i.e., F_{l+1}).

Step 6. IF the Q value is null, THEN end the sorting process,

ELSE $l = l + 1$ and back to Step 4.

END

4.2.4 Restarted Simulated Annealing Algorithm (RSA)

The Simulated Annealing (SA) algorithm was created in 1983 by Kirkpatrick et al. It simulates how metals are heated and cooled to improve solutions in

combinatorial optimization [46]. This effective and simple approach has also been applied to two sides layout. According to Li et al. (2019) which conducted an extensive study with 13 algorithms for TRALBP, it was concluded that SA and RSA achieved the second-highest performance level [37].

One of the main characteristics of Simulated Annealing (SA) is that it reduces the probability of getting trapped in local optima by selecting a less optimal solution based on a probability function. The process begins with an initial solution, S , followed by the generation of a neighboring solution, S' . If the new objective, $f(S')$, is lower than the current objective, given a minimization case, then the new solution is accepted. If not, the new solution may still be accepted based on a probability given by $\exp \frac{f(S')-f(S)}{T}$. Here, T represents the temperature, which acts as a control parameter that decreases as the search progresses [46].

RSA is an enhanced version of the SA algorithm, designed to improve the exploration of search space. In this research, the multi-objective restarted simulated annealing algorithm (RSA) developed by Li et al. to simultaneously optimize two objectives is adapted to manage three objectives [9]. The RSA algorithm employs a new acceptance criterion and a restart mechanism as diversification strategies to achieve a Pareto-optimal set. The process of the restarted SA algorithm is illustrated in Figure 9, where the main loop continues until the termination criterion is satisfied. Initially, a solution is randomly generated using the encoding procedure with a neighbor structure. Then, by applying the decoding steps outlined in Section 4.2.1, the objective function values of the solution are obtained. Subsequently, the Pareto optimal set is updated through the application of an

acceptance rule and a restart mechanism. Detailed information about the RSA algorithm presented in algorithm 1.

Algorithm 1 RSA for RTALB Problem

Input: RTALB data, RSA parameters, dn (Number of moves before restart)

Output: Pareto-optimal set (POS)

Begin:

Generate an initial solution S ;

$d := 0$; $ns := 0$;

while termination criterion is not satisfied **do**

while $ns < NS$ (Number of repetitions at each temperature level) **do**

 Obtain a new solution S' with a neighbor structure;

$ns := ns + 1$;

if S' is not dominated by the solutions in POS **then**

 Update POS;

else

$d := d + 1$;

if $d > dn$ **then**

 Select a new solution S'' with the restart mechanism and $S \leftarrow S''$;

else

 Select one objective randomly (f) and $\Delta = f(S'') - f(S)$;

if $\Delta \leq 0$ **then**

$S \leftarrow S''$;

else

$S \leftarrow S''$ with a probability of $\exp \frac{-\Delta}{T \times f(S)}$;

end if

end if

end if

end while

$T := \alpha T$;

end while

Pareto-Optimal Set Update To obtain the Pareto-optimal set, a systematic process is applied. First, a neighbor solution S is obtained by applying the neighbor structures. If this solution is non-dominated, meaning it provides a better objective function value on at least one criterion, it will replace the current solution. The Pareto-optimal set is then updated by removing any solutions dominated by

S' . The next iteration proceeds with S as the current solution.

Conversely, if the new solution is dominated by those in the Pareto-optimal set, a simple multinomial probability mass function is employed. Utilizing this probability function may result in accepting a suboptimal solution. However, this approach enhances the search space, potentially helping the algorithm in escaping local optima. This method has proven to be better in some cases with the techniques of weighting and scaling each objective. Its advantage lies in the ease of implementation [47]. In each iteration, the new solution with the current solution is compared based on this probability mass function, which randomly selects an objective. Given that the model considers three objectives, each objective has a selection probability of approximately 0.3 in the probability mass function.

For instance, if the new solution is dominated, it will be evaluated based on one of the three objective values $f(*)$ chosen randomly. The change in the selected objective function between the dominated solution and the current solution $\Delta = f(S') - f(S)$ is calculated. If the change is negative, the new solution is accepted to replace the current one, indicating that the dominated solution provided a better value in the selected objective. If the change is positive, the dominated solution is accepted with a probability of $\exp \frac{-\Delta}{T \times f(S)}$. It is important to note that, given the different magnitudes of the three objectives, $\exp \frac{-\Delta}{T \times f(S)}$ is used instead of the original probability $\exp \frac{-\Delta}{T}$ to ensure the objectives share the same temperature [9].

4.2.5 Restart Mechanism

In this study, the restart mechanism proposed by [9] is adopted as a diversification strategy to enhance the algorithm's ability to explore different regions of the solution space. This approach increases the chances of finding the global optimum and obtaining well-spread non-dominated solutions. Before the algorithm undergoes the restart or reinitialization phase, a number of individual solution updates, or "moves," referred to as (dn) , must be performed. If the algorithm is unable to obtain a new non-dominated solution after (dn) moves, a solution from the Pareto-optimal set is selected based on specific criteria. The criterion used for selection is the crowding distance operator, developed by [44]. This operator helps to provide a more spread-out set of solutions in the Pareto-optimal front by selecting the solution that is furthest from others on the Pareto frontier, thereby reducing gaps in the objective solution space. However, the algorithm may repeatedly select these isolated solutions with minimum or maximum objective values.

To address this issue, two modifications were introduced by [9]. First, in the original algorithm, the distances of the extreme solutions were set to infinity (∞). However, to prioritize these solutions during selection due to their dominance and importance, their distances are now set to 1 instead of infinity. Additionally, to reduce the possibility of selecting the same solution repeatedly, a selection counter is used to record the number of times each solution is selected. Consequently, the solution with the fewest selections will have the largest distance, increasing its likelihood of being chosen as the current solution. The detailed restart mechanism

procedure is shown in algorithm 2 [9].

Algorithm 2 Restart Mechanism Based on Modified Crowding Distance Operator

Input: POS (Pareto-optimal set), Obj (objective), dist (distance), SelectCounter (number of times selected)

Output: Solution with the largest distance to replace the incumbent one

Initialization: Setsize $\leftarrow |POS|$

for all objectives m **do**

 Sort POS using the current objective m

$f_m^{\max} \leftarrow$ maximum value for objective m

$f_m^{\min} \leftarrow$ minimum value for objective m

for $i = 2$ to Setsize $- 1$ **do**

$POS[i]_{\text{dist}} \leftarrow POS[i]_{\text{dist}} + \frac{(POS[i+1]_{\text{Obj}_m} - POS[i-1]_{\text{Obj}_m})}{(f_m^{\max} - f_m^{\min})}$

end for

$POS[1]_{\text{dist}} \leftarrow POS[1]_{\text{dist}} + \frac{1}{m}$

$POS[\text{Setsize}]_{\text{dist}} \leftarrow POS[\text{Setsize}]_{\text{dist}} + \frac{1}{m}$

end for

for $i = 1$ to Setsize **do**

$POS[i]_{\text{dist}} \leftarrow \frac{POS[i]_{\text{dist}}}{POS[i]_{\text{SelectCounter}}}$

end for

 Select the solution with the largest distance to replace the incumbent one

4.2.6 Cooling schedule

In RSA, the cooling schedule significantly impacts the overall search behavior of the algorithm, as it controls the temperature reduction over time [48]. After the algorithm executes for NS iterations, representing the local search phase, the temperature T is updated using a cooling rate parameter, which ranges from 0 to 1. As the search progresses and T decreases, the probability of accepting worse solutions is reduced.

In the simulated annealing algorithm, three primary parameters need to be calibrated: the initial temperature (T_0), the cooling rate (α), and the number of iterations for each temperature update (NS). Due to the integration of the restart

mechanism, an additional parameter, the number of moves before restart (dn), is also incorporated. Therefore, calibration experiments are performed to determine the appropriate levels for each parameter.

CHAPTER 5

EVALUATION EXPERIMENTS

The optimization performance of the developed model, including the performance of the meta heuristics algorithms, must be evaluated. Therefore, based on well-known benchmark cases in the two-sided assembly line field, benchmark sets are generated in the following section to incorporate the concept of line rebalancing in a two-sided layout. These problems are then solved using two methods: exact solutions with the Gurobi 9.5.2 solver for small-sized problems, and meta heuristics for all problems. All parameters of NSGA-II and RSA are calibrated based on statistical experiments. Finally, the results are compared to evaluate the performance and the model is analyzed based on multiple technical metrics. Both the exact model and meta heuristics are coded using Julia programming language and solved on a personal computer. The computer used for the experiments was equipped with an Intel Xeon W-1290P CPU at 3.70 GHz and 32.0 GB RAM.

5.1 Benchmark of instances

In general, the cases considered in the literature of two-sided assembly lines are classified into small-size and large-size problems. Problems P9, P12, P16, and P24 are considered small-size problems, whereas P65, P148, and P205 are considered large-size problems. The information about their precedence diagrams and task directions is taken from the literature [6], [7], [16]. The main property of these problems is that they represent actual manufacturing two-sided assembly lines. For instance, the large-sized problems, P65, P148, and P205, are taken from the automobile industry.

The energy consumption for each robot PC_r is classified into operating power consumption and standby power consumption SPC_r , with standby power consumption being 10% of the operating power consumption. The energy consumption data are taken from [9], where they are randomly generated for each robot per unit of time. The additional data required to address greenfield and brownfield balancing problems are taken from [11]. This data includes robot costs, which cover investment costs, processing costs, and savings. Using a uniform distribution, the robot costs are generated between $C_{\min}=5000\$$ and $C_{\max}=30000\$$.

To reflect the impact of cost on robot selection, where more expensive robots are more efficient in processing tasks t_{ir} , inverse scaling relative to the investment in robot cost (C_{new}) is used. Therefore, for each robot r , the processing time t for each task i is generated based on the following equation (5.1), which is also adapted from [11]. Here, t is the original processing time, with the lower bound

being $t_i \times 0.8$ and the upper bound $t_i \times 1.2$ as per the literature. In the original formula, the actual minimum and maximum cost values are used for calculations.

$$t_{ir} = \frac{(t_i \times 0.8) - (t_i \times 1.2)}{c_{\max} - c_{\min}} \times c_{r,\text{new}} - c_{\min} + (t_i \times 1.2) \quad \forall i, r \quad (5.1)$$

However, since the original costs are not available, the minimum and maximum cost values are used instead. Also, in [11], the generated processing costs are based on the robot's capability. For the savings, it is assumed that 10% of the generated robot investment costs are saved. To accommodate large problems in the two-sided model, this study considers the number of robots in the following ranges: 20, 40, 60, 80 alternatives using meta-heuristics, and for the exact method, a smaller number of robot alternatives is considered in the range 5, 10, 15, 20 to reduce the problem complexity on the solver.

The task types are initialized based on the distribution provided by [11], which is derived from assembly lines handling rigid small to medium-sized parts in the automotive industry. Since the model is simulated based on the automotive industry, this distribution is used. The distribution is as follows: 50% of the tasks are joining, 35% are handling, and 15% are separation.

In this study, the robots have different capabilities based on their investment cost. The higher the cost, the more task types a robot can perform. The initial capability distribution is also taken from [11], where 40% of robots perform joining, 35% handle, and 25% perform separation tasks. Subsequently, arbitrary values were selected to reflect that more expensive robots are more likely to have greater

production capabilities. The top 60% of the most expensive robots receive full capabilities, the next 20% receive two capabilities, and the last 20% are likely to receive only one capability. In the exact model, penalty values are used to represent the capabilities of the robots. Based on the generated capability matrix, large values are assigned to indicate that a particular robot cannot perform a specific task. This approach ensures that the model accurately reflects the capabilities of the robots, effectively preventing the assignment of unsuitable robots to certain tasks.

5.2 Parameter calibration

For each meta-heuristic algorithm, there are parameters that significantly influence its performance. Therefore, parameter tuning is conducted through experiments to optimize algorithm performance. Since different parameter levels do not impact the algorithm's performance in small-sized problems, the parameters are calibrated based on the experiments conducted on the largest case, P205, which considers the highest number of robots, totaling 80 alternatives.

A common approach for conducting experiments is to use a full factorial design, which involves testing all possible combinations of parameters. While this method offers high precision, it demands substantial resources, including increased time and cost [49]. Therefore, the Taguchi method is employed in this study. This method is more efficient than the full factorial design as it utilizes orthogonal arrays. With a smaller number of combinations, the entire range of parameter

levels can still be evaluated with reduced time and cost [50], [51].

In this method, the experimental scheme divides the factors into two types: controllable factors and uncontrollable factors, which introduce noise into the data. The aim of applying this method is to achieve the highest possible robust performance level by testing the impact of the controllable factors. It represents the main impact of each factor level, demonstrating their relative importance to the objective function [52]. Consequently, it addresses the deficiencies caused by uncontrollable factors, where eliminating their noise is impractical and potentially impossible.

To optimize each factor, the Taguchi method analyzes the experimental data by utilizing the Signal-to-Noise (S/N) ratio. A high S/N ratio is desirable, as it indicates a more robust system despite variations in the experimental data. This method determines the Signal-to-Noise (S/N) ratio calculations based on the response variable of the objective functions. There are three possible classifications: If the response variable needs to be minimized, the ratio is classified as 'the smaller, the better.' If it needs to be maximized, it is classified as 'the larger, the better.' If achieving a specific objective function value is the target, the classification used is 'nominal is best.' [51].

To evaluate a model with multiple objective functions where the objectives are not co-dimensional, a response variable must be selected that can explain the impact of factor levels on the objective functions. Therefore, in this study, the response variable considered is the convergence rate of the Pareto-optimal set. This measures the convergence of the obtained Pareto set to the approximated

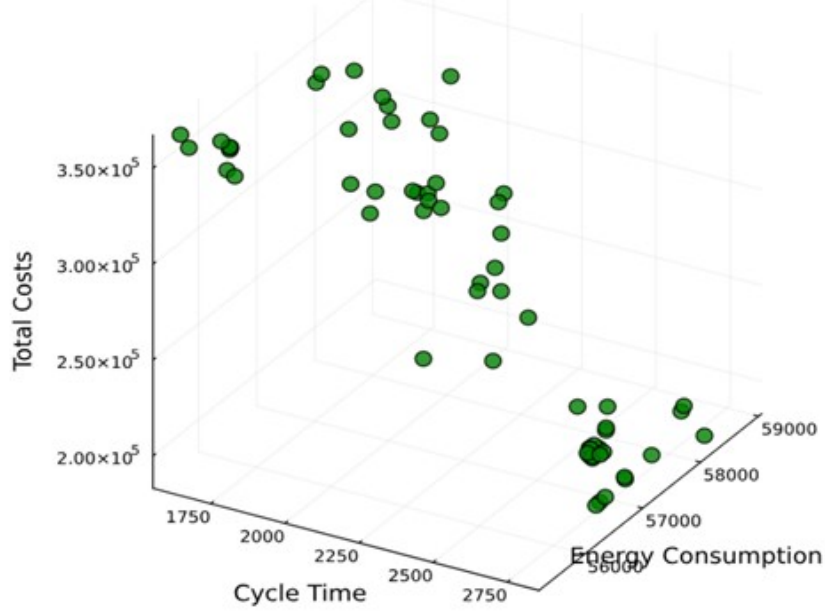


Figure 5.1: Non-dominated solutions in the TP

true Pareto set (TP).

The TP is the set of optimal solutions where no other solutions are better in all objectives. TP is approximated by running both algorithms, RSA and NSGA-II, for the highest termination condition, which is $nt \times nt \times 30$, twenty times for each. As shown in Figure 5.1, a total of 61 solutions represent non-dominated solutions in the TP.

To obtain the convergence rate, the Euclidean distance must first be calculated. According to Equation 5.2, the Euclidean distance d_{tl} represents the minimum distance between the l th solution from the Pareto front set and solution y in the TP. The maximum ($f_{\max}$) and minimum ($f_{\min}$) values of the m th objective are then used to normalize the distances. The Euclidean distance is subsequently divided by the number of solutions obtained in the Pareto front (P_s) as shown in Equation 5.3. A low convergence rate (CP) indicates a better approximation of

the true Pareto front.

$$d_{t_1} = \min_{n=1}^{|TP|} \sqrt{\sum_{m=1}^3 \left[\frac{f_m(x_l) - f_m(y_n)}{f_m^{\max} - f_m^{\min}} \right]^2} \quad (5.2)$$

$$CP(P_s) = \frac{\sum_{l=1}^{|P_s|} d_{t_l}}{|P_s|} \quad (5.3)$$

Since minimum values of the convergence rate are desirable, the S/N ratio would follow the 'smaller the better' case. As shown in Equation 5.4, Obj_e is the objective function value in the single-objective case [52]. To accommodate the multi-objective case, the objective function value is replaced by the response variable, which is the convergence rate in the e-th experiment. The squared convergence rate is divided by $nExp$, which is the number of observations for each parameter combination, with each combination replicated three times.

$$\frac{S}{N} = -10 \log_{10} \left(\frac{\sum_e obj_e^2}{nExp} \right) \quad (5.4)$$

5.3 Experiment Design

The orthogonal arrays are constructed for each algorithm based on the number of parameters and their corresponding levels. For RSA, there are four factors that must be considered, whereas NSGA-II requires three parameters to be calibrated [44]. The parameters and their levels considered are presented in Table 5.1.

Considering a full factorial design would result in a total of 54 combinations

Table 5.1: The parameters and their levels

Parameter	Levels
RSA	Initial temperature T_0 : 0.5, and 1
	Cooling rate: 0.9, 0.95, and 0.98
	Number of iterations (NS): 500, 1000, and 2000
	Number of moves (dn): 100, 200, and 300
NSGA-II	Population size (N_{popA}): 100, 200, and 300
	Crossover rate: 0.7, 0.8, and 0.9
	Mutation rate: 0.1, 0.2, and 0.3

for RSA and 27 combinations for NSGA-II. By applying the Taguchi method, the construction of orthogonal arrays reduced the number of combinations to 18 for RSA and 9 for NSGA-II. Tables 5.2 ,and 5.3 show the considered combinations for both algorithms.

5.4 Experiment Results

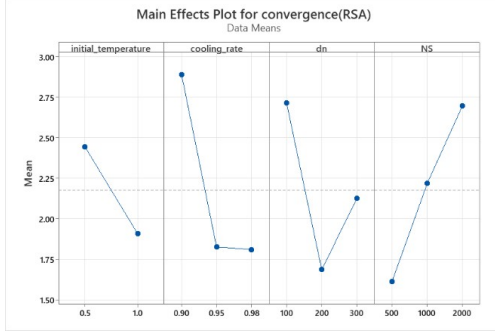
For each parameter, the selected level should have the minimum convergence rate compared to other levels and simultaneously provide the maximum S/N ratio. The Taguchi method is applied using Minitab software to construct the main cause-and-effect graphs. As shown in Figure 5.2, changing the RSA parameter levels impacts both, the convergence rate and S/N ratio among the experiment runs.

Table 5.2: Orthogonal arrays for RSA parameters

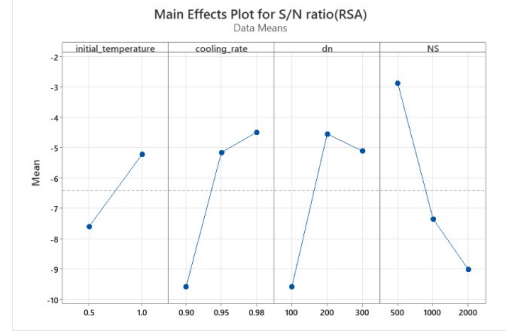
Combination no.	T0	Cooling rate	Ns	dn
1	0.5	0.9	500	100
2	0.5	0.9	1000	200
3	0.5	0.9	2000	300
4	0.5	0.95	500	200
5	0.5	0.95	1000	300
6	0.5	0.95	2000	100
7	0.5	0.98	500	300
8	0.5	0.98	1000	100
9	0.5	0.98	2000	200
10	1	0.9	500	100
11	1	0.9	1000	200
12	1	0.9	2000	300
13	1	0.95	500	200
14	1	0.95	1000	300
15	1	0.95	2000	100
16	1	0.98	500	300
17	1	0.98	1000	100
18	1	0.98	2000	200

Table 5.3: Orthogonal arrays for NSGA-II parameters

Combination no.	NpopA	Crossover rate	Mutation rate
1	100	0.7	0.1
2	100	0.8	0.2
3	100	0.9	0.3
4	200	0.7	0.2
5	200	0.8	0.3
6	200	0.9	0.1
7	300	0.7	0.3
8	300	0.8	0.1
9	300	0.9	0.2



(a) Main Effects Plot for Convergence



(b) Main Effects Plot for S/N Ratio

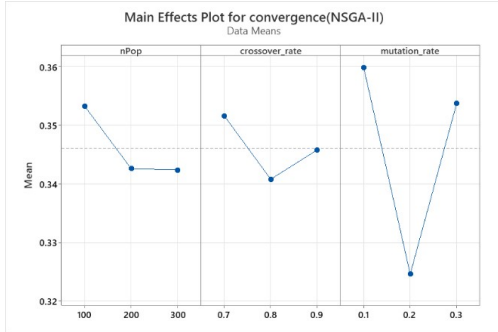
Figure 5.2: Main Effects Plots for Convergence and S/N Ratio (RSA)

First, the convergence clearly has lower values for an initial temperature of 1 compared to 0.5, which is explained by a higher initial temperature allowing the algorithm to explore the solution space more broadly. Additionally, the higher S/N ratio indicates that this level not only provides better convergence but also results in more robust performance across different runs. Second, a cooling rate that is too low (0.9) might cause the algorithm to cool down too quickly, leading to a higher convergence rate. Meanwhile, 0.95 and 0.98 provided a lower convergence rate and higher S/N ratio, with 0.98 providing a slightly higher S/N ratio. However, the selected level is 0.95 to balance exploration and exploitation, providing enough time to explore the solution space while gradually focusing on the best solutions.

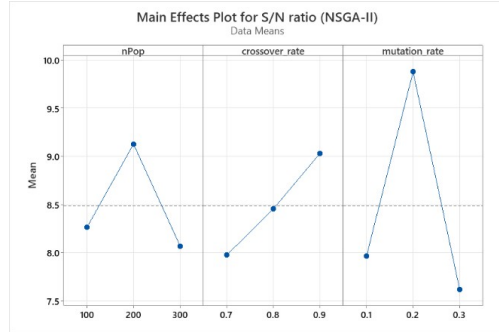
Third, for the lowest value of the number of moves before restart (dn), the algorithm may not have enough time to refine the current solution before restarting, leading to a higher convergence rate. Conversely, for the highest dn value, the algorithm might spend too much time refining a potentially suboptimal solution before exploring new areas, resulting in worse convergence values. Based on both values, 200 provided the lowest convergence value and highest S/N ratio, indicat-

ing it contributes to a good balance between refinement and exploration. Lastly, the highest S/N ratio and lowest convergence rate at 500 iterations confirm that more frequent temperature updates help the algorithm adapt dynamically, providing more robust performance. Higher iteration values result in less frequent updates, reducing the algorithm's ability to provide better solutions, which leads to lower S/N ratios.

For NSGA-II, the different levels of parameters also impact the overall convergence. As shown in Figure 5.3, the convergence rate improves as the population size increases from 100 to 200, then remains relatively stable from 200 to 300. A population size of 200 provides a good balance between exploration and exploitation, allowing the algorithm to converge efficiently, and also provides the highest S/N ratio. Increasing the population size beyond 200 does not significantly improve convergence and results in the lowest S/N ratio.



(a) Main Effects Plot for Convergence



(b) Main Effects Plot for S/N Ratio

Figure 5.3: Main Effects Plots for Convergence and S/N Ratio (NSGA-II)

For the crossover rate, the convergence improves as the crossover rate increases from 0.7 to 0.8, then slightly worsens at 0.9. Despite 0.9 providing a slightly higher S/N ratio, the selected level is 0.8 since it offers the best convergence. This is likely

due to an optimal balance between creating diverse offspring and preserving good solutions. It is important to note that higher crossover rates (0.9) may introduce too much diversity, adversely impacting the convergence process.

For the mutation rate, convergence is worst at a mutation rate of 0.1, then improves significantly at 0.2, and worsens again at 0.3. The S/N ratio also indicates that higher mutation rates (0.3) introduce too much variability, reducing robustness. The 0.2 value provides both the best convergence and the highest S/N ratio, indicating that it is the optimal setting for balancing performance and robustness. Table 5.4 presents the selected levels for all parameters of both RSA and NSGA-II algorithms.

Table 5.4: Selected levels for all parameters of both algorithms.

Parameter	Selected Level
RSA	
Initial temperature T_0	1
Cooling rate	0.95
Number of iterations (NS)	500
Number of moves (dn)	200
NSGA-II	
Population size (N_{popA})	200
Crossover rate	0.8
Mutation rate	0.2

5.5 Comparison matrices

To determine which algorithm delivers superior performance, two evaluation metrics are utilized in conjunction with the convergence metric: the ratio of non-dominated solutions and the spread metric, which are essential for assessing multi-objective optimization algorithms [53], [54]. The ratio of non-dominated solutions, denoted as $RP(P_s)$, quantifies the proportion of solutions within a Pareto set P_s that are not dominated by any other solution in the overall set P , which represents the union of all solution sets. Specifically, the formula 5.5 computes the number of solutions in P_s that are not dominated by any solution in P and divides this by the total number of solutions in P_s to determine the ratio. Here, P_s is a subset of P comprising Pareto-optimal solutions. The numerator is the count of solutions in P_s that are not dominated by any other solutions in P , and the denominator $|P_s|$ represents the total number of solutions in P_s . A high value of $RP(P_s)$ is desirable as it indicates that a significant proportion of the solutions in P_s are non-dominated by any other solutions in P .

$$RP(P_s) = \frac{|P - \{X \in P_s \mid \exists Y \in P : Y > X\}|}{|P_s|} \quad (5.5)$$

The spread metric, denoted as $SP(P_s)$, evaluates the distribution or spread of solutions within the Pareto front. It assesses the extent to which the solutions in the Pareto set P_s cover the trade-off surface, aiming to achieve a broad and uniform distribution of solutions. Equations 5.6 and 5.7 are employed to compute the spread metric. The terms d_{f_1} and d_{f_2} represent the Euclidean distances from

the extreme solutions in the Pareto set P_s to the boundary solutions in the true Pareto set, measuring the extent to which the most extreme solutions in the computed set diverge from the actual extremes in the optimal front.

The term sd_l denotes the Euclidean distance between two consecutive solutions in P_s , calculated for each pair of consecutive solutions within the set. The average distance between consecutive solutions, represented by $\bar{sd}$, is calculated next. Equation 5.7 quantifies the relative deviation of each inter-solution distance sd_l from the average distance $\bar{sd}$, normalized by the total spread, including d_{f_1} and d_{f_2} . This formula captures both the absolute extent of the solutions' spread and the uniformity of their distribution.

Subsequently, Equation 5.6 normalizes the Euclidean distance between each pair of consecutive solutions by the range of each objective. A smaller value of $SP(P_s)$ is indicative of a superior spread, as it signifies a minor deviation from the average distance $\bar{sd}$, suggesting a more uniform distribution of solutions along the Pareto front. Conversely, a larger value denotes a poorer spread, indicative of gaps or clusters within the solution set, which may be undesirable in decision-making contexts where a well-distributed set of trade-offs is crucial. If the SP value exceeds 1.0, it suggests an inadequate solutions distribution.

$$SP(P_s) = \frac{sd_{f_1} + sd_{f_2} + \sum_{l=1}^{|P_s|} |sd_l - \bar{sd}|}{sd_{f_1} + sd_{f_2} + (|P_s| - 1) \bar{sd}} \quad (5.6)$$

$$sd_l = \sqrt{\sum_{m=1}^3 \left[\frac{f_m(x_l) - f_m(x_{l+1})}{f_m^{\max} - f_m^{\min}} \right]^2} \quad (5.7)$$

Each problem is assessed using a variable number of alternatives. For small-sized problems, the number of alternatives considered (n_r) is 20 and 40, with a maximum of four mated stations m_{max} . For larger cases, the number of alternatives spans four scenarios: 20, 40, 60, and 80, with the maximum number of mated stations set at 8. To accommodate the inherent randomness typical of metaheuristic algorithms, each scenario was replicated 20 times. The analysis was conducted in two termination conditions $nt \times nt \times N$, for $N = 10$ and $N = 20$, to evaluate the stability and convergence of the algorithm over time. Accordingly, a total of 800 data samples (40×20) for each algorithm were analyzed under a termination condition of $nt \times nt \times 10$ milliseconds. Table 5.5 summarizes the average results from the three-evaluation metrics for both algorithms, NSGA-II and RSA.

Table 5.5: Average results from the three evaluation metrics for different problems.

Problem	m_{max}	r	$N = 10$						$N = 20$					
			NSGA-II			RSA			NSGA-II			RSA		
			RP	CP	SP	RP	CP	SP	RP	CP	SP	RP	CP	SP
P9	4	20	1	0	0.98	1	0.87	0.65	1	0	0.98	1	0.82	0.6
P9	4	40	1	0	0.99	1	0.23	0.22	1	0	0.99	1	0.21	0.21
P12	4	20	1	0.06	0.98	1	0.24	0.33	1	0.08	0.98	1	0.24	0.35
P12	4	40	1	-	-	1	-	-	1	-	-	1	-	-
P16	4	20	1	0.05	0.99	1	0.26	0.26	1	0.07	0.98	1	0.21	0.23
P16	4	40	1	0.35	0.98	1	0.33	0.33	1	0.3	0.98	1	0.45	0.37
P24	4	20	1	0.06	0.99	1	0.16	0.77	1	0.07	0.99	1	0.15	0.78
P24	4	40	1	0.1	0.99	1	0.17	0.75	1	0.1	0.98	1	0.17	0.75
P65	8	20	1	0.05	0.99	1	0.09	0.83	1	0.04	0.99	1	0.1	0.78
P65	8	40	1	0.07	0.99	1	0.09	0.81	1	0.07	0.99	1	0.1	0.76
P65	8	60	1	0.07	0.99	1	0.12	0.9	1	0.08	0.99	1	0.12	0.88
P65	8	80	1	0.1	0.99	1	0.11	0.9	1	0.09	0.99	1	0.12	0.89
P148	8	20	1	0.1	0.99	1	0.12	0.77	1	0.08	1	1	0.12	0.77
P148	8	40	1	0.09	0.99	1	0.08	0.79	1	0.07	0.99	1	0.09	0.76
P148	8	60	1	0.12	1	1	0.07	0.89	1	0.11	1	1	0.08	0.92
P148	8	80	1	0.07	0.99	1	0.15	0.89	1	0.07	0.99	1	0.12	0.79
P205	8	20	1	0.25	1	1	0.14	0.93	1	0.25	0.99	1	0.12	0.91
P205	8	40	1	0.06	0.99	1	0.09	0.96	1	0.05	0.99	1	0.09	0.96
P205	8	60	1	0.08	0.99	1	0.13	0.95	1	0.08	0.99	1	0.12	0.95
P205	8	80	1	0.06	0.99	1	0.12	0.93	1	0.07	0.99	1	0.12	0.92

- Indicates there is only one solution in the approximated true Pareto front.

According to the table presented above, the performance of the algorithms is consistent across multiple runs, which underscores their robustness. However, in the case of scenario P12 involving 40 robots, only one solution in the true Pareto front, thus the calculation of the convergence rate and spread metric is infeasible. Regarding the ratio of non-dominated solutions, it is observed that this metric equals 1 for all cases in both algorithms. An RP value of 1 indicates that every solution in each Pareto-optimal set P_s is not dominated by any other solution in the aggregate set P. This suggests that no solution in P_s can be considered dominated when all objectives are considered simultaneously.

Regarding the convergence rate, NSGA-II outperforms RSA in all cases except for two specific instances: scenario P148 with $r = 60$ and scenario P205 with $r = 20$. Conversely, RSA consistently outperforms NSGA-II in terms of the spread metric across all scenarios. Consequently, statistical tests are conducted to determine if there is a significant difference between the performance of the two algorithms. Following an initial normality test which revealed that the data are not normally distributed, a non-parametric Wilcoxon matched-pairs signed rank test is utilized [9].

The following results presented in Table 5.6 are obtained using Minitab 22. P-value (0.000) confirms that the difference in performance between NSGA-II and RSA is statistically significant. NSGA has a median CP value of 0.069885, and RSA has a median CP value of 0.120379. The estimated difference between the medians is -0.055974, with a confidence interval of (-0.0852618, -0.0370863), which means that NSGA's median CP is lower than RSA's median CP.

Table 5.6: Summary of Results for CP Metric

Descriptive Statistics	N	Median
NSGA_CP	38	0.069885
RSA_CP	38	0.120379

Estimation for Difference		
Difference	CI for Difference	Achieved Confidence
-0.0559704	(-0.0852618, -0.0370863)	95.10%

Test	W-Value	P-Value
Not adjusted for ties	956.00	0.000
Adjusted for ties	956.00	0.000

For the spread metric test results presented in Table 5.7, RSA has a lower median SP value of 0.784741 compared to NSGA's 0.991952. This implies that RSA achieves a better, more desirable spread of solutions across the Pareto front. The significant difference confirmed by the test with a p-value of 0.000 supports the conclusion that RSA consistently achieves a better distribution of solutions compared to NSGA-II.

NSGA-II outperforms RSA in terms of convergence performance, as evidenced by its significantly lower median CP value. Lower CP values are preferable as they indicate the algorithm reaches a near-optimal solution more efficiently with higher quality, suggesting that NSGA-II is more effective in converging to the Pareto front than RSA in the scenarios tested. However, RSA showed a better spread metric (SP), which suggests that while NSGA is stronger in driving solutions to the Pareto front more efficiently, RSA provides a broader distribution of solutions

across the Pareto front.

Table 5.7: Summary of Results for SP Metric

Descriptive Statistics	N	Median
NSGA_SP	38	0.991952
RSA_SP	38	0.784741

Estimation for Difference		
Difference	CI for Difference	Achieved Confidence
0.205533	(0.104056, 0.226796)	95.10%

Test	W-Value	P-Value
Not adjusted for ties	2185.00	0.000

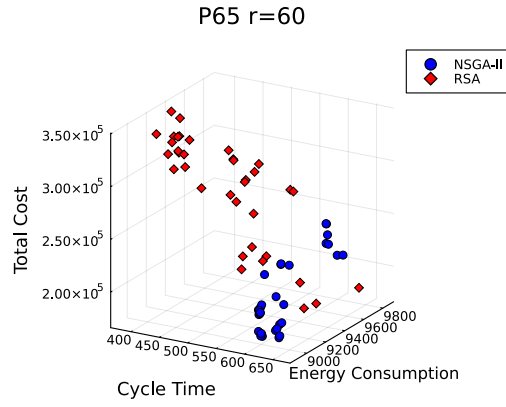
Considering both spread and convergence metrics, RSA tends to distribute its solutions more evenly across the Pareto front, whereas NSGA-II demonstrates more effective convergence to the front. To decide which method to choose, the performance of the two algorithms will be further analyzed in solving the three largest cases, particularly considering scenarios with a high number of alternatives, specifically 60 and 80.

The optimal solutions obtained from 20 runs for scenarios P65, P148, and P205 are documented. As depicted in Figure 5.4 and Figure 5.5, both algorithms exhibit comparable performance. The provided 3D plots, which illustrate the performance of NSGA-II and RSA across three optimization objectives—Total Cost, Cycle Time, and Energy Consumption—yield several noteworthy insights. NSGA-II appears to cluster solutions towards lower Energy Consumption and Total Cost values, emphasizing cost efficiency and energy savings.

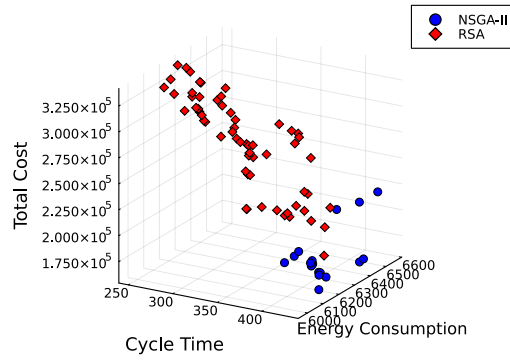
In contrast, RSA demonstrates a broader distribution across all dimensions, indicating a more balanced approach to exploring the trade-offs between the objectives. Significantly, RSA’s solutions extend further into the lower regions of Cycle Time, potentially offering solutions that achieve shorter cycle times without excessively compromising cost or energy consumption. Overall, while NSGA-II may be more suited for applications that prioritize cost and energy efficiencies, RSA exhibits strengths in delivering more consistently balanced solutions, particularly beneficial in scenarios where time efficiency is equally critical.

Table 18 illustrates the minimum objective values achieved by both NSGA-II and RSA across various problem instances. The data clearly indicates that RSA consistently provides lower cycle times in all cases compared to NSGA-II, highlighting its efficiency in minimizing operational delays. Similarly, RSA generally achieves better performance in reducing energy consumption, with the exceptions being cases P148 and P65 at $r=60$, where NSGA-II exhibits marginally lower energy usage. In terms of total cost, NSGA-II demonstrates a competitive advantage, consistently recording lower costs across the problems assessed. This trend aligns with the observed higher cycle times produced by NSGA-II, suggesting a trade-off where longer cycle times contribute to reduced overall costs.

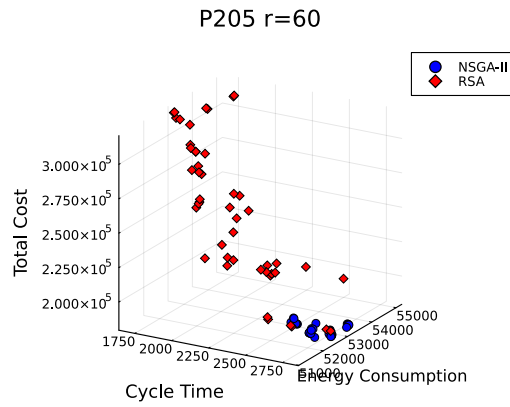
Given the crucial importance of cycle time in two-sided models, due to the nature of the layout where quick operations are essential, the final analysis will focus on utilizing the RSA algorithm. This decision is based on RSA’s superior ability to optimize cycle times, which is paramount in ensuring operational efficiency in such layouts. These observations present the strengths and trade-offs



(a) P65 $r=60$

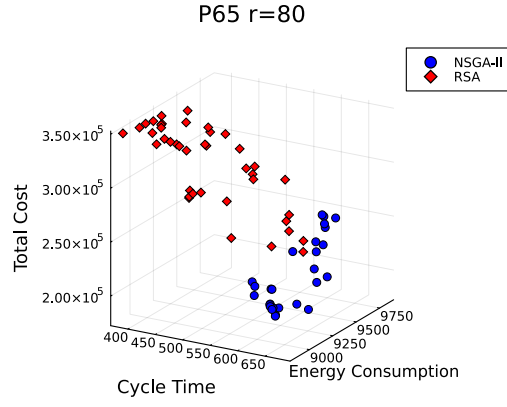


(b) P148 $r=60$

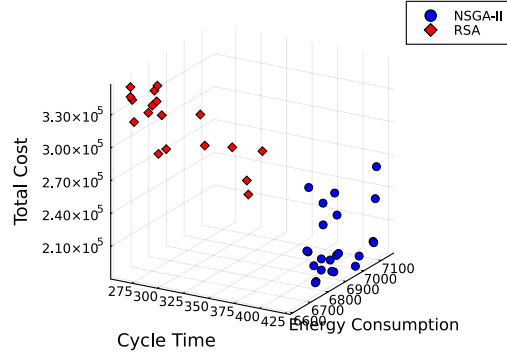


(c) P205 $r=60$

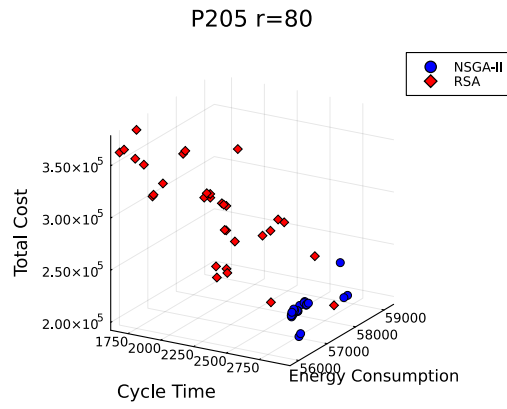
Figure 5.4: NSGA-II and RSA Pareto fronts at $r = 60$.



(a) P65 $r=80$
P148 $r=80$



(b) P148 $r=80$



(c) P205 $r=80$

Figure 5.5: NSGA-II and RSA Pareto fronts at $r = 80$.

Table 5.8: Minimum objective values achieved by both NSGA-II and RSA.

Problem	r	Algorithm	Min CT	Min TE	Min TC
P205	80	NSGA-II	2754	56481	197725
P205	80	RSA	1657	55797	221473
P205	60	NSGA-II	2748	51150	183483
P205	60	RSA	1676	51096	183509
P148	60	NSGA-II	414	5960	159650
P148	60	RSA	245	6108	186737
P148	80	NSGA-II	416	6715	185257
P148	80	RSA	259	6601	247648
P65	80	NSGA-II	587	8853	177569
P65	80	RSA	378	8825	231809
P65	60	NSGA-II	591	8857	171402
P65	60	RSA	373	8980	187393

CT: Cycle Time, TE: Total Energy, TC: Total Cost.

inherent to each algorithm. While RSA excels in optimizing operational efficiency and energy usage, making it suitable for scenarios where speed and sustainability are prioritized, NSGA-II offers distinct advantages in cost-sensitive environments where total expenditure is the critical constraint.

CHAPTER 6

RESULTS AND CONCLUSION

To analyze the behavior of the model across two operational modes—Greenfield and Brownfield—the following analyses are conducted. In the subsequent sections, the analysis is divided into two parts. The first part concerns the analysis of solver results, where small cases are solved using the Gurobi solver. The second part deals with the analysis of metaheuristics, where the largest cases are solved using the RSA algorithm. Below, a detailed analysis is provided considering technical metrics in addition to the objective values.

6.1 Evaluation metrics

Since the solver has limited ability to handle large-sized problems, four small cases—P9, P12, P16, and P24—are considered. Each case is solved using r values of 5, 10, 15, 20 for the operational modes—Greenfield and Brownfield—using the Gurobi solver. All the small cases solved considering the number of mated stations (nm) as two mated stations. The efficiency of the solutions and the runtime for

each case are analyzed in addition to the three objective values. Table 6.1 presents the evaluation metrics.

Table 6.1: Evaluation metrics

Name	Objective
Run time	Lower is better
Efficiency	Higher is better
Total cost	Lower is better
Cycle time	Lower is better
Total energy consumption	Lower is better

Efficiency is calculated based on Equation 6.1, where the average efficiency is considered for each set of solutions per run. This equation is based on [11], and it is extended to account for a two-sided layout.

$$E = \sum_{i=1}^{nt} \sum_{r=1}^{nr} \sum_{j=1}^{nm} \sum_{k=1}^2 \frac{x_{ijk} \cdot y_{rjk}}{nm \cdot CT} \quad (6.1)$$

6.2 Solver Capability

Before proceeding with the analysis, it is important to highlight the solver’s ability to tackle different problem instances across two scenarios. Table 6.2 demonstrates how problems (P9, P12, P16, and P24) were solved using the Gurobi solver in both Greenfield and Brownfield scenarios with varying numbers of robots (nr). Generally, an increase in tasks typically raises the problem’s dimensionality and computational burden on the solver. This can result in longer solving times and challenges in finding optimal solutions within set constraints. Surprisingly, all problems were successfully solved in the Brownfield scenario for all values of nr

(5, 10, 15, 20). In contrast, the Greenfield scenario posed a higher level difficulty for the solver as the number of robots increases, as seen in the case of P24, which fails to solve for $nr = 15$ and $nr = 20$. For this study, a time limit of 3,600 seconds was imposed, and thus, problems that were not solved to optimality within the time limit were considered unsolved.

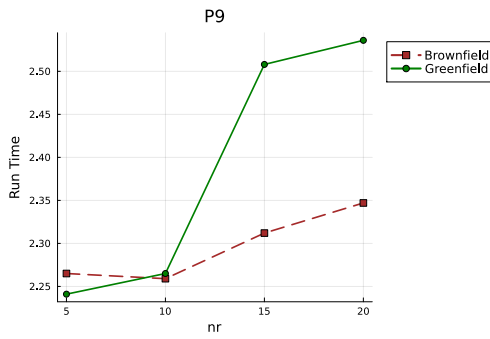
Table 6.2: Performance of Gurobi solver in solving problems across Greenfield and Brownfield scenarios

Problem	nm = 2	Greenfield				Brownfield			
	nr	5	10	15	20	5	10	15	20
P9		✓	✓	✓	✓	✓	✓	✓	✓
P12		✓	✓	✓	✓	✓	✓	✓	✓
P16		✓	✓	✓	✓	✓	✓	✓	✓
P24		✓	✓	×	×	✓	✓	✓	✓

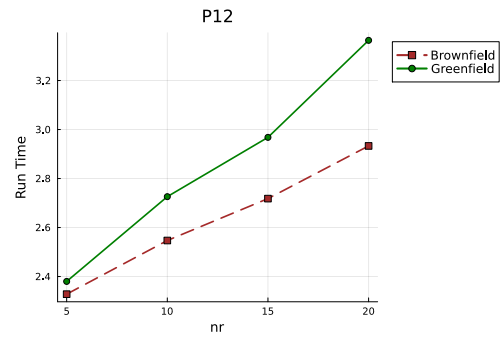
6.3 Exact Model Results

The interaction plots in Figure 6.7 for the run times of problems P9, P12, P16, and P24 illustrate the differences in solver performance between Brownfield and Greenfield scenarios as the number of robots increases. The Greenfield scenarios exhibit a noticeable increase in computation times with more robots, especially for problem P24, where the solver fails to find a solution within the time limit for $nr = 15$ and $nr = 20$. Conversely, the Brownfield computation times show a slower rate of increase or remain relatively stable across different values of nr .

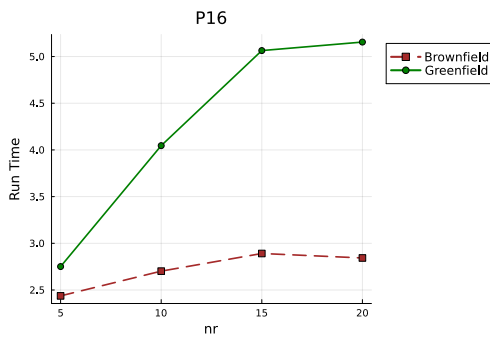
The interaction plots in Figure 6.2 for mean efficiency of problems P9, P12, P16, and P24 highlight the differences between Brownfield and Greenfield sce-



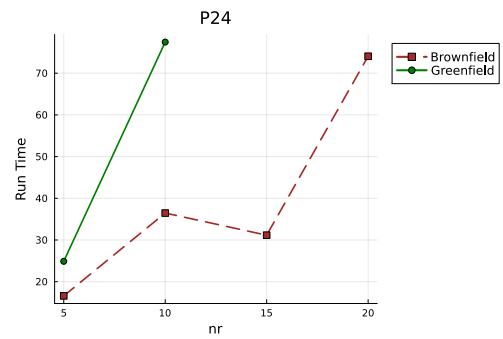
(a) Interaction Plot for P9 run Time



(b) Interaction Plot for P12 run Time



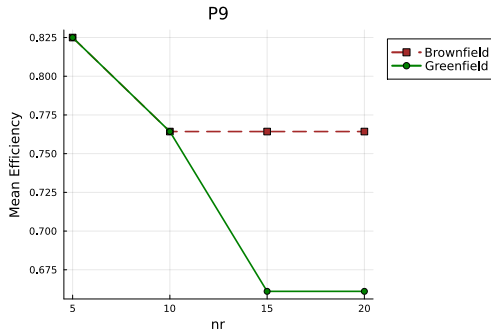
(c) Interaction Plot for P16 run Time



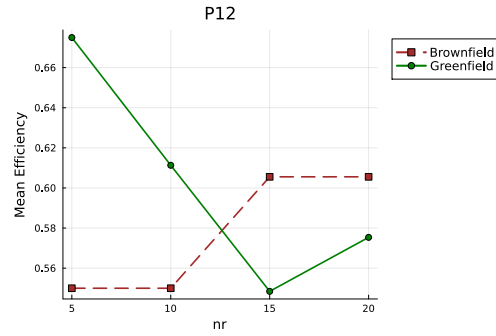
(d) Interaction Plot for P24 run Time

Figure 6.1: Interaction Plots of run times across different scenarios

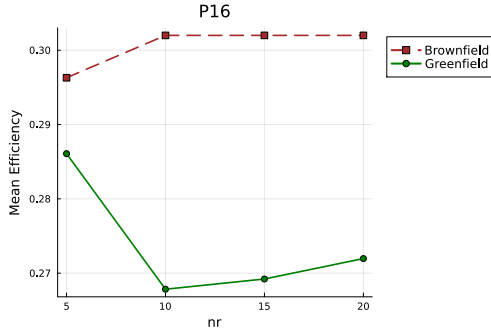
narios as the number of robots, nr , increases. In general, the Greenfield scenario shows a decrease in efficiency as the number of robots increases, while the Brownfield scenario demonstrates more stable efficiency across different nr values. Additionally, efficiency decreases as the problem size increases: for instance, P9 achieves around 80% efficiency when $nr = 5$, while P24 achieves only about 31% in the best scenario with $nr = 15$. Overall, the Brownfield scenario tends to maintain higher mean efficiency than the Greenfield scenario.



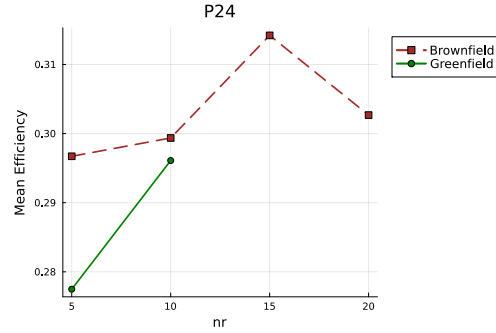
(a) Interaction Plot for P9 Mean Efficiency



(b) Interaction Plot for P12 Mean Efficiency



(c) Interaction Plot for P16 Mean Efficiency



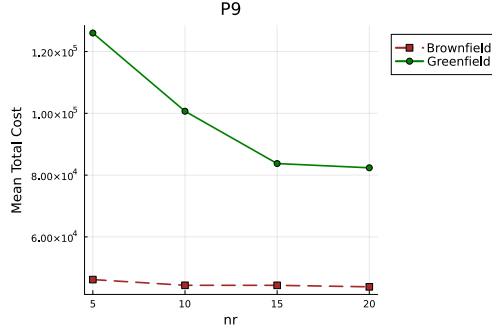
(d) Interaction Plot for P24 Mean Efficiency

Figure 6.2: Interaction Plots of mean efficiency across different scenarios

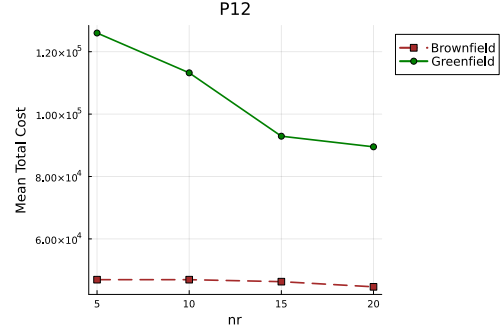
For the analysis of mean total cost, as illustrated in Figure 6.3, the Greenfield scenario consistently incurs higher total costs compared to the Brownfield scenario across all problems. Although the total cost for Greenfield shows a linear decrease

as nr increases, it remains significantly higher than that of the Brownfield scenario.

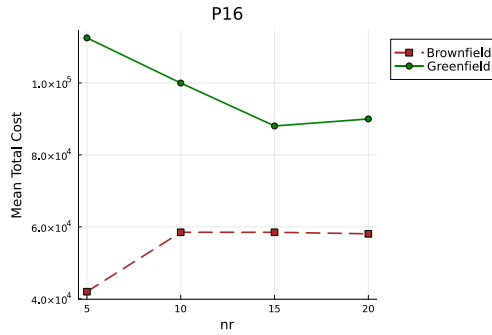
In contrast, the Brownfield scenario maintains a much lower and more stable mean total cost across all setups and problem instances.



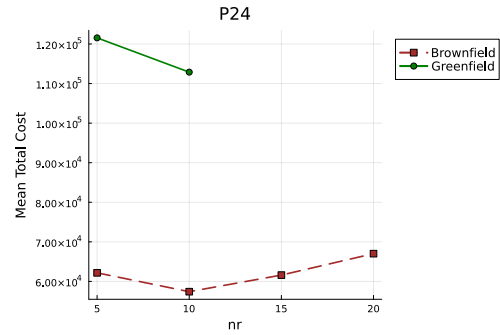
(a) Interaction Plot for P9 Mean Total Cost



(b) Interaction Plot for P12 Mean Total Cost



(c) Interaction Plot for P16 Mean Total Cost

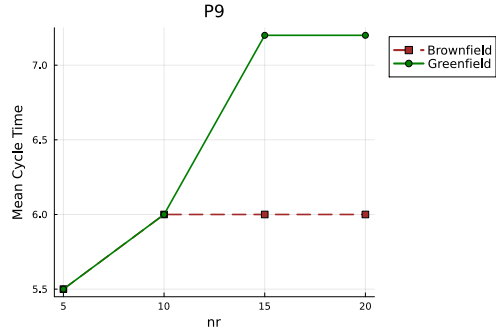


(d) Interaction Plot for P24 Mean Total Cost

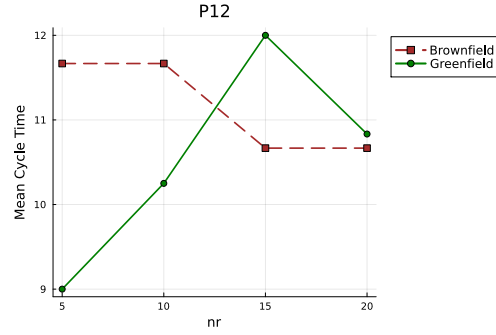
Figure 6.3: Interaction Plots of mean total cost across different scenarios

The interaction plots in Figure 6.4 for mean cycle time of problems P9, P12, P16, and P24 show notable differences between Brownfield and Greenfield scenarios as the number of robots increases. The Greenfield scenario exhibits more variability in cycle time as nr changes. For example, in P9, cycle time increases steadily with nr , while in P12 and P24, it fluctuates, initially increasing and then decreasing or vice versa. In contrast, the Brownfield scenario maintains more stable cycle times across all problem instances, showing minimal change as nr in-

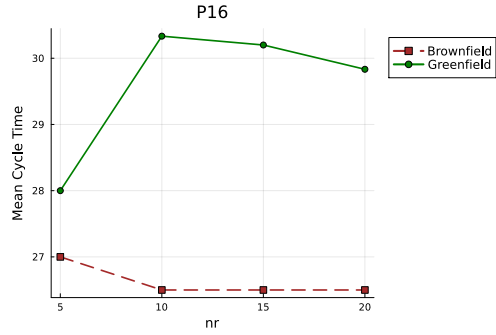
creases. These results were expected, given that the Brownfield scenario achieved better efficiency, leading to improved cycle times as well.



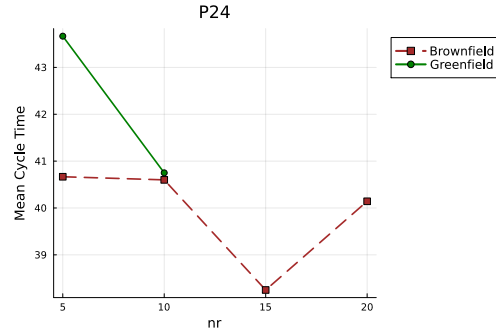
(a) Interaction Plot for P9 Mean Cycle Time



(b) Interaction Plot for P12 Mean Cycle Time



(c) Interaction Plot for P16 Mean Cycle Time

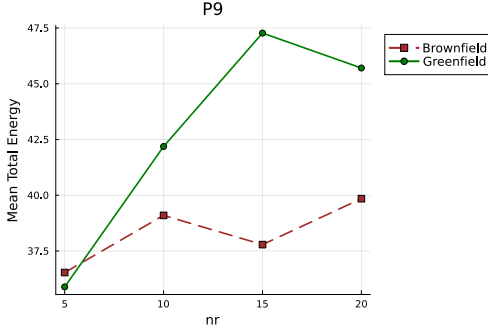


(d) Interaction Plot for P24 Mean Cycle Time

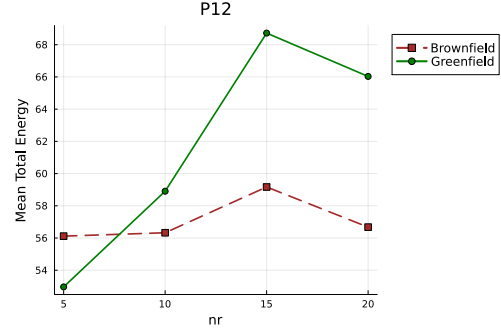
Figure 6.4: Interaction Plots of mean cycle time across different scenarios

The interaction plots in Figure 6.5 for mean total energy consumption of problems P9, P12, P16, and P24 highlight the differences between Brownfield and Greenfield scenarios as the number of robots increases. The Greenfield scenario generally exhibits higher energy consumption, with values increasing steadily as nr increases, particularly in P9 and P12. In contrast, the Brownfield scenario maintains more stable and lower energy consumption across different problem instances and nr values. These results were also expected, given that energy consumption is closely linked to cycle time; with the Brownfield scenario achieving

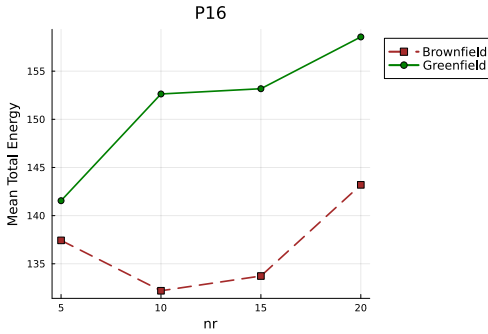
better cycle times, it naturally results in lower energy consumption. However, the random behavior observed in the results can be attributed to the nature of the energy consumption data, which is inherently random.



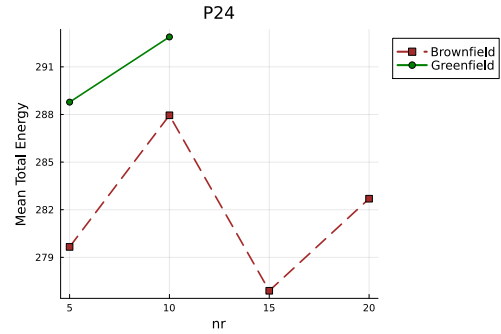
(a) Interaction Plot for P9 Mean Total Energy Consumption



(b) Interaction Plot for P12 Mean Total Energy Consumption



(c) Interaction Plot for P16 Mean Total Energy Consumption

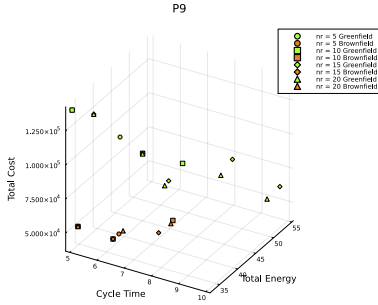


(d) Interaction Plot for P24 Mean Total Energy Consumption

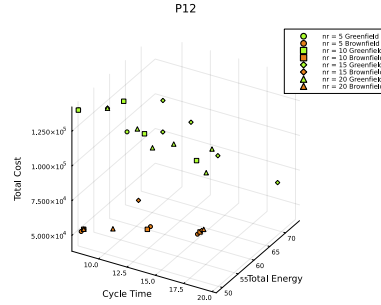
Figure 6.5: Interaction Plots of mean total energy consumption across different scenarios

Figure 6.6 presents 3D plots that display the solutions for all problems across different scenarios (Brownfield vs. Greenfield) and varying numbers of robots ($nr = 5, 10, 15, 20$), offering a comprehensive view of the interactions between cycle time, total energy, and total cost. The plots reveal that as the cycle time decreases, the total cost tends to increase, indicating that configurations achieving lower cycle times often involve higher costs. This suggests that more efficient robot

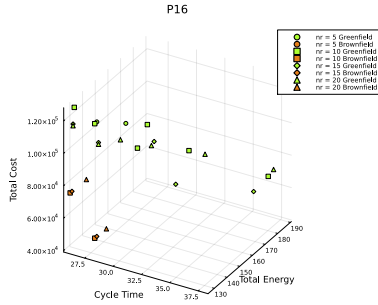
configurations (in terms of cycle time) are associated with higher operational expenses, reflecting a trade-off between efficiency and cost.



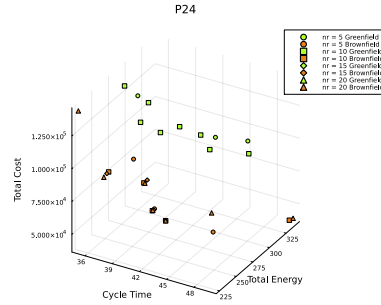
(a) 3D Plot for P9 - Greenfield vs Brownfield



(b) 3D Plot for P12 - Greenfield vs Brownfield



(c) 3D Plot for P16 - Greenfield vs Brownfield



(d) 3D Plot for P24 - Greenfield vs Brownfield

Figure 6.6: 3D Plots showing the interaction between cycle time, total energy, and total cost for different problem setups (P9, P12, P16, P24) in Greenfield and Brownfield scenarios

6.3.1 Results Discussion

The results demonstrate that the Brownfield scenario consistently outperforms its Greenfield counterpart across several dimensions—run time, efficiency, stability, and cost. Run times in the Brownfield scenario are substantially lower than in the Greenfield scenario across all values of nr . This supports the observation that Brownfield benefits from pre-existing solutions and conditions, making optimiza-

tion more straightforward and less computationally intensive. Brownfield’s run time increases more gradually, showing that its computational demands scale efficiently with increasing complexity. In contrast, Greenfield experiences noticeable increases in run time, particularly in setups like P24, suggesting that the solver struggles with greater complexity and a larger search space, especially without pre-existing infrastructure to guide the optimization process.

In terms of efficiency and stability, Brownfield again proves to be superior. It consistently demonstrates stable performance with only minor fluctuations, while Greenfield shows signs of instability and solver limitations, particularly as the complexity increases (i.e., as nr increases). The results also highlight the superiority of Brownfield in terms of total cost, as it consistently delivers lower and more stable costs compared to Greenfield. Greenfield’s cost instability and higher overall costs suggest that it is less effective in maintaining efficiency as problem complexity increases. The lower total cost in the Brownfield case can be attributed to leveraging the savings obtained from selling unused robots, which contributes significantly to overall cost efficiency. Moreover, Brownfield proves to be more efficient and stable in terms of cycle time, facilitating the solver’s ability to find more energy-efficient solutions as well. This reinforces Brownfield’s status as the more reliable and economical option in all problem setups.

Given that the only difference between problem P24 compared to P9, P12 and P16 is the number of tasks, while the mathematical formulation remains consistent across all problems, the results suggest that the increased number of tasks in problem P24 introduces specific challenges in the Greenfield scenario not

present in the Brownfield scenario. However, the Brownfield scenario includes an initial solution, accounting for existing resources and involves decisions such as selling unused robots for savings, reusing robots (thus incurring only operational costs), or investing in new robots. Meanwhile, the Greenfield scenario lacks initial solutions and any existing infrastructure, focusing solely on investments in new robots and operational costs.

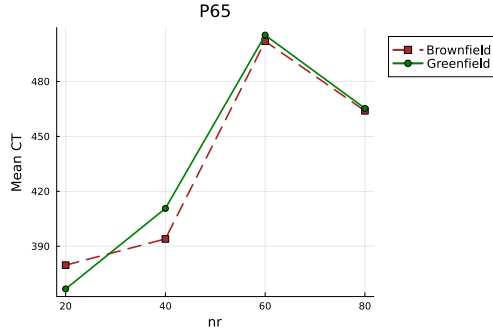
Although the Brownfield scenario appears more complex due to its inclusion of multiple decision layers (reuse, sell, buy new), it provides the solver with initial solutions and potential pathways that might simplify the search space. The solver has more concrete starting points, guiding it toward feasible solutions more quickly. In contrast, the Greenfield scenario’s lack of initial solutions means the solver starts from a broader search space without pre-existing paths to guide the optimization process. This could lead to increased computational difficulty, as the solver needs to explore a larger set of potential configurations from scratch, raising the likelihood of encountering challenging solution landscapes or convergence issues. Furthermore, the ability to factor in savings from selling unused robots or considering only the operational costs for reused robots in the Brownfield scenario may significantly alter the cost landscape, making it more favorable and potentially easier to find cost-effective solutions compared to the Greenfield scenario, where only new investments are considered.

6.4 RSA results

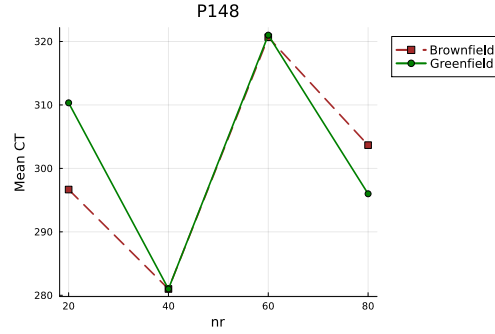
In this section, each of the three largest cases, P65, P148, and P205, is replicated three times to ensure robustness, with the average results calculated subsequently. Each run generates multiple solutions on the Pareto front; hence the average of each objective is computed for all solutions per run. The analysis includes technical aspects such as the average number of robots used and efficiency. Additionally, the objective values analyzed include total cost, cycle time, and energy consumption.

Starting with the interaction plot for average cycle time in P65 shown in Figure 6.7, both modes exhibit an increasing trend in cycle times as the number of robots increases from 20 to 60, peaking at 60, and then slightly decreasing at 80. In the interaction plot for P148, both modes show cycle times decreasing from 20 to 40, bottoming out at 40, and then increasing at 60 before leveling off slightly. The cycle times are very similar for Brownfield and Greenfield, but Greenfield shows marginally better performance at most points, except at 40 robots. Lastly, in the interaction plot for P205, cycle times increase sharply from 20 to 40, peak sharply at 40, and then decrease dramatically at 60, followed by a slight increase at 80.

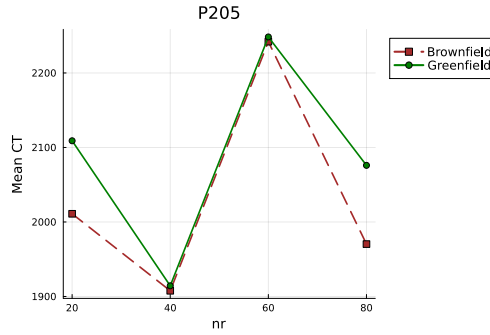
It is important to note that both Greenfield and Brownfield modes exhibit similar patterns in cycle time values. This similarity likely arises because the RSA algorithm processes cycle time calculations similarly across both scenarios, focusing primarily on the operational impact of the number of robots rather than the setup's nature. Notably, the investment cost data considered was identical across



(a) Interaction Plot for P65



(b) Interaction Plot for P148



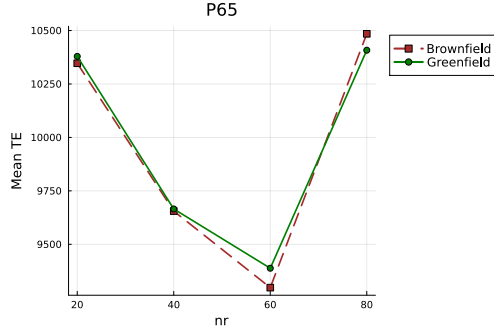
(c) Interaction Plot for P205

Figure 6.7: Comparative Analysis of Average Cycle Time (CT) for Large Cases Between Greenfield and Brownfield Modes Across Various Robot Alternatives ($nr = 20$ to 80).

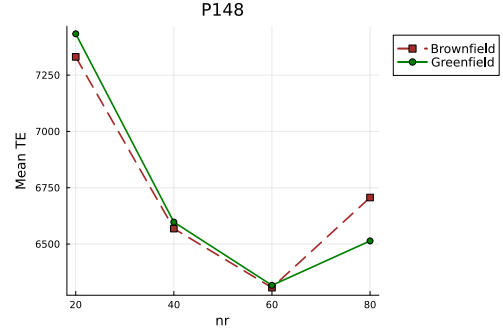
all scenarios, indicating that the observed differences in cycle time efficiency are not due to varying cost structures but are likely attributed to how robots are integrated within each setup—new or existing. However, the analysis indicates that simply increasing the number of robots does not necessarily enhance efficiency. In fact, beyond a certain point, adding more robots may lead to inefficiencies due to increased complexity in the problem space.

The interaction plots for total energy (TE) in Figure 6.8 show how the mean energy consumption varies across different numbers of robot alternatives (nr) for both Brownfield and Greenfield modes. First for P65, both modes exhibit a sharp decrease in mean energy consumption as the number of robots increases from

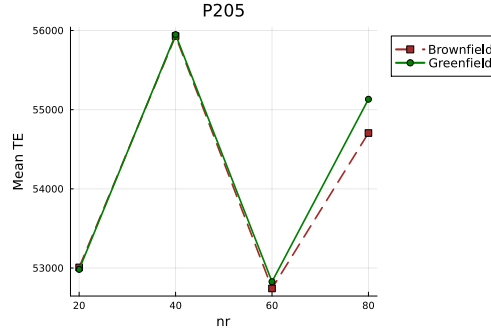
20 to 60, reaching a minimum at 60, and then a sharp increase at 80. This suggests that an intermediate number of robots, particularly around 60, might be optimal for energy efficiency. Furthermore, both modes follow a similar pattern, but Brownfield generally has slightly higher energy consumption across all points.



(a) Interaction Plot for P65



(b) Interaction Plot for P148



(c) Interaction Plot for P205

Figure 6.8: Comparative Analysis of Total Energy (TE) for Large Cases Between Greenfield and Brownfield Modes Across Various Robot Alternatives ($nr = 20$ to 80).

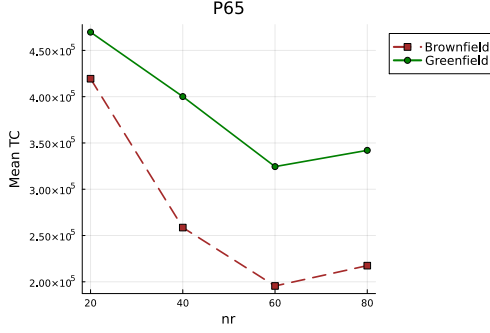
Regarding the interaction plot for P148, the data show a consistent decrease in mean energy consumption from $nr = 20$ to 60 robots, reaching its lowest point at 60, followed by a slight increase at 80. This pattern indicates that increasing the number of robots up to 60 could enhance energy efficiency. For the last case, P205, the interaction plot presents an increase in energy consumption from 20 to 40 robots, peaking sharply at 40, then decreasing significantly at 60, with a

further increase at 80. The peak at 40 suggests an inefficiency at this robot count, potentially due to operational or coordination issues. Across all scenarios, it is observed that both modes exhibit similar behavior, which is expected. Additionally, the plots indicate non-linear relationships between the number of robots and energy consumption.

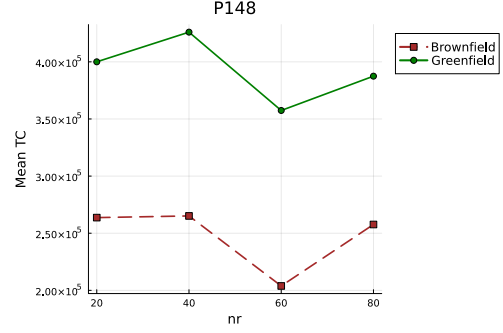
Figure 6.9 shows the total average costs between different operational modes and robot configurations. First, the Interaction Plot for P65 presents the average total cost for P65. In this plot, Greenfield mode shows a steep decline in total cost from $nr=20$ to 80 robots, indicating significant cost reductions as more robots are introduced. Despite this reduction, Brownfield starts and remains at a lower cost across all data points. Second, the Interaction Plot for P148 shows that in Greenfield mode, costs decrease from 20 to 60 robots, followed by stabilization between 60 and 80.

In contrast, Brownfield mode's costs slightly decrease from $nr=20$ to 40 and then remain relatively flat, consistently staying below Greenfield's costs. Third, the Interaction Plot for P205 reveals that Greenfield mode exhibits a sharp decrease in cost from 20 to 60, with the lowest cost at 60, followed by a minor increase at 80. Brownfield, starting higher at 20, quickly becomes less costly by 40 and maintains a lower cost at 60 and 80 compared to Greenfield.

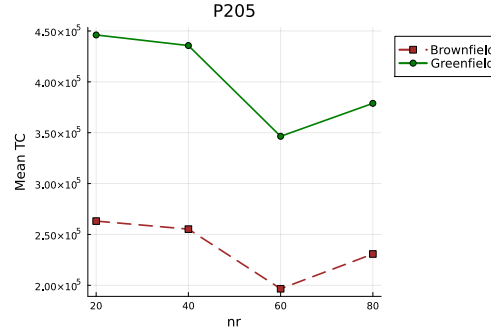
In all cases, as the number of robot alternatives increases, total cost tends to decrease, indicating that a greater number of alternatives enhances the probability of finding lower-cost options. More importantly, across all scenarios and numbers of alternatives, Brownfield maintains a cost advantage over Greenfield.



(a) Interaction Plot for P65



(b) Interaction Plot for P148



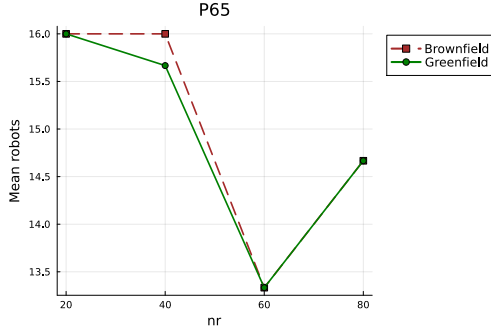
(c) Interaction Plot for P205

Figure 6.9: Comparative Analysis of Total Cost (TC) for Large Cases Between Greenfield and Brownfield Modes Across Various Robot Alternatives ($nr = 20$ to 80).

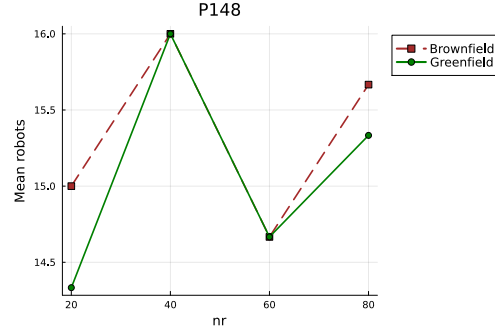
This suggests that modifying existing systems is more cost-effective than establishing new setups, primarily due to the utilization of existing infrastructure and the cost savings that can be derived from selling available, unused robots.

Figure 6.10 shows the Interaction Plots for the average number of robots used. In the Interaction Plot for P65, both modes start with a high number of robots at $nr=20$, then show a significant decrease at $nr=40$, reaching the lowest point at $nr=60$. Then, they both increase again at $nr=80$. However, Brownfield mode consistently uses slightly more robots than Greenfield across all values of nr . In the Interaction Plot for P148, both modes increase in the number of robots from $nr=20$ to $nr=40$, but then significantly decrease at $nr=60$ and continue to rise at

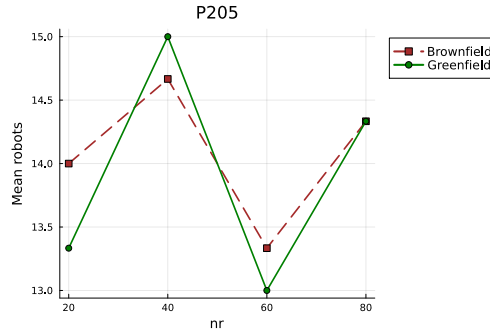
$nr=80$. The pattern is similar to P65, but the fluctuations are more pronounced, especially in the Greenfield mode which shows a sharper increase at $nr=40$.



(a) Interaction Plot for P65



(b) Interaction Plot for P148



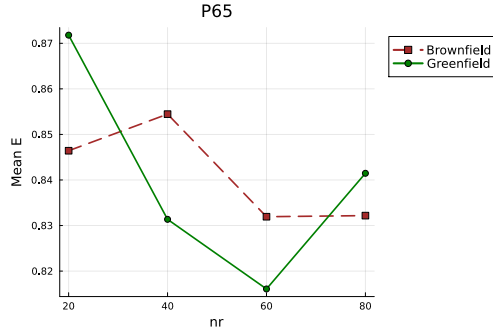
(c) Interaction Plot for P205

Figure 6.10: Comparative Analysis of Average Robot Numbers for Large Cases Between Greenfield and Brownfield Modes Across Various Alternatives ($nr = 20$ to 80).

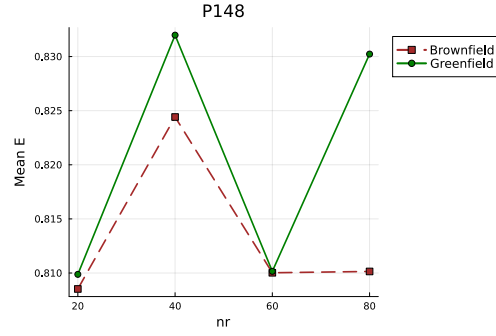
Lastly, in the Interaction Plot for P205, the number of robots increases sharply from $nr = 20$ to $nr=40$ for both modes, then decreases just as sharply at $nr=60$ and then increases again at $nr=80$. The Greenfield mode uses fewer robots at $nr=20$ and $nr=60$ compared to Brownfield. The consistent difference in robot usage between modes may indicate that Brownfield, which utilizes existing infrastructure, requires more robots to maintain or enhance productivity compared to Greenfield, which could be optimized with fewer robots. The trend across all plots indicates robot usage dips at $nr=60$ and then rises as nr increases.

Figure 6.11 depicts the efficiency of Brownfield and Greenfield modes. First, in the Interaction Plot for P65, both modes start at higher efficiency at $nr=20$, and then Greenfield experiences a notable decrease at $nr=40$. Subsequently, the efficiency continues to decrease slightly at $nr=60$ for both modes, before increasing at $nr=80$ for Greenfield, whereas Brownfield presents constant efficiency. It is important to note that Brownfield maintains higher efficiency compared to Greenfield across mid-range alternative values. Second, in the Interaction Plot for P148, there is a sharp increase in efficiency for both modes from $nr=20$ to $nr=40$, then it decreases again at $nr=60$. For Brownfield, the efficiency remains constant from $nr=60$ to $nr=80$ which is similar to P65, while for Greenfield, there is a steep rise in efficiency at $nr=80$. Overall, Greenfield has provided higher efficiency than Brownfield among the possible values of robots' alternatives.

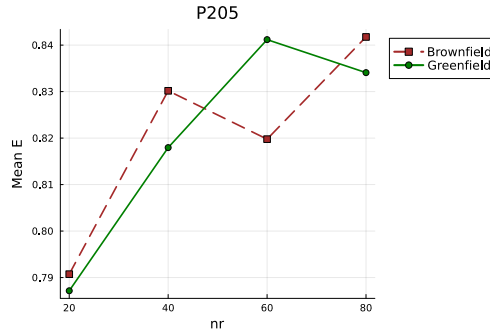
Lastly, in the Interaction Plot for P205, the efficiency dramatically improves from $nr=20$ to $nr=40$ for both modes. Then, it still increases for Greenfield and decreases for Brownfield at $nr=60$. After that, Brownfield shows better efficiency again at $nr=80$. In three out of four scenarios, Brownfield has provided higher efficiency in general. Therefore, Brownfield shows a higher starting point and maintains relatively higher efficiency throughout alternatives levels. The observed trends indicate that efficiency tends to fluctuate significantly with changes in the number of robots. Optimal robot counts for maximum efficiency differ among the scenarios, with mid-range counts often showing dips or peaks.



(a) Interaction Plot for P65



(b) Interaction Plot for P148



(c) Interaction Plot for P205

Figure 6.11: Comparative Analysis of Efficiency (E) for Large Cases Between Greenfield and Brownfield Modes Across Various Robot Alternatives ($r = 20$ to 80).

6.4.1 Results Discussion

For the cycle time values and energy consumption, there is no direct correlation between the number of robotic alternatives and their values. This is expected, as their performance depends on the selected robots, where the selection process is mainly derived from the robots' capabilities. Regarding total cost, in all cases among all possible scenarios, the total cost of the Brownfield mode is always less than that of the Greenfield mode, which is an important finding. In the Brownfield mode, using available resources reduces the total investment cost. Additionally, leveraging the savings from selling unused equipment explains this

significant reduction.

For the average number of robots, Greenfield mode typically uses a slightly lower number of robots compared to Brownfield mode. However, the number of robots fluctuates among different scenarios in all three cases, indicating there is no direct correlation between r values and the number of robots used. Similarly, efficiency varies with each case for different values of r , suggesting that there is no clear correlation between these variables. The lack of a clear correlation between the variables in the results is expected for several reasons.

Firstly, the algorithm applied to both modes utilizes the same procedural framework, except for cost calculations, ensuring uniformity across the scenarios. Secondly, variations in behavior across different problems, influenced by the number of tasks, the types of tasks, and the capabilities of the robots involved, contribute to the complexity. As the size and scope of the problem increases, so does the complexity of analyzing behavior, especially given that the model aims to optimize three objectives simultaneously which introduces inherent behavior variability.

To further analyze the results, a non-parametric Wilcoxon matched-pairs signed rank test is utilized to verify if there is a significant difference between the Brownfield and Greenfield modes. All the largest cases yield similar conclusions. Table 6.3 shows the results for the P65 case.

The table presents the sample size, median values, and p-values for each category. The test results show that Brownfield and Greenfield modes do not differ significantly in cycle time, total energy, the number of robots used, or efficiency.

Table 6.3: Test results of P65

Sample	N	Median	P-value
Brownfield CT	12	430	0.885
Greenfield CT	12	417	
Brownfield TE	12	9954	0.885
Greenfield TE	12	9944	
Brownfield TC	12	242900	0.010
Greenfield TC	12	382067	
Brownfield robots no.	12	15.5	0.931
Greenfield robots no.	12	15.5	
Brownfield E	12	0.839050	1.00
Greenfield E	12	0.842318	

However, a critical difference lies in the total costs, where Brownfield mode is significantly more cost-effective than Greenfield. Notably, the significant p-value of 0.010 for total costs (TC) between Brownfield and Greenfield modes indicates a statistically significant difference. This finding could influence decision-making regarding the selection of Brownfield versus Greenfield for specific projects, especially when cost is a major consideration.

6.5 RSA vs solver

To further explore the performance of the solver in comparison to the RSA, Table 21 presents random trial runs for problems P12 and P16, under Brownfield mode settings with two mated stations and twenty robot alternatives. In P12, both methods demonstrate their efficiency in achieving optimal cycle times. The solver, as recorded in reference [34], achieves an optimal cycle time of 6 when considering cycle time as the sole objective. This result is mirrored closely by the RSA, indicating that both approaches can effectively minimize cycle time while

addressing multiple objectives. Although the RSA exactly matches the optimal cycle time, the solver achieves a slightly higher cycle time but with a reduced total cost, demonstrating a trade-off between cycle time and cost efficiency.

Table 6.4: Random trial RSA runs for problems P12 and P16, under Brownfield mode settings with two mated stations given $nr = 20$.

Problem	Solver			RSA		
	CT	TC	TE	CT	TC	TE
P12	9	43200	58	6	75241	44
	10	41683	63	10	61138	44
	12	40538	66	7	61138	45
P16	26	120681	140	23	91830	171
	27	110255	162	24	91830	172
	28	94181	138	38	78423	186
	30	77155	155			

6.6 Conclusion

This research is driven by three primary objectives. The first objective is to extend existing reconfiguration models with task types from single-sided to two-sided assembly lines. This expansion addresses the unique requirements and challenges of two-sided configurations within the context of flexible production systems. The second objective focuses on multi-objective optimization within the Reconfiguration Model. It aims to integrate additional key performance indicators, specifically cycle time and energy consumption, alongside the existing objective of total reconfiguration cost. This integration aligns the production process with broader sustainability and efficiency goals. The third objective is to solve the newly de-

veloped reconfiguration model for two-sided assembly lines using both exact and meta-heuristic methods. This approach aims to demonstrate the model’s applicability and effectiveness in optimizing re-balancing processes across different solution methodologies.

The mathematical formulation of the TRALBP-TT is developed to include additional constraints that account for a two-sided production layout, while also integrating aspects of robots’ capabilities. Moreover, the integration of task types has enabled a more realistic simulation of real-world robots’ capabilities. This enhancement enforces the allocation of tasks of the same production type at specific workstations based on the robots assigned. By grouping the same types of tasks at particular workstations, robots can specialize and adapt their movements and operations more efficiently. This is especially advantageous in a two-sided line, where space and timing are critical constraints. Furthermore, the inclusion of energy consumption, alongside total reconfiguration cost and cycle time in the model, has contributed to reducing waste and enhancing sustainable production practices.

In this research, a composite dataset has been utilized that integrates benchmark problems for two-sided models. To adapt to these two-sided models, new task types and capabilities specifically tailored for these problems have been generated. Additionally, supplemental data on energy usage and costs, derived from prior studies, have been incorporated. The resulting dataset is publicly available for future investigations in the field.

Based on the benchmark evaluation results, the solver demonstrates higher

efficiency in Brownfield scenarios, where it benefits from existing infrastructure and initial conditions. Although the Brownfield scenario appears more complex due to its multiple decision layers (reuse, sell, buy new), it provides the solver with initial solutions and potential pathways that might simplify the search space. This efficiency is evident in faster solve times, lower total costs, and better management of cycle times and energy. In contrast, in Greenfield scenarios, where no existing infrastructure is leveraged, the solver faces greater challenges, particularly as the number of robots and the complexity of operations increase. This is evident from increased solve times and difficulties in maintaining cost and energy efficiency.

On the other hand, to manage large-sized cases, two meta-heuristics are implemented which are RSA, and NSGA-II. Consequently, new encoding and decoding procedures have been developed to handle the extended model, which includes task types and incorporates robots' capabilities. Meta-heuristics have demonstrated highly efficient performance in managing large-sized problems. However, based on evaluation metrics, the RSA has provided a more diverse set of high-quality solutions compared to NSGA-II. Furthermore, RSA achieved shorter cycle times not only in small-sized cases but also in solving large cases with a considerable number of robot alternatives. The advantage is evident in how the RSA provides consistent performance regardless of the number of robots or tasks. The results from RSA indicate that there are no significant differences in cycle time, total energy, the number of robots used, or efficiency between Brownfield and Greenfield modes. However, a critical difference lies in the total costs, where the Brownfield mode is significantly more cost-effective than Greenfield, which aligns

with the solver results.

To summarize, across all considered scenarios, the total cost in the Brownfield mode is consistently lower than that in the Greenfield mode, which is a significant finding. In the Brownfield mode, the use of available resources helps reduce total investment costs. Additionally, the financial benefits derived from selling unused robots contribute to this substantial cost reduction. However, the scalability of the solver is tested, revealing diminishing returns in efficiency and cost as more robots are added. This indicates limitations in the solver’s ability to optimize at larger scales, particularly in Greenfield scenarios.

For future applications or problem setups in Greenfield scenarios, strategies may need to be developed to better guide the solver. This could involve providing heuristic starts or simplified models in the initial stages to effectively manage the increased computational load. In contrast, the Brownfield approach generally appears more robust and scalable under the current settings and solver capabilities. The model presented in this study addresses only a single product in a two-sided layout. Future research could explore mixed-model TRALBP-TT scenarios considering additional task types. Moreover, human-robot interaction could be examined within the context of the Brownfield model. In terms of sustainability, future research could further incorporate environmental aspects by including energy consumption costs as part of the processing costs. Additionally, evaluating the current model in different assembly line configurations, such as U-shaped or parallel line layouts, could be explored.

APPENDIX A

SUPPLEMENTARY

MATERIAL

The following tables and lists represent a sample of the data used in this study, specifically the task dependencies, direction assignments, capability matrix and task types for a subset of the problem instances.

Table A.1: P9: Task Dependencies: Predecessors and Successors

Task ID	Full Predecessors	Full Successors
1	None	4, 7
2	None	5, 6, 7, 9, 8
3	None	6, 9
4	1	7
5	2	7, 8
6	2, 3	9
7	5, 4, 2, 1	None
8	5, 2	None
9	6, 2, 3	None

Table A.2: P9: Task Types Assignment

Task ID	Task Type
1	Type 1
2	Type 3
3	Type 2
4	Type 3
5	Type 3
6	Type 3
7	Type 2
8	Type 2
9	Type 2

Table A.3: P9: Robot Capabilities for Tasks 1 to 9

Robot	Capabilities (Tasks 1 to 9)
Robot 1	[1, 1, 1, 1, 1, 1, 1, 1, 1]
Robot 2	[1, 1, 1, 1, 1, 1, 1, 1, 1]
Robot 3	[1, 1, 0, 1, 1, 1, 0, 0, 0]
Robot 4	[1, 1, 1, 1, 1, 1, 1, 1, 1]
Robot 5	[1, 1, 1, 1, 1, 1, 1, 1, 1]

Table A.4: P9: Robot Processing Times for Tasks 1 to 9

Robot	Processing Times (Tasks 1 to 9)
Robot 1	[2.0, 3.0, 2.0, 3.0, 1.0, 1.0, 2.0, 2.0, 1.0]
Robot 2	[2.0, 3.0, 2.0, 3.0, 1.0, 1.0, 2.0, 2.0, 1.0]
Robot 3	[3.0, 4.0, 2000.0, 4.0, 2.0, 2.0, 2000.0, 2000.0, 2000.0]
Robot 4	[2.0, 3.0, 2.0, 3.0, 1.0, 1.0, 2.0, 2.0, 1.0]
Robot 5	[2.0, 3.0, 2.0, 3.0, 1.0, 1.0, 2.0, 2.0, 1.0]

Table A.5: Robot Costs

Cost Type	Values
Investment Costs (EC)	[18283, 20567, 8028, 22319, 28019]
Processing Costs (PC)	[7200, 21600, 7200, 7200, 14400]
Saving Costs (SC)	[828, 2056, 802, 2231, 2801]

Table A.6: Energy Consumption Per Robot (PCr)

Robot	Energy Consumption
Robot 1	2.4
Robot 2	1.6
Robot 3	1.5
Robot 4	2.4
Robot 5	1.5

Table A.7: P24: Robot Processing Times for Tasks 1 to 24

Robot	Processing Times (Tasks 1 to 24)
Robot 1	[3.0, 7.0, 7.0, 5.0, 4.0, 3.0, 4.0, 3.0, 6.0, 4.0, 4.0, 3.0, 3.0, 9.0, 5.0, 9.0, 2.0, 7.0, 9.0, 9.0, 8.0, 8.0, 9.0, 9.0]
Robot 2	[3.0, 7.0, 7.0, 5.0, 4.0, 3.0, 4.0, 3.0, 6.0, 4.0, 4.0, 3.0, 3.0, 9.0, 5.0, 9.0, 2.0, 7.0, 9.0, 9.0, 8.0, 8.0, 9.0, 9.0]
Robot 3	[4.0, 9.0, 9.0, 6.0, 5.0, 4.0, 5.0, 4.0, 7.0, 5.0, 5.0, 4.0, 4.0, 11.0, 6.0, 11.0, 3.0, 9.0, 11.0, 11.0, 10.0, 10.0, 11.0, 11.0]
Robot 4	[3.0, 7.0, 7.0, 5.0, 4.0, 3.0, 4.0, 3.0, 6.0, 4.0, 4.0, 3.0, 3.0, 9.0, 5.0, 9.0, 2.0, 7.0, 9.0, 9.0, 8.0, 8.0, 9.0, 9.0]
Robot 5	[3.0, 6.0, 6.0, 5.0, 4.0, 3.0, 4.0, 3.0, 5.0, 4.0, 4.0, 3.0, 3.0, 8.0, 5.0, 8.0, 2.0, 6.0, 8.0, 8.0, 7.0, 7.0, 8.0, 8.0]

Table A.8: P24: Capability Matrix for Robots (Tasks 1 to 24)

Robot	Capabilities (Tasks 1 to 24)
Robot 1	[0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Robot 2	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Robot 3	[0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1]
Robot 4	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Robot 5	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Appendix: Sample Code Implementation

Listing A.1: Sample Julia Code for Precedence Graph and Task Direction Assignments

```
1  # Define immediate predecessors based on precedence graph
2  P_i = Dict(
3      1 => Int[],
4      2 => Int[],
5      3 => Int[],
6      4 => Int[],
7      5 => [2],
8      6 => [2, 3],
9      7 => [3],
10     8 => [5],
11     9 => [6],
12     10 => [7],
13     ...
14     24 => [20]
15 )
16
17 # Initialize an empty dictionary for immediate successors
18 S_i = Dict{Int, Int[]} for i in 1:nt
19
20 # Populate the S_i dictionary using the P_i dictionary
21 for (task, predecessors) in P_i
```

```

22     for predecessor in predecessors
23         push!(S_i[predecessor], task)
24     end
25 end
26
27 # Initialize empty dictionaries for all predecessors and successors
28 All_P_i = Dict{<i>i</i>, Set{Int}}() for i in 1:nt)
29 All_S_i = Dict{<i>i</i>, Set{Int}}() for i in 1:nt)
30
31 # Define a recursive function to find all predecessors for a given
    task
32 function find_all_predecessors(task, P_i, All_P_i)
33     for pred in P_i[task]
34         push!(All_P_i[task], pred)
35         find_all_predecessors(pred, P_i, All_P_i)
36         union!(All_P_i[task], All_P_i[pred])
37     end
38 end
39
40 # Recursive function to find all successors
41 function find_all_successors(task, S_i, All_S_i)
42     for succ in S_i[task]
43         push!(All_S_i[task], succ)
44         find_all_successors(succ, S_i, All_S_i)
45         union!(All_S_i[task], All_S_i[succ])
46     end
47 end

```

Listing A.2: Model Constraints Implementation

```

1  # Constraint to ensure each task is assigned to only one side of a
   workstation within the mated-station (8)

2  for i in I
3      @constraint(model, sum(X[i, j, k] for j in J, k in K_i[i]) == 1)
4  end

5
6  for i in setdiff(I, P0) # Exclude tasks with no immediate
   predecessors
7      for h in P_i[i] # Immediate predecessors of task i
8          @constraint(model, sum(g * X[h, g, k] for g in J, k in K_i[h
          ]) <=
9
10                                     sum(j * X[i, j, k] for j in J, k in K_i[
11                                     i]))
12
13     end
14 end

15
16 # Add finishing time constraints (10)
17
18 for i in I
19     @constraint(model, tf[i] <= CT)
20 end

21
22 # When task h is the immediate predecessor of task i at the same
   mated-station (11)
23
24 for i in I
25     for h in P_i[i], j in J
26         @constraint(model, tf[i] - tf[h] +

```

```

22         * (1 - sum(X[i, j, k] for k in K_i[i])) +
23         * (1 - sum(X[h, j, k] for k in K_i[h])) >=
24         sum(tir[(r, h)] * y[r, j, k] for r in 1:nr, k in
25             K_i[i]))
26     end
27
28 # Checks that only one robot is assigned to each station (15)
29 for j in J
30     for k in 1:2 # Assuming there are two sides, 1 and 2
31         @constraint(model, sum(y[r, j, k] for r in 1:nr) == 1)
32     end
33 end
34
35 # To test the original model: enforces that each robot can be
36     assigned to only one station
37
38 for r in 1:nr
39     @constraint(model, sum(y[r, j, k] for j=1:nm, k=1:2) <= 1)
40 end
41
42 # Checks that the finishing time of task i is greater than or equal
43     to its operational time (14)
44
45 for i in I
46     for j in J
47         for k in K_i[i]
48             @constraint(model, tf[i] + * (1 - X[i, j, k]) >=
49                 sum(tir[r, i] * y[r, j, k] for r in

```

```

1:nr))
46     end
47 end
48 end

```

Listing A.3: Definition of binary variables, and their associated constraints for linearizing the product of binary variables.

```

1  # Define the binary variable zip with two indices i and p
2  @variable(model, zip[i in I, p in setdiff(I, union(All_P_i[i],
3      All_S_i[i], C_i[i]))], Bin)
4
4  # Define the binary variable z_ijkp for the linearized multiplication
5  terms
6
6  @variable(model, z_ijkp[i in I, p in setdiff(I, union(All_P_i[i],
7      All_S_i[i], C_i[i])), j in J, k in intersect(K_i[i], K_i[p])],
8      Bin)
9
9  # Add constraints to linearize the product  $X[i, j, k] * X[p, j, k]$ 
10
10  for i in I
11      for p in setdiff(I, union(All_P_i[i], All_S_i[i], C_i[i]))
12          if i < p
13              for j in J
14                  for k in intersect(K_i[i], K_i[p])
15                      #  $z_{ijkp}[i, p, j, k] \leq X[i, j, k]$ 
16                      @constraint(model, z_ijkp[i, p, j, k] <= X[i, j,
17                          k])
18

```

```

16         # z_ijkp[i, p, j, k] <= X[p, j, k]
17         @constraint(model, z_ijkp[i, p, j, k] <= X[p, j,
18             k])
19
20         # z_ijkp[i, p, j, k] >= X[i, j, k] + X[p, j, k] -
21             1
22         @constraint(model, z_ijkp[i, p, j, k] >= X[i, j,
23             k] + X[p, j, k] - 1)
24
25     end
26
27 end
28
29 # Add the linearized constraint for zip[i, p]
30 @constraint(model, zip[i, p] == sum(z_ijkp[i, p, j, k]
31     for j in J, k in intersect(K_i[i], K_i[p])))
32
33 end
34
35 end
36
37 end

```

Listing A.4: Cost calculations including investment, savings, and processing costs, along with their respective constraints and auxiliary variables.

```

1  # Define auxiliary binary variables for investments and savings
2  @variable(model, z_inv[r=1:nr, j=1:nm, k=1:2], Bin)
3  @variable(model, z_sav[r=1:nr, j=1:nm, k=1:2], Bin)
4
5  # Constraints for investment
6  @constraint(model, inv_constraints[r=1:nr, j=1:nm, k=1:2],
7      z_inv[r, j, k] == y[r, j, k] * (1 - D_3D[r, j, k]))

```

```

8 C_inv = sum(EC[r] * z_inv[r,j,k] for r=1:nr, j=1:nm, k=1:2)
9
10 # Constraints for savings to activate z_sav_cond when D_3D is 1 and y
    is 0
11 @constraint(model, activate_sav[r=1:nr, j=1:nm, k=1:2],
12             z_sav[r,j,k] == D_3D[r,j,k] * (1 - y[r,j,k])) # z_sav_cond
    can only be 1 if D_3D is 1
13 C_sav = sum(SC[r] * z_sav[r,j,k] for r=1:nr, j=1:nm, k=1:2)
14
15 # Expression for processing costs
16 @expression(model, C_prc, sum(PC[r] * sum(y[r, j, k] for j=1:nm, k
    =1:2) for r=1:nr))

```

Listing A.5: Restarted Simulated Annealing (RSA) code for task allocation and optimization.

```

1 while true
2     # Calculate elapsed time
3     elapsed_time = Millisecond(now() - start_time).value / 1000 #
    Convert milliseconds to seconds
4     # Check if the termination criterion has been met
5     if elapsed_time >= max_cpu_time
6         println("Termination criterion met. Maximum CPU time elapsed.
    ")
7         break # Terminate the loop
8     end
9
10 while ns < NS

```

```

11      # Generate a new neighbor solution including both task
12      permutation and robot allocation
13
14      new_task_permutation = generate_neighbor(current_solution[1])
15
16      # Decode the new solution to get the cycle time and energy
17      consumption
18
19      new_cycle_time, new_energy_consumption, new_total_cost,
20      new_robot_allocation = decode_individual(
21      new_task_permutation, capability_matrix, CT)
22
23      ns += 1
24
25      # Check for CT_best update
26
27      if new_cycle_time < CT_best
28
29          CT_best = new_cycle_time
30
31          CT = CT_best - 1 # Update CT as per the iterative
32          mechanism
33
34      end
35
36      # Check if the new solution is not dominated by the solutions
37      in POS
38
39      if is_not_dominated(new_task_permutation,
40      new_robot_allocation, new_cycle_time,
41      new_energy_consumption, new_total_cost, POS)
42
43          # If non-dominated, update POS with the new solution
44
45          update_POS!(POS, new_task_permutation,
46      new_robot_allocation, new_cycle_time,
47      new_energy_consumption, new_total_cost)

```

```

28         filter_dominated_solutions!(POS)
29     else
30         d += 1  # Moves counter
31         if d > dn
32             # Perform a restart
33             calculate_crowding_distances!(POS)  # Calculate
34             crowding distances since POS changed
35             current_solution =
36                 select_solution_with_largest_distance!(POS)  #
37                 Restart with largest distance
38             new_cycle_time, new_energy_consumption,
39             new_total_cost, _ = decode_individual(
40                 current_solution[1], capability_matrix, CT)
41         else
42             # Select one objective randomly and calculate  $\Delta$ 
43             objective_to_consider = rand(1:num_objectives)
44             if objective_to_consider == 1 $\Delta$ 
45                 = new_cycle_time - current_cycle_time
46                 fs = current_cycle_time
47             elseif objective_to_consider == 2 $\Delta$ 
48                 = new_energy_consumption -
49                 current_energy_consumption
50                 fs = current_energy_consumption
51             elseif
52                 = new_total_cost - current_total_cost
53                 fs = current_total_cost
54             end

```

```

49
50         if  $\Delta \leq 0$  || exp( $\Delta / T * fs$ ) > rand()
51             current_solution = (new_task_permutation,
52                                 new_robot_allocation)
53             current_cycle_time = new_cycle_time
54             current_energy_consumption =
55                 new_energy_consumption
56             current_total_cost = new_total_cost
57         end
58     end
59     # Cooling down
60     T *= cooling_rate
61 end

```

Listing A.6: Non-dominated Sorting Genetic Algorithm II (NSGA-II) for optimization.

```

1  # Main Genetic Algorithm Function
2  function nsga_ii(nPop, crossover_rate, mutation_rate, num_tasks,
3      capability_matrix, initial_CT, , Nt, combined_population)
4      max_cpu_time = Nt * Nt * / 1000
5      start_time = now()
6      # Initialize arrays for plotting data
7      cycle_times = Float64[]
8      energy_consumptions = Float64[]
9      total_costs = Float64[]

```

```

9
10 # Step 1: Initialize population
11 population = generate_initial_population(nPop, num_tasks,
12     capability_matrix, initial_CT)
13
14 # Step 2: Sort the population using non-dominated sorting
15 fronts, first_front = fast_non_dominated_sort(population)
16 calculate_crowding_distances!(fronts, population)
17
18 # Iterate through generations
19 iteration = 1
20 while true
21     # Calculate elapsed time
22     elapsed_time = Millisecond(now() - start_time).value / 1000
23     # Convert milliseconds to seconds
24     # Check if the termination criterion has been met
25     if elapsed_time >= max_cpu_time
26         println("Termination criterion met. Maximum CPU time
27             elapsed.")
28         break # Terminate the loop
29     end
30
31     # Step 3: Selection
32     parents = perform_selection(population)
33
34     # Step 4: Crossover and mutation to generate new offspring
35     offspring = []

```

```

33     while length(offspring) < length(parents)
34         parent1, parent2 = sample(parents, 2, replace=false)
35         child = crossover(parent1, parent2, crossover_rate,
36                             initial_CT)
37         child = mutate(child, mutation_rate, initial_CT)
38         push!(offspring, child)
39
40     end
41
42     # Step 5: Sort combined population (parents + offspring)
43     combined_population = vcat(parents, offspring)
44     # Update Cycle Times for combined population
45     update_cycle_time!(combined_population)
46     fronts, first_front = fast_non_dominated_sort(
47         combined_population)
48     calculate_crowding_distances!(fronts, combined_population)
49
50     # Step 6: Environment selection
51     population = select_new_population(fronts,
52         combined_population, nPop)
53
54     iteration += 1
55
56 end
57
58 for idx in first_front
59     individual = combined_population[idx]
60     $(individual.energy_consumption), Total Cost: $(
61         individual.total_cost)

```

```
56         push!(cycle_times_rank1, individual.cycle_time)
57         push!(energy_consumptions_rank1, individual.
           energy_consumption)
58         push!(total_costs_rank1, individual.total_cost)
59     end
60 end
61 return population, cycle_times, energy_consumptions, total_costs,
           combined_population, first_front
62 end
```

REFERENCES

- [1] Y. K. Kim, W. S. Song, and J. H. Kim, “A mathematical model and a genetic algorithm for two-sided assembly line balancing,” *Computers & Operations Research*, vol. 36, no. 3, pp. 853–865, Mar. 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0305054807002304>
- [2] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel, “Reconfigurable Manufacturing Systems,” *CIRP Annals*, vol. 48, no. 2, pp. 527–540, Jan. 1999. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0007850607632326>
- [3] A. Yelles-Chaouche, E. Gurevsky, N. Brahimi, and A. Dolgui, “Reconfigurable manufacturing systems from an optimisation perspective: a focused review of literature,” *International Journal of Production Research*, vol. 59, pp. 6400–6418, Oct. 2020.
- [4] Z. Li, I. Kucukkoc, and J. M. Nilakantan, “Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem,” *Computers & Operations Research*, vol. 84, pp. 146–161,

- Aug. 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0305054817300618>
- [5] M. N. Janardhanan, Z. Li, G. Bocewicz, Z. Banaszak, and P. Nielsen, “Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times,” *Applied Mathematical Modelling*, vol. 65, pp. 256–270, Jan. 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0307904X18304050>
- [6] J. J. Bartholdi, “Balancing two-sided assembly lines: a case study,” *International Journal of Production Research*, vol. 31, no. 10, pp. 2447–2461, Oct. 1993. [Online]. Available: <https://www.tandfonline.com/doi/abs/10.1080/00207549308956868>
- [7] Y. Kim, Y. Kim, and Y. Kim, “Two-sided assembly line balancing: A genetic algorithm approach,” *Production Planning and Control*, vol. 11, no. 1, pp. 44–53, 2000. [Online]. Available: <https://doi.org/10.1080/095372800232478>
- [8] G. Levitin, J. Rubinovitz, and B. Shnits, “A genetic algorithm for robotic assembly line balancing,” *European Journal of Operational Research*, vol. 168, no. 3, pp. 811–825, Feb. 2006. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0377221704004874>
- [9] Z. Li, Q. Tang, and L. Zhang, “Minimizing energy consumption and cycle time in two-sided robotic assembly line systems using restarted simulated annealing algorithm,” *Journal of Cleaner Production*, vol. 135, pp.

- 508–522, Nov. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0959652616308083>
- [10] Z. Li, I. Kucukkoc, and Q. Tang, “A comparative study of exact methods for the simple assembly line balancing problem,” *Soft Computing*, vol. 24, no. 15, pp. 11 459–11 475, Aug. 2020. [Online]. Available: <https://doi.org/10.1007/s00500-019-04609-9>
- [11] M. Albus and M. F. Huber, “Resource reconfiguration and optimization in brownfield constrained Robotic Assembly Line Balancing Problems,” *Journal of Manufacturing Systems*, vol. 67, pp. 132–142, Apr. 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0278612523000018>
- [12] X. Hu, E. Wu, and Y. Jin, “A station-oriented enumerative algorithm for two-sided assembly line balancing,” *European Journal of Operational Research*, vol. 186, no. 1, pp. 435–440, Apr. 2008. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0377221707001427>
- [13] H. Xiaofeng, W. Erfei, B. Jinsong, and J. Ye, “A branch-and-bound algorithm to minimize the line length of a two-sided assembly line,” *European Journal of Operational Research*, vol. 206, no. 3, pp. 703–707, Nov. 2010. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0377221710001657>

- [14] E.-F. Wu, Y. Jin, J.-S. Bao, and X.-F. Hu, “A branch-and-bound algorithm for two-sided assembly line balancing,” *The International Journal of Advanced Manufacturing Technology*, vol. 39, no. 9, pp. 1009–1015, Nov. 2008. [Online]. Available: <https://doi.org/10.1007/s00170-007-1286-3>
- [15] C. P. Tomazella and M. S. Nagano, “A comprehensive review of Branch-and-Bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem,” *Expert Systems with Applications*, vol. 158, p. 113556, Nov. 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0957417420303808>
- [16] T. O. Lee, Y. Kim, and Y. K. Kim, “Two-sided assembly line balancing to maximize work relatedness and slackness,” *Computers & Industrial Engineering*, vol. 40, no. 3, pp. 273–292, Jul. 2001. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835201000298>
- [17] U. Özcan and B. Toklu, “Balancing two-sided assembly lines with sequence-dependent setup times,” *International Journal of Production Research*, vol. 48, no. 18, pp. 5363–5383, Sep. 2010. [Online]. Available: <https://doi.org/10.1080/00207540903140750>
- [18] A. Scholl and C. Becker, “State-of-the-art exact and heuristic solution procedures for simple assembly line balancing,” *European Journal of Operational Research*, vol. 168, no. 3, pp. 666–693, Feb. 2006. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0377221704004795>

- [19] J. Rubinovitz, J. Bukchin, and E. Lenz, “RALB – A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines,” *CIRP Annals*, vol. 42, no. 1, pp. 497–500, Jan. 1993. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0007850607624949>
- [20] J. Gao, L. Sun, L. Wang, and M. Gen, “An efficient approach for type II robotic assembly line balancing problems,” *Computers & Industrial Engineering*, vol. 56, no. 3, pp. 1065–1080, Apr. 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835208002222>
- [21] J. Mukund Nilakantan, G. Q. Huang, and S. G. Ponnambalam, “An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems,” *Journal of Cleaner Production*, vol. 90, pp. 311–325, Mar. 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0959652614012219>
- [22] Z. A. Çil, S. Mete, E. Özceylan, and K. Ağpak, “A beam search approach for solving type II robotic parallel assembly line balancing problem,” *Applied Soft Computing*, vol. 61, pp. 129–138, Dec. 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1568494617304829>
- [23] A. S. Michels, T. C. Lopes, C. G. S. Sikora, and L. Magatão, “The Robotic Assembly Line Design (RALD) problem: Model and case studies with practical extensions,” *Computers & Industrial Engineering*, vol. 120,

- pp. 320–333, Jun. 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835218301463>
- [24] Z. Li, M. N. Janardhanan, Q. Tang, and P. Nielsen, “Mathematical model and metaheuristics for simultaneous balancing and sequencing of a robotic mixed-model assembly line,” *Engineering Optimization*, vol. 50, no. 5, pp. 877–893, May 2018. [Online]. Available: <https://doi.org/10.1080/0305215X.2017.1351963>
- [25] Z. Li, M. N. Janardhanan, A. S. Ashour, and N. Dey, “Mathematical models and migrating birds optimization for robotic U-shaped assembly line balancing problem,” *Neural Computing and Applications*, vol. 31, no. 12, pp. 9095–9111, Dec. 2019. [Online]. Available: <https://doi.org/10.1007/s00521-018-3957-4>
- [26] B.-q. Sun, L. Wang, and Z.-p. Peng, “Bound-guided hybrid estimation of distribution algorithm for energy-efficient robotic assembly line balancing,” *Computers & Industrial Engineering*, vol. 146, p. 106604, Aug. 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835220303387>
- [27] B. Zhou and Q. Wu, “Decomposition-based bi-objective optimization for sustainable robotic assembly line balancing problems,” *Journal of Manufacturing Systems*, vol. 55, pp. 30–43, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0278612520300157>

- [28] B. Zhang, L. Xu, and J. Zhang, “Balancing and sequencing problem of mixed-model U-shaped robotic assembly line: Mathematical model and dragonfly algorithm based approach,” *Applied Soft Computing*, vol. 98, p. 106739, Jan. 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1568494620306773>
- [29] H. Soysal-Kurt and S. K. İşleyen, “Multi-objective optimization of cycle time and energy consumption in parallel robotic assembly lines using a discrete firefly algorithm,” *Engineering Computations*, vol. 39, no. 6, pp. 2424–2448, Jan. 2022, publisher: Emerald Publishing Limited. [Online]. Available: <https://doi.org/10.1108/EC-12-2020-0747>
- [30] L. Belkharroubi and K. Yahyaoui, “Solving the energy-efficient Robotic Mixed-Model Assembly Line balancing problem using a Memory-Based Cuckoo Search Algorithm,” *Engineering Applications of Artificial Intelligence*, vol. 114, p. 105112, Sep. 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0952197622002494>
- [31] C. Zhang, J. Dou, and P. Wang, “Configuration design of reconfigurable single-product robotic assembly line for capacity scalability,” *Computers and Industrial Engineering*, vol. 185, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0360835223007064>
- [32] Y. Lahrichi, D. Damand, L. Deroussi, N. Grangeon, and S. Norre, “Investigating two variants of the sequence-dependent robotic assembly

- line balancing problem by means of a split-based approach,” *International Journal of Production Research*, vol. 61, no. 7, pp. 2322–2338, Apr. 2023. [Online]. Available: <https://doi.org/10.1080/00207543.2022.2062266>
- [33] Y. Li, Z. Qiao, Y. Chi, L. Guo, and R. Yan, “Robotic assembly line balancing considering the carbon footprint objective with cross-station design,” *Computers & Industrial Engineering*, vol. 190, p. 110045, Apr. 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835224001669>
- [34] Z. Li, M. N. Janardhanan, Q. Tang, and P. Nielsen, “Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem,” *Advances in Mechanical Engineering*, vol. 8, no. 9, p. 1687814016667907, Sep. 2016. [Online]. Available: <https://doi.org/10.1177/1687814016667907>
- [35] Z. Li, N. Dey, A. S. Ashour, and Q. Tang, “Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem,” *Neural Computing and Applications*, vol. 30, no. 9, pp. 2685–2696, Nov. 2018. [Online]. Available: <https://doi.org/10.1007/s00521-017-2855-5>
- [36] M. Aghajani, R. Ghodsi, and B. Javadi, “Balancing of robotic mixed-model two-sided assembly line with robot setup times,” *The International Journal of Advanced Manufacturing Technology*, vol. 74, no. 5, pp. 1005–1016, Sep. 2014. [Online]. Available: <https://doi.org/10.1007/s00170-014-5945-x>

- [37] Z. Li, M. N. Janardhanan, Q. Tang, and S. G. Ponnambalam, “Model and metaheuristics for robotic two-sided assembly line balancing problems with setup times,” *Swarm and Evolutionary Computation*, vol. 50, p. 100567, Nov. 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2210650218308460>
- [38] . Aslan, “Mathematical model and a variable neighborhood search algorithm for mixed-model robotic two-sided assembly line balancing problems with sequence-dependent setup times,” *Optimization and Engineering*, vol. 24, no. 2, pp. 989–1016, Jun. 2023. [Online]. Available: <https://doi.org/10.1007/s11081-022-09718-3>
- [39] Y. Zhang, X. Hu, and C. Wu, “Heuristic Algorithm for Type II Two-sided Assembly Line Rebalancing Problem with Multi-objective,” *MATEC Web of Conferences*, vol. 175, p. 03063, 2018, publisher: EDP Sciences. [Online]. Available: https://www.matec-conferences.org/articles/mateconf/abs/2018/34/mateconf_ifcae-iot2018_03063/mateconf_ifcae-iot2018_03063.html
- [40] —, “A modified multi-objective genetic algorithm for two-sided assembly line re-balancing problem of a shovel loader,” *International Journal of Production Research*, vol. 56, no. 9, pp. 3043–3063, May 2018. [Online]. Available: <https://doi.org/10.1080/00207543.2017.1402136>
- [41] G. Mavrotas, “Effective implementation of the ϵ -constraint method in Multi-

- Objective Mathematical Programming problems,” *Applied Mathematics and Computation*, vol. 213, no. 2, pp. 455–465, Jul. 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0096300309002574>
- [42] A. Rezaei, N. Manavizadeh, M. Rabbani, and S. Mohammadzade, “Multi-Objective Two-Sided Robotic Mixed-Model Assembly Line Balancing Problem Considering Energy Consumption and Smoothing Workload,” *Advances in Industrial Engineering*, vol. 54, no. 4, pp. 401–421, Oct. 2020. [Online]. Available: <https://doi.org/10.22059/jieng.2021.325607.1780>
- [43] M. Asghari, A. M. Fathollahi-Fard, S. M. J. Mirzapour Al-e hashem, and M. A. Dulebenets, “Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey,” *Mathematics*, vol. 10, no. 2, p. 283, Jan. 2022. [Online]. Available: <https://www.mdpi.com/2227-7390/10/2/283>
- [44] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/996017>
- [45] H. Babazadeh, M. H. Alavidooost, M. H. Fazel Zarandi, and S. T. Sayyari, “An enhanced NSGA-II algorithm for fuzzy bi-objective assembly line balancing problems,” *Computers & Industrial Engineering*, vol. 123, pp. 189–208, Sep. 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835218302870>

- [46] S. Kirkpatrick, C. D. J. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[Online]. Available: <https://pubmed.ncbi.nlm.nih.gov/17813860/>
- [47] S. Kulturel-Konak, A. Smith, and B. Norman, "Multi-objective tabu search using a multinomial probability mass function," *European Journal of Operational Research*, vol. 169, no. 3, pp. 918–931, 2006. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0377221704005697>
- [48] D. Khorasanian, S. Hejazi, and G. Moslehi, "Two-sided assembly line balancing considering the relationships between tasks," *Computers and Industrial Engineering*, vol. 66, no. 4, pp. 1096–1105, 2013.
[Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S036083521300260X>
- [49] D. C. Montgomery, *Design and Analysis of Experiments*. John Wiley & Sons, 2017.
- [50] G. Taguchi, *Introduction to quality engineering: designing quality into products and processes*. Asian Productivity Organization, 1986. [Online]. Available: <https://trid.trb.org/View/1179550>
- [51] M. H. Alavidooost, M. H. F. Zarandi, M. Tarimoradi, and Y. Nemati, "Modified genetic algorithm for simple straight and U-shaped assembly line balancing with fuzzy processing times," *Journal of Intelligent*

- Manufacturing*, vol. 28, no. 2, pp. 313–336, Feb. 2017. [Online]. Available: <https://doi.org/10.1007/s10845-014-0978-4>
- [52] M. Alavidoost, M. Tarimoradi, and M. Zarandi, “Fuzzy adaptive genetic algorithm for multi-objective assembly line balancing problems,” *Applied Soft Computing Journal*, vol. 34, pp. 655–677, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S1568494615003464>
- [53] P. Chutima and P. Chimklai, “Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge,” *Computers and Industrial Engineering*, vol. 62, no. 1, pp. 39–55, 2012.
- [54] H. D. Purnomo and H.-M. Wee, “Maximizing production rate and workload balancing in a two-sided assembly line using Harmony Search,” *Computers & Industrial Engineering*, vol. 76, pp. 222–230, Oct. 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360835214002186>

VITAE

- Name: Bayan Ahmed Alzain
- Nationality: Saudi
- Date of Birth: 20/04/1998
- Email: *bayanwork19@outlook.com*
- Permenant Address: Building Number 6629, Street: Sharf Adeen Alshareef,
Postal Code: 23233, Jeddah
- Experience: King Fahd University of Petroleum and Minerals, Graduate
Assistant, Dhahran, Saudi Arabia, 2022 – Present
- Research interests:
 - Optimization
 - Robotic Assembly Lines
 - Mathematical Modeling
 - Multi-Objective Optimization
 - Meta-Heuristics Approaches